CMP

Windows 95 API How-To

Windows 95
API开发人员指南

(美) Mattbew Tells
Andrew Cooke 著

计算机软件开发
与程序设计
系列丛书

机械工业出版社 西蒙与舒斯特国际出版公司

计算机软件开发与程序设计系列丛书

Windows 95 API 开发人员指南

（美）Mattbew Telles
Andrew Cooke 著

吴冰 姚彦忠 等译

孙绍麟 审校

机 械 工 业 出 版 社
西蒙与舒斯特国际出版公司

JS146/04

本书以问答的形式组织起来100多个有关Windows 95 API使用技巧的专题。涉及到的范围包括：系统的信息，文件和目录，任务调度，图形设备接口，打印，应用程序间通信，多媒体，以及对话框、编辑控制、列表框、菜单等界面组件。介绍了一些窗口操作的实现，如：置前，置后，改变大小和移动位置等。同时还介绍了一些编程技巧以及如何完善应用程序。在每个专题中都有具体的例子程序的实现，既为读者示范了实现的步骤，同时还为读者提供了进一步开发的模板。

本书主要面向中高级开发人员，同时也考虑到了初学者，因此本书对不同层次的读者都是适用的。

Mattbew Telles, Andrew Cooke: Windows 95 API How-to.
Authorized translation from the English edition Published by Waite Group Press.
Copyright 1996 by Waite Group Press.
All rights reserved. For sale in Mainland China only.

本书中文简体字版由机械工业出版社和美国西蒙与舒斯特国际出版公司合作出版，未经出版者书面许可，本书的任何部分不得以任何方式复制或抄袭。
本书封面贴有Prentice Hall防伪标签，无标签者不得销售。
版权所有，翻印必究。

本书版板登记号：图字：01-96-1224

图书在版编目（CIP）数据

Windows 95 API开发人员指南/（美）特利斯（Telles，M.），（美）库克（Cooke，A.）著；吴冰等译.—北京：机械工业出版社，1997.6
（计算机软件开发与程序设计系列丛书）
书名原文：Windows 95 API How-to
ISBN 7-111-05463-6

Ⅰ.W… Ⅱ.①特… ②库… ③吴… Ⅲ.软件接口，API-指南 Ⅳ.TP334

中国版本图书馆CIP数据核字（97）第05475号

出 版 人：马九荣（北京百万庄南街1号 邮政编码100037）
责任编辑：何伟新
三河永和印刷有限公司印刷·新华书店北京发行所发行
1997年6月第1版第1次印刷
787mm×1092mm 1/16·42.25印张·1039千字
0001-7000册
定价：74.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

译 者 序

Windows 95 的推出曾在世界各地引起了很大的轰动，它那强大的功能和新颖的用户界面赢得了广大用户的青睐。当前，大量的 16 位应用程序纷纷移植到这个新的 32 位平台上，而且更多的新软件正在开发之中。尽管在此 32 位平台上已有不少优秀的可视开发环境，如：Visual C++、Delphi、Visual Basic 等等。但是，在有些情况下直接使用 Windows 95 API 来开发应用程序可能会更高效、更实用。而且，有些功能在可视开发环境中是难以实现的，必须直接使用 API 函数来实现。因此，作为当前的 Windows 程序员，了解 Windows 95 API 是很有必要的。

本书是以一种非常容易查阅的问答形式组织起来的，详细介绍了 Windows 95 API 的使用技巧。涉及的范围非常广泛，不仅包括 Windows 95 中的新特征，而且还包括常用的 Windows 功能。因此本书不仅便于 Windows 程序员迅速转向新的平台，同时也适用于 Windows 编程的初学者。在本书的翻译过程中，由于译者的水平有限，错误和不妥之处难免，恳请广大读者指正。

参加本书翻译的有：孙绍麟老师翻译了前言，吴冰翻译了第 1、2、3、4、7、12、15 章，姚彦忠翻译了第 8、9、10、11、13、14 章，李跃宁翻译了第 5 章和第 6 章。孙绍麟老师最后审校了全书。译者对华章公司的编辑们所提供的帮助和支持表示感谢。

前　　言

起初，Windows 应用编程接口（API）曾风靡一时。程序员对 API 称道“好极了！”，他们借助 API 开发了 Windows 程序，用户对 Windows 程序情不自禁地说：“真棒！”。由数以百计称为“函数”的魔块所组成的 API 似乎至高无上，随后，API 轰动了计算机界。

当 Windows 程序设计领域处于发展初期时，Windows 程序员可使用的编程工具唯有 API 函数。但这些函数难以理解，易于误用，还会导致出错。

随着可视编程环境的到来情况发生了巨变，程序员能采用拖放（drag and drop）方式来开发完美的全功能应用程序，它们的界面友好、操作简便。Borland 和 Microsoft 创建的类库替代了 API 的神秘功能。其实，在当今应用程序的类库和定制控制是构筑在 Windows API 的基础上的。这些类库和函数加速了 Windows 应用编程的开发。程序员第一次把梦想做的事情变成了现实，而不必忧虑必须编制数以千行计的代码。这也导致了非常多的程序员在类库前面“故步自封”，对下层 API 函数的强大功能一无所知。

但当前，如果程序员要编写符合当代标准的程序，则没有比理解 API 的性能更重要。虽然类库和定制控制使开发应用程序容易得多，但它们只触及到 Microsoft 的 Windows 系统功能的皮毛。

本书的宗旨是指导用户去实现只利用 API 调用来完成开发应用程序的所有技巧。读者不仅会学到如何深入 Windows 的内部，而且将会鉴赏到这个图形用户界面的强大功能和灵活性。可以这样说，程序员只有真正理解 Windows API 的内含，才能成功地扩展那些类库和可视开发环境。

本书以烹饪书的编写方式展开。因而，读者可以根据当前应用程序开发的需要，随意跳到有关任务的专门章节去参阅所需的内容。

本书共分为十五章，每章集中讨论应用编程接口函数的一个独立部分。

第 1 章：获取系统信息。本章介绍有关确定正在运行的计算机处理器，正在使用的 Windows 版本及应用程序可用的内存有多少等的简单方法。另外，还将讨论找出按下哪些键和接通 NUMLOCK 键的编程小技巧。本章中还说明如何确定计算机的网络名（如果联网的话）及如何查找当前登录的用户名。最后，将考虑在 Windows 95 中新设立的注册表（Registry），并说明在应用程序中如何利用这一功能。

第 2 章：文件和目录。本章介绍如何处理在 Windows 95 下的严重错误，以及如何获取有关磁盘、驱动器和目录方面的信息。并将说明如何识别 CD 为音乐、照片或普通的数据光盘。另外，还介绍有关如何清除磁盘缓冲区和拷贝文件。

第 3 章：应用程序和任务控制。本章阐述包括查找当前运行在系统上的其他程序或任务，以及启动或终止任务所需步骤的一些技巧和提示。并将说明如何确保在给定时间内只运行应用程序的单一实例，如何实现后台处理，以及如何从应用程序中终止和重新启动 Windows 系统。

第 4 章：绘图和图形设备接口（GDI）。本章将阐述 GDI 的来龙去脉，以及如何利用 Win-

dows API 来描绘一个优美的世界。这里还将讨论装入、显示、缩放和旋转位图，以及利用鼠标来绘制“橡皮带线”。同时还介绍如何利用简单方法和新的 WinG 图形库来绘制图表和图形，甚至创作动画。

第 5 章：对话框。本章将讨论如何利用对话框作为应用程序的主窗口，如何处理在对话框内的快捷键，如何改变对话框中的字体和颜色及对话框中的单独控制。另外将说明如何通过显示和隐藏控制来激活和禁止对话框中控制及改变对话框的过程。Windows 95 的核心功能之一是广泛应用的对话框内属性单（property sheets），这里将说明如何能在应用程序中内含这些特性。

第 6 章：编辑控制。本章介绍有关如何实现只读编辑控制，以及如何能够获取口令，查找和替换文本，添加文本，撤消用户动作和确认输入。

第 7 章：列表框。本章介绍如何能在应用程序中使列表框达到最大程度的实用，如何在纵向和横向滚动列表框，如何在列表框内添加多列，以及如何创建自绘制的列表框。此外，将考察有关 Windows 95 的新的树（Tree）控制及说明如何在应用程序中有效地利用这种控制。

第 8 章：菜单。本章介绍如何能激活、禁止、添加和删除菜单项。并说明在菜单中如何选择项及如何改变核选框的外观。最后，将讨论有关应用程序窗口的系统菜单及如何修改它。

第 9 章：文档和编辑器。本章介绍如何自动创建新的 MDI 窗口，如何利用对话框作为 MDI 子窗口，以及如何能够利用简单的 Windows 95 API 命令来实现一个完整的文本编辑器的基本功能。在本章中还考察了在 Windows 95 中新的富文本编辑（Rich Text Edit）控制及在应用程序中利用它时必须做什么。

第 10 章：打印。本章涉及读者感兴趣的几个功能，包括查找已安装了哪些打印机，并确定其性能。另外还介绍了如何将用户数据复写于文件中，以及如何查找任何特定打印机的可用字体。

第 11 章：应用程序之间的通信。本章涉及有关 DDE、OLE 和 Windows 剪贴板的讨论。对这些在 Windows 95 中较难的概念提供了广泛的例子。

第 12 章：音效和音乐。本章中介绍有关播放声音，读光盘信息及记录 MIDI 信息等。

第 13 章：有关窗口的应用。本章介绍编写屏幕保护程序的基础，还讨论了使用户窗口如何总是保持在顶部或底部，并如何移动和缩放它们。另外还介绍了如何装入其他应用程序中的图标信息，以及如何改变用户自己的程序图标。

第 14 章：程序设计的技巧。本章讨论有关确认指针，置版本信息于文件中，以及利用和装入动态连接库等。

第 15 章：完善应用程序。本章包括了有关利用状态条、工具条、在线帮助等信息。并介绍如何使程序图标动起来，以及如何在对话框或窗口的背景上显示位图。

本书面向的主要对象

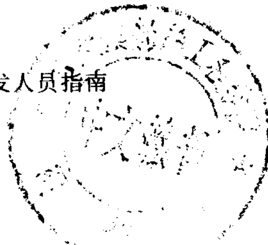
本书宗旨是给习惯于继续使用可视开发环境的高级程序员作为媒介来参考。本书目的是为了解决在 Windows 应用程序的正规开发中所遇到的问题。如果开发人员正在使用 Visual C＋＋、Visual Basic、或 Delphi 创建新的应用程序，而且要求超出 MFC、VB 控制及 Delphi 组件所提供范围的功能，那么本书将提供帮助。

此外，本书对于那些愿意进一步提高 Windows 编程知识的学生和业余爱好者也是一本优

秀的参考书。在此，给出的大量具体例子适合于各种不同层次的程序员所需的程序设计问题和方法。

最后，本书分成两种编码“风格”。第一种风格是在单纯的C环境下使用原始的API函数编程的例子，这种风格适合于那些习惯于使用没有一个与“实际”操作系统相分离的高层库的程序员。第二种风格是利用Visual C++的MFC开发环境编程的例子，这些例子是针对那些要求MFC的强大功能但需要超出Microsoft所支持的功能范围的程序员。在本书中汇集了这两种不同的编程风格，为应用程序的开发提供了一个自由创造的理想天地。

405960



The Microsoft Windows95 开发人员指南	86.00
Visual Basic4 开发人员指南	73.00
Visual FoxPro3.0 开发指南	119.00
Visual Basic4.0API 程序设计	83.00
I/O 接口程序设计入门与应用	28.00
精通 VISUAL BASIC4.0 多媒体程序设计	15.00

Internet 实务系列丛书

快快乐乐畅游 Internet	34.00
WWW 文件设计-HTML 语言实务	29.00
Java/VRML 设计大全	32.00
Web 动态技术入门	23.00
Java 轻松上手	26.00
WWW 上站与架站实务	20.00
Mosaic 与 WWW	10.00
Archie-Internet 标题检索工具	10.00
Netscape Navigator Gold3.0 与网络资源搜寻	20.00
JavaScript 快速查询手册	12.00

北京华章图文信息有限公司
地址：北京市百万庄南街 14 号
邮政编码：100037
电话：68326444　　传真：68326444
E-mail：huazhang@public3.bta.net.cn
户名：北京华章图文信息有限公司
帐号：014-144738-13-47
开户行：北京工商行百万庄分理处信息院代办处
增值税号：110102101444065

目　录

前言

译者序

第1章　获取系统信息

1.1　确定 Windows 的当前版本 …… 2
1.2　获取有关显示器、鼠标及系统的配置信息 …… 3
1.3　确定计算机的处理器类型 …… 7
1.4　确定多少系统内存是可用的 …… 8
1.5　获取系统上的用户注册信息 …… 9
1.6　获取计算机的网络名 …… 12
1.7　找出计算机上当前登录入网的用户 …… 15
1.8　找出键盘上当前按下的键 …… 18
1.9　使用 Windows 95 的注册表存取信息 …… 22
1.10　找出程序或 DLL 的版本号 …… 30

第2章　文件和目录

2.1　查找 Windows 目录和 System 目录 …… 36
2.2　显示 I/O 错误信息 …… 41
2.3　检测驱动器中是否有盘 …… 50
2.4　简单地实现文件拷贝 …… 56

第3章　应用程序和任务控制

3.1　找出系统上正在运行的任务 …… 66
3.2　激活另一任务 …… 73
3.3　关闭其他的应用程序 …… 78
3.4　找出应用程序的执行文件名 …… 83
3.5　确保应用程序同时只能运行一个实例 …… 85
3.6　在后台运行其他任务 …… 88
3.7　启动另外的程序并等待其运行结束 …… 94
3.8　从应用程序中终止和重启动 Windows …… 98

第4章　绘图和图形设备接口

4.1　显示 256 色的位图 …… 104
4.2　改变位图中的颜色 …… 116
4.3　旋转位图 …… 128
4.4　随鼠标拖动绘制“橡皮带线” …… 144
4.5　捕捉窗口或部分屏幕 …… 155
4.6　生成动画 …… 165
4.7　实现位图的拖放 …… 178

第5章　对话框

5.1　利用对话框作为应用程序的主窗口 …… 191
5.2　改变对话框中的字体 …… 197
5.3　改变对话框中控制的字体 …… 203
5.4　改变对话框的背景颜色 …… 208
5.5　激活和禁止对话框中的控制 …… 211
5.6　改变对话框中显示的控制 …… 218
5.7　在对话框中使用属性单 …… 221
5.8　在对话框中的按钮上绘制图象 …… 225
5.9　在执行另一操作的同时显示“进度”对话框 …… 228

第6章　编辑控制

6.1　使编辑控制只读 …… 236
6.2　利用编辑控制获取口令 …… 243
6.3　改变编辑控制的背景颜色 …… 249
6.4　在编辑控制中替换文本 …… 256
6.5　给编辑控制添加撤消功能 …… 262
6.6　确认编辑控制中的输入有效 …… 269
6.7　利用剪贴板进行剪切和粘贴操作 …… 277

第7章　列表框

7.1　随同列表项存储信息 …… 287
7.2　捕捉列表框中的双击 …… 295
7.3　通过拖放在列表框中移动列表项 …… 298
7.4　滚动列表框 …… 305
7.5　实现宽列表的水平滚动 …… 308
7.6　在列表框中右对齐数字 …… 311
7.7　实现自绘制列表框 …… 318
7.8　在列表框中存放更多的列表项 …… 324
7.9　实现层次列表（树） …… 344

第8章　菜单

8.1　激活和禁止菜单项 …… 356
8.2　添加和删除菜单项 …… 364
8.3　为菜单项添加核选标记 …… 374

8.4 利用自己定义的核选标记 …………… 382
8.5 为系统菜单添加选项 ………………… 397
8.6 设计点击鼠标右键后弹出的菜单 …… 403

第9章 文档和编辑器

9.1 自动打开MDI子窗口 ………………… 416
9.2 在MDI窗口中利用无模式对话框 …… 419
9.3 随MDI子窗口调整列表框或编辑控制的大小 ………………………………… 422
9.4 创建简单的文件阅览器 ……………… 426
9.5 查找和替换文本 ……………………… 430
9.6 显示大于64KB的文件 ……………… 435
9.7 改变编辑控制中插入光标的类型 …… 458
9.8 改变插入光标的闪烁速度 …………… 461

第10章 打印

10.1 确定打印机的性能 …………………… 465
10.2 确定当前打印机的页面大小和方向 ……………………………………… 469
10.3 利用通用打印机对话框 ……………… 472
10.4 打印到文件 …………………………… 477
10.5 确定可用的打印机字体 ……………… 479
10.6 确定打印队列的状态 ………………… 482

第11章 应用程序之间的通信

11.1 支持从系统程序拖放到应用程序 …… 487
11.2 支持从应用程序拖放到另一应用程序 ……………………………… 490
11.3 查看剪贴板的内容 …………………… 496
11.4 利用剪贴板进行剪切、拷贝和粘贴 ……………………………………… 499
11.5 编写动态数据交换客户程序 ………… 502
11.6 编写动态数据交换服务程序 ………… 506
11.7 在动态数据交换中支持系统主题 …… 512
11.8 使文件对象的链接和嵌入（OLE）兼容 ……………………………………… 547
11.9 创建OLE服务程序对象 …………… 552

第12章 音效和音乐

12.1 播放音效 ……………………………… 555
12.2 读取CD中的曲目信息 ……………… 560
12.3 播放CD音乐 ………………………… 565
12.4 播放MIDI音乐 ……………………… 574

第13章 有关窗口的应用

13.1 编写屏幕保护程序 …………………… 584
13.2 创建无标题条的窗口 ………………… 590
13.3 保持一个窗口在所有其他窗口的上面 ……………………………………… 593
13.4 把一个窗口移到所有其他窗口的下面 ……………………………………… 595
13.5 改变鼠标光标的形状 ………………… 596
13.6 改变应用程序窗口极小化时的图标 ……………………………………… 599
13.7 装入另一应用程序的图标 …………… 601
13.8 移动和缩放应用程序中的窗口 ……… 605

第14章 程序设计的技巧

14.1 确定指针是否有效 …………………… 610
14.2 确定字符串是否有效 ………………… 612
14.3 在应用程序中放置版本信息 ………… 614
14.4 编写动态连接库 ……………………… 617
14.5 利用动态连接库 ……………………… 622
14.6 创建各种分辨率下都能显示的应用程序 ……………………………… 624

第15章 完善应用程序

15.1 实现上下文相关帮助 ………………… 627
15.2 创建状态条 …………………………… 631
15.3 创建工具条 …………………………… 634
15.4 实现在运行时修改工具条 …………… 637
15.5 从在线帮助中显示范例 ……………… 642
15.6 在应用程序启动时显示About框 … 647
15.7 显示扉屏（splash screen） ………… 653
15.8 确定应用程序的图标 ………………… 660
15.9 显示作为窗口或对话框背景的位图 ……………………………………… 662

第1章　获取系统信息

设计一个应用程序，运行在各种版本的 Windows 操作系统上，与简单地设计数据表格和菜单相比，程序员需要考虑更多方面的问题。现在的用户都希望应用程序能自动地检测运行环境，比如自动测试运行的 Windows 版本，自动测试屏幕的大小以便将窗口和数据表格置于屏幕的中央，以及自动获取某些网络信息用于应用程序中。更重要的是，应用程序要能默认完成某些操作，而不需要用户的介入。如果如此的话，用户将不需要为每一个应用程序配置键盘，或为每一个应用程序确定 CAPSLOCK 键和 NUMLOCK 键是开状态还是闭状态。

应用程序的所有这些小调整的实现，以及要确保用户操作错误及设置错误不会引起混乱和灾难，这些都是程序员的责任。

1. 如何确定 Windows 的当前版本

如果应用程序使用的某些特性依赖于运行的 Windows 版本，便需要确定用户机器上安装的 Windows 版本。在本节中将介绍如何确定用户的操作系统是 Windows 3.x，Windows 95，还是 Windows NT。

2. 如何获取有关显示器、鼠标及系统的配置信息

确定这些信息是非常重要的：用户是否使用鼠标（因为键盘是很难实现双击的），使用多少磁盘驱动器及什么类型，显示器的大小。所有这些信息都可利用 Windows API 函数得到。在本节中将介绍如何查看磁盘驱动器的数目及其可能的类型，如何确定是否安装了鼠标及用户是否互换了左右键。最后，将介绍如何获取屏幕的大小及如何使窗口或对话框居中。

3. 如何确定计算机的处理器类型

确定用户计算机的处理器类型是很有用的，可以将此信息保存起来，以便于测试程序存取，或用于桌面帮助信息中。假如用户需要各种应用程序信息时，只需简单地按一下键，便将弹出一个对话框，显示用户需要的所有信息。这是使应用程序界面友好的绝妙途径。

4. 如何确定多少系统内存是可用的

在设计一个使用大量系统内存的应用程序时，最好在其运行前能检查是否有足够的资源。为了防止严重的内存和磁盘操作崩溃，最好不断确认程序是否用光了系统内存或其他系统资源。在本节中将介绍如何确定可用的系统内存数，以防止内存和磁盘操作崩溃。

5. 如何获取系统上的用户注册信息

专业版的程序一般能自动检测 Windows 软件的原注册用户。任何人都能取到此信息，但需要巧用 Windows API 函数。在本节中将介绍如何从 Windows 中获取此信息，同时介绍如何巧取其他程序的资源以用于自己的应用程序中。

6. 如何获取计算机的网络名

在任何网络环境中，最好能知道所使用的计算机的网络名。可以使用 Windows API 函数来获得此信息，用于显示或用于用户登录系统。在本节中将介绍如何获取计算机的当前网络名。

7. 如何找出计算机上当前登录入网的用户

一旦获取了计算机网络名，自然也想获取当前登录的用户名。此信息可用于显示，或用在应用程序中定制一些基于用户名的选项。这样用户便可简单地登入计算机，使所有的程序都按自己的需求来运行。这些都可使用 API 函数来实现。

8. 如何找出键盘上当前按下的键

假如用户要滚动复杂的文档，如果应用程序并不检查键↓按下的同时用户是否按下键 CTRL 或 SHIFT，则显然是非常不方便的。而且，你可以想象假如用户同时按下几个键而应用程序却少记录了一个，这样的应用程序用户能使用吗？在本节中将介绍如何检测键盘上所有键的状态。

9. 如何使用 Windows 95 的注册表存取信息

如果你习惯于使用 Windows 3.x API 编程，那么就可能对初始化文件（.INI）进行过大量的操作。这些文件很容易被破坏，或是因为用户使用了文本编辑器编辑，或是因为被其他的应用程序覆盖。Windows 95 通过引入注册文件的思想解决了此问题。应用程序利用 Windows 注册函数来维护此文件。在本节中将介绍如何在注册文件中添加新项，以及如何从此文件中读出已存在的项。

10 如何找出程序或 DLL 的版本号

迟早，所有的程序员都会被过时文件的问题所困扰。有时，此问题只是烦人，应用程序不能按要求运行。有时则是灾难性的：破坏数据，内存操作崩溃或死机。在本节中将介绍如何获取文件的版本号，及如何把此信息放入应用程序中以便在运行时进行确认。

表 1-1 列出了在本章中使用的 Windows 95 API 函数。

表 1-1 在第 1 章中使用的 Windows 95 API 函数

GetVersionEx	VerQueryValue	GetFileVersionInfoSize	GetFileVersionInfo
GetSystemInfo	GetDriveType	GetSystemMetrics	
RegOpenKeyEx	GlobalMemoryStatus	RegOpenKey	
WNetGetConnection	RegEnumValue	RegCloseKey	
RegEnumKey	WNetGutUser	GetKeyboardState	
RegCreateKey	RegQueryValue	RegSetValue	

1.1 确定 Windows 的当前版本

问题

有时，程序员需要确定哪一个版本的 Windows 正在运行，以便根据版本号而采取相应的步骤。

方法

已编入文档的 Windows API 函数 GetVersionEx 可以用来确定哪一个版本的 Microsoft Windows 操作系统正在运行。Windows NT 表示为 3.51，指当前发行的 Windows NT 3.51；Windows 95 表示为 Windows 4.0。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从菜单 SysInfo 中选择菜单项 Windows Version，将弹出一个对话框，显示 Windows 的版本号。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH11.MAK。

2. 进入 AppStudio，在菜单 IDR_MAINFRAME 中添加新的菜单 SysInfo。在 SysInfo 中添加新的菜单项 &Windows Version，其对话框 ID 为 ID_WIN_VERSION。

3. 启动 ClassWizard。从下拉列表中选择对象 CMainFrame，从对象列表中选择 ID_WIN_VERSION，选择消息 COMMAND 以定义消息处理。点击按钮 Add Function，键入函数名 OnWinVersion，然后选择按钮 Edit Code。

4. 在 OnWinVersion 中，输入下列代码：

```
void CMainFrame:: OnWinVersion ()
{
    char buffer [80];
    OSVERSIONINFO osinfo;

    // Get version info

    osinfo.dwOSVersionInfoSize = sizeof (OSVERSIONINFO);
    GetVersionEx (&osinfo);

// Break into the major and minor versions of Windows

    sprintf (buffer, "Version %ld. %ld", osinfo.dwMajorVersion, osinfo.dwMinorVersion);

    MessageBox (buffer, "Windows", MB_ICONINFORMATION | MB_OK);

}
```

用法

Windows API 函数 GetVersionEx 返回一个包含版本信息的结构给应用程序。在本节中，我们只对结构中的域 dwMajorVersion 和 dwMinorVersion 感兴趣，这两个域包含 DWORD 值（即双整数值）。

应该注意，调用函数 GetVersionEx 前必须先在域 dwOSVersionInfoSize 中设置结构的大小，这是为了将来结构的改变而不影响已有的代码，以后新增的所有域都将附加在结构的末尾，而函数将只返回程序员所要求的部分。

注释

对当前所支持的 Windows 95 版本，函数 GetVersionEx 的返回值为 4.0。对当前版本的 Windows NT，其返回值为 3.51。函数 GetVersionEx 是专门的 32 位函数，不能用在 Windows 3.x 中。要得到 Windows 3.x 的版本号，应该使用 16 位的函数 GetVersion。

1.2 获取有关显示器、鼠标及系统的配置信息

问题

有时，程序员希望能够确定用户目前正在使用的是什么配置，此信息应该包括驱动器的数目和类型，以及用户是否使用鼠标，当然，最好还能够确定用户使用的显示器的类型以及显示器的分辨率。

方法

确定所有的信息需要好几步。首先可以调用 Windows API 函数 GetDriveType 来确定机器安装了哪些驱动器以及它们是什么类型（硬盘，CD-ROM，软盘，或网络驱动器）。

使用 Windows API 函数 GetSystemMetrics 可以获取信息的第二部分（是否有鼠标以及显示器的大小）。此函数返回给程序员大量的信息，在本节中我们只使用其中的一部分。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 SysInfo 以及菜单项 Drives，将弹出一个对话框（如图 1-1 所示）。对话框包含所有可能的驱动器符（A～Z）及对应驱动器的类型。类型是下列中的一个：

- 不确定的：此驱动器可能在系统中不存在。
- 可移动驱动器：通常为软盘驱动器。
- 固定驱动器：通常为硬盘驱动器或 CD-ROM。
- 网络驱动器或 CD-ROM：通常为网络驱动器，但某些 CD-ROM 驱动器也归为此类。

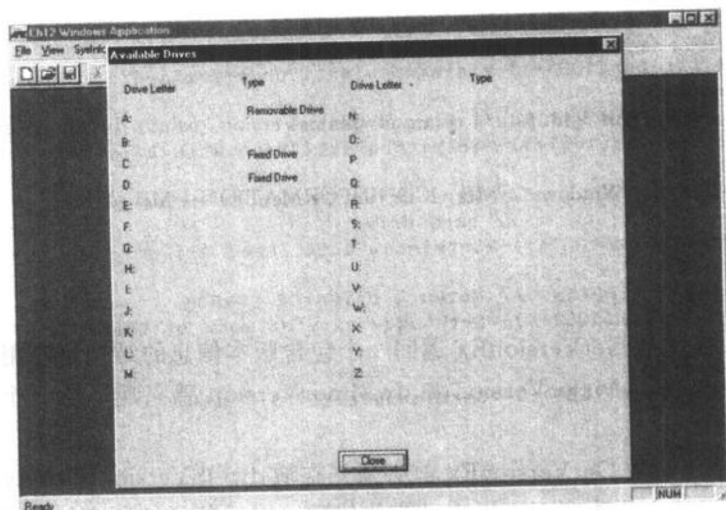


图 1-1　系统可用的驱动器

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH12.MAK。利用 AppStudio，创建新的菜单 SysInfo。在菜单 SysInfo 中添加菜单项 Drives，标识符为 ID_DRIVE_INFO。

2. 在 AppStudio 中，创建新的对话框 IDD_DIALOG1。添加 26 个静态文本域，标题为驱动器符（A:，B:，C: 等等）。与其对齐，添加另外 26 个静态文本域。与驱动器符同顺序，ID 分别赋为 1001，1002 等等。

3. 在 AppStudio 中，选择 ClassWizard，然后选择 New Class 来创建新的对话框类。新类

命名为 CDriveDlg，并接受所有其他的缺省值。为消息 WM_INITDIALOG 添加一个函数（命名为 OnInitDialog）。

4. 在刚创建的函数 OnInitDialog 中，添加下列代码：

```
BOOL CDriveDlg:: OnInitDialog ()
{
    CDialog:: OnInitDialog ();
    char buffer [10];

    for ( int i=0; i<26; ++i ) {
      sprintf (buffer, "%c: \\", 'A' + i );
        WORD ret = GetDriveType (buffer);
        switch ( ret ) {
         case 1: // This indicates this is beyond the last drive defined.
            break;
         case 0: // Non-determined drive type
           GetDlgItem (1001+i) ->SetWindowText ("Undetermined");
           break;
         case DRIVE_REMOVABLE: // Floppy drives
           GetDlgItem (1001+i) ->SetWindowText ("Removable Drive");
           break;
         case DRIVE_FIXED: // Hard drive OR mapped drive on hard-drive
           GetDlgItem (1001+i) ->SetWindowText ("Fixed Drive");
           break;
         case DRIVE_REMOTE: // Network drive OR CD-ROM
           GetDlgItem (1001+i) ->SetWindowText ("Network or CDROM");
          break;
         default:
           GetDlgItem (1001+i) ->SetWindowText ("");
           break;
        }
        }

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

5. 进入 ClassWizard，从下拉列表中选择对象 CMainFrame，从对象列表中选择对象 ID_DRIVE_INFO，从消息列表中选择消息 COMMAND。点击按钮 Add Function，新函数命名为 OnDriveInfo，在 CMainFrame 的函数 OnDriveInfo 中添加下列代码：

```
void CMainFrame:: OnDriveInfo ()
{
```

```
    CDriveDlg dlg;

    dlg.DoModal ();
}
```

6. 在文件 MAINFRM.CPP 顶部的 include 文件列表中添加下列 include 文件行：

```
#include "drivedlg.h"
```

7. 编译并运行此例子程序。

用法

API 函数 GetDriveType 通常用来确定安装的驱动器类型。尽管此函数可能会被某些 CD-ROM 驱动器、网络驱动器及软件模拟磁盘驱动器所“欺骗”，但仍是一个确定当前系统上哪一个驱动器符可用的理想方法。这些驱动器接着就可以用在应用程序中。

函数 GetSystemMetrics 通常用来获取多种有关系统及其性能的信息。在本节中只使用此 API 函数获取的某些有用的信息。程序员可以使用此函数来将窗口置于屏幕的中央，同样也可使对话框和显示的信息居中。

注释

使用 Delphi 也可完成同样的任务。按照下列步骤实现一个例子程序，用来获取有关鼠标及显示器的信息。运行此例子程序，将显示出一个包含有关信息的表单。在例子程序中还使用了函数 GetSystemMetrics 来将表单置于屏幕的中央，此函数为获取屏幕大小的有效途径。

按照下列步骤实现此例子程序：

1. 创建新的项目文件 SYSINFO.DPR。在表单中添加三个文本域，标题分别为 Mouse Present、Screen Width 和 Screen Height。添加另外三个文本域，命名为 MousePresent、ScreenWidth 和 ScreenHeight。

2. 双击表单区中的任何地方（除了前面定义的任一文本域外），在表单的方法 FormCreate 中添加下列代码：

```
procedure TForm1.FormCreate (Sender: TObject);
Var
  screen_height, screen_width: Integer;
  ht, wt: Integer;
begin
  screen_width := GetSystemMetrics (SM_CXSCREEN);
  screen_height := GetSystemMetrics (SM_CYSCREEN);

  form1.top := (screen_height div 2) - (form1.height div 2);
  form1.left := (screen_width div 2) - (form1.width div 2);
  if (GetSystemMetrics (SM_MOUSEPRESENT) <> 0) then
    begin
        MousePresent.Caption := 'Yes';
```

```
    end
  else
    MousePresent.Caption := 'No';
  ScreenWidth.Caption := IntToStr (screen_width);
  ScreenHeight.Caption := IntToStr (screen_height);
end;
```

3. 运行此例子程序。显示在屏幕上的表单如图 1-2 所示。

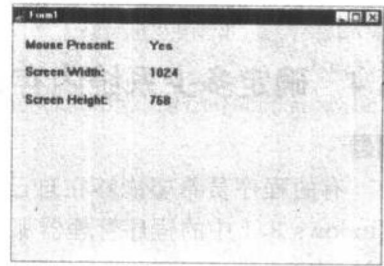


图 1-2　包含屏幕大小和鼠标安装信息并被居中的表单

1.3　确定计算机的处理器类型

问题

有的程序员希望知道运行自己程序的机器安装的是什么处理器，然后就此信息用于桌面帮助信息，或只是用来确定自己的应用程序能否在此机器上运行。

方法

可以使用 Windows API 函数 GetSystemInfo 来获取处理器信息。此函数返回多种信息，其中就包括处理器的类型。此函数识别当前的这个（或这些）（因为 Windows NT 可以运行在多处理机上）处理器为 386、486 或 Pentium。由于 Windows NT 和 Windows 95 不能运行在 80286 或更低型号的处理器上，所以此函数不识别这些处理器型号。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 SysInfo，选择菜单项 Processor Type。将弹出一个消息框，显示当前机器上安装的处理器类型。消息框为下列选择之一：80386，80486，Pentium。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH13.MAK。将菜单 SysInfo 添加到主菜单中，在菜单 SysInfo 中添加新菜单项 &Processor Type，ID 为 ID_PROCESSOR。

2. 打开 ClassWizard。选择对象 CMainFrame，选择对象 ID 为 ID_PROCESSOR，选择消息 ID 为 COMMAND。选取按钮 Add Function，命名方法为 OnProcessor。

3. 选择按钮 Edit Code，在方法 OnProcessor 中输入下列代码：

```
void CMainFrame::OnProcessor ()
{
  SYSTEM_INFO sinfo;

  GetSystemInfo ( &sinfo );
  if ( sinfo.dwProcessorType == PROCESSOR_INTEL_386 )
    MessageBox ( "80386 Processor", "Information", MB_ICONINFORMATION | MB_OK );
  if ( sinfo.dwProcessorType == PROCESSOR_INTEL_486 )
    MessageBox ( "80486 Processor", "Information", MB_ICONINFORMATION | MB_OK );
```

```
  if ( sinfo.dwProcessorType == PROCESSOR_INTEL_PENTIUM )
    MessageBox ( "Pentium Processor", "Information", MB_ICONINFORMATION | MB_OK );
}
```

1.4 确定多少系统内存是可用的

问题

有的程序员希望能够在自己的应用程序的 About 框中显示剩余的系统内存数，就如同 Windows 3.1 中的程序管理器那样，如何来实现呢？

方法

函数 GlobalMemoryStatus 是专为 Windows 95（和 Windows NT）设计的，向程序员提供必要的信息来确定程序是否可在当前的系统配置下运行。无论是物理内存还是虚拟内存都可用此函数来确定，因此应用程序也可以确定在运行期间系统是否会切换存储。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Help|About 并查看弹出的对话框。在对话框的底部有两个数值，分别表示物理内存和虚拟内存的可用字节数。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH14.MAK。

2. 在 AppStudio 中，修改对话框资源 IDD_ABOUTDLG。添加两个静态文本域，标题分别为 Free System Memory 和 Free Virtual Memory。紧挨刚添加的两个静态文本域再添加另外两个静态文本域，标题为空，ID 分别为 ID_MEM_AVAIL 和 ID_VIRTUAL_AVAIL。

3. 进入 ClassWizard。从下拉列表中选择类 CAboutDlg，从对象列表中选择对象 CAboutDlg，从消息列表中选择消息 WM_INITDIALOG。点击按钮 Add Function，在 CAboutDlg 的方法 OnInitDialog 中添加下列代码：

```
BOOL CAboutDlg::OnInitDialog ()
{
  CDialog::OnInitDialog ();
  MEMORYSTATUS memStatus;

  memStatus.dwLength = sizeof (MEMORYSTATUS);

  GlobalMemoryStatus (&memStatus);

  // Get the physical memory free
  DWORD mem = memStatus.dwAvailPhys;
  // Get the virtual memory free
  DWORD res = memStatus.dwAvailVirtual;
  // Set the dialog items
  char buffer [80];
  sprintf (buffer, "%ld", mem );
```

```
  GetDlgItem (ID_MEM_AVAIL) ->SetWindowText ( buffer );
  sprintf (buffer, "%ld", res );
  GetDlgItem (ID_VIRTUAL_AVAIL) ->SetWindowText ( buffer );
  return TRUE;    // return TRUE unless you set the focus to a control
}
```

4. 编译并运行此例子程序。

用法

对应用程序来说，可用的内存总量是机器的实际内存和操作系统的虚拟内存的总和。利用函数 GlobalMemoryStatus，程序员就可以确定在当前的系统上可以运行多大的程序。

应该注意，在使用函数 GlobalMemoryStatus 时，必须将结构（MEMORYSTATUS）的地址传入函数，此结构包含有关机器整个内存状态的信息域。而且在调用此函数前还必须用结构的大小来初始化结构中的某个域（dwLength），以确保将来结构的改变不影响已有的代码。

还应注意的是，在结构 MEMORYSTATUS 中，除了域 dwAvailPhys 和 dwAvailVirtual 外，还有域 dwTotalPhys 和 dwTotalVirtual。查看这两个域可以确定机器上安装的实际内存量和分配的虚拟内存量。

注：在 Windows 系统中，可用的内存字节数是实际内存和虚拟内存的总和，可能与可分配的最大连续块的字节数并不相等。因此在使用任何内存分配函数时应该检查是否内存分配失败。

1.5 获取系统上的用户注册信息

问题

有的程序员希望能够显示当前安装的 Windows 版本的注册用户的信息，此信息包括用户名和单位。但似乎找不到一个 Windows API 函数来完成此功能。

方法

由于没有直接的方法来实现此功能，因此也就找不到单个 Windows API 函数来显示注册信息，只能由程序员以高超的编程技巧联合好几个 Windows API 函数才能完成此任务。

有关 Windows 系统注册用户的姓名和单位的信息实际上是保存在 Windows 系统的注册表中。要获取这些信息，则需要打开注册表，找到正确的条目，然后取回所查找的关键字的值。

在本节中，将介绍如何在新的 Windows 95 注册表数据库中查找适当的关键字及其值，从而完成检索用户信息的任务。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从菜单 SysInfo 中选择菜单项 Registration，则弹出一个对话框（如图 1-3 所示），显示 Windows 系统注册用户的姓名和单位。

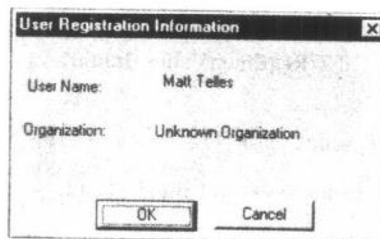


图 1-3 包含当前用户信息的对话框 User Registration Information

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH15. MAK。

2. 进入 AppStudio 并创建新的对话框，添加两个 ID 为 ID_STATIC 静态文本域，标题分别为 User Name 和 Organization。

3. 与前两个静态文本域对齐，创建另外两个静态文本域。ID 分别为 ID_USER_NAME 和 ID_ORGANIZATION。

4. 改变对话框名为 ID_USER_INFO，改变对话框的标题为 User Registration Information。

5. 进入 ClassWizard，选择按钮 Add Class。命名新类为 CUserRegDlg，选择基类为 CDialog，并接受源代码文件名和头文件名的缺省值。

6. 在 ClassWizard 中，从对象 ID 中选择对象 CUserRegDlg，选择消息 WM_INITDIALOG，点击按钮 Add Function，选择按钮 Edit Code，在函数 OnInitDialog 中输入下列代码：

```
BOOL CUserRegDlg:: OnInitDialog ()

{
  HKEY hkRoot, hSubKey;

  CDialog:: OnInitDialog ();

  if (RegOpenKey (HKEY_LOCAL_MACHINE, NULL, &hkRoot) == ERROR_SUCCESS) {
      if ( RegOpenKeyEx (hkRoot, "SOFTWARE\\MICROSOFT\\Windows\\CurrentVersion\\",
                              0,
                              KEY_ENUMERATE_SUB_KEYS |
                              KEY_EXECUTE |
                              KEY_QUERY_VALUE,
                              &hSubKey) == ERROR_SUCCESS ) {
        char ValueName [256];
        unsigned char DataValue [256];
        unsigned long cbValueName = 256;
        unsigned long cbDataValue = 256;
        DWORD dwType;

        if ( RegEnumValue (hSubKey,
                           4,
                           ValueName,
                           &cbValueName,
                           NULL,
                           &dwType,
                           DataValue,
                           &cbDataValue) == ERROR_SUCCESS ) {
```

```
        GetDlgItem (ID_USER_NAME) ->SetWindowText ( (char * ) DataValue );
        }

      cbValueName = 256;
      cbDataValue = 256;

        if ( RegEnumValue (hSubKey,
                          5,
                          ValueName,
                          &cbValueName,
                          NULL,
                          &dwType,
                          DataValue,
                          &cbDataValue) == ERROR_SUCCESS ) {

          GetDlgItem (ID_ORGANIZATION) ->SetWindowText ( (char * ) DataValue );
        }
        // Don't forget to close the open subkey...

        RegCloseKey (hSubKey);
      }

      // And certainly don't forget to close the root key.

      RegCloseKey (hkRoot);
  }

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

7. 返回 AppStudio，选择菜单 IDR_MAINFRAME。在主菜单 SysInfo 中添加新的菜单项 User Registration，ID 为 ID_USER_REGISTRATION。

8. 进入 ClassWizard，选择对象 CMainFrame，从对象 ID 列表中选择 ID_USER_REGISTRATION，从消息列表中选择 COMMAND。点击按钮 Add Function，函数命名为 OnUserRegistration。

9. 选择按钮 Edit Code，在方法 OnUserRegistration 中输入下列代码：

```
void CMainFrame:: OnUserRegistration ()
{
  CUserRegDlg dlg;

  dlg.DoModal ();
```

10. 同时，在文件 MAINFRM.CPP 的顶部添加下列行：

```
#include  "userregd.h"
```

11. 编译并运行此例子程序。

用法

要检索用户信息，首先要打开并浏览 Windows 95 的注册数据库。在此数据库中是一个包含 Windows 95 系统各方面信息的关键字的层次集合。其中一个关键字可以按路径 SOFTWARE\\MICROSOFT\\Windows\\CurrentVersion\来查找，此关键字包含检索用户名和单位的有用信息。

首先，必须打开注册数据库。要做到这一点，需要选择数据库中的一个根层次关键字并打开它。在本节的例子程序中，此根层次关键字为 HKEY_LOCAL_MACHINE，调用 API 函数 RegOpenKey（或 RegOpenKeyEx）打开此根层次关键字，然后调用函数 RegOpenKeyEx（在例子程序中示范了这两个函数的使用，其实任一函数都可用于这两种情况）打开子关键字，并且获得关键字 CurrentVersion 的句柄。

一旦从注册数据库中检索到句柄，就可以调用函数 RegEnumValue 来检索此关键字的某个值。在本节的例子程序中，我们感兴趣的两个值序号分别为 4 和 5。一旦检索到这两个值，就将其放置在对话框的静态文本域中，并通过关闭所有打开的关键字来关闭数据库。关闭注册关键字句柄是通过调用 API 函数 RegCloseKey 来完成的。

1.6 获取计算机的网络名

问题

有的程序员希望能够知道自己计算机上的驱动器的网络名。尽管知道驱动器为网络驱动器是很有用的，但是有时还必须向用户提供他们所要打开的文件的“扩展”网络名。如何使用 Windows API 函数来完成此功能呢？

方法

用户希望能够知道他们正在操作的文件的“实际”名字。更重要的是，他们希望能够确认文件的网络名是否与他们认为的一样。假如用户对某文件操作了一段时间后，才发现实际要操作的文件在另一网络驱动器上，而正在操作的文件仅仅是实际文件的一个拷贝，还有什么事比这更让用户沮丧的呢？显示驱动器的网络名是解决此类问题的绝妙办法。

Windows 95 提供了一个绝妙的网络函数集，程序员可以在自己的应用程序中使用这些函数。本节的例子程序建立在 1.2 节的例子程序的基础上，获取计算机上的所有驱动器以便用户可以任意检测每个驱动器。如果有的驱动器有连接的网络名，则用户也就可以知道此驱动器的网络名了。

最后，我们将使用前面使用过的 API 函数 GetDriveType 来确定机器上可用的驱动器。我们将假设只有潜在的网络驱动器和固定驱动器才可用于网络使用，而忽略软盘驱动器和不存在的驱动器。一旦用户给出要检测的驱动器符，我们将使用网络函数 WNetGetConnection 来

查找此驱动器的网络名。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从菜单 Networks 中选择菜单项 Get Network Name，将弹出一个对话框（如图 1-4 所示），显示此机器上安装的驱动器。当从列表中选择一个驱动器后，文本域将改变为此驱动器连接的网络名。如果此驱动器只是一个本地驱动器，则改变为字符串“Not network drive”。

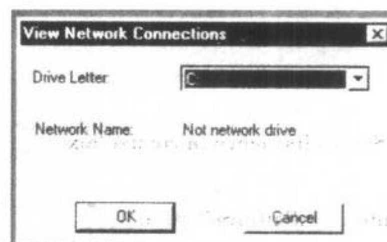


图 1-4　对话框 View Network Connections

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH16. MAK。

2. 进入 AppStudio 并创建新的对话框。在对话框中，添加标题分别为 Drive Letter：和 Network Name：的两个文本域。添加一个组合框，风格设置为 Drop Down List。添加一个静态文本域，并命名为 ID _ NET _ NAME。

3. 将对话框 ID 改为 ID _ NETWORK _ CONNECTION，并设置对话框的标题为 View Network Connections，保存对话框。

4. 在菜单 IDR _ MAINFRAME 中，添加新的菜单 Networks。在菜单 Networks 中，添加菜单项 Get Network Name，其 ID 为 ID _ NETWORK _ NAME。

5. 进入 ClassWizard，选择按钮 Add Class，在名字域中键入名字 CNetworkConnectDlg，选择 CDialog 为基类，选择 ID _ NETWORK _ CONNECTION 为对话框 ID。

6. 在 ClassWizard 的下拉列表中选择 CNetworkConnectDlg，从对象 ID 列表中选择对象 CNetworkConnectDlg，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Edit Code。

7. 在 CNetworkConnectDlg 的方法 OnInitDialog 中输入下列代码：

```
BOOL CNetworkConnectDlg:: OnInitDialog ()
{
  CDialog:: OnInitDialog ();
  CComboBox *combo = (CComboBox *) GetDlgItem (IDC_COMBO1);

  for ( int i=0; i<26; ++i ) {
    char buffer [10];
    sprintf (buffer, "%c: \\", 'A'+i);
    WORD ret = GetDriveType (buffer);
    switch ( ret ) {
        case DRIVE_FIXED: // Hard drive OR mapped drive on hard-drive
          sprintf (buffer, "%c:", 'A'+i);
          combo->AddString ( buffer );
          break;
        case DRIVE_REMOTE: // Network drive OR CD-ROM
```

```
            sprintf (buffer, "%c:", 'A'-i);
            combo->AddString ( buffer );
            break;
                •
        }
    }
// Set to first entry in combo box

combo->SetCurSel (0);
OnNewDrive ();

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

8. 在ClassWizard中，从对象ID列表中选择对象IDC_COMBO1，选择命令CBN_SELCHANGE，点击按钮Add Function，命名方法为OnNewDrive。点击按钮Edit Code，并在CNetworkConnectDlg的方法OnNewDrive中输入下列代码：

```
void CNetworkConnectDlg:: OnNewDrive ()

  char drive_letter [20];
  char network_name [256];
  unsigned long size=256; (=256 为译者所加)

  // Update the data
  UpdateData ();

  // Get the selection from the combo box

  CComboBox *combo = (CComboBox * ) GetDlgItem (IDC_COMBO1);
  int sel = combo->GetCurSel ();

  // Get the text associated with the item

  combo->GetLBText ( sel, drive_letter );

  if ( WNetGetConnection (drive_letter, network_name, &size) == 0 ) {
    GetDlgItem (ID_NET_NAME) ->SetWindowText ( network_name );

  else
    GetDlgItem (ID_NET_NAME) ->SetWindowText ( "Not network drive" );
```

9. 从下拉列表中选择对象 CMainFrame，从对象列表中选择对象 ID _ NETWORK _ NAME，从消息列表中选择 COMMAND，点击按钮 Add Function 并命名方法为 OnNetworkName，点击按钮 Edit Code，在 CMainFrame 的方法 OnNetworkName 中输入下列代码：

```
void CMainFrame:: OnNetworkName ()
{
  CNetworkConnectDlg dlg;
  dlg.DoModal ();
}
```

10. 在文件 MAINFRM.CPP 的顶部添加下列行：

```
#include "networkc.h"
```

11. 编译并运行此例子程序。

用法

函数 GetDriveType 在前面的 1.2 节中讨论过，用来识别系统上所有 Windows 95 支持的驱动器。函数 WNetGetConnection 使用驱动器符来向服务器查询此驱动器符连接的网络名。如果驱动器是映射驱动器（驱动器映射系统的简单逻辑扩充），则认为是正确的网络名，但如果是本地驱动器，则返回错误并标记为本地驱动器。

1.7 找出计算机上当前登录入网的用户

问题

有的程序员希望能够显示用户当前登录的用户名。这使得程序员可以识别当前企图执行某个操作的用户，也可用来通知用户机器已被某个用户以另外的用户名登录。如何使用 Windows 95 的网络函数来完成此功能呢?

方法

前一节介绍了如何找出网络的连接信息。在 Windows 95 和 Windows NT 中，用户信息直接同网络连接相关联。在 Windows for Workgroups 中，则需要一个更迂回的方法。

利用 Windows API 函数 WNetGetUser 可以确定当前登录的用户。在 WIN32 系统中（Windows 95 和 Windows NT），此函数接受本地驱动器名，返回用户名及其大小（用户名缓冲区的大小）。在 Windows 3.x 系统中，函数 WNetUser 只接受用户名和大小（用户名缓冲区的大小）。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Networks，选择菜单项 User Names，将弹出一个对话框（如图 1-5 所示），显示此机器上安装的驱动器。当从列表中选择一个驱动器后，文本域将改变为与此驱动器相连接的用户名或字符串 "None"。

实现例子程序的具体步骤如下：

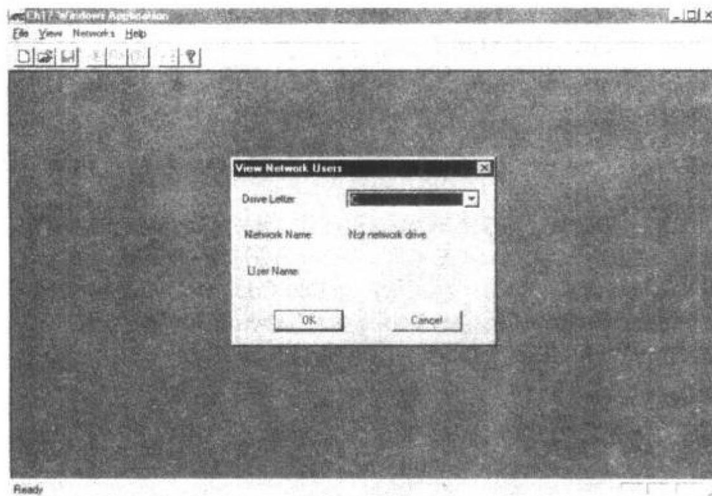


图 1-5 对话框 View Network Users

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH17.MAK。

2. 进入 AppStudio 创建新的对话框。在对话框中，添加三个文本域，标题分别为 Drive Letter:、Network Name: 和 User Name:。添加一个组合框，并设置风格为 Drop Down List。添加两个静态文本域，标识符分别为 ID_NET_NAME 和 ID_NET_USER。

3. 将对话框 ID 改为 ID_NETWORK_USER，并设置标题为 View Network Users，保存对话框。

4. 进入 ClassWizard，选择按钮 Add Class，为新对话框创建新的对话框类。类名为 CNetworkUserDlg，基类为 CDialog，对话框 ID 为 ID_NETWORK_USER。

5. 在 ClassWizard 中，从下拉列表中选择 CNetworkUserDlg，从对象 ID 列表中选择对象 CNetworkUserDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Edit Code。

6. 在 CNetworkUserDlg 的方法 OnInitDialog 中输入下列代码：

```
BOOL CNetworkUserDlg:: OnInitDialog ()

  CDialog:: OnInitDialog ();

  char buffer [20];

  CComboBox *combo = (CComboBox *) GetDlgItem (IDC_COMBO1);

  for ( int i=0; i<26; ++i ) {
     sprintf (buffer, "%c: \\", 'A'+i);
    WORD ret = GetDriveType (buffer);
    switch ( ret ) {
```

```
        case DRIVE_FIXED: // Hard drive OR mapped drive on hard-drive
          sprintf (buffer, "%c:", 'A'+i);
          combo->AddString ( buffer );
          break;
        case DRIVE_REMOTE: // Network drive OR CD-ROM
          sprintf (buffer, "%c:", 'A'+i);
          combo->AddString ( buffer );
          break;

      }
    }
  combo->SetCurSel (0);
  OnDriveChange ();

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

7. 在ClassWizard中，从对象ID列表中选择对象IDC_COMBO1，选择命令CBN_SELCHANGE，点击按钮Add Function，命名方法为OnDriveChange。点击按钮Edit Code，在CNetworkUserDlg的方法OnDriveType中输入下列代码：

```
void CNetworkUserDlg:: OnDriveChange ()
{
  char drive_letter [20];
  char network_name [256];
  char user_name [256];
  unsigned long size=256;

  // Update the data
  UpdateData ();

  // Get the selection from the combo box

    CComboBox *combo = (CComboBox *) GetDlgItem (IDC_COMBO1);
  int sel = combo->GetCurSel ();

  // Get the text associated with the item

  combo->GetLBText ( sel, drive_letter );

  if ( WNetGetConnection (drive_letter, network_name, &size) == 0 ) {
     GetDlgItem (ID_NET_NAME) ->SetWindowText ( network_name );
```

```
    }
    else
        GetDlgItem (ID_NET_NAME) -->SetWindowText ( "Not network drive" );

    if ( WNetGetUser (drive_letter, user_name, &size) ) {
        GetDlgItem (ID_NET_USER) ->SetWindowText ( user_name );
    }
    else
        GetDlgItem (ID_NET_USER) ->SetWindowText ( "None" );
}
```

8. 进入 AppStudio，添加新的主菜单 Networks。在菜单 Networks 中添加新的菜单项 User Names，ID 为 ID_NETWORK_USERS。

9. 从下拉列表中选择对象 CMainFrame，从对象列表中选择对象 ID_NETWORK_USERS，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名方法为 OnNetworkUsers。点击按钮 Edit Code，在 CMainFrame 的方法 OnNetworkUsers 中输入下列代码：

```
void CMainFrame:: OnNetworkUsers ()
{
    CNetworkUserDlg dlg;
    dlg.DoModal ();
}
```

10. 在文件 MAINFRM.CPP 的顶部添加下列行：

```
#include "networku.h"
```

11. 编译并运行此例子程序。

用法

本节例子程序中的大部分代码是相当直接的。首先检查驱动器是否有效，如果是，将驱动器符添加到组合框中。当用户从组合框中选择驱动器后，例子程序接着向网络软件查询此驱动器逻辑相连的网络名。最后，由于在同一台工作站上可以用多个用户名登录多个网络，所以向网络软件查询的是与驱动器相连接的用户名。

1.8 找出键盘上当前按下的键

问题

有的程序员希望能够识别键盘上所有按下的键，以便在应用程序中能正确地处理它们。特别希望知道的是用户是否按下了键 SHIFT、CTRL、NUM LOCK、或 CAPS LOCK。对于键 SHIFT 和 CTRL，程序员关心的是用户是否当前按下此键；对于键 NUM LOCK 和 CAPS LOCK，程序员只关心用户是否在应用程序取得控制之前选取了此键。

方法

这是一个非常有趣的问题。如何在多任务、多窗口系统中确定键盘上的哪个键是按下的，以使键盘状态对所有的应用程序来说都是一致的呢?由于有的键可能在另一个窗口中被按下，

所以不能简单地捕捉键的按下操作。由于键 NUM LOCK 有可能在应用程序启动前已被按下，所以也不能简单地监视此键是否被按下。

因此，要完成此任务所需要的是一个能够随时获取键盘状态的方法。幸运的是，Windows API 对此问题提供了一个简单的解决方案，即 API 函数 GetKeyboardState。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Keyboard，从下拉列表中选择菜单项 Keys Pressed。将弹出一个显示一列键的对话框，每个键的旁边没有文本。按下几个列出键并点击按钮 Show，则列表右边的状态值将改变以反映这些键的当前状态。对话框如图 1-6 所示。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH18.MAK。

2. 进入 AppStudio 并创建新的对话框。添加四个文本域，标题分别为 Control Key、Shift Key、Num Lock Key 和 Caps Lock Key。

3. 与刚创建的四个文本域对齐，创建另外四个文本域。标题为空，ID 分别为 ID_CONTROL_KEY、ID_SHIFT_KEY、ID_NUM_LOCK_KEY 和 ID_CAPS_LOCK_KEY。

4. 改变对话框的标题为 View Key States。

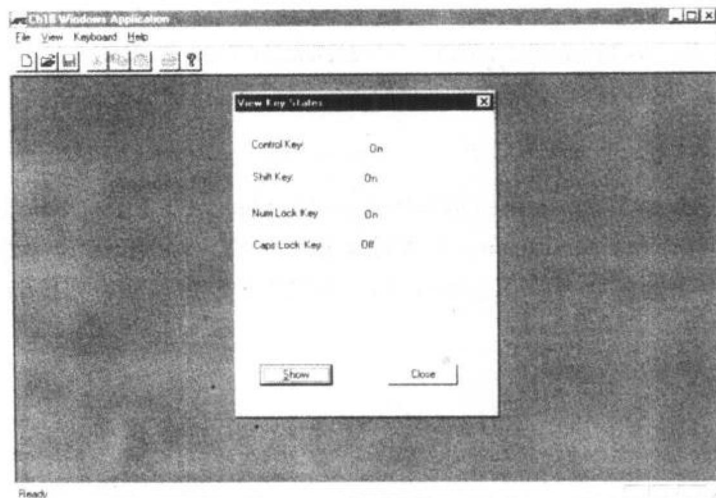


图 1-6 对话框 View key States

5. 选择 ClassWizard，点击按钮 Add Class，并命名对话框类为 CKeyShowDlg。

6. 在 ClassWizard 中，从下拉列表中点击对象 CKeyShowDlg，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 COMMAND，点击按钮 Add Function，函数命名为 OnButton1。

7. 在对象 CKeyShowDlg 的方法 OnButton1 中输入下列代码：

```
void CKeyShowDlg::OnButton1 ()
{
```

```
  unsigned char kbuf [256];
  GetKeyboardState ( kbuf );

  if ( kbuf [VK_CAPITAL] & 1 )
     GetDlgItem (ID_CAPS_LOCK_KEY) ->SetWindowText ("On");
  else
     GetDlgItem (ID_CAPS_LOCK_KEY) ->SetWindowText ("Off");

  if ( kbuf [VK_SHIFT] & 128 )
     GetDlgItem (ID_SHIFT_KEY) ->SetWindowText ("On");
  else
     GetDlgItem (ID_SHIFT_KEY) ->SetWindowText ("Off");

  if ( kbuf [VK_CONTROL] & 128 )
     GetDlgItem (ID_CTRL_KEY) ->SetWindowText ("On");
  else
     GetDlgItem (ID_CTRL_KEY) ->SetWindowText ("Off");

  if ( kbuf [VK_NUMLOCK] & 1 )
     GetDlgItem (ID_NUM_LOCK_KEY) ->SetWindowText ("On");
  else
     GetDlgItem (ID_NUM_LOCK_KEY) ->SetWindowText ("Off");

}
```

8. 在菜单 Keyboard 中添加新的菜单项 Keys Pressed，ID 为 ID_KEYS_PRESSED。进入 ClassWizard，在对象 CMainFrame 中为对象 ID_KEYS_PRESSED 和消息 COMMAND 添加新的函数，命名此函数为 OnKeyPressed，并在此函数中添加下列代码：

```
void CMainFrame:: OnKeysPressed ()
{
  CKeyShowDlg dlg;
  dlg.DoModal ();
}
```

9. 在文件 MAINFRM.CPP 的顶部添加下列行：

```
#include "keyshowd.h"
```

10. 编译并运行此例子程序。

用法

Windows API 函数 GetKeyboardState 返回键盘上每一键的当前状态。对于可以按下和

松开的键，如 SHIFT，CTRL，TAB 以及所有的字符键，键的“状态”值或是高位设置（表示键被按下）或是高位清除（表示键未被按下，即松开的）。对于切换选择的键，如 NUM LOCK 或 CAPS LOCK，低位设置表示此键当前是选中（开）状态，低位清除表示此键当前是未选中（关）状态。

通过在预定的时间检查键盘的状态，可以随时得到被按下的任何键和所有键。例如：如果用户按下键 CTRL-ALT-SHIFT-F10，则在键盘缓冲区中将设置下列键值的高位：

VK _ SHIFT

VK _ CONTROL

VK _ MENU（为 ALT 键）

VK _ F10

在线帮助或 include 文件 Windows.H 中都列出了键盘上所有这些键的虚拟键码。

注释

使用 Visual Basic 也能完成同样的任务。按照下列步骤在 Visual Basic 的表单中实现同样的功能，表单如图 1-7 所示。

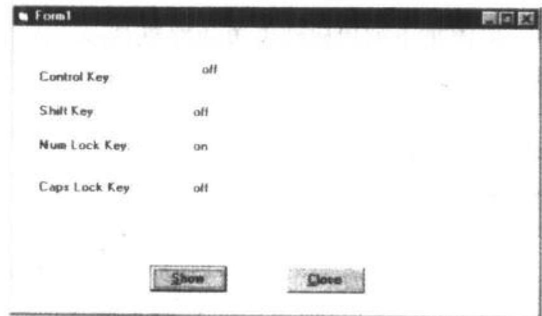


图 1-7　显示按下的键的 Visual Basic 表单

1. 创建新的 Visual Basic 项目文件，并创建新的表单。添加四个标号域，标题分别为 Control Key:、Shift Key:、Num Lock Key: 和 Caps Lock Key:。同时对应每一个标号域创建另外一个标号域，标题为空。在表单中添加两个按钮，Show 和 Close。

2. 双击按钮 Show，在方法中添加下列代码：

```
Sub Command1 _ Click
    Dim keys As String
    keys = Space $ (256)
    GetKeyboardState keys

    If &H1 = (Asc (Mid (keys, VK _ CONTROL + 1, 1)) And &H1) Then
       Label2. Caption = "on"
    Else
       Label2. Caption = "off"
    End If

    If &H1 = (Asc (Mid (keys, VK _ SHIFT + 1, 1)) And &H1) Then
       Label4. Caption = "on"
    Else
       Label4. Caption = "off"
    End If

    If &H1 = (Asc (Mid (keys, VK _ NUM LOCK + 1, 1)) And &H1) Then
       Label6. Caption = "on"
```

```
    Else
        Label6.Caption = "off"
    End If

    If &H1 = (Asc (Mid (keys, VK_CAPITAL + 1, 1)) And &H1) Then
        Label8.Caption = "on"
    Else
        Label8.Caption = "off"
    End If
End Sub
```

3. 添加下列代码到表单的按钮 Close 中：

```
Sub Command2_Click

End
End Sub
```

4. 最后，在表单的 General 部分添加下列行：

```
Const VK_SHIFT = &H10
Const VK_CAPITAL = &H14
Const VK_NUMLOCK = &H90
Private Declare Sub GetKeyboardState Lib "User32" (ByVal LpKeyState As String)
```

1.9 使用 Windows 95 的注册表存取信息

问题

许多程序员对新的 Windows 95 的注册表都有了相当多的了解，知道注册表是用来替代 Windows 3.1 中的初始化文件 (.INI)。但是，如何使用此注册表呢？如何在注册表中保存信息以便将来检索呢？是把应用程序的所有信息都放入注册表，还是继续使用 .INI 文件呢？

方法

Windows 95 的注册表实际上是 Windows 3.1 中已存在的注册函数的扩充。这些函数用来为 OLE 服务器定义 OLE (Object Linking and Embedding) 信息，以用于响应 OLE 命令而启动应用程序。

但是在 Windows 95 中，Microsoft 已扩展了此注册功能，以允许应用程序在系统级保存更多的信息。由于大多数的 Windows 95 应用程序都支持拖放功能和 OLE 功能，所以能够为 OLE 应用程序搜集信息就显得更加重要。

尽管 OLE 命令不在本节讨论之列，但知道注册表中有什么信息以及如何获取还是很重要的。同时，在本节中还将介绍如何在注册表中添加自己应用程序的信息以及如何检索这些信息。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从菜单 Registry 中选择菜单项 Show Keys，将弹出对话框显示注册表的主层次关键字及其值。对于每一个主层次关键字，都将列出此关键字的零个或多个子关键字，关键字名前加双破折号（——）表示子关键字。对话框 Show Keys 如图 1-8 所示。

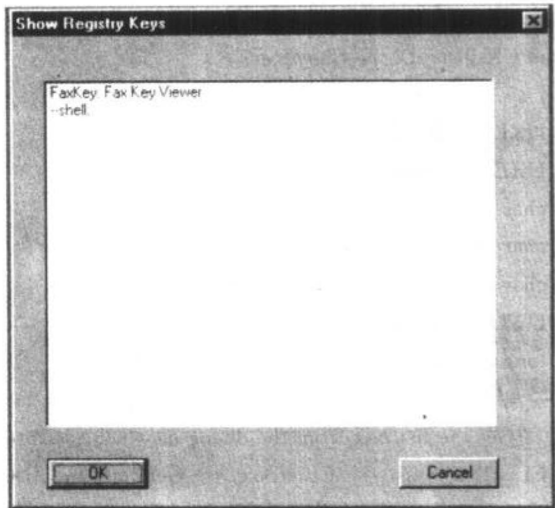


图 1-8　对话框 Show Registry Keys

在菜单 Registry 中还定义了另外两个菜单项。菜单项 Find Key 用来查找某一主层次关键字和此关键字的某个子关键字（子关键字是任选的）。从菜单 Registry 中选择菜单项 Find Key，将弹出一个有两个编辑框的对话框（如图 1-9 所示）。选择第一个编辑框并键入注册表中某个关键字的名字，关键字通常是应用程序的名字。你可以从 Show Keys 列表框中选取某个应用程序的名字用在这里，也可以直接键入一个。第二个编辑框用于子关键字的名字，如果你在 Show Keys 列表框中注意到一个包含子关键字的关键字，那么可以在此编辑框中输入此子关键字，对话框将在域 Value（在两个编辑框的下面）中显示此关键字和子关键字的联合值。

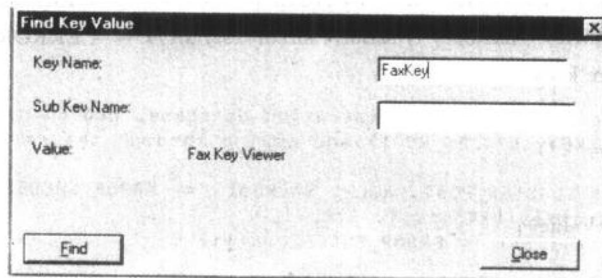


图 1-9　对话框 Find Key Value

实现例子程序的具体步骤如下：

1. 在 Visual C＋＋，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH110. MAK。

2. 进入 AppStudio 并添加新的对话框，在此对话框中添加一个列表框控制。进入 Class Wizard，为此对话框模板生成新的对话框类，并命名新类为 CRegistryDlg，接受其他的所有缺省值。

3. 进入 ClassWizard，选择类 CRegistryDlg，选择消息 WM _ INITDIALOG，在方法 OnInitDialog 中添加下列代码：

```
ShowKeys ();
```

4. 在类的源文件（.CPP）中添加下列代码：

```
void CRegistryDlg:: ShowKeys ()
{
  HKEY hkRoot;
  HKEY  hSubKey;
  char buffer [256];
  char szBuff [80], szValue [256];
  char szBuff1 [80], szValue1 [256];
  DWORD i, i1;
  long cb;

  // Get the list box from the dialog in which to store the information
  CListBox *list = (CListBox * ) GetDlgItem (IDC_LIST1);

  // Clear out existing entries

  list->ResetContent ();

  // Loop through the keys in the registration database, beginning with
  // the top level (HKEY_CLASSES_ROOT) and moving through the rest.
  if (RegOpenKey (HKEY_CLASSES_ROOT, NULL, &hkRoot) == ERROR_SUCCESS) {
    for (i = 0; RegEnumKey (hkRoot, i, szBuff, sizeof (szBuff)) == ERROR_SUCCESS; ++i) {
        // Ignore system keys

        if ( * szBuff == '.')
            continue;
        cb = sizeof (szValue);

        // Get the value of this key
        if (RegQueryValue (hkRoot, (LPSTR) szBuff, szValue,
              &cb) == ERROR_SUCCESS) {

           // Add the key and value to the list box

           sprintf (buffer, "%s: %s", szBuff, szValue );
           list->AddString ( buffer );
        }
        // Do the subkeys

        if ( RegOpenKey (hkRoot, szBuff, &hSubKey) == ERROR_SUCCESS)
         {
           for (i1 = 0; RegEnumKey (hSubKey, i1, szBuff1,
```

```
        sizeof (szBuff1) ) == ERROR_SUCCESS; ++i1) {
        if ( * szBuff1 == '.')
            continue;
        cb = sizeof (szValue1);

        // Get the value of this subkey
        if ( (RegQueryValue (hSubKey, (LPSTR) szBuff1, szValue1,
                &cb)) == ERROR_SUCCESS) {
            sprintf (buffer, "--%s: %s", szBuff1, szValue1 );
            list->AddString ( buffer );
        }
      }
    }

            // Don't forget to close all open keys...

            RegCloseKey (hSubKey);

    }
  }
//And certainly don't forget to close the root key.
RegCloseKey (hkRoot);
}
```

5. 在头文件（.H）中添加下列代码：

```
void  ShowKeys (void);
```

6. 现在返回 AppStudio 并创建新的对话框。在此对话框中，添加三个文本域，标题分别为 Key Name:、Sub Key Name: 和 Value:。邻接文本域 Value: 添加新的文本域，标识符为 ID_KEY_VALUE，邻接文本域 Key Name: 和文本域 Sub Key Name: 添加两个编辑域。

7. 改变按钮 IDOK 的标题为 Find，改变按钮 IDCANCEL 的标题为 Close。

8. 在 ClassWizard 中，为此对话框模板生成新的对话框类。选择对象 ID 为 IDOK，选择消息为 BN_CLICKED，在类的方法 OnOk 中添加下列代码：

```
void CFindRegKey:: OnOK ()
{
  char keyName [256];
  char subKeyName [256];
  HKEY hkRoot, hSubKey;
  char szValue [80];
  char szValue1 [80];
```

```
LONG cb;
// Get the key name from the edit box

GetDlgItem (IDC_EDIT1) ->GetWindowText ( keyName, 256 );

// Get the sub-key name (if any) from the edit box

GetDlgItem (IDC_EDIT2) ->GetWindowText ( subKeyName, 256 );

// Look them up.

if (RegOpenKey (HKEY_CLASSES_ROOT, NULL, &hkRoot) == ERROR_SUCCESS) {

        // Get the value of this key

        cb = sizeof (szValue);
        if (RegQueryValue (hkRoot, (LPSTR) keyName, szValue, &cb) ==
ERROR_SUCCESS) {

            // If they asked for a sub-key, ask for it.
            if ( strlen (subKeyName) ) {
                if (RegOpenKey (hkRoot, keyName, &hSubKey) == ERROR_SUCCESS) {
                 cb = sizeof (szValue1);
                 if (RegQueryValue (hSubKey, (LPSTR) subKeyName,
szValue1, &cb) == ERROR_SUCCESS) {
                        // Put it in the value string
                        GetDlgItem (ID_KEY_VALUE) ->SetWindowText ( szValue1 );
                    }
                    // Don't forget to close the open subkey...

                    RegCloseKey (hSubKey);

                }
            }
            else {
                // Put this value in the value string.

                GetDlgItem (ID_KEY_VALUE) ->SetWindowText ( szValue );
            }

        }
}
```

```
// And certainly don't forget to close the root key.

RegCloseKey (hkRoot);

}
```

9. 最后，再一次进入 AppStudio。创建新的对话框，在对话框中添加三个文本域，标题分别为 Key Name:、Sub Key Name: 和 Value:。邻接这三个文本域添加三个编辑域，编辑域要与对应的文本域在一条水平线上。

10. 改变按钮 IDOK 的标题为 Add，改变按钮 IDCANCEL 的标题为 Close。

11. 进入 ClassWizard，为此对话框生成新的对话框类。新类命名为 CAddKeyDlg，并接受其他的所有缺省值。

12. 从对象列表中选择对象 ID 为 IDOK，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function，在方法 OnOK 中添加下列代码：

```
void CAddKeyDlg:: OnOK ()
{
  char keyName [256];
  char subKeyName [256];
  HKEY hkRoot, hKey, hSubKey;
  char value [80];
  LONG cb;

  // Get the key name from the edit box

  GetDlgItem (IDC_EDIT1) ->GetWindowText ( keyName, 256 );

  // Get the sub-key name (if any) from the edit box

  GetDlgItem (IDC_EDIT2) ->GetWindowText ( subKeyName, 256 );

  // And get the value (if any) from the edit box
  GetDlgItem (IDC_EDIT3) ->GetWindowText ( value, 80 );

  cb = strlen (value);

  // Look them up. First, open the root key

  if (RegOpenKey (HKEY_CLASSES_ROOT, NULL, &hkRoot) == ERROR_SUCCESS) {

    // Open the main key here.
```

```
if (RegOpenKey (hkRoot, keyName, &hKey) == ERROR_SUCCESS) {

    // If they asked for a sub-key, ask for it.

    if ( strlen (subKeyName) ) {

        if (RegOpenKey (hKey, subKeyName, &hSubKey) == ERROR_SUCCESS)
{

            // Key already exists -- Error

            RegCloseKey (hSubKey);
            RegCloseKey (hKey);
            RegCloseKey (hkRoot);
            MessageBox ("Key Already Exists!", "Error", MB_OK );
        }
        else {
            // Add sub key here.
            if (RegCreateKey (hKey, subKeyName, &hSubKey) == ERROR_SUCCESS) {
                RegSetValue (hSubKey, NULL, REG_SZ, value, cb);
            }
            else

                MessageBox ("Unable to create sub key", "Error", MB_OK);
        }
    }
    else {
        // Key already exists -- Error
        RegCloseKey (hkRoot);
        RegCloseKey (hKey);
        MessageBox ("Key Already Exists!", "Error", MB_OK );
    }
}   // End of open of main key.
else {

    // Create main key
    if (RegCreateKey (hkRoot, KeyName, &hKey) ==ERROR_SUCCESS) {

        // Create sub key
        if (RegCreateKey (hKey, subKeyName, &hSubKey)   ==
          ERROR_SUCCESS) {

            // And set the value as requested
```

```
            RegSetValue (hSubKey, NULL, REG_SZ, value, cb);

            RegCloseKey (hKey);
            RegCloseKey (hSubKey);
            RegCloseKey (hkRoot);
         }
         else
            MessageBox ("Unable to create sub key", "Error", MB_OK);

      }
      else
         MessageBox ("Cannot create main key", "Error", MB_OK);
   }
 }
 else
   MessageBox ( "Can't open root key!", "Error", MB_OK );
   CDialog:: OnOK ();
}
```

13. 在菜单 Registry 中添加新菜单项 Show Keys，标识符为 ID_SHOW_KEYS。进入 ClassWizard，在对象 CMainFrame 中为对象 ID_SHOW_KEYS 和消息 COMMAND 添加新的函数，并命名此函数为 OnShowKeys。在此函数中添加下列代码：

```
void CMainFrame:: OnShowKeys ()
{
  CRegistryDlg   dlg;

  dlg.DoModal ();
}
```

14. 在菜单 Registry 中添加新菜单项 Find Keys，标识符为 ID_FIND_KEYS。进入 ClassWizard，在对象 CMainFrame 中为对象 ID_FIND_KEYS 和消息 COMMAND 添加新的函数，并命名此函数为 OnFindKeys。在此方法中添加下列代码：

```
void CMainFrame:: OnFindKeys ()
{
  CFindRegDlg    dlg;

  dlg.DoModal ();
}
```

15. 在菜单 Registry 中添加新菜单项 Add Keys，标识符为 ID_ADD_KEYS。进入 ClassWizard，在对象 CMainFrame 中为对象 ID_ADD_KEYS 和消息 COMMAND 添加新的函数，并命名此函数为 OnAddKeys。在此方法中添加下列代码：

```
void CMainFrame::OnAddKeys ()
{
  CAddKeyDlg    dlg;

  dlg.DoModal ();
}
```

16. 在源文件 MAINFRM.CPP 的顶部添加下列行：

```
#include "registry.h"
#include "findregk.h"
#include "addkeydl.h"
```

用法

Windows 95 的注册表简单地说是一个关键字的层次树，每个关键字是作为父关键字的子关键字存储的，从而形成倒立的树结构。树顶部的根是 Microsoft 预定义的关键字，其中包括 HKEY_CLASSES_ROOT。所有用户定义的关键字是根关键字的后代，并且要利用根关键字来查询。

要获取树中的某个关键字，必须打开根关键字并向下浏览，只有通过其父关键字才能找到它。添加一个关键字也类似，必须先找到要插入关键字的父关键字，然后才能插入。

Windows 95 API 函数 RegOpenKey 和 RegCreateKey 的功能基本上一致，它们定位树中的给定关键字并返回此关键字的“句柄”。对于函数 RegCreateKey，如果此关键字不存在，则创建它；如果存在，则打开它。

API 函数 RegSetValue 用来给关键字赋值。如果关键字是其他树节点的子关键字，就需要浏览此子关键字以便给其赋值。函数 RegCloseKey 仅仅用来关闭关键字，并使句柄无效。

应该注意，尽管这些函数在 Windows 95（和 Windows NT）中可以运行得非常好，但它们已经被 32 位的函数所替代。RegOpenKeyEx 替代 RegOpenKey，RegCreateKeyEx 替代 RegCreateKey，其他的类似。在 Windows 95 中，这些函数的差别很小，使用哪一个函数都是可以的。但在 Windows NT 中，考虑到操作系统的安全，应尽量选择使用新函数。

1.10 找出程序或 DLL 的版本号

问题

有的程序员需要确保自己的程序运行使用某一特定版本的 DLL。有时，用户也需要确定当前运行程序的版本，以便确定是否有适合自己使用的某些功能。如何利用 Windows 95 API 函数来完成这些任务呢？

方法

在 Windows 系统目录下的 VERSION.DLL 文件中可以找到 Windows 的版本信息函数。这些版本函数可以用来确定某个文件中是否有版本信息以及版本信息是什么。

在这里不妨回顾一下。版本信息是通过资源语句 VERSIONINFO 保存在应用程序中的，此语句创建的资源在应用程序联编时编入应用程序，以后可以被任何需要此资源的应用程序存取。可用的信息包括可执行程序或 DLL 的版本号，以及创建单位。

有关的 Windows 95 API 函数是 GetFileVersionInfoSize、GetFileVersionInfo 和 Ver-

QueryValue。注意，使用这些函数需要动态连接库 VERSION.DLL，而且输出库 VERSION.LIB 必须连接在应用程序中。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 SysInfo 和菜单项 File Version Info，将弹出一个包含编辑框和数个文本域的对话框。键入一个 Windows 可执行程序或 DLL 文件的全路径名(一个好例子是键入\Windows\SYSTEM\DDEML.DLL)，并点击按钮 Inspect，注意显示的信息。对话框如图 1-10 所示。

图 1-10　对话框 File Version Information

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH111.MAK。

2. 利用 AppStudio 创建新的对话框。对话框包含下列控制：

• 文本域。标题分别为：Company Name:，File Description:，File Version:，Internal Name:，Original File Name:，Product Name:，Product Version:，File Name to Inspect:。

• 空白标题文本域。标识符分别为：ID_VER_COMPANY_NAME，ID_VER_FILE_DESC，ID_VER_FILE_VERSION，ID_VER_INTERNAL_NAME，ID_VER_ORIG_FILENAME，ID_VER_PRODUCT_NAME，及 ID_VER_PRODUCT_VERSION。

3. 改变按钮 IDCANCEL 的标题为 Close，删除按钮 IDOK，添加标识符为 ID_INSPECT 和标题为 Inspect 的新按钮。接着与编辑域水平对齐添加另外一个按钮，标题为 &Browse。

4. 设置对话框的标题为 File Version Information。

5. 利用 ClassWizard 为此对话框生成新的对话框类，并命名此类为 CVersionDlg，接受其他所有的缺省值。

6. 从对象 ID 列表中选择对象 ID_INSPECT，从消息列表中选择消息 BN_CLICKED。点击按钮 Add Function，新函数命名为 OnInspect。

7. 在函数 OnInspect 中输入下列代码：

```
void CVersionDlg::OnInspect ()
{
    UpdateData ();

    // Get the edit field value

    char ebuffer [256];
    GetDlgItem (IDC_EDIT1) ->GetWindowText ( ebuffer, 256 );
```

```
BYTE        block [1024];
DWORD FAR  * translation;
DWORD FAR  * buffer;
DWORD        handle;
DWORD        bytes;
char        name      [2048];        // StringFileInfo data block.
char        data [256];

// Get the actual size of the information block.
bytes = GetFileVersionInfoSize (ebuffer, &handle);
if (bytes)
{
    // Get the actual block for the version information

       bytes = 2048;

    if (GetFileVersionInfo (ebuffer, handle, bytes, block))
    {
        // Good. First, get the Product Version information

        if (VerQueryValue (block, "\\VarFileInfo\\Translation",
              (VOID FAR * FAR * ) &translation, (UINT FAR * ) &bytes))
        {
            wsprintf (name, "\\StringFileInfo\\%04x%04x\\ProductVersion",
                    LOWORD ( * translation), HIWORD ( * translation));
            if (VerQueryValue (block, name, (VOID FAR * FAR * ) &buffer,
                    (UINT FAR * ) &bytes))
                lstrcpy (data, (char far * ) buffer);
            GetDlgItem (ID_VER_PRODUCT_VERSION) - >SetWindowText (data);
        }
        else {
            MessageBox ( "Unable to get translation type", "Error", MB_OK );
        }

        // Next, get the Company name information

        wsprintf (name, "\\StringFileInfo\\%04x%04x\\CompanyName",
                LOWORD ( * translation), HIWORD ( * translation));
        if (VerQueryValue (block, name, (VOID FAR * FAR * ) &buffer,
                (UINT FAR * ) &bytes))
            lstrcpy (data, (char far * ) buffer);
            GetDlgItem (ID_VER_COMPANY_NAME) - >SetWindowText (data);
        }
```

```
// The Original File name for this file

wsprintf (name, "\\StringFileInfo\\%04x%04x\\OriginalFilename",
        LOWORD (*translation), HIWORD (*translation));

if (VerQueryValue (block, name, (VOID FAR * FAR *) &buffer,
        (UINT FAR *) &bytes)) {
    lstrcpy (data, (char far *) buffer);
    GetDlgItem (ID_VER_ORIG_FILENAME) ->SetWindowText (data);
}

// The File Description name for this file

wsprintf (name, "\\StringFileInfo\\%04x%04x\\FileDescription",

        LOWORD (*translation), HIWORD (*translation));
if (VerQueryValue (block, name, (VOID FAR * FAR *) &buffer,
      (UINT FAR *) &bytes)) {
    lstrcpy (data, (char far *) buffer);
    GetDlgItem (ID_VER_FILE_DESC) ->SetWindowText (data);
}
// The File Version for this file

wsprintf (name, "\\StringFileInfo\\%04x%04x\\FileVersion",
        LOWORD (*translation), HIWORD (*translation));

if (VerQueryValue (block, name, (VOID FAR * FAR *) &buffer,
        (UINT FAR *) &bytes)) {
    lstrcpy (data, (char far *) buffer);
    GetDlgItem (ID_VER_FILE_VERSION) ->SetWindowText (data);
}

// The Internal Name for this file

wsprintf (name, "\\StringFileInfo\\%04x%04x\\InternalName",
        LOWORD (*translation), HIWORD (*translation));

if (VerQueryValue (block, name, (VOID FAR * FAR *) &buffer,
        (UINT FAR *) &bytes)) {
    lstrcpy (data, (char far *) buffer);
    GetDlgItem (ID_VER_INTERNAL_NAME) ->SetWindowText (data);
}
```

```
        // The Product Name for this file
        wsprintf (name, "\\StringFileInfo\\%04x%04x\\ProductName",
                LOWORD ( * translation), HIWORD ( * translation));

        if (VerQueryValue (block, name, (VOID FAR * FAR * ) &buffer,
                (UINT FAR * ) &bytes)) {
            lstrcpy (data, (char far * ) buffer);
            GetDlgItem (ID_VER_PRODUCT_NAME) ->SetWindowText (data);
        }

    }
  }
  UpdateData ();

}
```

8. 在ClassWizard中，从对象列表中选择对象IDC_BUTTON1，从消息列表中选择消息BN_CLICKED，选择按钮Add Function，并命名新函数为OnBrowse。在CVersionDlg的方法OnBrowse中输入下列代码：

```
void CVersionDlg:: OnBrowse ()
{
  // Put up a file open dialog

  static char szFilter [] = "DLLs (*.dll) | *.DLL | Executables
(*.exe) | *.exe | All Files (*.*) | *.* ||";
  CFileDialog dlg ( TRUE, "DLL", NULL, OFN_HIDEREADONLY, szFilter, this );

  if ( dlg.DoModal () == IDOK ) {
     GetDlgItem (IDC_EDIT1) ->SetWindowText ( dlg.GetPathName () );
  }
}
```

9. 利用AppStudio，在菜单SysInfo中添加新菜单项File Version Info，标识符为ID_FILE_VERSION_INFO。利用ClassWizard，从下拉列表中选择对象CMainFrame，从对象列表中选择标识符ID_FILE_VERSION_INFO，从消息列表中选择消息COMMAND，点击按钮Add Function，新函数命名为OnFileVersionInfo。

10. 在方法OnFileVersionInfo中添加下列代码：

```
void CMainFrame:: OnFileVersionInfo ()
{
  CVersionDlg dlg;

  dlg.DoModal ();
```

11. 在文件 MAINFRM.CPP 中，#include "stdafx.h"语句后，添加下列行：
#include "versiond.h"

12. 编译并运行此例子程序。

用法

当用户从菜单中选择菜单项 File Version Info 后，对话框将弹出。当用户在编辑框中键入文件名并点击按钮 Inspect 后，例子程序将分下列几步进行：

首先，调用函数 GetFileVersionInfoSize。此函数用来确定存放（指定文件的）资源 VERSIONINFO 信息所需要的缓冲区大小。此函数的实际操作是打开文件，寻找文件中版本信息的起始点，并读出资源头的大小，实际资源的大小就存放在此结构中。接着将此数字返回给调用者，注意，如果文件中没有版本信息或文件不存在，则函数的返回值为 0。

接着，调用函数 GetFileVersionInfo。此函数实际将版本信息块从文件中装入缓冲区中，拷贝的最多字节数由函数 GetFileVersionSize 指定。此信息块处于"原"状态，即它不是可存取的简单结构，而是复杂的可变长的缓冲数据。

函数 VerQueryValue 使用名字从原信息中析取数据。此函数接受缓冲区和名字，缓冲区由函数 GetFileVersionInfo 获得，名字为要析取的数据"块"的名字。析取出的数据接着保存在用户提供的缓冲区中。第一次调用函数 VerQueryValue 是析取名为\\VarFileInfo\\Translation 的数据块，此信息用来翻译版本资源中的其他数据块。指定翻译表后，再一次调用函数 VerQueryValue，本次的名字使用下列表达式来确定：
wsprintf (name," \\ StringFileInfo \\%04x%04x \\ ProductVersion", LOWORD (* translation), HIWORD (* translation))

此函数用来为版本信息块构造新"关键字"，使用第一个函数调用得到的翻译表来表示关键字 ProductVersion。

函数 VerQueryValue 的第二个以及后续的调用返回的缓冲区就是保存在版本资源中的实际信息。这些信息被显示在屏幕上。

第2章　文件和目录

在早期的 Windows 版本中，大多数文件和设备函数都调用 DOS。但在 Windows 95 中，这些函数被封装在 Windows 操作系统中。本章将介绍这些重要的磁盘和目录函数，从而使得应用程序更健壮和更易于使用。

表 2-1 列出了在本章中使用的 Windows 95 API 函数。

表 2-1　在第 2 章中使用的 Windows API 函数

GetWindowsDirectory	GetSystemDirectory
GetTempPath	SetErrorMode
MessageBox	GetLastError
GetLogicalDrives	CopyFile
MoveFile	FillMemory
GetOpenFileName	GetSaveFileName
CreateDirectory	RemoveDirectory

1. 如何查找 Windows 目录和 System 目录

有时应用程序需要获取某些有关运行环境的信息。在本节中，将介绍如何查找 Windows 目录、System 目录和临时文件目录。

2. 如何显示 I/O 错误信息

如果应用程序需要执行某些文件或设备 I/O 操作，除非是一些最简单的操作，否则有可能出现操作错误。有时，由应用程序来处理硬件错误信息是很有用的，可以提供给用户一些针对此应用程序的信息。在本节中，将介绍如何阻止系统显示缺省的错误信息，以及如何用自己的错误信息替代它们。

3. 如何检测驱动器中是否有盘

无论是编写可靠的备份工具，还是做安装程序盘，或者仅仅是少量数据的写盘，最好都能在操作前检查驱动器中是否有盘。实现此功能的一种方法是存取此驱动器并获取错误信息，此方法在 2.2 节中已介绍过。但是检查可移动驱动器中是否有盘，Windows 提供了一个更好的方法，本节将介绍此方法。

4. 如何简单地实现文件拷贝

许多程序员都希望有更简单的方法来实现文件拷贝，而不希望调用 DOS 函数或者编写字节到字节的拷贝函数。Windows 95 API 提供了两个新函数来实现文件拷贝和重新命名，本节将介绍这两个函数。

2.1　查找 Windows 目录和 System 目录

问题

应用程序经常需要确定计算机系统上文件的目录位置。许多程序员在应用程序中对需要的目录位置采取硬编码的方法，但是，假如用户没有把 Windows 系统安装在缺省目录中，而

是安装在其他目录中，那么在应用程序中采取硬编码的方法就有可能出现问题。另外，网络安装也有可能将 System 目录安装在共享文件服务器上，因此，需要一个能确定这些目录位置的方法。

方法

Windows 95 API 提供了许多目录函数。在本节的例子程序中，只使用了最有用的三个。函数 GetWindowsDirectory 返回 Windows 目录的位置，函数 GetSystemDirectory 返回 Windows System 目录的路径，函数 GetTempPath 用来确定保存临时文件的最好位置。

应该注意，尽管三个函数具有同样的参数，但参数的顺序在函数 GetTempPath 中是颠倒的。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，则对话框弹出，显示 Windows 目录、System 目录和临时文件目录。对话框如图 2-1 所示。

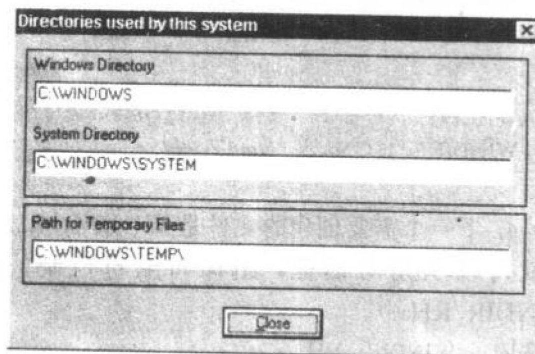


图 2-1　显示 Windows 目录、System 目录和临时文件目录

实现例子程序的具体步骤如下：

1. 创建本例子程序的工作目录 WINDIR，用来保存例子程序的所有源程序。

2. 为例子程序准备图标。利用资源编辑器，创建一个 16 色或少于 16 色的 32×32 象素的图标，并保存在文件 WINDIR.ICO 中。

3. 现在为例子程序创建资源文件。资源文件包括例子程序的图标和例子程序运行时显示的对话框的定义。使用文本编辑器创建新文件 WINDIR.RC，在此文件中输入下列文本：

```
/*----------------------------------------------------------------------*/
/*                                                                      */
/* MODULE: WINDIR.RC                                                    */
/* PURPOSE: Defines the dialog and application icon for the sample.    */
/*                                                                      */
/*----------------------------------------------------------------------*/
#include <windows.h>
#include "windir.rh"
IDD_WINDIRMAIN DIALOG 6, 15, 262, 144
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
        WS_CAPTION | WS_SYSMENU
CAPTION "Directories used by this system"
FONT 8, "MS Sans Serif"
{
   DEFPUSHBUTTON "&Close", IDOK, 106, 124, 50, 14
   CONTROL "", -1, "static", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE,
           5, 7, 251, 69
```

```
    LTEXT      "Windows Directory", -1, 9, 9, 90, 8
    EDITTEXT       IDE_WindowsDIR, 9, 21, 240, 12, ES_AUTOHSCROLL |
                   ES_READONLY | WS_BORDER | WS_TABSTOP
    LTEXT       "System Directory", -1, 9, 41, 90, 8
    EDITTEXT      IDE_SYSTEMDIR, 9, 53, 240, 12, ES_AUTOHSCROLL |
                  ES_READONLY | WS_BORDER | WS_TABSTOP
    CONTROL       "", -1, "static", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE,
                  5, 79, 251, 37
    LTEXT           "Path for Temporary Files", -1, 9, 83, 90, 8
    EDITTEXT IDE_TEMPPATH, 9, 95, 240, 12, ES_AUTOHSCROLL | ES_READONLY |
            WS_BORDER | WS_TABSTOP


IDI_WINDIR    ICON      "windir.ico"
```

4. 下一个需要创建的文件是 WINDIR.RH。下列代码定义资源标识符，因此在例子程序中就可以引用对话框、图标和编辑控制。在新文件中输入下列代码并保存在文件 WINDIR.RH 中。

```
#ifndef __WINDIR_RH
/* -------------------------------------------------------------- */
/*                                                                */
/* MODULE: WINDIR.RH                                              */
/* PURPOSE: Defines resource identifiers for the application.     */
/*                                                                */
/* -------------------------------------------------------------- */
//  Our application icon
#define IDI_WINDIR          201
// The  dialog  and  its  control  IDs
#define   IDD_WINDIRMAIN      1001
#define   IDE_WindowsDIR      101
#define   IDE_SYSTEMDIR       102
#define   IDE_TEMPPATH        104

#endif
```

5. 创建新文件 WINDIR.C。在此文件的顶部输入下列代码段，此代码段包含 Windows 头文件和在第三步中创建的资源头文件，同时还定义预处理器符号 STRICT，表示在 Windows 文件中对菜单和窗口句柄进行严格的类型检查。

```
/* -------------------------------------------------------------- */
/*                                                                */
/* MODULE: WINDIR.C                                               */
/* PURPOSE: This C program demonstrates Windows 95 API calls which */
```

```
/*           can be used to find out the location of the Windows,          */
/*           System, and Temp directories.                                 */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#define     STRICT
#include    <windows.h>
#include    <winnt.h>
#include    <stdlib.h>
#include    "windir.rh"
```

6. 现在添加函数 SetupDialog。此函数用来响应消息 WM_INITDIALOG，此消息在对话框创建时发送。函数调用 GetWindowsDirectory 来获取 Windows 目录的路径，并传送缓冲区以保存路径和说明缓冲区的大小。在本例子程序中使用的是 Windows 定义的常量 MAX_PATH。

下一个函数调用是 GetSystemDirectory，使用同样的参数获取 Windows System 目录，最后，此函数调用函数 GetTempPath 获取临时文件目录。注意，在此函数中缓冲区参数和大小参数的顺序是颠倒的。获取的三个路径放入对话框的只读编辑框中。

```
/*-------------------------------------------------------------------------*/
/* This little function demonstrates our API routines. It calls the        */
/* GetWindowsDirectory () function, the GetSystemDirectory () Function,    */
/* and the GetTempPath () function. Note that the arguments for the        */
/* latter are in the opposite order to the first two. All three            */
/* functions return the length of the string, of 0 if they fail. If        */
/* the length of the string is greater than the size of the buffer         */
/* supplied, the functions return the length of the buffer required.       */
/* We use MAX_PATH to make sure that the buffer is large enough.           */
/*-------------------------------------------------------------------------*/
void SetupDialog (HWND hWnd)
{
    char *dirbuf;
    UINT  size;

    if ( (dirbuf = malloc (MAX_PATH)) == NULL)
        return;
    if ( ( (size = GetWindowsDirectory (dirbuf, MAX_PATH)) != 0) && (size < MAX_PATH))
        SetDlgItemText (hWnd, IDE_WindowsDIR, dirbuf);

    if ( ( (size = GetSystemDirectory (dirbuf, MAX_PATH)) != 0) && (size < MAX_PATH))
        SetDlgItemText (hWnd, IDE_SYSTEMDIR, dirbuf);

    if ( ( (size = GetTempPath (MAX_PATH, dirbuf)) != 0) && (size < MAX_PATH))
        SetDlgItemText (hWnd, IDE_TEMPPATH, dirbuf);
```

```
    free (dirbuf);
}
```

7. 现在添加函数 HandleDialog。这是一个回调函数，用来接收并处理发送给对话框的消息。此函数调用 SetupDialog 来初始化对话框以响应消息 WM _INITDIALOG。此函数还响应 WM _COMMAND 消息 IDCLOSE，此消息在用户点击按钮 Close 时产生。最后，此函数关闭例子程序以响应对话框的关闭。

```
/* ----------------------------------------------------------------------- */
/*  This function receives messages for the Dialog box. It processes       */
/*  the WM_INITDIALOG message, and the ID_CLOSE dialog message.            */
/* ----------------------------------------------------------------------- */
BOOL CALLBACK HandleDialog (HWND hWnd, UINT message,
                            WPARAM wparam, LPARAM lparam)
{
    switch (message)
     {
        case WM_INITDIALOG:
             SetupDialog (hWnd);
             return TRUE;
        case WM_COMMAND:
             if (wparam == IDOK)
             {
                DestroyWindow (hWnd);
                return TRUE;
             }
             break;
        case WM_CLOSE:
             DestroyWindow (hWnd);
             return TRUE;
        case WM_DESTROY:
             PostQuitMessage (0);
             return TRUE;
    }
    return FALSE;
}
```

8. 最后添加的代码是函数 WinMain。在传统的 C 程序中（如本例），函数 WinMain 是应用程序的入口，此函数创建并显示对话框，接着处理消息直到应用程序结束。一旦在源文件中添加了这段代码，便可编译并测试此例子程序。

```
/* ----------------------------------------------------------------------- */
/* Our main function - set up and run the application.                     */
/* ----------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
```

```
{
    MSG   msg;
    HWND hWnd;
    if ( (hWnd = CreateDialog (hInstance,
                              MAKEINTRESOURCE (IDD_WINDIRMAIN),
                              NULL, (DLGPROC) HandleDialog)) == NULL)
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
        if (! IsDialogMessage (hWnd, &msg))
         {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }

    return msg.wParam;
}
```

9. 编译并运行此例子程序。

注释

小心使用 Windows 目录系统是很重要的。不鼓励应用程序将任何文件都放在 Windows 目录下，动态连接库和其他的共享文件可以放在 System 目录下，但如果此目录在网络磁盘上，则需要有相应的网络权限。实际上，应该将共享文件放在登记表中定义的 Common Files 目录中，私有的或应用程序专有的文件应该放在应用程序主目录下的 System 子目录中。登记函数（在第一章中介绍过）提供了存取保存在 Windows 登记表中的大量系统和应用程序位置信息的方法。

2.2 显示 I/O 错误信息

问题

对于 I/O 硬件错误，有的程序员希望阻止系统显示缺省的错误信息，而由应用程序来处理并显示错误信息。是否有简单的方法来显示程序员自己的错误信息呢？

方法

使用 Windows API 函数 SetErrorMode，应用程序可以控制硬件错误信息的显示。如果应用程序调用 I/O 函数并出现错误，Windows 通常在函数返回应用程序前警告用户并向用户询问错误处理操作。图 2-2 所示的便是用户使用 Windows 资源管理器存取没有软盘的驱动器时发生的错误。通过禁止 Windows 处理致命的错误，应用程序便可以强制 I/O 函数调用失败时不显示对话框，而是返回可供分析的错误代码。

步骤

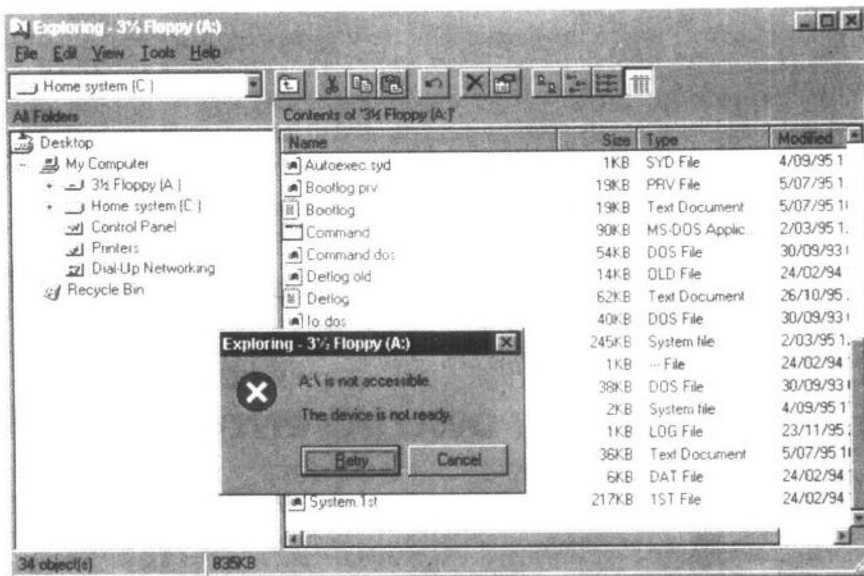


图 2-2　标准的 Windows 错误信息框

按照下列步骤实现一个例子程序。运行此例子程序，则列表框中列出可用的磁盘驱动器。双击某一驱动器或点击按钮 Write File，例子程序就试着在驱动器的根目录下创建文件。如果没有选中核选框 Handle Errors，则由 Windows 来处理操作错误。向有写保护的盘写入，或向未插盘的软盘驱动器写入，都将由 Windows 95 来显示错误信息。

如果选中核选框 Handle Errors，点击按钮 Write File，则程序将直接处理所有的操作错误，显示的信息如图 2-3 所示。

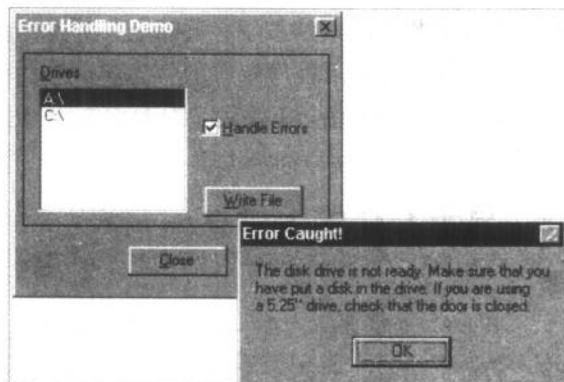


图 2-3　由例子程序处理的错误信息

可以在各种不同的情况下测试此例子程序：驱动器中无盘，磁盘满，磁盘写保护，或磁盘未格式化。例子程序根据核选框 Handle Errors 是否被选中来决定是否使用函数 SetErrorMode 和处理错误信息。

实现例子程序的具体步骤如下：

1. 创建新目录 GETERRS，使用此目录作为开发本例子程序的工作目录。

2. 现在为例子程序创建资源脚本。使用文本编辑器创建新文件 GETERRS.RC，并在此文件中添加下列代码。此资源脚本定义例子程序的菜单和图标。

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/* MODULE: GETERRS.RC                                                 */
/* PURPOSE: Resource script for use in the GETERRS application.       */
/*          The application uses SetErrorMode to control the display  */
```

```
/*          of errors.                                                    */
/*                                                                        */
/*------------------------------------------------------------------------*/
#include    <windows.h>
#include    "geterrs.rh"

IDD_TESTDIALOG  DIALOG  6, 15, 162, 119
STYLE  DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
       WS_CAPTION | WS_SYSMENU
CAPTION   "Error Handling Demo"
FONT   8,    "MS Sans Serif"
{
CONTROL  "", -1, "static", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE,
         3, 6, 154, 81
LTEXT     "&Drives", -1, 12, 11, 34, 8
LISTBOX     IDL_DRIVELIST, 12, 23, 73, 59, LBS_STANDARD | WS_TABSTOP
AUTOCHECKBOX     "&Handle Errors", IDC_HANDLEERRORS, 93, 35, 60, 12
DEFPUSHBUTTON     "&Write File", IDTEST, 94, 68, 50, 14
PUSHBUTTON         "&Close", IDCLOSE, 56, 96, 50, 14
}

IDI_ERRICON   ICON   "geterrs.ico"
```

3. 为例子程序准备图标文件。使用资源编辑器创建 32×32 象素的图标，并保存在工作目录中的文件 GETERRS. ICO 中。

4. 在刚创建的资源脚本中引用了 include 文件 GETERRS. RH，此文件包含菜单项的资源定义，以便于资源脚本引用这些资源，同样也便于例子程序的源代码引用。创建新文件并存为 GETERRS. RH，在此文件中输入下列定义：

```
#ifndef __GETERRS_RH
/*------------------------------------------------------------------------*/
/*                                                                        */
/* MODULE: GETERRS.RH                                                     */
/* PURPOSE: Defines resource identifiers.                                 */
/*                                                                        */
/*------------------------------------------------------------------------*/
#define      __GETERRS_RH

#define     IDD_TESTDIALOG          200
#define     IDC_HANDLEERRORS        101
#define     IDL_DRIVELIST           102
#define     IDTEST                  103
```

```
#define     IDCLOSE                104

#define     IDI_ERRICON            201

#endif // not     __GETERRS_RH
```

5. 现在创建例子程序的源文件。本例子程序用普通C语言写成，可以用大多数C语言和C++语言编译器编译。创建新文件GETERRS.C，在此文件的顶部插入下列代码，此段代码包含编译Windows应用程序所需要的文件，另外还定义测试文件的名字和两行写入文件的文本。

```
/* -------------------------------------------------------------- */
/*                                                                */
/* MODULE: GETERRS.C                                              */
/* PURPOSE: This C program demonstrates a method of avoiding the  */
/*          default system display of critical errors, and receiving */
/*          the results in the application instead. The application */
/*          uses SetErrorMode to control the display of errors.   */
/*                                                                */
/* -------------------------------------------------------------- */
/*                                                                */
/* Copyright 1995, Andrew Cooke. All rights reserved.             */
/*                                                                */
/* -------------------------------------------------------------- */
#define     STRICT
#include    <windows.h>
#include    <winnt.h>
#include    <windowsx.h>
#include    <stdio.h>
#include    "geterrs.rh"

static  char  *FileNameToWrite = "x: TestFile.txt";
static  char  *DataToWrite = "Sample data file\r\n"
                             "Created by GETERRS example application\r\n";
```

6. 在同一文件中添加下列代码。函数DisplayErrorMessages将传送来的错误代码解释成要显示的字符串，并查找写文件时常碰到的错误，switch语句的最后一种情况用来处理所有其他的错误。

```
/* -------------------------------------------------------------- */
/* This function displays a dialog with an error message. It uses a */
/* large switch statement to generate useful messages for a group */
/* of common I/O errors.                                          */
/* -------------------------------------------------------------- */
void DisplayErrorMessages (HWND hParent, DWORD errorCode)
```

```
{
    char *errMsg;
    switch (errorCode)
     {
        case ERROR_WRITE_PROTECT:
        case ERROR_ACCESS_DENIED:
            errMsg = "The disk that you have inserted is\n"
                     "write-protected. Move the write-protect\n"
                     "tab and reinsert the disk.";
            break;
        case ERROR_BAD_DEVICE:
        case ERROR_BAD_UNIT:
        case ERROR_INVALID_DRIVE:
            errMsg = "The requested disk device does not appear to\n"
                     "be available. Perhaps the device is not properly\n"
                     "installed on your system.";
            break;
        case ERROR_CRC:
        case ERROR_SECTOR_NOT_FOUND:
            errMsg = "A CRC error was encountered when writing to\n"
                     "the disk. Please use ScanDisk to check the\n"
                     "surface of this disk for errors.";
            break;
        case ERROR_DISK_CHANGE:
        case ERROR_FILE_INVALID:
        case ERROR_WRONG_DISK:
            errMsg = "You appear to have changed the disk while this\n"
                     "application had a file open on it.";
            break;
        case ERROR_DISK_CORRUPT:
        case ERROR_FLOPPY_UNKNOWN_ERROR:
        case ERROR_GEN_FAILURE:
        case ERROR_NOT_DOS_DISK:
            errMsg = "This disk does not appear to be formatted. Use\n"
                     "Explorer to format the disk, or insert another\n"
                     "disk.";
            break;
        case ERROR_DISK_FULL:
        case ERROR_HANDLE_DISK_FULL:
            errMsg = "There is no room on this disk to write any more\n"
                     "data. Please insert another disk.";
            break;
        case ERROR_NOT_READY:
```

```
            errMsg = "The disk drive is not ready. Make sure that you\n"
                     "have put a disk in the drive. If you are using\n"
                     "a 5.25\" drive, check that the door is closed.";
            break;
        default:
            errMsg = "The application caught an error message which\n"
                     "there is not an explanation for. A complete\n"
                     "example would handle this instance.";
            break;
    }
    MessageBox (hParent, errMsg,"Error Caught!", MB_OK);
}
```

7. 现在在文件中添加函数 TestFileIO。此函数试着向选中的驱动器写入一个小的文本文件，在此函数中，首先检查核选框 Handle Errors 当前是否选中的，如果选中，则调用函数 SetErrorMode（参数为 SEM_FAILCRITICALERRORS），从而禁止 Windows 显示错误信息并使 I/O 函数返回错误代码。原模式保存在变量 oldErrorState 中，以便将来恢复。

此函数接着在选中的驱动器上试着打开、写入和关闭文件。驱动器符由列表框中得到，如果发生错误且缺省的错误处理被关闭，此函数调用 GetLastError 来取回错误代码，接着调用 DisplayErrorMessages 显示结果。

```
/* ------------------------------------------------------------------- */
/* This function attempts to write a few lines to a file on the        */
/* specified disk drive. It retrieves the error code (if any) . If     */
/* the handleErrors argument is non-zero, the function uses the        */
/* SetErrorMode API function to SEM_FAILCRITICALERRORS,                */
/* and displays a message box depending upon the result of the I/O     */
/* functions.                                                          */
/* ------------------------------------------------------------------- */
void TestFileIO (HWND hWnd)
{
    UINT        oldErrorState;
    BOOL        handleErrors, hadErrors;
    DWORD       errorCode;
    char        fileName [14], filePath [8];
    int         selItem;
    FILE        *fileHandle;

    // Find out if we should handle errors or not.
    handleErrors = SendDlgItemMessage (hWnd, IDC_HANDLEERRORS,
                                       BM_GETCHECK, 0, 0);

    // Disable the default Windows error handler, if required.
```

```
    if (handleErrors)
        oldErrorState = SetErrorMode (SEM_FAILCRITICALERRORS);
    // Make up the file name to open.
    selItem = SendDlgItemMessage (hWnd, IDL_DRIVELIST,
                                  LB_GETCURSEL, 0, 0);
    if (selItem == LB_ERR)
        MessageBox (hWnd,"Please select a drive to write to.",
                  "Warning", MB_OK | MB_ICONEXCLAMATION);
    SendDlgItemMessage (hWnd, IDL_DRIVELIST, LB_GETTEXT,
                       selItem, (LPARAM) filePath);
    lstrcpy (fileName, FileNameToWrite);
    fileName [0] = filePath [0];

    // Attempt to open and write to a file.
    hadErrors = FALSE;
    fileHandle = fopen (fileName,"wt");
    if (fileHandle)
     {
        if (fwrite (DataToWrite, 1, lstrlen (DataToWrite), fileHandle) == 1)
            hadErrors = (fclose (fileHandle) != 0);
    }
    else
        hadErrors = TRUE;

    // Test for errors.
    if ( (hadErrors) && (handleErrors))
     {
        errorCode = GetLastError ();
        DisplayErrorMessages (hWnd, errorCode);
    }

    if (! hadErrors)
        MessageBox (hWnd,"File written successfully.",
                         "No Errors", MB_ICONINFORMATION | MB_OK);
    // Restore the original error mode.
    if (handleErrors)
        SetErrorMode (oldErrorState);
}
```

8. 在同一源文件中添加函数 SetupDialog。此函数使用 API 函数 GetLogicalDrives 获取一个 DWORD 的屏蔽码，此屏蔽码定义系统上可用的驱动器，低 26 位的每一位对应一个驱动器。置 1 表示驱动器可用，置 0 表示驱动器不可用。此函数反复检查屏蔽码，为每个可用的驱动器产生一个字符串，如"A：\"，字符串插入在列表框中。

```
/* ------------------------------------------------------------------------ */
/* The SetupDialog function is called in response to the                    */
/* WM_INITDIALOG message. It uses GetLogicalDrives () to                     */
/* retrieve a  mask specifying the available drives, then converts          */
/* that mask to strings, placing each string into the list box.             */
/* ------------------------------------------------------------------------ */
static void SetupDialog (HWND hWnd)
{
    char            driveString [8];
    DWORD           drives;
    int             i;
    HWND            hWndList;

    // Get a handle to the list box.
    hWndList = GetDlgItem (hWnd, IDL_DRIVELIST);

    // Retrieve a mask of drives.
    drives = GetLogicalDrives ();

    // Test 26 bits in the mask, to see if those drives are available.
    for (i = 0; i < 26; i++)
        if (drives & (1L << i))
         {
         wsprintf (driveString,"%c: \\", i + 'A');
         SendMessage (hWndList, LB_ADDSTRING, 0, (LPARAM) driveString);
         }
}
```

9. 函数 HandleDialog 接收发送给对话框的消息。为响应消息 WM_INITDIALOG，函数获取可用的驱动器并显示在列表框中。为响应消息 IDCLOSE（此消息在点击 Close 按钮时发送），函数保存选中的驱动器符并关闭对话框。为响应消息 IDTEST 或列表框中的双击，则调用前面添加的函数 TestFileIO。

```
/* ------------------------------------------------------------------------ */
/* This callback function handles the messages sent to the dialog.          */
/* ------------------------------------------------------------------------ */
LRESULT CALLBACK HandleDialog (HWND hWnd, UINT message,
                                WPARAM wParam, LPARAM lParam)
{
    UINT notifycode, idcode;

    switch (message)
     {
```

```
        case WM_INITDIALOG:
            SetupDialog (hWnd);
            return TRUE;
        case WM_COMMAND:
            idcode = GET_WM_COMMAND_ID (wParam, lParam);
            notifycode = GET_WM_COMMAND_CMD (wParam, lParam);
            if (idcode == IDCLOSE)
            {
                DestroyWindow (hWnd);
                return TRUE;
            }
            else
                if ( (idcode == IDTEST) ||
                    ( (idcode == IDL_DRIVELIST) &&
                      (notifycode == LBN_DBLCLK)))
                {
                    TestFileIO (hWnd);
                    return TRUE;
                }
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            return TRUE;
    }
    return FALSE;
}
```

10. 最后，在文件中添加函数 WinMain。此函数创建并显示对话框，接着处理消息直到例子程序结束，一旦添加此函数，便可编译并测试此例子程序。

```
/* ---------------------------------------------------------------------- */
/* The Windows entry point of the application. Initialise the dialog      */
/* and process messages until the application is closed.                  */
/* ---------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG         msg;
    HWND        hWnd;

    if ( (hWnd = CreateDialog (hInstance,
                        MAKEINTRESOURCE (IDD_TESTDIALOG),
                        NULL, (DLGPROC) HandleDialog)) == NULL)
        return FALSE;
```

```
    while (GetMessage (&msg, NULL, 0, 0))
        if (! IsDialogMessage (hWnd, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }

    return msg.wParam;
}
```

11. 编译并运行此例子程序。

注释

为了替换缺省的错误信息，尤其是为了使应用程序具有更加友好的用户界面，Windows API 函数 SetErrorMode 提供了一个有用的解决方案，应用程序显示的错误信息，应尽量的具体清晰，以便于用户理解。

2.3 检测驱动器中是否有盘

问题

在向软盘写入前，应用程序最好检测一下驱动器中是否有盘。在驱动器中没盘时，能否不要 Windows 通知用户，而由应用程序来实现此功能呢？

方法

检测驱动器中是否有盘的最简单的方法是存取此驱动器。如果驱动器中没盘或盘没有格式化，Windows 就会显示错误信息。为了避免这样，可以调用 API 函数 SetErrorMode，使用参数 SEM_FAILCRITICALERRORS 来禁止 Windows 显示错误消息，而使用函数 GetLastError 来获取错误代码。

在本节的例子程序中，也使用了函数 GetLogicalDrives 来获取所有可用的驱动器，并显示在对话框中，以便于用户选取。函数 GetLogicalDrives 返回一个 DWORD 的屏蔽码，每个驱动器用一位来表示，26 个可能的驱动器用屏蔽码的 26 位来表示。如果某位置 1，则相应的驱动器存在。例如：第 0 位置 1，则 A：驱动器存在；第 3 位置 1，则 C：驱动器存在。

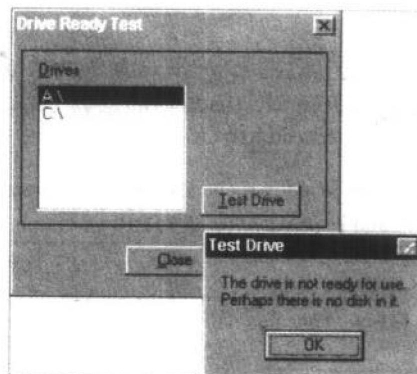


图 2-4 检测驱动器中是否有盘

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将弹出对话框，显示可用的逻辑驱动器。点击某个驱动器并点击按钮 Test Drive，例子程序将弹出消息框以指示驱动器中是否有盘，如图 2-4 所示。

实现例子程序的具体步骤如下：

1. 创建新目录 DISKRDY，使用此目录保存例子程序开发时的源文件。

2. 接着，创建资源脚本以定义例子程序中使用的对话框。使用文本编辑器创建新文件 DISKRDY.RC，在此文件中添加下列代码行：

```
/* ------------------------------------------------------------ */
```

```
/*                                                                        */
/* MODULE: DISKRDY.RC                                                     */
/* PURPOSE: This resource script defines the dialog and icon.             */
/*                                                                        */
/* ---------------------------------------------------------------------- */
#include <windows.h>
#include "diskrdy.rh"

IDD_DISKRDYMAIN    DIALOG   6, 15, 162, 119
STYLE  DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
       WS_CAPTION | WS_SYSMENU
CAPTION    "Drive Ready Test"
FONT    8,    "MS Sans Serif"
{
CONTROL  "", -1, "static", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE,
          3, 6, 154, 81
LTEXT       "&Drives", -1, 12, 11, 34, 8
LISTBOX     IDL_DRIVELIST, 12, 23, 73, 59, LBS_STANDARD | WS_TABSTOP
DEFPUSHBUTTON     "&Test Drive", IDTEST, 94, 68, 50, 14
PUSHBUTTON     "&Close", IDCLOSE, 56, 96, 50, 14
}

IDI_APPICON    ICON    "diskrdy.ico"
```

3. 在资源脚本中包含了另外一个文件 DISKRDY.RH，此文件定义对话框和控制的标识符，使用文本编辑器创建此文件，并输入下列定义：

```
#ifndef __DISKRDY_RH
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: DISKRDY.RH                                                     */
/* PURPOSE: Defines resource identifiers for use in the application.      */
/*                                                                        */
/* ---------------------------------------------------------------------- */
#define __DISKRDY_RH

#define    IDD_DISKRDYMAIN          301
#define    IDL_DRIVELIST            101
#define    IDTEST                   102
#define    IDCLOSE                  103

#define    IDI_APPICON              302
#endif //  not    __DISKRDY_RH
```

4. 现在创建例子程序的源文件 DISKRDY. C，在此文件的顶部添加下列代码段。此代码段包含 Windows 的头文件，定义预处理器符号 STRICT，此符号表示对程序中使用的窗口句柄和类型进行严格的类型检查。

```
/*-----------------------------------------------------------*/
/*                                                           */
/* MODULE: DISKRDY.C                                         */
/* PURPOSE: This small application demostrates how to test if there  */
/*          is a formatted disk in the drive.                */
/*                                                           */
/*-----------------------------------------------------------*/
#define     STRICT
#include    <windows.h>
#include    <windowsx.h>
#include    <io.h>
#include    "diskrdy.rh"
```

5. 在同一源文件中添加函数 SetupDialog。此函数响应消息 WM _ INITDIALOG，此消息在对话框第一次创建时发送。此函数调用 API 函数 GetLogicalDrives，接着分析返回值的低 26 位，如果某位为 1，则相应的驱动器符（A－Z）就显示在列表中。

```
/*-----------------------------------------------------------*/
/* The SetupDialog function is called in response to the     */
/* WM_INITDIALOG message. It uses GetLogicalDrives () to get */
/* a mask of available drives, then converts the mask to strings, placing */
/* each string in the list box.                              */
/*-----------------------------------------------------------*/
static  void  SetupDialog (HWND  hWnd)
{
    char        driveString [8];
    DWORD       drives;
    int         i;
    HWND        hWndList;

    // Get a handle to the list box.
    hWndList = GetDlgItem (hWnd, IDL_DRIVELIST);

    // Retrieve a mask of drives.
    drives = GetLogicalDrives ();

    // Test 26 bits in the mask, to see if those drives are available.
    for (i = 0; i < 26; i++)
        if (drives & (1L << i))
```

```
        {
        wsprintf (driveString," %c: \\", i + 'A');
        SendMessage (hWndList, LB_ADDSTRING, 0, (LPARAM) driveString);
        }
}
```

6. 在同一文件中添加下列代码，其运行机制为：函数 TestDriveReady 使用 C 库函数 access 来测试给定驱动器的根目录是否可存取，如果因某些原因此驱动器不可存取，则函数 access 返回非零值。通常，如果应用程序存取一个未准备好的驱动器，则 Windows 会弹出一个对话框。为了避免这种情况，下列代码在存取驱动器前调用函数 SetErrorMode（参数为 SEM_FAILCRITICALERRORS）来禁止 Windows 在错误发生时显示错误信息，从而应用程序可检测函数 access 的返回值以确定驱动器是否可用。

使用函数 SetErrorMode，应该存储原错误模式并在操作完成后予以恢复。在下列代码段中，变量 oldErrorMode 用来保存原错误模式，最后再调用函数 SetErrorMode 来进行恢复。此段代码显示消息框以指示驱动器是否可存取。

```
/* ---------------------------------------------------------------- */
/* The TestDriveReady function uses a DeviceIoControl API to test if  */
/* the specified drive is ready. It puts up a message box to show     */
/* the result. If the user has not selected a list box item, they     */
/* are asked to.                                                      */
/* ---------------------------------------------------------------- */
void TestDriveReady (HWND hWnd)
{
    int         curItem;
    char        name [8];
    UINT        oldErrorMode;

    // Find the selected item in the listbox.
    if ( (curItem =SendDlgItemMessage (hWnd, IDL_DRIVELIST,
                                LB_GETCURSEL, 0, 0)) == LB_ERR)
     {
        MessageBox (hWnd,"You have not selected a drive to test.\n"
                    "Please select one of the drives in the\n"
                    "list, then choose Test Drive.","Test Error", MB_OK);
        return;
    }
    // Retrieve the root directory path from the listbox.
        SendDlgItemMessage (hWnd, IDL_DRIVELIST, LB_GETTEXT, curItem, (LPARAM)
name);

    // Turn on error handling.
    oldErrorMode = SetErrorMode (SEM_FAILCRITICALERRORS);
```

```
    // Use the C runtime library function access to check the drive.
    if (access (name, 0) == 0)
        MessageBox (hWnd,"The drive is ready for use.\n",
                    "Test Drive", MB_OK);
    else
        MessageBox (hWnd,"The drive is not ready for use.\n"
                          "Perhaps there is no disk in it.",
                    "Test Drive", MB_OK);
    // Restore the old error mode.
    SetErrorMode (oldErrorMode);
}
```

7. 现在添加下列函数。函数 HandleDialog 用来处理发送给对话框的消息，当响应信息 WM_INITDIALOG 时，此函数设置可用驱动器列，当响应消息 IDTEST 和 IDCOLSE 时，此函数相应地测试驱动器或关闭例子程序。函数 WinMain 是例子程序的入口点，创建并显示对话框，接着处理消息直到例子程序关闭。

```
/* ------------------------------------------------------------------- */
/* This callback function handles the messages sent to the dialog.     */
/* ------------------------------------------------------------------- */
LRESULT CALLBACK HandleDialog (HWND hWnd, UINT message,
                               WPARAM wParam, LPARAM lParam)
{
    UINT notifycode, idcode;

    switch (message)
     {
        case WM_INITDIALOG:
             SetupDialog (hWnd);
             return TRUE;
        case WM_COMMAND:
             idcode = GET_WM_COMMAND_ID (wParam, lParam);
             notifycode = GET_WM_COMMAND_CMD (wParam, lParam);
             if (idcode == IDCLOSE)
             {
                 DestroyWindow (hWnd);
                 return TRUE;
             }
             else
                 if ( (idcode == IDTEST) ||
                      ( (idcode == IDL_DRIVELIST) &&
                        (notifycode == LBN_DBLCLK)))
                 {
```

```
                TestDriveReady (hWnd);
                return TRUE;
            }
        break;
    case WM_DESTROY:
        PostQuitMessage (0);
        return TRUE;
    }
    return FALSE;
}
/* ---------------------------------------------------------------- */
/* The Windows entry point of the application. Initialise the dialog */
/* and process messages until the application is closed.             */
/* ---------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG         msg;
    HWND        hWnd;

    if ( (hWnd = CreateDialog (hInstance,
                        MAKEINTRESOURCE (IDD_DISKRDYMAIN),
                        NULL, (DLGPROC) HandleDialog)) == NULL)
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
        if (! IsDialogMessage (hWnd, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }

    return msg.wParam;
}
```

8. 准备例子程序的图标文件。使用资源编辑器创建一个 32×32 象素的图标，并保存在工作目录中的文件 DISKRDY.ICO 中。

注释

本节示范了一个相当有效的方法，以用来在使用驱动器前检测驱动器是否准备好。Windows 的将来版本可能有更好的检测方法。在 Windows NT 中实现了函数 DeviceIoControl，参数为 IOCTL_DISK_CHECK_VERIFY 时允许用户透明地检测驱动器的状态，可是，在 Windows 95 中没有实现此 Win32 功能，将来能否实现还需拭目以待。

2.4 简单地实现文件拷贝

问题

重新命名文件可以使用编程语言中的库函数 rename 来实现，但是源文件和目的文件都要求在同一驱动器上。如何不编写代码就可以实现驱动器间的文件拷贝和移动呢？

方法

Windows 95 提供了两个用于文件拷贝和移动的函数，函数 CopyFile 为单个文件创建拷贝，函数 MoveFile 用来移动文件的位置，也可用来重新命名文件。例如：可以使用函数 MoveFile 把文件从一个驱动器移动到另一个驱动器上；在同一个驱动器上，可以移动一个目录及其所有内容或对其进行重新命名。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从菜单 File 中选择菜单项 Copy File。例子程序将使用标准的打开文件对话框，提示用户选取要拷贝的源文件，接着使用标准的文件存盘对话框，提示用户选取要拷贝的目的文件，可以覆盖已存在的文件或创建新文件，但文件不能是只读的，不能在写保护的驱动器或目录中，接着开始拷贝文件。本例子程序不允许拷贝文件到自身。使用此例子程序拷贝文件如图 2-5 所示。

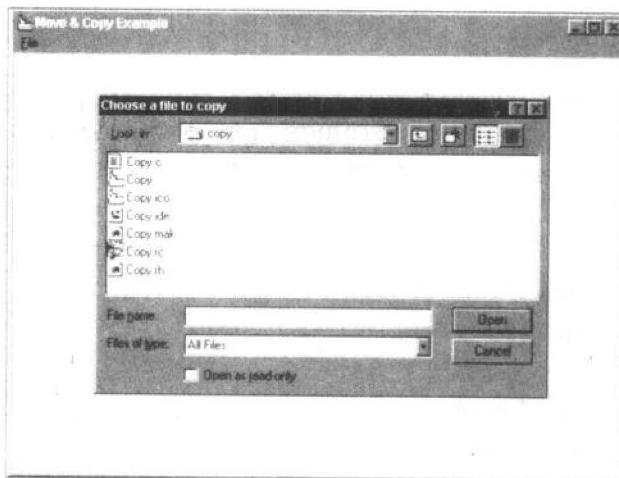


图 2-5 使用 API 函数拷贝文件

从菜单 File 中选择菜单项 Move File。同拷贝文件一样，此例子程序将提示用户选取源文件和目的文件，接着文件被移动或重新命名，源文件则消失。可以使用资源管理器来验证此例子程序的操作是否正确。

实现例子程序的具体步骤如下：

1. 首先为例子程序创建工作目录 COPY，在开发例子程序时使用此目录来保存所有的源文件。

2. 现在定义例子程序的对话框。使用文本编辑器创建新文件 COPY.RC，在此文件中添

加下列资源脚本：

```
/* ------------------------------------------------------------------------------------- */
/*                                                                                       */
/* MODULE: COPY.RC                                                                       */
/* PURPOSE: Resource script for use in the COPY application.                             */
/*          Supplies the menu and the application icon.                                  */
/*                                                                                       */
/* ------------------------------------------------------------------------------------- */
#include    <windows.h>
#include    "copy.rh"

IDM_COPYMENU    MENU
{
  POPUP    "&File"
   {
     MENUITEM    "&Copy File",    CM_FILECOPY
     MENUITEM    "&Move File",    CM_FILEMOVE
     MENUITEM    SEPARATOR
     MENUITEM    "E&xit",    CM_EXIT
  }
}

IDI_COPY    ICON    "copy.ico"
```

3. 创建定义资源标识符的 include 文件，此文件用在资源脚本中，也用在源代码中，以引用对话框及其控制。创建新文件 COPY.RH，在此文件中输入下列定义：

```
#ifndef __COPY_RH
/* ------------------------------------------------------------------------------------- */
/*                                                                                       */
/* MODULE: COPY.RH                                                                       */
/* PURPOSE: Defines resource identifiers for use in the COPY                             */
/*          application. The identifiers are used to display the                         */
/*          menu and respond to commands, and to set the app icon.                       */
/*                                                                                       */
/* ------------------------------------------------------------------------------------- */
#define __COPY_RH

#define    IDM_COPYMENU    201
#define    CM_FILECOPY     101
#define    CM_FILEMOVE     102
#define    CM_EXIT         103

#define    IDI_COPY        202
```

```
#endif  // not  __COPY_RH
```

4. 现在创建例子程序的源文件。创建新文件COPY.C，在此文件的顶部添加下列代码。此段代码包含所需要的 Windows 头文件，接着定义过滤器，以用在文件对话框中，此过滤器使用文件通配符 *.* 来请求 "All Files"，过滤器字符串用空字符 (\0) 来分隔字符串，在编译时C编译器将在字符串末再添加一个空字符，在一行中有两个空字符则表示过滤器列的结束。

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/* MODULE: COPY.C                                                  */
/* PURPOSE: This application demonstrates the use of two Windows   */
/*          API functions. CopyFile copys a file from one name,    */
/*          device or directory to another. MoveFile renames a file, */
/*          or directory, or moves the file or directory to another */
/*          location. MoveFile can move files across devices, but  */
/*          cannot move directories across devices.                */
/*                                                                 */
/*-----------------------------------------------------------------*/
#define    STRICT
#include   <windows.h>
#include   <windowsx.h>
#include   <commdlg.h>
#include   "copy.rh"

static  char  *WindowClassName  =  "SampleCopyWindow";
static  char  *FilterStrings  =  "All Files\0*.*\0";
```

5. 现在添加函数 Copy。此函数初始化 OPENFILENAME 结构，并使用此结构来从用户那里获取源文件名 oldName，这是通过调用 API 函数 GetOpenFileName 来完成的，接着重新初始化同一结构，以用在函数 GetSaveFileName 中，此函数取回目的文件名并保存在变量 newName 中。

应该注意，在调用函数 GetOpenFileName 时，结构只设置了标志 OFN_FILEMUSTEXIST；当此标志用于函数 GetSaveFileName 时，用来指示路径必须存在，如果试图覆盖已存在的文件则应该警告用户，此标志还指示选取的文件不能是只读的或在写保护目录中。

接着调用 API 函数 CopyFile，传送源文件名和目的文件名。如果目的文件已经存在，那么用户已经被警告过，所以此函数的第三个参数 fFailIfExists 设置为 FALSE。

```
/*-----------------------------------------------------------------*/
/* This function uses GetOpenFileName and GetSaveFileName           */
/* common dialog box API functions to get the old and new names    */
/* for the file to copy. It then uses CopyFile to perform the operation. */
/* The fFailIfExists argument is always set to FALSE, because the  */
/* user will have already been warned if an old file would be      */
```

```
/* overwritten.                                                              */
/* ------------------------------------------------------------------------- */
void Copy (HWND hWnd)
{
    char          oldName [MAX_PATH], newName [MAX_PATH];
    char          extension [4];
    OPENFILENAME  ofn;

    oldName [0] = newName [0] = 0;
    FillMemory (&ofn, sizeof (OPENFILENAME), 0);
    ofn.lStructSize = sizeof (OPENFILENAME);
    ofn.hwndOwner = hWnd;
    ofn.lpstrFilter = FilterStrings;
    ofn.nFilterIndex = 1;
    ofn.lpstrFile = oldName;
    ofn.nMaxFile = MAX_PATH;
    ofn.lpstrTitle = "Choose a file to copy";
    ofn.Flags = OFN_FILEMUSTEXIST;
    if (GetOpenFileName (&ofn))
     {
        lstrcpy (newName, oldName);
        if (ofn.nFileExtension)
            lstrcpy (extension, &oldName [ofn.nFileExtension]);
        else
            extension [0] = 0;
        FillMemory (&ofn, sizeof (OPENFILENAME), 0);
        ofn.lStructSize = sizeof (OPENFILENAME);
        ofn.hwndOwner = hWnd;
        ofn.lpstrFilter = FilterStrings;
        ofn.nFilterIndex = 1;
        ofn.lpstrFile = newName;
        ofn.nMaxFile = MAX_PATH;
        ofn.lpstrTitle = "Choose a name to copy the file to";
        if (extension [0])
            ofn.lpstrDefExt = extension;
        ofn.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
        OFN_OVERWRITEPROMPT | OFN_NOREADONLYRETURN;
        if (GetSaveFileName (&ofn))
            if (lstrcmp (oldName, newName) ! = 0)
             {
                if (CopyFile (oldName, newName, FALSE))
                    MessageBox (hWnd,"Copy completed", NULL,
                                MB_ICONINFORMATION | MB_OK);
```

```
            else
                MessageBox (hWnd,"Error during copy", NULL,
                            MB_ICONSTOP | MB_OK);
        }
        else
            MessageBox (hWnd,"Old and new file names cannot\n"
                             "be the same.",
                        NULL, MB_ICONSTOP | MB_OK);
    }
}
```

6. 下一个要添加的函数 Move 与前面的函数 Copy 非常类似。获取文件名的代码同函数 Copy 中的基本一致，不同的是源文件（用函数 GetOpenFileName 获取）不能是只读文件或在写保护目录中，这是因为函数 MoveFile 要删除源文件。接着调用函数 MoveFile，同函数 CopyFile 有同样的参数(源文件名，目的文件名和标志 fFailIfExists)。如果试图覆盖已存在的文件，那么用户已经被警告过，所以此函数的第三个参数 fFailIfExists 设置为 FALSE。

```
/* ------------------------------------------------------------ */
/* This function uses GetOpenFileName and GetSaveFileName      */
/* common dialog box API functions to get the old and new names for */
/* the file to copy. It then uses MoveFile to perform the operation. The */
/* fFailIfExists argument is always set to FALSE, because the user */
/* will have already been warned if an old file would be overwritten. */
/* Note that MoveFile can also be used to move directories (and all */
/* their files and subdirectories in the same operation), providing */
/* that the destination is on the same device.                  */
/* ------------------------------------------------------------ */
void Move (HWND hWnd)
{
    char          oldName [MAX_PATH], newName [MAX_PATH];
    char          extension [4];
    OPENFILENAME ofn;

    oldName [0] = newName [0] = 0;
    FillMemory (&ofn, sizeof (OPENFILENAME), 0);
    ofn.lStructSize = sizeof (OPENFILENAME);
    ofn.hwndOwner = hWnd;
    ofn.lpstrFilter = FilterStrings;
    ofn.nFilterIndex = 1;
    ofn.lpstrFile = oldName;
    ofn.nMaxFile = MAX_PATH;
    ofn.lpstrTitle = "Choose a file to move";
    ofn.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
    OFN_NOREADONLYRETURN;
    if (GetOpenFileName (&ofn))
```

```
    {
        strcpy (newName, oldName);
        if (ofn.nFileExtension)
            lstrcpy (extension, &oldName [ofn.nFileExtension]);
        else
            extension [0] = 0;
        memset (&ofn, 0, sizeof (OPENFILENAME));
        ofn.lStructSize = sizeof (OPENFILENAME);
        ofn.hwndOwner = hWnd;
        ofn.lpstrFilter = FilterStrings;
        ofn.nFilterIndex = 1;
        ofn.lpstrFile = newName;
        ofn.nMaxFile = MAX_PATH;
        ofn.lpstrTitle = "Choose a name to move the file to";
        if (extension [0])
            ofn.lpstrDefExt = extension;
        ofn.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
        OFN_OVERWRITEPROMPT | OFN_NOREADONLYRETURN;
        if (GetSaveFileName (&ofn))
            if (lstrcmp (oldName, newName) ! = 0)
             {
                if (CopyFile (oldName, newName, FALSE))
                    MessageBox (hWnd,"Move completed", NULL,
                                MB_ICONINFORMATION | MB_OK);
                else
                    MessageBox (hWnd,"Error during move", NULL,
                                MB_ICONSTOP | MB_OK);
            }
            else
                MessageBox (hWnd,"Old and new file names cannot\n"
                                "be the same.",
                            NULL, MB_ICONSTOP | MB_OK);
    }
}
```

7. 下一个要添加的函数是 MainWndProc。此函数处理发送给例子程序主窗口的消息，尤其是来自菜单的命令 CM_FILECOPY 和 CM_FILEMOVE，此函数分别调用函数 Copy 或 Move 来响应。

```
/* --------------------------------------------------------------- */
/* This callback function handles the messages for the window class.   */
/* --------------------------------------------------------------- */
LRESULT CALLBACK MainWndProc (HWND hWnd, UINT message,
                              WPARAM wParam, LPARAM lParam)
```

```
{
    switch (message)
    {
        case WM_COMMAND:
            switch (GET_WM_COMMAND_ID (wParam, lParam))
            {
                case CM_EXIT:
                    DestroyWindow (hWnd);
                    break;
                case CM_FILECOPY:
                    Copy (hWnd);
                    break;
                case CM_FILEMOVE:
                    Move (hWnd);
                    break;
                default:
                    break;
            }
            break;
        case WM_QUIT:
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return (DefWindowProc (hWnd, message, wParam, lParam));
    }
    return 0;
}
```

8. 最后，添加例子程序的框架函数，这三个函数建立起例子程序并使其运行。函数 InitApplication 注册主窗口类，此函数只有在此类窗口不存在时才被调用。函数 InitInstance 创建并显示主窗口。最后一个函数 WinMain 是例子程序的入口点，调用其他的函数来建立例子程序，接着处理消息直到例子程序结束。

```
/* ----------------------------------------------------------------- */
/* Register the window class for the application's main window.     */
/* ----------------------------------------------------------------- */
BOOL InitApplication (HINSTANCE  hInstance)
{
    WNDCLASS  wndClass;

    wndClass.style = 0;
    wndClass.lpfnWndProc = (WNDPROC) MainWndProc;
    wndClass.cbClsExtra = 0;
```

```
    wndClass.cbWndExtra = 0;
    wndClass.hInstance = hInstance;
    wndClass.hIcon=LoadIcon (hInstance, MAKEINTRESOURCE (IDI_COPY));
    wndClass.hCursor = LoadCursor (NULL, IDC_ARROW);
    wndClass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wndClass.lpszMenuName = MAKEINTRESOURCE (IDM_COPYMENU);
    wndClass.lpszClassName = WindowClassName;

    return (RegisterClass (&wndClass));
}
/* ------------------------------------------------------------------ */
/* Perform initialisation specific to this instance of the app. In    */
/* this case, creation and display of the main window.               */
/* ------------------------------------------------------------------ */
BOOL InitInstance (HINSTANCE hInstance, int nCmdShow)
{
    HWND  hWnd;

    // Create a main window for this instance of the application.
    hWnd = CreateWindow (WindowClassName,"Move & Copy Example",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    // Display the window and force it to be repainted.
    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);
    return TRUE;
}
/* ------------------------------------------------------------------ */
/* The main entry point for the application. This function starts    */
/* the application, then processes messages until it is closed.      */
/* ------------------------------------------------------------------ */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG       msg;
    if   (! FindWindow (WindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;
```

```
    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    // Loop, pumping messages.
    while (GetMessage (&msg, NULL, 0, 0))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
     }
    return msg.wParam;    // Return the value from PostQuitMessage.
}
```

注释

使用API函数CopyFile和MoveFile，可以为应用程序增加一些高级特性，实现一些类似于Windows资源管理器中的功能。API函数MoveFile不支持驱动器间移动目录，要实现此功能，需要递归遍历要移动的目录，使用函数CreateDirectory创建目的目录，接着使用函数MoveFile移动此目录中的每一个文件，当源目录为空时，使用API函数RemoveDirectory删除此目录。

第 3 章　应用程序和任务控制

对于初学 Windows 编程的程序员来说，他们所开发的应用程序将不再是系统上唯一运行的应用程序，多任务以及其他有关的概念，是初学者最需掌握的概念，也是他们最难掌握的概念。

以前开发的应用程序，除了有时使用 Windows 的剪贴板拷贝文本或使用 Print Manager 脱机打印文件外，很少考虑周围的运行环境，好象应用程序运行在单任务系统上似的。不过，现在的程序员已经越来越多地利用其运行环境来开发应用程序。例如：利用 Program Manager 来编写安装程序，利用 Windows 中的 Notepad 或 Write 在应用程序中读写文档。

在 Windows 95 中，有大量可以利用的控制、工具以及其他可编程的设备。例如：如果在应用程序中要压缩文件，可以启动一个 ZIP 程序；如果要查看动画信息，可以启动一个多媒体浏览器。所有这些功能在 Windows 操作环境下都是非常容易实现的，同时，有些 Windows 应用程序也有大量的功能可以用于应用程序的开发。所以可以认为 Windows 95 是一个由组件构成的或是面向对象的操作系统。

在利用其他的应用程序前，最好能清楚 Windows 是如何理解应用程序的。在 Windows 95 中，每个运行的应用程序都是一个任务，每个任务运行固定的时间片，然后交回控制给操作系统。因为时间分隔成多个时间片，轮流运行每个任务，所以这种模式称为分时。由于 Windows 清楚每个任务的运行情况，所以可以在任务间安全地进行切换。

在本章中，将介绍在应用程序中如何充分利用 Windows 提供的功能来解决实际问题。介绍如何在应用程序中启动另外的程序，如何在应用程序中终止另外的程序，以及如何找出正在运行的应用程序，还介绍了如何获得应用程序的信息：应用程序所在的目录，应用程序运行了几个实例，这些在本章中都将得到解答。

1. 如何找出系统上正在运行的任务

本节将介绍几个方法，分别用来查找 Windows 95 上正在显示的主窗口、正在运行的任务(可能有窗口显示也可能没有)以及已装入的 DLL。

使用这些方法，可以非常容易地实现一个任务管理器，用来替代 Windows for Workgroups 或 Windows 3.1 中的任务管理器，也可以实现一个任务控制条来替代 Windows 95 中的任务控制条。

2. 如何激活另一任务

有时为了实现某功能，需要从应用程序中启动另一应用程序，如果此应用程序已在运行，只需将其窗口置前，本节将介绍如何置前一个已运行的应用程序。确定需要启动的应用程序是否已在运行，可以利用 3.1 节中介绍的方法。

3. 如何关闭其他的应用程序

有时需要启动某一应用程序，当然有时也需要关闭某些正在运行的任务。可能是应用程序不能同某任务同时运行，也可能是因为这些任务是此应用程序启动的。本节将介绍如何实现此功能。

4. 如何找出应用程序的执行文件名

程序员经常需要确定执行文件所在的目录。例如：在执行文件所在的目录中存储应用程序的信息。如果将成百上千个应用程序的配置文件都存放在 Windows 的 SYSTEM 目录下，文件系统将会相当混乱。本节将介绍如何确定执行文件的目录。

5. 如何确保应用程序同时只能运行一个实例

在 Windows 3.1 中，为了限定应用程序同时只运行一个实例，对内存使用作了许多限制。在 Windows 95 中，大大减少了对内存使用的限制，但是有很充足的理由不使用这些改进。

如果应用程序需要互斥的存取一些数据库、文件或资源，则同时只运行应用程序的一个实例便是需要的。在本节中，将介绍如何确定应用程序是否已有实例在运行，如果实例已经在运行，则如何激活此实例，而不是创建新的实例。

6. 如何在后台运行其他任务

用户在前台继续工作，而后台可以运行一些任务也是非常有用的。本节将介绍如何在用户继续工作的情况下，启动一些应用程序并监控其运行过程。例如：可以在用户浏览信息时，在后台运行 UNZIP 程序进行安装。

7. 如何启动另外的程序并等待其运行结束

本节将介绍如何启动另一 Windows 应用程序并等待其结束。在 3.6 节中，运行另外的应用程序时用户可以继续工作，但在本节中将介绍如何强制用户等待，直到另外的应用程序结束。需要调用外部程序的任何系统有时会需要这些功能。

8. 如何从应用程序中终止和重启动 Windows

安装程序经常需要重新启动 Windows。例如安装了新的驱动程序，修改了 PATH 语句，或者应用程序需要某一启动程序。最好不要询问用户是否停止操作并重新启动 Windows，而是由应用程序自动来完成，这样不仅可以使用户界面友好，而且可以更安全，因为保证了运行应用程序前重新启动 Windows。

表 3-1 列出了在本章中使用的 Windows 95 API 函数。

表 3-1 在第 3 章中使用的 Windows 95 API 函数

Process32First	Process32Next	Module32First
Module32Next	EnumWindows	MakeProcInstance
FreeProcInstance	BringWindowToTop	GetWindow
GetWindowTextLength	GetWindowText	PostMessage
GetModuleFileName	FindWindow	GetLastActivePopup
ShowWindow	CreateThread	PeekMessage
TranslateMessage	DispatchMessage	PostQuitMessage
WinExec	CreateProcess	WaitForSingleObject
NotifyRegister	NotifyUnregister	ExitWindowsEx

3.1 找出系统上正在运行的任务

问题

有的程序员希望能够向用户列出当前正在运行的任务，但对任务和窗口的区别又不是很

确定。是否可以同时列出呢？还是需要确定显示哪一个呢？

方法

列出任务和窗口是相当有用的。任务是运行着的 Windows 应用程序，无论此应用程序是否显示窗口。一个任务也可以显示好几个窗口，主窗口即父窗口，其他的窗口为父窗口的子窗口，子窗口通常显示在父窗口内，但也可显示在父窗口外。无论是显示在父窗口内还是显示在父窗口外，所有的子窗口都同属于这一个任务。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Tasks，从下拉菜单中选择菜单项 View Tasks，弹出对话框，对话框含有一个列表框和四个按钮，如图 3-1 所示。选择按钮 Process，在列表框中将显示出所有当前正运行的任务的名字。

并非所有任务都显示窗口。选择第二个按钮 Windows，列表框中将显示出所有当前正运行的窗口的名字。选择按钮 Modules，列表框中将显示出所有已装入的模块。按钮 Close 用来关闭对话框。

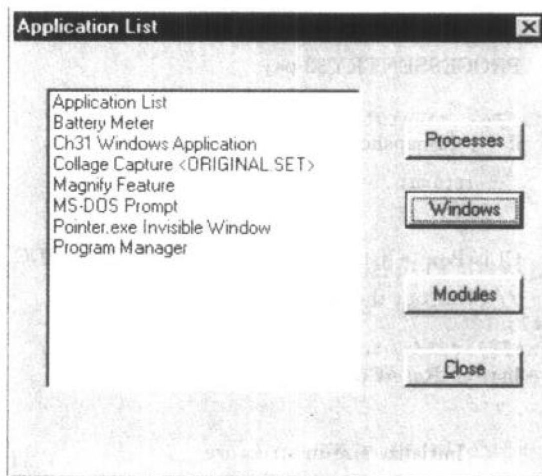


图 3-1 对话框 View Tasks

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新项目文件，并命名此项目文件为 CH31.MAK。

2. 进入 AppStudio 创建新的对话框。在对话框中，添加一个列表框和三个按钮，按钮的标题分别为 Processes、Windows 和 Modules，删除按钮 Cancel，改变按钮 Ok 的标题为 Colse。

3. 选择 ClassWizard，为此对话框创建对话框类，类名为 CTaskDialog。从对象列表中选择对象 CTaskDialog，从消息列表中选择消息 WM_INITDIALOG。点击按钮 Add Function，在方法 OnInitDialog 中添加下列代码：

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

      // Make the dialog window centered in our application

      CenterWindow();

      // Init toolhelp 32 functions

      if ( ! InitToolhelp32 () ) {
        MessageBox(" Unable to initialize toolhelp functions!", " Error", MB_OK );
```

```
        EndDialog(IDCANCEL);
        return FALSE;
    }

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

4. 从对象列表中选择对象 IDC _BUTTON1，从消息列表中选择消息 BN _CLICKED，命名方法为 OnProcessList，并在此方法中添加下列代码：

```
void CTaskDialog:: OnProcessList()
{
    // Get a snapshot of the thread li
    HANDLE hSnapshot=pCreateToolhelp32Snapshot( TH32CS _SNAPPROCESS, 0 );
    PROCESSENTRY32 pe;

    if ( ! hSnapshot )
        return;

    CListBox *list=(CListBox *)GetDlgItem(IDC _LIST1);
    // Clear out the list box

    list->ResetContent();

      // Initialize size in structure

    pe.dwSize=sizeof(pe);
    for ( int i=pProcess32First( hSnapshot, &pe ); i; i=pProcess32Next( hSnapshot, &pe ) )
     {

      HANDLE          hModuleSnap=NULL;
      MODULEENTRY32 me;

      // Take a snapshot of all modules in the specified process.
      hModuleSnap=pCreateToolhelp32Snapshot(TH32CS _SNAPMODULE,
                                                  pe.th32ProcessID );
      if (hModuleSnap==(HANDLE)-1)
            return;

      // Fill the size of the structure before using it.
      me.dwSize=sizeof(MODULEENTRY32);

      // Walk the module list of the process, and find the module of
```

```
    // interest. Then copy the information to the buffer pointed
    // to by lpMe32 so that it can be returned to the caller.
    if (pModule32First(hModuleSnap, &me)) {
          do {
                if (me.th32ModuleID==pe.th32ModuleID) {
                      list->AddString ( me.szExePath );
                      break;
                }
          }
                while (pModule32Next(hModuleSnap, &me));
          }
   }
   CloseHandle( hSnapshot );      // Done with this snapshot.   Free it

}
```

5. 从对象列表中选择对象 IDC _BUTTON2，从消息列表中选择消息 BN _CLICKED，命名方法为 OnWindowList，并在此方法中添加下列代码：

```
void CTaskDialog:: OnWindowList()
{
   // Get the list box from the dialog

   CListBox *list=(CListBox *)GetDlgItem(IDC_LIST1);

   // Clear out the list box

   list->ResetContent();

   // Make a callback procedure for Windows to use to iterate through the
   // Window list.

      FARPROC   EnumProcInstance   =   MakeProcInstance  (   ( FARPROC )  EnumWindowsProc,
AfxGetInstanceHandle() );

   // Call the EnumWindows function to start the iteration

   EnumWindows ( (WNDENUMPROC)EnumProcInstance, (LPARAM)list );

   // Don't forget to free up the allocated memory handle

   FreeProcInstance ( EnumProcInstance );

   // Make sure the dialog gets updated
```

```
    UpdateData();
}
```

6. 在方法 OnWindowList 的前面添加下列代码：

```
BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)
{
    // Get the list box

    CListBox *list=(CListBox *)lParam;

    // Get the window text to insert

    char buffer [256];
    GetWindowText(hwnd, buffer, 256);

    // Insert it into the list box

    if ( strlen(buffer) ) {
        list->AddString ( buffer );
    }

    return TRUE;
}
```

7. 从对象列表中选择对象 IDC_BUTTON3，从消息列表中选择消息 BN_CLICKED，命名方法为 OnModuleList，并在此方法中添加下列代码：

```
void CTaskDialog::OnModuleList()
{
    MODULEENTRY32 me;

    // Get the list box from the dialog

    CListBox *list=(CListBox *)GetDlgItem(IDC_LIST1);

    // Clear out the list box

    list->ResetContent();

    // Initialize the MODULEENTRY structure to 0 and set the
    // size of the structure in the dwSize element.
```

```
    memset ( &me, 0, sizeof(me) );
    me.dwSize=sizeof(MODULEENTRY32);
    HANDLE hSnapshot=pCreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );

    if ( ! hSnapshot )
        return;

    // Clear out the list box

    list->ResetContent();

    for ( int i=pModule32First( hSnapshot, &me ); i; i=pModule32Next( hSnapshot, &me ) )
    {
        list->AddString ( me.szExePath );

    }

    CloseHandle( hSnapshot );    // Done with this snapshot.  Free it

}
```

8. 在文件 TASKDIAL.CPP 的顶部添加下列代码：

```
// Type definitions for pointers to call tool help functions.
typedef BOOL (WINAPI *MODULEWALK)(HANDLE hSnapshot,
    LPMODULEENTRY32 lpme);
typedef BOOL (WINAPI *THREADWALK)(HANDLE hSnapshot,
    LPTHREADENTRY32 lpte);
typedef BOOL (WINAPI *PROCESSWALK)(HANDLE hSnapshot,
    LPPROCESSENTRY32 lppe);
typedef HANDLE (WINAPI *CREATESNAPSHOT)(DWORD dwFlags,
    DWORD th32ProcessID);
// File scope globals. These pointers are declared because of the need
// to dynamically link to the functions.  They are exported only by
// the Windows 95 kernel. Explicitly linking to them will make this
// application unloadable in Microsoft(R) Windows NT(TM) and will
// produce an ugly system dialog box.
static CREATESNAPSHOT pCreateToolhelp32Snapshot=NULL;
static MODULEWALK  pModule32First  =NULL;
static MODULEWALK  pModule32Next   =NULL;
static PROCESSWALK  pProcess32First  =NULL;
static PROCESSWALK  pProcess32Next   =NULL;
static THREADWALK  pThread32First  =NULL;
```

```
static THREADWALK   pThread32Next   =NULL;

// Function that initializes tool help functions.
BOOL InitToolhelp32 (void)
{
    BOOL    bRet   =FALSE;
    HINSTANCE hKernel=NULL;

    // Obtain the module handle of the kernel to retrieve addresses of
    // the tool helper functions.
    hKernel=GetModuleHandle(" KERNEL32.DLL");
    if (hKernel) {
        pCreateToolhelp32Snapshot=
            (CREATESNAPSHOT)GetProcAddress(hKernel,
            " CreateToolhelp32Snapshot");

        pModule32First   =(MODULEWALK)GetProcAddress(hKernel,
            " Module32First");
        pModule32Next    =(MODULEWALK)GetProcAddress(hKernel,
            " Module32Next");

        pProcess32First=(PROCESSWALK)GetProcAddress(hKernel,
            " Process32First");
        pProcess32Next   =(PROCESSWALK)GetProcAddress(hKernel,
            " Process32Next");

        pThread32First   =(THREADWALK)GetProcAddress(hKernel,
            " Thread32First");
        pThread32Next    =(THREADWALK)GetProcAddress(hKernel,
            " Thread32Next");
        // All addresses must be non-NULL to be successful.
        // If one of these addresses is NULL, one of
        // the needed lists cannot be walked.
        bRet=   pModule32First && pModule32Next   && pProcess32First &&
                pProcess32Next && pThread32First && pThread32Next &&
                pCreateToolhelp32Snapshot;
    }
    else
        bRet=FALSE; // could not even get the module handle of kernel

    return bRet;
}
```

9. 最后，在文件 TASKDIAL.CPP 中添加下列 include 文件行：

```
#include <tlhelp32.h>
```

10. 返回 AppStudio，在主菜单中添加新的菜单 Tasks，在菜单 Tasks 中添加新的菜单项 View Tasks，ID 为 ID_VIEW_TASKS。

11. 进入 ClassWizard，从下拉列表中选择对象 CMainFrame，从对象列表中选择对象 ID_VIEW_TASKS，从消息列表中选择消息 COMMAND，点击按钮 Add Function，方法命名为 OnViewTasks。

12. 在 CMainFrame 的成员函数 OnViewTasks 中输入下列代码：

```
void CMainFrame::OnViewTasks()
{
  CTaskDialog dlg;
  dlg.DoModal();
}
```

13. 在文件 MAINFRM.CPP 的顶部，#include "mainfrm.h"行的后面输入下列行：

```
#include "taskdial.h"
```

14. 编译并运行此例子程序。

用法

本节介绍了三个不同的 Windows API 函数集，用来列出当前正在运行的任务、正在运行的窗口和已装入的 DLL。

Windows 95 API 函数 Process32First 和 Process32Next，用来列出当前正在运行的进程。要使用这些函数，必须从 kernel32.DLL 中装入这些函数，从而完成 ToolHelp 功能的初始化，例子程序的第 8 步，函数 InitToolhelp32 即用来完成 ToolHelp 功能的初始化。这些 API 函数将遍历进程链表，返回每一个进程的进程标识符，使用这些进程标识符，调用模块函数(Module32First 和 Module32Next)遍历模块链表，从而找出相应的进程名。

Windows API 函数 MakeProcInstance 和 EnumWindows，用来列出当前正在运行的窗口。在例子程序中，函数 MakeProcInstance 接受函数 EnumWindowsProc 的地址，返回 FARPROC 类型的值，接着使用函数 EnumWindows 将此返回值传送给 Windows 95，从而列出所有的主窗口。函数 EnumWindowsProc 具有两个参数，第一个参数为窗口句柄，第二个参数为用户定义参数，由函数 EnumWindows 传送。在本节的例子程序中，此参数为 CListBox 的实例，用来存储数据。在函数 EnumWindowsProc(因为被 Windows 95 调用，所以也称为回调函数)中，窗口的标题显示在列表框中，以便于用户查看。

3.2 激活另一任务

问题

有的程序员希望能够列出当前正在运行的任务，并允许用户任选一个来激活。只允许用户激活具有可见窗口的任务，这是因为用户只能使用这样的应用程序。如何利用 Windows API 函数来实现呢？

方法

在前一节中已经实现了本节任务的一半，列出窗口。在本节中，将介绍如何激活其他应用程序的窗口，即显示在前。

本节的实现方法类似于前一节。列出所有的窗口，以便于用户选择要激活的任务，不同之处在于列出窗口后，用户要任选一个任务，然后使用 Windows API 函数 BringWindowsToTop 激活此窗口。此函数将使此窗口显示在前，并接收所有键盘和鼠标的输入。

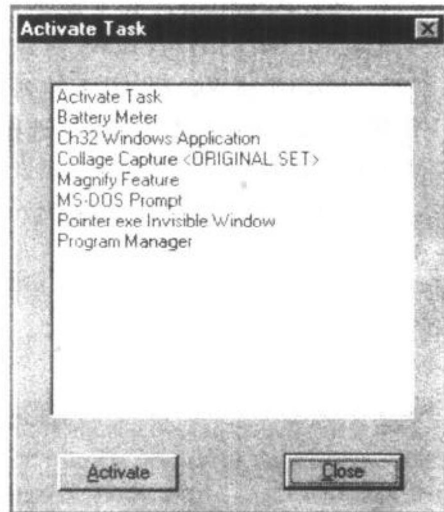


图 3-2 对话框 Activate Task

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Tasks，从下拉菜单中选择菜单项 Activate Task，将弹出一个对话框，显示系统上正在运行的所有窗口，如图 3-2 所示。选择一个窗口并点击按钮 Activate，对话框将关闭，选中的窗口被置前。

实现例子程序的具体步骤如下：

1. 利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH32.MAK。进入 AppStudio 创建新的对话框。

2. 在对话框中，添加一个列表框，删除按钮 Cancel，重新命名按钮 Ok 为 Close，对话框的标题为 Activate Task。

3. 在对话框中添加新的按钮 Activate。

4. 启动 ClassWizard。为刚创建的对话框创建对话框类，类名为 CTaskActivateDlg。从对象列表中选择 CTaskActivateDlg，从消息列表中选择消息 WM_INITDIALOG。点击按钮 Add Function，在方法 OnInitDialog 中添加下列代码：

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

    // Get the list box from the dialog

    CListBox *list=(CListBox *)GetDlgItem(IDC_LIST1);

    // Clear out the list box

    list->ResetContent();
```

```
    // Make a callback procedure for Windows to use to iterate through the
    // Window list.

      FARPROC  EnumProcInstance  =  MakeProcInstance  (  ( FARPROC )  EnumWindowsProc,
AfxGetInstanceHandle() );

    // Call the EnumWindows function to start the iteration

    EnumWindows ( (WNDENUMPROC)EnumProcInstance, (LPARAM)list );

    // Don't forget to free up the allocated memory handle

    FreeProcInstance ( EnumProcInstance );

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

5. 在文件 TASKACTI.CPP 中，方法 OnInitDialog 的前面，添加下列代码：

```
static BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)
{
   // Get the list box

   CListBox *list=(CListBox *)lParam;

   // Get the window text to insert

   char buffer [256];
   GetWindowText(hwnd, buffer, 256);

   // Insert it into the list box

   if ( strlen(buffer) ) {
       int idx=list->AddString ( buffer );
       list->SetItemData ( idx, (DWORD)hwnd );
   }

   return TRUE;
}
```

6. 从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED。点击按钮 Add Function，命名方法为 OnActivateTask，在方法中添加下列代码：

```
void CTaskActivateDlg:: OnActivateTask()
{
```

```
    // Get the list box from the dialog

    CListBox *list=(CListBox *)GetDlgItem(IDC_LIST1);

    // Get the selected window number

    int idx  =list->GetCurSel();

    // See if there was a selection

    if ( idx==LB_ERR ) {
        MessageBox(" You must select a window to activate!", " Error",
                   MB_APPLMODAL | MB_OK );
        return;
    }

    // Get the handle of the window to activate

    HWND hWnd=(HWND)list->GetItemData ( idx );

    ::BringWindowToTop (hWnd);

    // Close the dialog

    EndDialog ( IDOK );
}
```

7. 在菜单 Tasks 中添加新的菜单项 Activate Task，标识符为 ID_ACTIVATE_TASK。

8. 在 ClassWizard 中，从下列列表中选择对象 CMainFrame，选择对象 ID_ACTIVATE_TASK，选择消息 COMMAND。点击按钮 Add Function，在方法 OnActivateTask 中添加下列代码：

```
void CMainFrame::OnActivateTask()
{
    CTaskActivateDlg dlg;
    dlg.DoModal();
}
```

9. 在文件 MAINFRM.CPP 的顶部，#include "mainfrm.h"行的下面，添加下列行：

```
#include  " taskacti.h"
```

10. 编译并运行此例子程序。

用法

类似于前一节，本节同样使用 API 函数 EnumWindows 及回调函数 EnumWindowsProc 来列出窗口，但是，传送给函数 EnumWindowsProc 的窗口句柄同窗口的标题一齐被存储，当用户选择窗口后，将取回窗口句柄。

实际过程如下：用户从列表框中选择窗口标题并点击按钮 Activate Task 后，CTaskActivateDlg 的方法 OnActivateTask 就被调用，用来取回存储的窗口句柄，此句柄接着传送给 Windows 95 API 函数 BringWindowToTop(因为所有 Cwnd 派生的对象都有方法 BringWindowToTop，所以范围操作符::是需要的)，此函数将使此窗口置前。

注：函数 SetWindowPos 也能将窗口置于其他窗口的前面。在本书的后面章节中将讨论此函数。

注释

使用 Visual Basic 也能实现同样的功能。Visual Basic 不能创建真正的 Windows 回调函数，所以只有用另外的方法来创建窗口列表。步骤如下：

1. 创建新的项目文件，或者在已有的项目文件中添加新的表单。在表单中，添加列表框，命名为 list1，添加两个按钮，标题分别为 Activate 和 Close。

2. 在新表单的方法 FormLoad 中添加下列代码：

```
Sub Form_Load
CurrWnd=GetWindow(Form1.hWnd, GW_HWNDFIRST)
        While CurrWnd <> 0
               Length=GetWindowTextLength(CurrWnd)
               Item$ =Space$(Length + 1)
               Length=GetWindowText(CurrWnd, Item$, Length + 1)
               If  Length > 0 Then
                   List1.AddItem  Item$
               End If
               CurrWnd=GetWindow(CurrWnd, GW_HWNDNEXT)
                 x=DoEvents()
          Wend
End Sub
```

3. 在表单的“general”部分添加下列代码：

```
DefInt  A-Z
Const  GW_HWNDFIRST=0
Const  GW_HWNDNEXT=2
DefInt  A-Z
Const  GW_HWNDFIRST=0
Const  GW_HWNDNEXT=2
Private Declare Function GetWindow Lib "user32" (ByVal  hwnd%, ByVal cmd%) As Integer
Private Declare Function GetWindowText Lib " user32" Alias
" GetWindowTextA"   (ByVal hwnd As Long, ByVal lpString As String, ByVal  cch As Long) As Long
```

```
Private Declare Function GetWindowTextLength Lib " user32" Alias
" GetWindowTextLengthA" (ByVal hwnd As Long) As Long
```

4. 在按钮 Activate 的事件处理程序中添加下列代码：

```
Sub Command1 _Click
  If List1. ListIndex <> -1 Then
        Item $ =List1. List(List1. ListIndex)
        On Local Error Resume Next
        AppActivate   Item $
  Else
        MsgBox  " No item selected to activate"
   End If
End Sub
```

5. 最后，在按钮 Close 的事件处理程序中添加下列代码：

```
Sub Command2 _Click()
     End
End Sub
```

6. 运行此例子程序。显示的表单如图 3-3 所示。

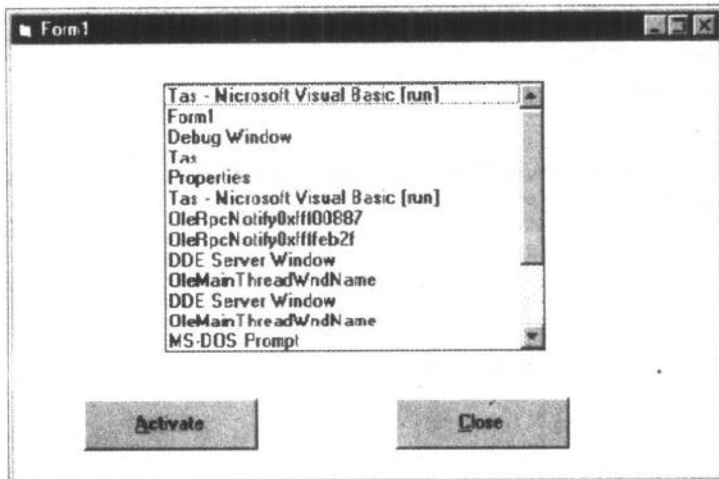


图 3-3　Visual Basic 表单 Activate Task

3.3　关闭其他的应用程序

问题

有的程序员希望能够使用户从自己的应用程序中启动其他的应用程序，并在操作完成后关闭此应用程序。例如：从应用程序中启动 Windows 应用程序 Write，作记录或写备忘录，然

后关闭此应用程序。有时，还希望能够忽略应用程序 Write 的消息框，消息框是用来询问用户是否保存文件的改变。

如何使用 Windows 95 API 函数尽量简单地实现这两个功能呢?

方法

在 3.2 节中，介绍了如何列出当前正在运行的任务以及如何激活选中的任务。本节要实现的功能类似，但在应用程序中要激活和要关闭的任务一般是确定的。不过为了使本节的例子程序具有普遍性，仍然由用户来选择要关闭的应用程序，并决定使用的关闭方法。

为了实现这些功能，需要使用 Windows API 函数 PostMessage。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Tasks，从下拉菜单中选择菜单项 Terminate Task，将弹出一个对话框，显示当前运行的窗口，如图 3-4 所示。选择窗口，点击按钮 Close App 或按钮 Quit App，对话框将关闭，选中的窗口也将关闭。

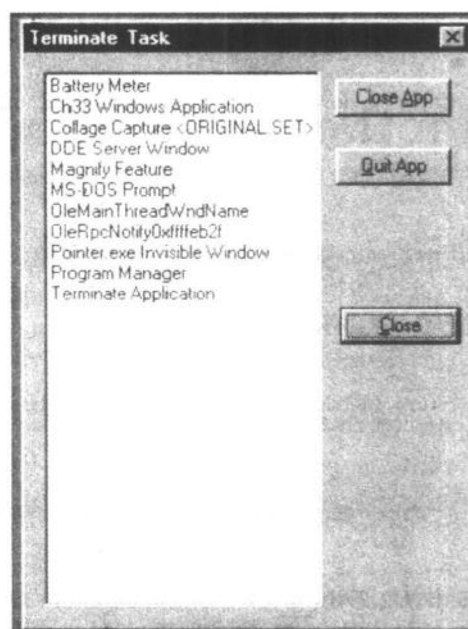


图 3-4　对话框 Terminate Task

启动 Windows 应用程序 WordPad，并键入一些文本，测试两个按钮的功能，会发现例子程序有时询问是否存储已改变的文本，而有时不询问。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH33.MAK。进入 AppStudio 创建新的对话框。

2. 在对话框中，添加列表框，删除按钮 Cancel，重新命名按钮 Ok 为 Close，对话框的标题设置为 Terminate Task。

3. 在对话框中添加两个按钮，标题分别为 Close App 和 Quit App。

4. 启动 ClassWizard。为刚创建的对话框创建对话框类，类名为 CTerminateTaskDlg。从对象列表中选择 CTerminateTaskDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function，在方法 OnInitDialog 中添加下列代码：

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

    // Get the list box from the dialog

    CListBox *list=(CListBox *)GetDlgItem
    (IDC_LIST1);
```

```
    // Clear out the list box

    list->ResetContent();

    // Make a callback procedure for Windows to use to iterate through the
    // Window list.

        FARPROC EnumProcInstance = MakeProcInstance ( ( FARPROC ) EnumWindowsProc,
AfxGetInstanceHandle());

    // Call the EnumWindows function to start the iteration

    EnumWindows ( (WNDENUMPROC)EnumProcInstance, (LPARAM)list );

    // Don't forget to free up the allocated memory handle

    FreeProcInstance ( EnumProcInstance );

    CenterWindow();

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

5. 在文件 TASKACT1.CPP 中，方法 OnInitDialog 的前面，添加下列代码：

```
static BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)
{
   // Get the list box

   CListBox *list=(CListBox *)lParam;

   // Get the window text to insert

   char buffer [256];
   GetWindowText(hwnd, buffer, 256);

   // Insert it into the list box

   if ( strlen(buffer) ) {
      int idx=list->AddString ( buffer );
      list->SetItemData ( idx, (DWORD)hwnd );
   }
```

```
  return TRUE;
}
```

6. 从对象列表中选择对象 IDC _BUTTON1，从消息列表中选择消息 BN _CLICKED，点击按钮 Add Function，方法名为 OnCloseApp，并在此方法中添加下列代码：

```
void CTerminateTaskDlg:: OnCloseApp()
{
  // Get the list box from the dialog

  CListBox * list=(CListBox * )GetDlgItem(IDC _LIST1);

  // Get the selected window number

  int idx   =list->GetCurSel();

  // See if there was a selection

  if ( idx= =LB _ERR ) {
     MessageBox(" You must select a window to close!", " Error",
               MB _APPLMODAL | MB _OK );
     return;
  }

  // Get the handle of the window to close

  HWND hWnd=(HWND)list->GetItemData ( idx );

  :: PostMessage( hWnd, WM _CLOSE, 0, 0L );

}
```

7. 从对象列表中选择对象 IDC _BUTTON2，从消息列表中选择消息 BN _CLICKED，点击按钮 Add Function，方法命名为 OnQuitApp，并在此方法中添加下列代码：

```
void CTerminateTaskDlg:: OnQuitApp()
{
  // Get the list box from the dialog

  CListBox * list=(CListBox * )GetDlgItem(IDC _LIST1);

  // Get the selected window number
```

```
  int idx   =list->GetCurSel();

  // See if there was a selection

  if ( idx==LB_ERR ) {
     MessageBox(" You must select a window to quit!", " Error",
               MB_APPLMODAL | MB_OK );
     return;
  }

  // Get the handle of the window to quit
  HWND hWnd=(HWND)list->GetItemData ( idx );

  :: PostMessage( hWnd, WM_QUIT, 0, 0L );

}
```

8. 在菜单 Tasks 中添加新的菜单项 Terminate Task，并设置菜单项的标识符为 ID_TERMINATE_TASK。

9. 在 ClassWizard 中，从下拉列表中选择对象 CMainFrame，选择对象 ID_TERMINATE_TASK 和消息 COMMAND，点击按钮 Add Function，在方法 OnTerminateTask 中添加下列代码：

```
void CMainFrame:: OnTerminateTask()
{
CTerminateTaskDlg dlg;
   dlg.DoModal();
}
```

10. 在文件 MAINFRM.CPP 的顶部，#include "mainfrm.h"行的下面，添加下列行：

```
#include  " terminat.h"
```

11. 编译并运行此例子程序。

用法

同前一节一样，本节也使用了 API 函数 EnumWindows 及回调函数 EnumWindowsProc 来列出窗口，同样，也将 Windows 句柄存放在列表框附加数据部分。

当用户从对话框中选择了按钮 Close App 后，选中的列表项及其句柄将被取回。使用此句柄传送标识符为 WM_CLOSE 的消息给窗口，表示窗口应被关闭。使用消息 WM_CLOSE 关闭窗口，将不会提示用户保存已改变的文本。

当用户从对话框中选择了按钮 Quit App 后，选中的列表项及其句柄将被取回。使用此句

柄传送标识符为 WM _ QUIT 的消息给窗口，表示窗口应被关闭。使用消息 WM _ QUIT 关闭窗口，将会提示用户保存已改变的文本，并允许用户撤消关闭窗口的请求。

3.4 找出应用程序的执行文件名

问题

同一个应用程序由不同的用户使用时，应用程序的名字有可能不同。有时需要找出应用程序实际使用的名字，以便于找到其所在的目录，也可根据实际名字显示版本信息，如何实现呢？

方法

许多 Windows 95 API 函数是专门为应用程序开发者设计的。本节所要讨论的问题，实际上是 Windows 程序开发者常遇见的问题，Windows 95 开发组在开发 API 时也考虑到了此问题。

Windows API 函数 GetModuleFileName 返回执行文件名，执行文件由应用程序的实例句柄确定。在本节的例子程序中，只显示当前正在运行的应用程序的文件名，而不考虑运行在系统上的其他应用程序。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Applications，从下拉菜单中选择菜单项 Executable File Name，将弹出对话框，如图 3-5 所示，显示当前正在运行的可执行文件的全路径名。

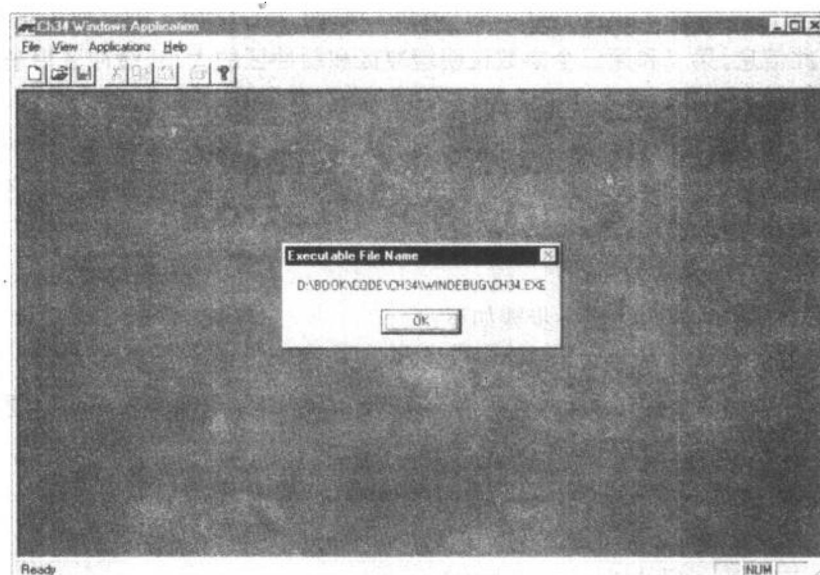


图 3-5　显示的消息框 Executable File Name

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH34.MAK。

2. 在 AppStudio 的主菜单中添加新的菜单 Applications，在菜单 Applications 中添加新的菜单项 Executable File Name，标识符为 ID_EXE_FILE_NAME。

3. 进入 ClassWizard。从下拉列表中选择类 CMainFrame，从对象列表中选择 ID_EXE_FILE_NAME，从消息列表中选择消息 COMMAND，点击按钮 Add Function，并命名方法为 OnExeFileName。

4. 在 CMainFrame 的方法 OnExeFileName 中添加下列代码：

```
void CMainFrame::OnExeFileName()
{
  char fileName [_MAX_PATH];

  GetModuleFileName ( AfxGetInstanceHandle(), fileName, _MAX_PATH );
  MessageBox ( fileName, " Executable File Name", MB_APPLMODAL | MB_OK);
}
```

5. 编译并运行此例子程序。

用法

在 Visual C++开发的应用程序中，用户选择菜单项 Exectable File Name，在 Delphi 开发的应用程序中，用户点击按钮 GetName，应用程序便调用 Windows API 函数 GetModuleFileName。此函数有三个参数，第一个参数是执行程序的实例句柄，在 Visual C++中，应用程序的实例句柄由函数 AfxGetInstanceHandle()取回，在 Delphi 中，使用全局变量 Hinstance 来保存此信息。第二和第三个参数说明缓冲区和缓冲区的大小，缓冲区用来存放取回的文件名，缓冲区的大小用字节数表示。

一旦函数返回，消息框将弹出，显示缓冲区中的内容。应该注意，当 Windows 95 应用程序运行在长文件名系统上时，可能会返回长文件名，因此使用缓冲区存放文件名前应检查文件系统中是否有长文件名。

注释

使用 Delphi 完成同样的功能，步骤如下：

1. 启动 Delphi 创建新的表单，或在已有的应用程序中添加新的表单。

2. 在表单中添加两个按钮，标题分别为 Get Name 和 Close。改变表单的标题为 Get Executable File Name，并命名按钮 Get Name 为 GetName。

3. 双击按钮 Get Name，在方法 TForm1.GetNameClick 中添加下列代码：

```
procedure TForm1.GetNameClick(Sender: Tobject);
var
   fileName  : PChar;
   ret       : Integer;
begin
   fileName := StrAlloc(251);
```

```
    GetModuleFileName(HInstance, fileName, 250);
    ret := Application.MessageBox(fileName, 'Executable File Name', mb_ok);
    StrDispose(fileName);
end;
```

4. 双击按钮 Close，在方法 TForm1.Button1Click 中添加下列代码：

```
procedure  TForm1.Button1Click(Sender: TObject);
begin
    Close;
end;
```

5. 编译并运行此例子程序，点击按钮 Get Name，显示的对话框如图 3-6 所示。

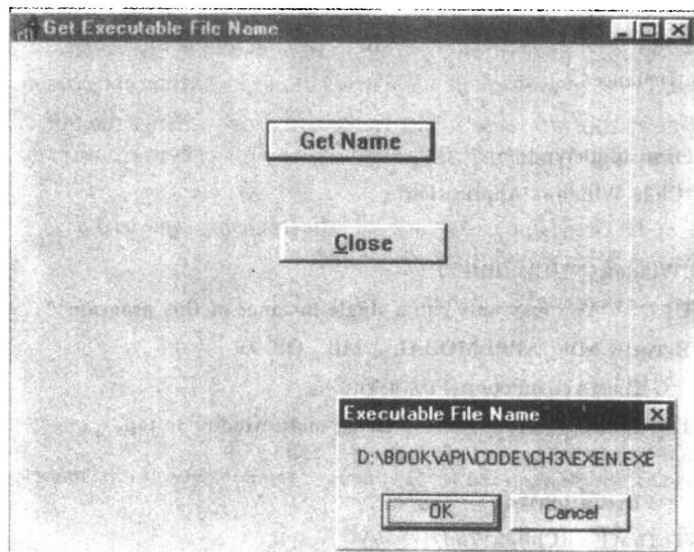


图 3-6 Delphi 中显示的 Executable File Name

3.5 确保应用程序同时只能运行一个实例

问题

有的程序员希望限制用户启动应用程序的多个实例。在 Windows 3.x 中，可以简单在函数 WinMain 中设置数据域 hPrevInstance，但在 Windows 95 中，已经不支持此方法。那么如何实现此功能呢？

方法

在 Windows 95 中已不再使用变量 hPrevInstance，但为了保持同 Windows 3.x 的应用程序兼容，函数 WinMain 的参数 hPrevInstance 仍然存在。只是在 Windows 95 中，此参数总为 NULL。

本节介绍一个“标准”的方法，用来确定应用程序是否已有实例在运行。本节的例子程

序在第二次被启动时，将激活已运行的实例。在 Windows 95 中往往会搞不清楚应用程序是否已启动，所以本节介绍的方法尤其有用。

步骤

按照下列步骤实现一个例子程序。运行此例子程序并将其极小化，然后重新运行此例子程序，例子程序的第二个实例开始运行，弹出一个消息框，提示用户不能同时运行例子程序的多个实例(如图 3-7)，当关闭此消息框后，例子程序的前一实例显示出来并恢复原来的大小。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH35.MAK。

2. 在文件 CH35.CPP 中找到方法 InitInstance，对 CCh35App 的方法 InitInstance 做下列修改，添加的代码用黑体表示。

```
BOOL CCh35App::InitInstance()
{
  HWND FirsthWnd, FirstChildhWnd;
  static char *title="Ch35 Windows Application";

  if (FirsthWnd=FindWindow(NULL, title)) {
      MessageBox ( NULL, " You may only run a single instance of this program!",
                  " Error", MB_APPLMODAL | MB_OK );
      FirstChildhWnd=GetLastActivePopup(FirsthWnd);
      BringWindowToTop(FirsthWnd);          // Bring main window to top.

      if (FirsthWnd != FirstChildhWnd)
         BringWindowToTop(FirstChildhWnd);

      // Don't forget to restore the state to normal

      ShowWindow(FirsthWnd, SW_SHOWNORMAL);

      return (FALSE);                       // Do not run second instance.
  }

     // Standard initialization
     // If you are not using these features and wish to reduce the size
     //  of your final executable, you should remove from the following
     //  the specific initialization routines you do not need.

     SetDialogBkColor();          // Set dialog background color to gray
```

```
    LoadStdProfileSettings();    // Load standard INI file options (including MRU)
    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.
    CMultiDocTemplate * pDocTemplate;
    pDocTemplate=new CMultiDocTemplate(
        IDR_CH35TYPE,
        RUNTIME_CLASS(CCh35Doc),
        RUNTIME_CLASS(CMDIChildWnd),          // standard MDI child frame
        RUNTIME_CLASS(CCh35View));
    AddDocTemplate(pDocTemplate);

    // create main MDI Frame window
    CMainFrame * pMainFrame=new CMainFrame;
    if (! pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd=pMainFrame;

    if (m_lpCmdLine [0] ! ='\0')
     {
        // TODO: add command line processing here
    }

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}
```

3. 编译并运行此例子程序。无论是否启动此例子程序的第二个实例，都将弹出一个对话框。

用法

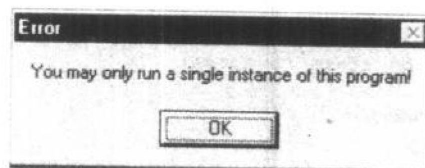


图 3-7 第二个实例对话框

当用户启动例子程序的第二个实例时，MFC(Microsoft Foundation Classes)为例子程序 CCh35App 创建一个实例，接着调用例子程序的处理程序 InitInstance 进行初始化实例。

处理程序 InitInstance 调用 API 函数 FindWindow 来查找具有同样标题的已运行的应用程序，标题存放在应用程序的资源文件中。如果找到一个标题匹配的窗口，例子程序将显示消息框，提示用户此例子程序同时只允许一个实例运行。窗口句柄用来查找应用程序的所有弹出窗口，并将所有的窗口置前，通过调用函数 ShowWindow 激活窗口并恢复到原来的大小。最后，函数返回 FALSE 给 MFC，表示不应该再创建此实例。

最后应该注意，Windows 95 API 有 32 位的函数 FindWindowEx，此函数的功能同函数 FindWindow 基本一致，但有另外两个参数：主窗口用来说明查找的范围(NULL 表示整个桌面)，子窗口用来说明需要查找的子窗口(NULL 表示所有的子窗口)。

3.6 在后台运行其他任务

问题

有的程序员希望能够在 Windows 的后台运行一些任务，以便用户可以继续工作。需要在后台运行的任务包括：打印、文件拷贝以及在窗口中显示动画。在保持 Windows 继续处理用户请求的情况下，如何实现这些功能呢？

方法

多任务操作系统都能在后台运行任务，当然 Windows 也不例外。实现的方法有两个：第一个方法是使用具有优先权的消息循环，Windows 3.x、Windows 95 及 Windows NT 都支持此方法，本节也主要介绍此方法。具有优先权的循环是接收消息的简单方法，消息来自操作系统，传送给相应的窗口，实现此功能不是使用应用程序的主循环，而是在应用程序中创建消息循环来传送消息。第二个方法在本节中只是简要的提一下，只能用于 Windows 95 和 Windows NT 中，此方法利用 32 位 API 函数 CreateThread 来创建一个新的线程，在本节的注释部分，将讨论线程是如何工作的以及如何利用线程来实现真正的后台处理。

本节中用到的重要函数有 PeekMessage，TranslateMessage，DispatchMessage 以及 PostQuitMessage。这些函数构成了 Windows 95 消息系统的核心，也是后台处理模块的核心，应基于 Windows 的消息系统来开发后台处理模块，从而实现对应用程序的控制。应该注意保持 Windows 系统的多个任务的平滑运行。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，点击工具条上的图标(左边第一个)或选择菜单 File|New menu，在例子程序的主窗口中将出现一个空白的窗口，如图 3-8 所示。

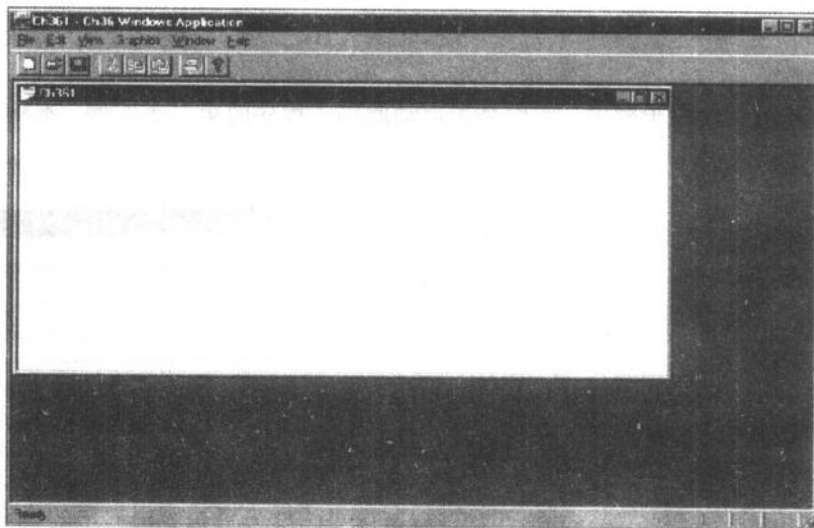


图 3-8 显示的空白新窗口

接着选择菜单 Graphics 并在下拉菜单中点击菜单项 Start Graphics，空白窗口中将随机地显示出一些彩色四边形，这时用户可以切换应用程序、浏览帮助信息或完成其他的一些操作，而绘制操作在同时进行，从菜单 Graphics 中选择菜单项 Stop Graphics，例子程序将停止四边形的绘制，如图 3-9 所示。

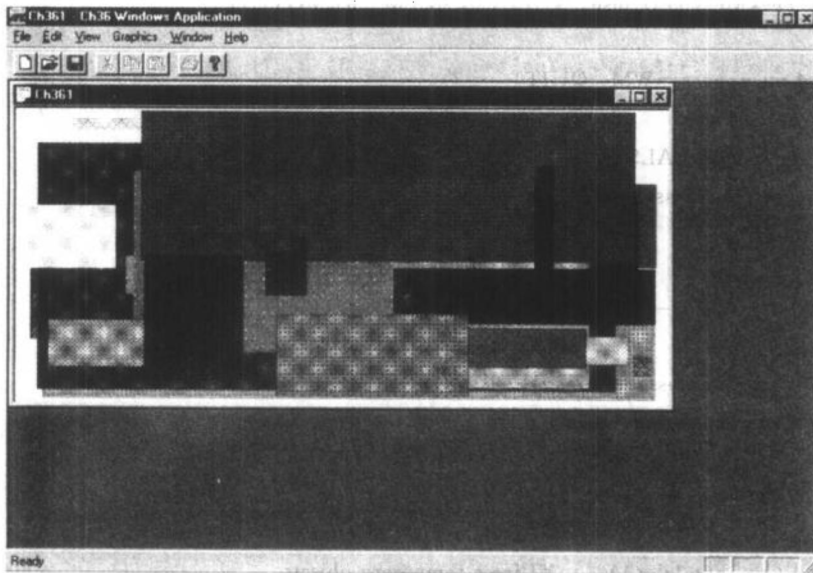


图 3-9　中止后台运行后显示的有图窗口

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH36.MAK。进入 ClassWizard，点击按钮 Add Class，类名为 CGfxView，基类为 CView，其他的取缺省值。

2. 选择源文件 CFXVIEW.CPP，在类的构造方法和析构方法中添加下列代码：

```
CGfxView::CGfxView()
{
   bProcessing=0;
}

CGfxView::~CGfxView()
{
   bProcessing=0;
}
```

3. 在源文件 GFXVIEW.CPP 中添加下列代码：

```
void CGfxView::DoIdleProcess()
{
```

```
// Loop until we are finished or the user tells us to
MSG msg;

while (bProcessing)
{
    while (::PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
      if (msg.message==WM_QUIT)
      {
          bProcessing=FALSE;
          ::PostQuitMessage(0);
          break;
      }
      if (!AfxGetApp()->PreTranslateMessage(&msg))
      {
          ::TranslateMessage(&msg);
          ::DispatchMessage(&msg);
      }
    }
    AfxGetApp()->OnIdle(0);    // updates user interface
    AfxGetApp()->OnIdle(1);    // frees temporary objects

    CDC *dc=GetDC();
    CPen pen;

    int red=rand() % 256;
    int green=rand() % 256;
    int blue=rand() % 256;

    pen.CreatePen ( PS_SOLID, 1, RGB(red, green, blue) );
    CBrush brush;

    brush.CreateSolidBrush ( RGB(red, green, blue) );

    dc->SelectObject ( &pen );
    dc->SelectObject ( &brush );

    CRect r;
    GetClientRect(&r);

    int height=r.bottom - r.top + 1;
    int width  =r.right - r.left + 1;
```

```
        int st_x=(rand() % width) + 1;
        int st_y=(rand() % height) + 1;
        int end_x=(rand() % width) + 1;
        int end_y=(rand() % height) + 1;

        if ( st_x > end_x ) {
            int temp=st_x;
            st_x=end_x;
            end_x=temp;
        }
        if ( st_y > end_y ) {
            int temp=st_y;
            st_y=end_y;
            end_y=temp;
        }

        dc->Rectangle ( st_x, st_y, end_x, end_y );

        ReleaseDC(dc);
    }
}
```

4. 接着，在头文件 GFXVIEW.H 中修改类的定义如下（修改的代码用黑体表示）：

```
// gfxview.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CGfxView view

class CGfxView : public CView
{
private:
    BOOL bProcessing;
        DECLARE_DYNCREATE(CGfxView)
protected:
        CGfxView();                 // protected constructor used by dynamic creation
    void DoIdleProcess();

// Attributes
public:

// Operations
public:
```

```
// Implementation
protected:
        virtual ~CGfxView();
        virtual     void OnDraw(CDC * pDC);        // overridden to draw this view
        // Generated message map functions
protected:
        //{{AFX_MSG(CGfxView)
        afx_msg void OnGfxStart();
        afx_msg void OnGfxStop();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
//////////////////////////////////////////////////////////////////////////////
```

5. 进入 AppStudio。在主菜单中添加新的菜单 Graphics，添加两个新的菜单项 Start Graphics 和 Stop Graphics，其标识符分别为 ID_GFX_START 和 ID_GFX_STOP。

6. 进入 ClassWizard，从下拉列表中选择类 CGfxView，从对象列表中选择对象 ID_GFX_START，从消息列表中选择消息 COMMAND。新方法命名为 OnGfxStart 并在此方法中输入下列代码：

```
void CGfxView::OnGfxStart()
{
   bProcessing=1;
   DoIdleProcess();
}
```

7. 进入 ClassWizard，从下拉列表中选择类 CGfxView，从对象列表中选择对象 ID_GFX_STOP，从消息列表中选择消息 COMMAND。新方法命名为 OnGfxStop 并在此方法中输入下列代码：

```
void CGfxView::OnGfxStop()
{
   bProcessing=0;
}
```

8. 在源文件 CH36.CPP 中，对 CCh36App 的方法 InitInstance 做下列修改（修改的代码用黑体表示）：

```
BOOL CCh36App::InitInstance()
{
        // Standard initialization
```

```
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

    SetDialogBkColor();          // Set dialog background color to gray
    LoadStdProfileSettings();    // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    // serve as the connection between documents, frame windows and views.

    CMultiDocTemplate* pDocTemplate;
    pDocTemplate=new CMultiDocTemplate(
        IDR_CH36TYPE,
        RUNTIME_CLASS(CCh36Doc),
        RUNTIME_CLASS(CMDIChildWnd),        // standard MDI child frame
        RUNTIME_CLASS(CGfxView));
    AddDocTemplate(pDocTemplate);

    // create main MDI Frame window
    CMainFrame* pMainFrame=new CMainFrame;
    if (! pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd=pMainFrame;

    // create a new (empty) document
    OnFileNew();

    if (m_lpCmdLine [0] ! ='\0')
    {
        // TODO: add command line processing here
    }

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}
```

9. **编译并运行此例子程序。**

用法

当用户打开一个新的视图时，例子程序便创建一个 GfxView，接着便等待消息。点击菜单项 Start Graphics，发送消息 ID_START_GFX，视图接收到消息后，便开始无限循环地

显示随机的四边形并不断检查消息队列。当用户选择了菜单项 Stop Graphics 后，发送消息 ID _ STOP _ GFX 给视图，此消息将设置全局标志并终止无限循环。

注释

在本节的开始已经提到，Windows NT 和 Windows 95 允许创建线程来执行任务。下列程序段将说明如何创建线程来执行任务：

```
DWORD  ThreadFunc(LPDWORD  param)
{
    MessageBox(NULL, " Got an argument of  %ld\n", * param);
    return 0;
}
// Inside of your function...
DWORD  id, param=12345;
HANDLE  hThread=CreateThread(NULL, 0,
(LPTHREAD _ START _ ROUTINE)ThreadFunc, &param, 0, &id );
//More code below.
```

3.7 启动另外的程序并等待其运行结束

问题

有的程序员希望能够在自己的应用程序中启动另外的 Windows 应用程序并在执行其他操作前等待其结束。如何启动另外的应用程序并确定其结束呢?

方法

在 Windows 中，从一个应用程序中启动另外一个应用程序是相当常见的。API 函数 WinExec 即可实现此功能，在 Visual Basic 中可以使用函数 ShellExecute，但是要确定另外的应用程序是否已经停止运行，则需要使用本章已经介绍过的方法以及新函数 WinExec，本节将主要介绍此方法。另外的实现方法，如利用 TOOLHELP.DLL 中的功能，将在注释部分中讨论。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Application，从下拉菜单中选择菜单项 Terminate and Wait，将启动 Windows 应用程序 Notepad，如图 3-10 所示。结束应用程序 Notepad 将重新显示 CH37 应用程序，并在应用程序的前面弹出一个消息框，显示消息 “Application is done”，如图 3-11 所示。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，使用 AppWizard 创建新的项目文件，并命名此项目文件为 CH37.MAK，进入 AppStudio 编辑菜单 IDR _ MAINFRAME，在主菜单 Application 中添加新的菜单项 Terminate and Wait，标识符为 ID _ TERM _ WAIT。

2. 进入 ClassWizard，从下拉列表中选择对象 CMainFrame，从对象列表中选择对象 ID _ TERM _ WAIT，从消息列表中选择消息 COMMAND。点击按钮 Add Function。

3. 新方法命名为 OnTermWait，点击按钮 Edit Code，在此方法中添加下列代码：

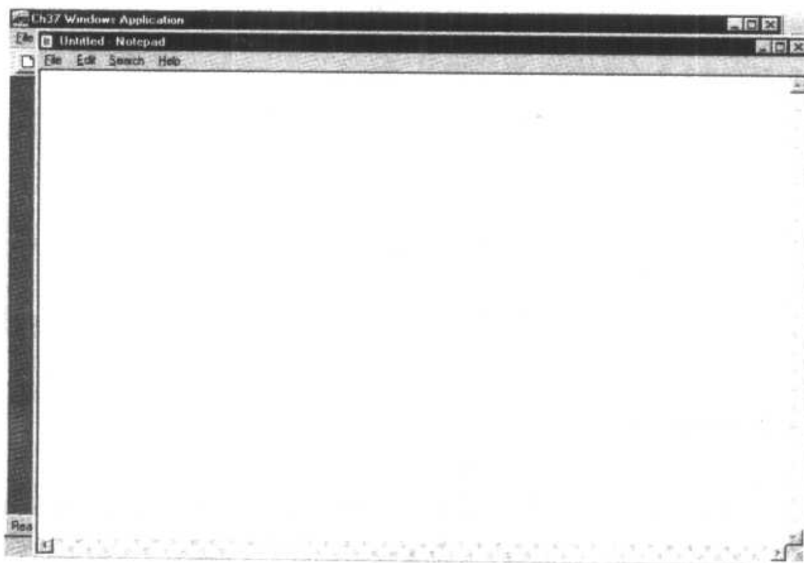


图 3-10 应用程序 CH37 启动 Notepad

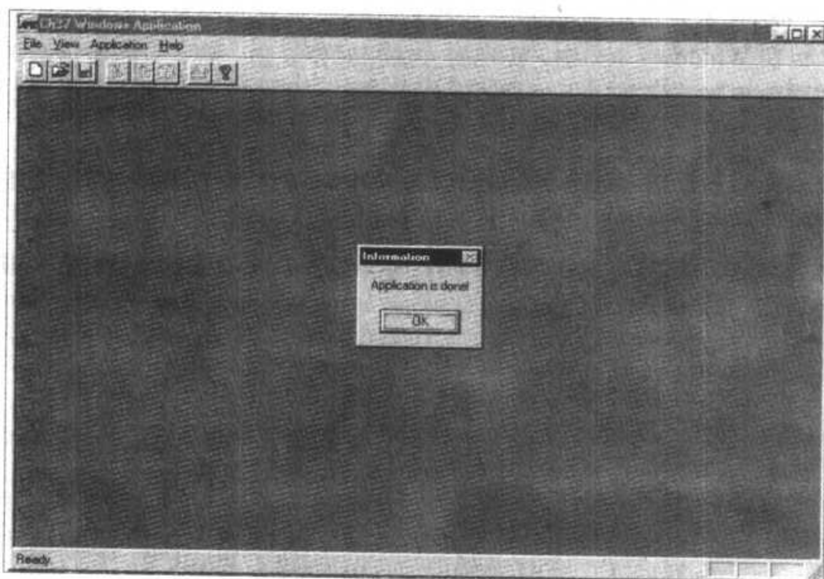


图 3-11 Notepad 退出后应用程序显示消息框

```
void CMainFrame:: OnTermWait()
{

    STARTUPINFO StartupInfo= {0};
    PROCESS_INFORMATION ProcessInfo;

    StartupInfo.cb=sizeof(STARTUPINFO);
```

```
if (CreateProcess(NULL, "notepad.exe", NULL, NULL, FALSE,
                    0, NULL, NULL, &StartupInfo, &ProcessInfo))
{
    WaitForSingleObject(ProcessInfo.hProcess, INFINITE);
    MessageBox ( " Application is done!", " Information", MB_OK );
}
else
{
    MessageBox ( " Unable to run application!", " Error", MB_OK );
}

}
```

4. 编译并运行此例子程序。

用法

当用户从菜单 Application 中选择了菜单项 Terminate and Wait 后，对象 CMainFrame 的方法 OnTermWait 将被调用，此方法使用 Windows 95 API 函数 CreateProcess 创建新的进程，以此进程运行应用程序 Notepad，并检查是否启动成功。

如果应用程序 Notepad 被正常启动，则方法 OnTermWait 将调用 Windows 95 API 函数 WaitForSingleObject 来等待此应用程序(进程)的结束，此函数简单的检查线程并等待此线程结束再返回。但在 Windows 3.x 中，实现的方法相对要复杂一些，只能使用循环不断检查窗口的名字，或者使用下面将介绍的 TOOLHELP 功能，等待窗口关闭的通知。

注释

确定应用程序是否已结束还有另外一个方法，此方法利用 TOOLHELP 功能实现回调函数来通知应用程序的关闭或启动。使用 Delphi 实现的步骤如下：

1. Delphi 需要 TOOLHELP 的回调函数在 DLL 文件中，所以首先创建新的项目文件，并使用下列代码替代项目文件的源代码：

```
Library  Hook;

uses  HookForm;

exports
    HookProc,
    InstallHook,
    UnInstallHook;

begin
   targetHWnd: =0;
end.
```

2. 接着，在项目文件中添加下列代码：

```
Unit  HookForm;

interface
{$F+}
{$k+}
Uses WinTypes, Messages, WinProcs, Toolhelp;

Const
    WM_NOTIFY=WM_USER+$100;
Procedure  UnInstallHook; export;
Procedure  InstallHook( notifyWindow: HWnd ); export;
Function  HookProc( wId: Word; dwData: LongInt ): Bool; export;

Var
    targetHWnd:   HWnd;

implementation

Function  HookProc(wID: Word; dwData: LongInt ): Bool;
    Begin
        PostMessage( targetHWnd, WM_NOTIFY,   wID, dwData );
        Result : =False
    End;

Procedure  InstallHook( notifyWindow: HWnd );
    Begin
        If targetHWnd=0 Then Begin
          If not NotifyRegister( 0, HookProc, NF_NORMAL )
          Then  Begin
                MessageBox( notifyWindow, 'NotifyRegister failed! ',
                            'Error! ', MB_OK+MB_ICONSTOP );
          End
          Else  Begin
             targetHWnd: =notifyWindow;
             PostMessage( notifyWindow, WM_USER+1, $8976, 0);
          End;
        End;
    End;

Procedure  UnInstallHook;
    Begin
        If  targetHWnd <> 0 Then Begin
          NotifyUnregister( 0 );
```

```
        targetHWnd := 0;
    End

end;

end.
```

3. 现在编译此项目文件，创建一个 DLL 文件，输出两个函数 InstallHook 和 UnInstallHook，这两个函数可用来获取来自 TOOLHELP.DLL 中的消息。函数 HookProc 用来传送消息给应用程序，消息包含消息 ID 和域 wParam，消息 ID 为 WM_USER+$100，域 wParam 用来指明通知的标识符，NFY_EXITTASK 表示任务结束，NFY_STARTTASK 表示任务启动。

4. 为了测试此程序，创建新的表单，包含单个按钮 Run。双击按钮 Run，在编辑窗口中输入下列代码：

```
Procedure  UnInstallHook;  far;  external  'HOOK';
Procedure  InstallHook( notifyWindow: HWnd ); far; external 'HOOK';

Procedure  TForm1.Button1Click(Sender: TObject);
begin
   InstallHook(Handle);
   Application.OnMessage := AppMessage;
   WinExec('notepad.exe', SW_SHOW);
end;

procedure  TForm1.AppMessage(var  Msg:  TMsg; var Handled: Boolean);
begin
   if (  Msg.message=WM_USER+$100 ) then
     begin
        case  Msg.wparam of
            NFY_EXITTASK:
                begin
                   MessageBox(0,  'Task Ended',  'Info', MB_OK );
                end;
        end;
     end;
end;
```

5. 编译并运行此例子程序。

3.8 从应用程序中终止和重启动 Windows

问题

有的程序员希望能够从自己的应用程序中重新启动 Windows 系统。例如：安装应用程序时常对 Windows 操作系统作些修改，因此在运行应用程序前便需要终止并重新启动 Windows 操作系统。最好不要依赖用户来终止并重新启动操作系统，因为假如用户忘记了重新启动系统，就有可能引起应用程序的崩溃。

方法

因为从应用程序中重新启动 Windows 系统是相当重要的，所以 Microsoft 提供了两种重新启动方式：只重新启动 Windows，热启动整个计算机，热启动计算机可以装入需要的设备驱动程序，也可以将某些驱动程序清出内存，如果修改了文件 AUTOEXEC.BAT，热启动计算机将使修改生效。

Microsoft 提供了函数 ExitWindowsEx 来实现重启动功能。此函数具有两个参数，第一个参数用来指明重新启动 Windows 的方式，EWX _ SHUTDOWN 表示重新启动操作系统，EWX _ REBOOT 表示热启动整个计算机，无论是哪一种方式，都需要“或”操作 EWX _ FORCE，表示不允许其他的应用程序中断。

步骤

按照下列步骤重新实现一个例子程序。运行此例子程序，从菜单 Application 中选择菜单项 Exit Windows，将弹出一个对话框，用来确认是否重新启动 Windows。由于重新启动 Windows 或热启动整个计算机是不可逆转的操作，所以确认一下是非常有用的。弹出的对话框如图 3-12 所示。

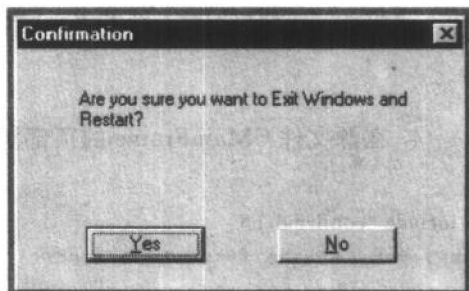


图 3-12　确认是否重新启动 Windows 的对话框

选择菜单 Application，从下拉列表中选择菜单项 Exit Windows and Reboot，将热启动整个计算机。同样也弹出一个对话框，用来确认用户的操作，如图 3-13 所示，点击按钮 Yes，计算机将重新启动。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH38.MAK。进入 AppStudio，编辑菜单 IDR _ MAINFRAME，在菜单 Application 中添加新的菜单项 Exit Windows，标识符为 ID _ EXIT _ WINDOWS。

2. 在 AppStudio 中创建新的对话框，添加静态文本域，标题为“Are you sure you want to Exit Windows and Restart?”。进入 ClassWizard，为此对话框创建新的对话框类 CConfirmDlg，从对象列表中选择 CConfirmDlg，从消息列表中选择 WM _ INITDIALOG，在方法 OnInitDialog 中添加下列代码：

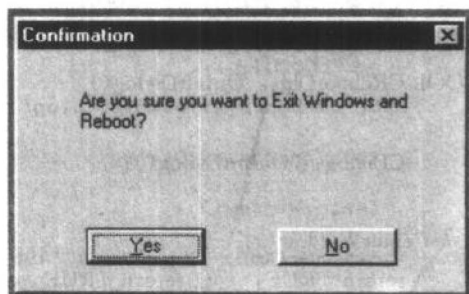


图 3-13　确认是否热启动计算机的对话框

```
BOOL CConfirmDlg::OnInitDialog()
{
```

```
  CDialog:: OnInitDialog();

  CenterWindow();

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

3. 进入ClassWizard，从下拉列表中选择对象CMainFrame，从对象列表中选择ID_EXIT_WINDOWS，从消息列表中选择COMMAND。点击按钮Add Function，方法命名为OnExitWindows，在此方法中添加下列代码：

```
void CMainFrame:: OnExitWindows()
{
   CConfirmDlg dlg;

   if ( dlg.DoModal()==IDOK ) {
      ExitWindowsEx(EWX_FORCE | EWX_SHUTDOWN, 0);
   }
}
```

4. 在源文件CMainFrame的顶部添加下列include文件行：

```
#include "confirmd.h"
```

5. 进入AppStudio，编辑菜单IDR_MAINFRAME，在菜单Application中添加新的菜单项Exit Windows and Reboot，标识符为ID_EXIT_REBOOT。

6. 在AppStudio中创建新的对话框，添加静态文本域，标题为“Are you sure you want to Exit Windows and Reboot?”。进入ClassWizard，为此对话框创建新的对话框类CRebootDlg。从对象列表中选择CRebootDlg，从消息列表中选择WM_INITDIALOG，在方法OnInitDialog中添加下列代码：

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

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

7. 进入ClassWizard，从下拉列表中选择对象CMainFrame，从对象列表中选择ID_EXIT_REBOOT，从消息列表中选择COMMAND。点击按钮Add Function，方法命名为OnExitReboot，并在此方法中添加下列代码：

```
void CMainFrame:: OnExitReboot()
```

```
{
  CRebootDlg dlg;

  if ( dlg.DoModal()==IDOK ){
     ExitWindowsEx(EWX_FORCE | EWX_REBOOT, 0);
  }

}
```

8. 在源文件CMainFrame的顶部添加下列include文件行：

```
#include  " rebootdl.h"
```

9. 编译并运行此例子程序。

用法

选择菜单项Exit Windows后，类CMainFrame的方法OnExitWindows将被调用，此方法调用Windows API函数ExitWindowsEx，第一个参数为EWX_FORCE|EWX_SHUTDOWN，表示重新启动Windows，并忽略其他应用程序的不许重新启动的请求。

选择菜单项Exit Windows and Reboot后，类CMainFrame的方法OnExitReboot将被调用，此方法调用了Windows API函数ExitWindowsEx，第一个参数为EWX_FORCE|EWX_REBOOT，表示热启动整个计算机。

注释

API函数ExitWindowsEx非常容易调用，而且可以使用任何语言。下列步骤便是使用Visual Basic调用函数ExitWindowsEx的例子。

1. 创建新的项目文件或在已有的项目文件中添加新的表单，在表单的general部分添加下列说明：

```
Const EWX_LOGOFF=0
Const EWX_SHUTDOWN=1
Const EWX_REBOOT=2
Const EWX_FORCE=4

Private Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As
Long, ByVal dwReserved As Long) As Long
```

2. 接着，在表单中添加两个按钮Exit Windows和Exit and Reboot。

3. 双击按钮Exit Windows，在函数Command1_Click中添加下列代码：

```
Private Sub Command1_Click()
```

```
    x%=ExitWindowsEx(EWX _FORCE Or EWX _SHUTDOWN, 0)
End Sub
```

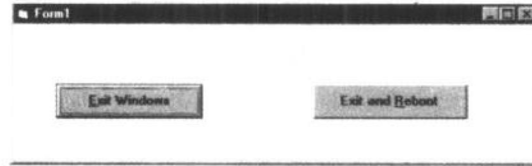


图 3-14 重启动 Windows 和热启动计算机的 Visual Basic 表单

4. 双击按钮 Exit and Reboot，在函数 Command2 _ Click 中添加下列代码：

```
Private Sub Command2 _ Click()
    x%=ExitWindowsEx(EWX _FORCE Or EWX _REBOOT, 0)
End  Sub
```

5. 保存此项目文件。运行此例子程序，将弹出一个对话框，如图 3-14 所示。点击按钮 Exit Windows，则 Windows 系统将重新启动。点击按钮 Exit and Reboot，则 Windows 系统将退出，而整个计算机系统将重新启动。

第 4 章　绘图和图形设备接口

图形设备接口（GDI）是一个函数库，是 Windows 接口的主要组成部分，这些函数用来实现所有的屏幕输出以及其他设备如打印机和绘图仪的输出。本章中的问题都是程序员使用 GDI 时常遇到的问题，在本章中介绍了许多解决方法，可以使得应用程序的图形功能的开发更加出色。

表 4-1 列出了所有在本章中使用的 Windows API 函数。

表 4-1　第 4 章中使用的 Windows API 函数

LoadBitmap	CreatePalette	SelectPalette
RealizePalette	SetDIBits	FindResource
LoadResource	LockResource	BitBlt
SetROP2	GetWindowRect	GetClientRect
ClientToScreen	ClipCursor	GetDesktopWindow
GetForegroundWindow	StretchBlt	timeBeginPeriod
timeSetEvent	timeKillEvent	timeEndPeriod

1. 如何显示 256 色的位图

许多程序员设计了非常漂亮的屏幕、背景以及其他位图，准备在应用程序中显示，但是，创建漂亮的图象仅仅是问题的一部分，而将这些 256 色的位图显示在屏幕上并不象说起来那么容易。本节将介绍如何来装入和显示设备无关位图。

2. 如何改变位图中的颜色

本节介绍了处理位图的第二个重要技术——操作调色板。讨论了在位图中使用调色板的各种不同的方法，示范了一个简单的方法来改变位图中的颜色。

3. 如何旋转位图

本节示范了一个简单的方法来以 90 度增量旋转位图，同时也展示了如何在不使用 GDI 函数的情况下存取设备无关位图中的内部数据。

4. 如何随鼠标拖动绘制“橡皮带线”

无论是编写一个功能强大的绘图程序，还是编写一个流行的字处理软件，都需要提供给用户某些功能，使得用户可以非常方便地拖动鼠标来绘制图形。橡皮带线技术就可以使得用户移动鼠标来绘制矩形或其他的一些形状，并可以移动和改变这些形状，使用橡皮带线来绘制图形可以不断地给用户以信息反馈。本节介绍了如何使用 Windows API 函数来快捷地绘制橡皮带线。

5. 如何捕捉窗口或部分屏幕

本节介绍了如何使用 GDI 来捕捉屏幕或任何窗口的位图图象，这样便可以捕捉其他应用程序的图象，用于自己的某些需要。

6. 如何生成动画

本节介绍了如何使用 Windows 95 API 来生成简单的位图动画。利用了前面几节介绍的

位图操作，以及本节介绍的实现模拟透明位图的方法。

7. 如何实现位图的拖放

在 Windows 95 的对象链接和嵌入（OLE）中大量地使用了拖放技术。即使不在 OLE 中，在 Windows 接口和应用程序的其他部分也大量使用了拖放技术，为的是简化操作。在本节中将介绍实现位图拖放的方法。

4.1 显示 256 色的位图

问题

显示位图似乎相当简单。在应用程序的资源中添加一张漂亮的位图，使用函数 LoadBitmap 将位图装入内存，然后将位图选入设备描述表，再使用函数 BitBlt 便可显示出位图。但是这样显示的位图最多只能有 16 种颜色，如何显示 256 色的位图呢？

方法

要显示位图，首先要了解 WINDWOS 95 中的位图的概念。位图有两种基本格式：设备相关位图（DDB）和设备无关位图（DIB）。在设备相关位图中，只有大小信息和位图数据，没有位图的设备分辨率信息，也没有位图的颜色信息。

在 Windows 3.0 中引进了一种新的位图格式：设备无关位图。这种格式的位图包含位图的分辨率信息和颜色信息，资源编辑器创建的 .BMP 文件便是此种位图格式。图 4-1 说明了设备相关位图和设备无关位图间的区别。

在编译应用程序时，开发环境将把资源数据联编在应用程序末，位图以设备无关位图的格式联编，包含所有颜色的调色板信息。但 Windows API 中的位图句柄（HBITMAP）所指的位图为设备相关位图，设备相关位图只有在选入设备描述表后才能操作，设备描述表为位图提供颜色、大小和分辨率信息。

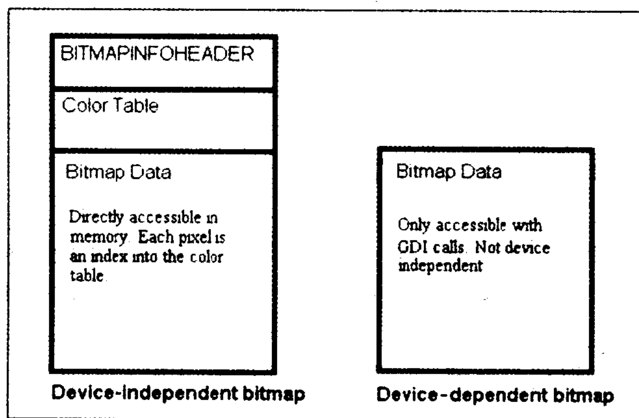


图 4-1 设备无关位图和设备相关位图的不同结构

当调用 API 函数 LoadBitmap 时，Windows 便将设备无关位图装入内存，并自动转换成 4 位（16 色）的设备相关位图，所以会有颜色信息的丢失。在本节的例子程序中将调用更底层的 API 函数，自己将位图资源装入内存，并建立指针以便应用程序可存取，接着调用 GDI 函数将设备无关的位图转换成设备相关的位图，以便于显示。使用此方法，便可保留所有需

要的颜色信息。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示出一个有菜单的简单窗口，从菜单 File 中选择菜单项 LoadBitmap，例子程序便调用 API 函数 LoadBitmap 装入位图资源，接着将其选入设备描述表并显示在窗口中。

从菜单 File 中选择菜单项 LoadResource，例子程序将调用更底层的 API 函数来装入位图资源并将其转换成设备相关位图。比较这两种位图显示的效果，在 256 色或更多颜色的显示模式下区别会更明显。选择菜单项 LoadResource 后显示的例子程序主窗口如图 4-2 所示。注：如果例子程序运行在单色显示器的系统上，或显示器的驱动程序只支持 16 色，那么这两个菜单项的显示将没有任何区别。

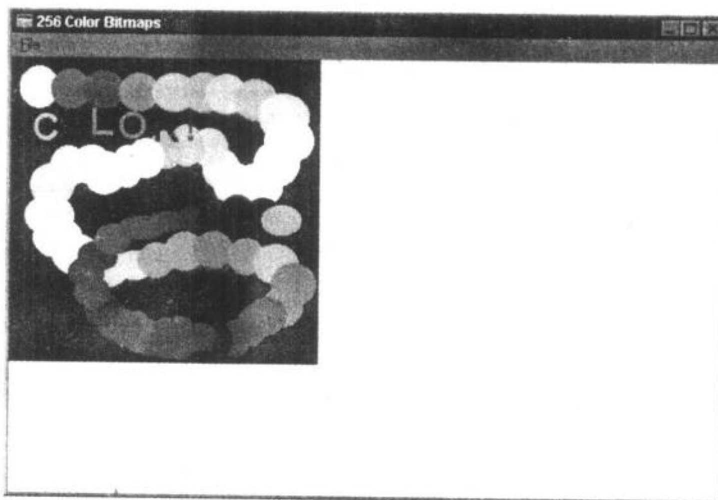


图 4-2　例子程序在选择菜单项 LoadResource 后

实现例子程序的具体步骤如下：

1. 创建新的目录 256COLOR，用来存放此例子程序的所有文件。

2. 创建一个位图，联编到例子程序中并在运行时显示。可以使用资源编辑器或应用程序 paint 创建一个 256×256 象素的位图，要确保位图是 256 色的(8 位)，并存成标准的 Windows 位图文件，命名为 SAMPLE.BMP。保存文件时不要使用 RLE (run length encoding) 压缩，因为有的显示驱动程序不支持此种压缩算法。

3. 为例子程序准备图标，可以使用资源编辑器来创建此图标。

4. 创建资源脚本。资源脚本中包含菜单、图标和位图，这些资源将联编到例子程序中。用户或许通常使用资源编辑器来创建资源脚本，但这里使用文本编辑器创建文件 256COLOR.RC，在此文件中键入下列文本：

```
/*------------------------------------------------------------------*
/*                                                                  */
/* MODULE: 256COLOR.RC                                              */
/* PURPOSE: This resource script defines the menu and application   */
/*          icon associated with this application, and also includes */
```

```
/*              a bitmap to be loaded and displayed.                                  */
/*                                                                                    */
/*------------------------------------------------------------------------------------*/
#include "256color.rh"

IDM_BITMAPMENU MENU
{
POPUP "&File"
{
  MENUITEM "Use &LoadBitmap", CM_USELOADBITMAP
  MENUITEM "Use &DIB Functions", CM_USEDIBFUNCS
  MENUITEM SEPARATOR
  MENUITEM "E&xit", CM_EXIT
}
}

IDI_APPICON ICON "256color.ico"
IDBM_SAMPLEBITMAP BITMAP "sample.bmp"
```

5. 刚创建的资源脚本定义了此例子程序的资源，但包含了一个没有创建的文件256COLOR.RH，此文件为菜单项、图标和位图定义标识符，这些标识符在例子程序中用来引用相应的资源。创建新文件256COLOR.RH并在此文件中键入下列代码：

```
#ifndef __256COLOR_RH
/*------------------------------------------------------------------------------------*/
/*                                                                                    */
/* MODULE: 256COLOR.RH                                                                */
/* PURPOSE: Defines identifiers used to address resources.                            */
/*                                                                                    */
/*------------------------------------------------------------------------------------*/
#define __256COLOR_RH

#define IDM_BITMAPMENU                301
#define CM_USELOADBITMAP              101
#define CM_USEDIBFUNCS                102
#define CM_EXIT                       103

#define IDI_APPICON                   1
#define IDBM_SAMPLEBITMAP             3567

#endif
```

6. 最后开始编写例子程序代码。本例子程序的代码使用普通的C语言，所以可以使用任何C语言编译器进行编译。使用文本编辑器创建新文件256COLOR.C并键入下列代码，代码包含了本例子程序所需要的 Windows 头文件以及包含资源标识符定义的文件

256COLOR.RH，另外定义了预处理符号STRICT，用来表示对Windows应用程序中使用的窗口句柄、菜单句柄和位图句柄进行严格的类型检查。

```
/*-----------------------------------------------------------------------*
/*                                                                        *
/* MODULE: 256COLOR.C                                                     *
/* PURPOSE: This C program loads and displays a 256 color bitmap          *
/*          using the Windows 95 API routines. If you use the API         *
/*          LoadBitmap to load a bitmap from a resource, a DDB            *
/*          (device dependant bitmap) is created with only 16 colors.     *
/*          by loading the bitmap ourselves and converting it from        *
/*          a DIB (device independant bitmap) to a DDB, we retain         *
/*          the correct colors.                                           *
/*                                                                        */
/*-----------------------------------------------------------------------*/
#define STRICT

#include <windows.h>
#include <windowsx.h>
#include "256color.rh"
```

7. 在同一源文件中添加下列定义。MainWindowClassName定义了此例子程序使用的窗口类的名字，其他的句柄和指针是全局变量，用在整个例子程序中，初始化为零或NULL是为了避免以后出现错误。

```
char                  *MainWindowClassName = "256ColorTest";
HBITMAP               hDDBitmap = NULL;
HPALETTE              hPalette = NULL;
BITMAPINFOHEADER      *pInfoHeader = NULL;
HINSTANCE             hInstance = NULL;
```

8. 在源文件中添加下列函数。函数CreateDIBPalette用来从设备无关位图中取出颜色表信息，创建逻辑调色板，创建和显示设备相关位图时使用此调色板。传送给函数的是BITMAPINFOHEADER结构的指针，指向内存中的设备无关位图。此函数的返回值是创建的逻辑调色板的句柄，如果此函数失败，则句柄为NULL。

```
/*-----------------------------------------------------------------------*/
/* This function creates a logical palette structure from the color     */
/* table supplied with our device independant bitmap. The palette is    */
/* saved into the global hPalette variable for use when converting      */
/* and displaying the image.                                             */
/*-----------------------------------------------------------------------*/
HPALETTE CreateDIBPalette (BITMAPINFOHEADER *info)
{
    LOGPALETTE          *palPtr;
    RGBQUAD             *colorTable;
    WORD                i;
```

```
    DWORD                  numEntries;
    HPALETTE               hPal;

    /* Allocate space for a LOGPALETTE structure and array of
       entries. Using LocalAlloc with the LMEM_FIXED argument
       gives us a pointer without having to lock and unlock it. */
    if (info->biClrUsed != 0)
        numEntries = info->biClrUsed;
    else
        numEntries = 1 << info->biBitCount;

    palPtr = (NPLOGPALETTE) LocalAlloc (LMEM_FIXED,
           sizeof (LOGPALETTE) + numEntries * sizeof (PALETTEENTRY));
    if (! palPtr)
        return NULL;

    palPtr->palVersion = 0x300;
    palPtr->palNumEntries = (WORD) numEntries;

    // Now fill in the array of palette entries from the color table.
    colorTable = (RGBQUAD *) ( (LPSTR) info + (WORD) info->biSize);
    for (i = 0; i < numEntries; i++)
     {
        palPtr->palPalEntry [i] .peRed = colorTable [i] .rgbRed;
        palPtr->palPalEntry [i] .peGreen = colorTable [i] .rgbGreen;
        palPtr->palPalEntry [i] .peBlue = colorTable [i] .rgbBlue;
        palPtr->palPalEntry [i] .peFlags = 0;
    }

    hPal = CreatePalette (palPtr);
    LocalFree ( (HLOCAL) palPtr);
    return hPal;
}
```

9. 在源文件中添加下列代码。函数 ConvertDIBToDDB 用来把设备无关位图转换成设备相关位图，而设备相关位图可以被快速地显示在屏幕上。此函数的实现方法为：首先创建一个内存设备描述表，同显示位图的屏幕相兼容，接着在设备描述表中创建设备相关位图，大小要同设备无关位图的大小一样，在设备描述表中调入前面刚创建的逻辑调色板，并实现调色板，最后，使用函数 SetDIBits 拷贝位图，按要求转换颜色。DIB_RGB_COLORS 表示设备无关位图中的颜色表的颜色是 RGB 值，而不是系统调色板的索引。

```
/*-------------------------------------------------------------------------*/
/* This function converts a DIB to a DDB created for a specific DC        */
/* and palette. Note that we use the screen DC as we are going to         */
```

```
/* place the image in a window. If we were going to use another          */
/* device we should create a compatible DC for that device. Usually      */
/* when printing to a printer device, time constraints are not so        */
/* important, so conversion from DIB to DDB and the actual painting      */
/* can be done in one step.                                              */
/*-----------------------------------------------------------------------*/
HBITMAP ConvertDIBToDDB (HWND hWnd, BITMAPINFOHEADER *info,
                         HPALETTE *hPalette)
{
    HDC             hDC, hMemDC;
    HPALETTE        hOldPalette;
    HBITMAP         hOldBitmap, hDDBitmap;
    DWORD           numEntries;

    hDC = GetDC (hWnd); // Get a handle to our device context.
    if ( (*hPalette = CreateDIBPalette (info)) != NULL)
    {
        hMemDC = CreateCompatibleDC (hDC);
        hOldPalette = SelectPalette (hMemDC, *hPalette, FALSE);
        RealizePalette (hMemDC);
        hDDBitmap = CreateCompatibleBitmap (hDC,
                                            info->biWidth,
                                            info->biHeight);
        hOldBitmap = SelectObject (hMemDC, hDDBitmap);
        if (info->biClrUsed != 0)
            numEntries = info->biClrUsed;
        else
            numEntries = 1 << info->biBitCount;
        /* Use SetDIBits to place the device independent bitmap data
           into the new bitmap that we have created, hDDBitmap. */
        SetDIBits (hMemDC, hDDBitmap,
                   0, info->biHeight,
                   (LPSTR) info + (info->biSize +
                     (numEntries * sizeof (RGBQUAD))),
                   (BITMAPINFO *) info, DIB_RGB_COLORS);
        SelectObject (hMemDC, hOldBitmap);
        SelectPalette (hMemDC, hOldPalette, FALSE);
        DeleteDC (hMemDC);
    }
    ReleaseDC (hWnd, hDC);

    return hDDBitmap;
```

10. 在同一源文件中添加函数 LoadBitmapResource。此函数联合使用函数 FindResource、LoadResource 和 LockResource 来将需要的位图资源装入内存，返回一个 BITMAPINFOHEADER 结构的指针，指向内存中的设备无关位图的开始处。

使用这些函数应该注意两件事情：第一件事情是资源是只读的。尽管函数 LockResource 返回一个指向资源的指针，但任何修改资源的企图都会引起保护错误，要修改位图（象后面章节实现的）需要创建一个拷贝，并且只能修改拷贝。

应该注意的第二件事情是 Windows 95 和 Windows NT 跟踪应用程序使用的资源，当应用程序结束后，Windows 负责进行释放，所以应用程序可以不考虑资源的释放。

```
/* ---------------------------------------------------------------- */
/* This function opens a bitmap resource attatched to our executable */
/* file and loads it into memory as a device independant bitmap.     */
/* The function returns a pointer to the resource data, or NULL if   */
/* it fails.                                                          */
/* ---------------------------------------------------------------- */
BITMAPINFOHEADER *LoadBitmapResource (HINSTANCE hInstance, WORD resId)
{
    HRSRC          hResource;
    HGLOBAL        hDib;

    if ( ( (hResource = FindResource (hInstance,
                              MAKEINTRESOURCE (resId),
                              RT_BITMAP)) != NULL) &&
         ( (hDib = LoadResource (hInstance, hResource)) != NULL))
        return (BITMAPINFOHEADER *) LockResource (hDib);
    return NULL;
}
```

11. 在源文件中添加函数 Paint。每当 Windows 操作系统发送消息 WM_PAINT 给例子程序窗口时，此函数将被调用，这时，位图已经转换为内存中的设备相关位图和调色板，所以可以相当迅速地将位图显示在屏幕上，将调色板选入当前屏幕的设备描述表并实现调色板，以便显示正确的图象颜色，接着将位图选入内存设备描述表，调用 API 函数 BitBlt 将位图显示在屏幕上。

```
/* ---------------------------------------------------------------- */
/* This is the actual drawing function called when WM_PAINT is      */
/* received. Because the conversion from DIB to DBB is done once    */
/* when the bitmap is loaded, this routine is a small and QUICK     */
/* Realization of the palette and BitBlt into the device context.   */
/* ---------------------------------------------------------------- */
void Paint (HWND hWnd, HPALETTE hPalette, HBITMAP hBitmap,
                         int BitmapWidth, int BitmapHeight)
{
    PAINTSTRUCT          ps;
```

```
    HDC                     hDC, hMemDC;
    HPALETTE                hOldPalette;
    HBITMAP                 hOldBitmap;

    hDC = BeginPaint (hWnd, &ps);

    // Realize the palette so that the colors are correct.
    if (hPalette)
    {
        hOldPalette = SelectPalette (hDC, hPalette, FALSE);
        RealizePalette (hDC);
    }
    // Select the bitmap into a memory DC then Blit it to the screen.
    if ( (hMemDC = CreateCompatibleDC (hDC)) ! = NULL)
    {
        hOldBitmap = SelectObject (hMemDC, hBitmap);
        BitBlt (hDC, 0, 0, BitmapWidth, BitmapHeight,
                                      hMemDC, 0, 0, SRCCOPY);
        SelectBitmap (hMemDC, hOldBitmap);
        DeleteDC (hMemDC);
    }
    if (hPalette)
      SelectPalette (hDC, hOldPalette, FALSE);
    EndPaint (hWnd, &ps);
}
```

12. 现在已经键入了此例子程序的大部分代码，在下面几步中所添加的函数构成了本例子程序的框架。下面的函数 MainWndProc 是一个回调函数，用来处理所有发送给例子程序窗口的消息。在本例子程序中，需要响应三个菜单项：CM _ USELOADBITMAP 对应菜单项 LoadBitmap，CM _ USEDIBFUNCS 对应菜单项 LoadResource，CM _ EXIT 对应菜单项 Exit。当用户选择了菜单项 LoadResource 后，此函数将调用前面刚创建的函数，装入位图，创建调色板，并将设备无关位图转换成设备相关位图，接着调用函数 InvalidateRect 刷新窗口，从而显示出新装入的位图。

当用户从菜单中选择了菜单项 LoadBitmap 后，例子程序将丢弃前面已装入的位图，调用 API 函数 LoadBitmap 从资源中装入位图，Windows 将对位图中的颜色进行转换，生成 16 色的设备相关位图。

```
/*-----------------------------------------------------------------------*/
/* This function is the main window callback function which will be     */
/* called by Windows to process messages for our window.                 */
/*-----------------------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                     WPARAM wParam, LPARAM lParam)
{
```

```
switch (message)
  {
    case WM_COMMAND:
        switch (wParam)
          {
            /* Usually the first method tried — load the resource
               directly into a DDB — Windows does the color conversion */
            case CM_USELOADBITMAP:
                /* Dont bother with anything if a DDB is already
                   loaded but no DIB */
                if ( (hDDBitmap) && (! pInfoHeader))
                    break;
                /* If the DIB stuff is around, remove it */
                if (pInfoHeader)
                    pInfoHeader = NULL;
                if (hDDBitmap)
                  {
                    DeleteObject (hDDBitmap);
                    hDDBitmap = NULL;
                  }
                if (hPalette)
                  {
                    DeleteObject (hPalette);
                    hPalette = NULL;
                  }

                if ( (hDDBitmap = LoadBitmap (hInstance,
                   MAKEINTRESOURCE (IDBM_SAMPLEBITMAP))) ! = NULL)
                InvalidateRect (hWnd, NULL, FALSE);
                break;
            /* The user would like to load the resource as a DIB and
               have it converted to a DDB to display, retaining color
               information */
            case CM_USEDIBFUNCS:
                // Don't bother to reload if already exists
                if (pInfoHeader)
                     break;
                //  Remove the old Bitmap
                if (hDDBitmap)
                  {
                    DeleteObject (hDDBitmap);
                    hDDBitmap = NULL;
```

```
            }
            /* Now load a DIB, convert it to DDB (creating a
               palette along the way */
            if ( (pInfoHeader = LoadBitmapResource (hInstance,
                                 IDBM_SAMPLEBITMAP)) != NULL)
             {
               if ( (hDDBitmap = ConvertDIBToDDB (hWnd,
                                 pInfoHeader, &hPalette)) != NULL)
                  InvalidateRect (hWnd, NULL, FALSE);
               else
                 MessageBox (hWnd," Unable to create DDB",
                             NULL, MB_ICONSTOP | MB_OK);
            }
            else
              MessageBox (hWnd," Unable to load DIB",
                          NULL, MB_ICONSTOP | MB_OK);
            break;
        case CM_EXIT:
            DestroyWindow (hWnd);
            break;
        default:
            return (DefWindowProc (hWnd, message, wParam, lParam));
       }
       break;
case WM_PALETTECHANGED:
     // Force a repaint
     if ( ( (HWND) wParam != hWnd) && (hDDBitmap != NULL))
         InvalidateRect (hWnd, NULL, TRUE);
     break;
case WM_PAINT:
     if (hDDBitmap != NULL)
     {
         if (pInfoHeader)
             Paint (hWnd, hPalette, hDDBitmap,
                    pInfoHeader->biWidth, pInfoHeader->biHeight);
         else
             Paint (hWnd, hPalette, hDDBitmap, 256, 256);
     }
     else
         return (DefWindowProc (hWnd, message, wParam, lParam));
  break;
case WM_DESTROY:
     if (hDDBitmap)
```

```
            DeleteObject (hDDBitmap);
        if (hPalette)
            DeleteObject (hPalette);
        PostQuitMessage (0);
        break;
    default:
        return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

13. 在源文件中添加函数 InitApplication。例子程序第一个实例运行时调用此函数，接着只有在窗口类不再存在时才调用此函数，此函数为例子程序的主窗口初始化并登记窗口类。

```
/*-------------------------------------------------------------------------*/
/* This function initialises a WNDCLASS structure and uses it to            */
/* register a class for our main window.                                    */
/*-------------------------------------------------------------------------*/
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName = MAKEINTRESOURCE (IDM_BITMAPMENU);
    wc.lpszClassName = MainWindowClassName;
    return RegisterClass (&wc);
}
```

14. 添加函数 InitInstance。例子程序的每个实例运行都将调用此函数，用来创建并显示此例子程序的主窗口。

```
/*-------------------------------------------------------------------------*/
/* This function creates an instance of our main window. The window         */
/* is given a class name and a title, and told to display anywhere.        */
/* the nCmdShow argument passed to the program determines how the           */
/* window will be displayed.                                                */
/*-------------------------------------------------------------------------*/
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
```

```
    HWND hWnd;

    hInstance = hInst; // Store in global variable.

hWnd = CreateWindow (MainWindowClassName," 256 Color Bitmaps",
            WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
            NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);          // Send a WM_PAINT message
    return TRUE;
}
```

15. 此例子程序的最后一段代码是函数 WinMain。此函数是 Windows 操作系统上的应用程序的入口点，完成窗口类和窗口的初始化，接着循环处理消息直到例子程序结束。

```
/*---------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if       */
/* the main window class name has already been registered, if it has    */
/* not, call InitApplication to register it. Call InitInstance to       */
/* create an instance of our main window, then pump messages until      */
/* the application is closed.                                           */
/*---------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
     }
    return (msg.wParam);
}
```

16. 编译并运行此例子程序。

注释

在 Windows 中显示 256 色的位图决不是一件简单的任务。本节的例子程序只适合于显示静态的位图，显示动画则效果不是很好，因为大量时间花费在完成逻辑调色板向物理调色板的转换上，以及将设备无关位图中 RGB 值转换到设备相关位图中的调色板中，所以使处理速度非常缓慢。其他处理调色板的方法将在下一节和 4.5 节中讨论。

4.2 改变位图中的颜色

问题

有时需要改变位图中某些颜色，如何实现呢？

方法

8 位的设备无关位图中的每一字节都是某一调色板的索引。调色板可能是位图中的颜色表，也可能是当前实现的逻辑调色板，或者是设备的物理调色板。由于位图(类型为 DIB_RGB_COLORS)使用颜色表进行解释，所以可以通过改变颜色表中的相应项来改变图象中任何颜色。如果知道颜色的颜色表索引，可以直接修改颜色表中的相应值，如果只知道颜色的 RGB 值而不知道索引值，可以遍历颜色表进行 RGB 值比较，修改所有匹配的项。本节的例子程序便是使用此方法。

在修改颜色表时应注意资源是不可直接修改的。在 32 位的 Windows 软件中，资源是只读的。在本节的例子程序中，为了修改颜色表，对资源做了拷贝，为了节约时间和空间，只拷贝了需要修改的部分 (BITMAPINFOHEADER 和颜色表)，DIB 的数据部分没有拷贝，但使用指针指向了这部分数据，使其可以同拷贝的信息头和颜色表一起用来创建设备相关位图和调色板。如果需要改变位图的数据，那么数据部分也必须进行拷贝。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示出一个 256 色非常简单的位图。如果从菜单 File 中选择菜单项 Change Colors，显示的位图将会改变，某些字消失，而另一些改变颜色，但位图的数据并没有改变，只是改变了用来解释位图的调色板，再选择菜单项 Change Colors 图象将恢复到原来的状态。颜色改变前的显示如图 4-3 所示。

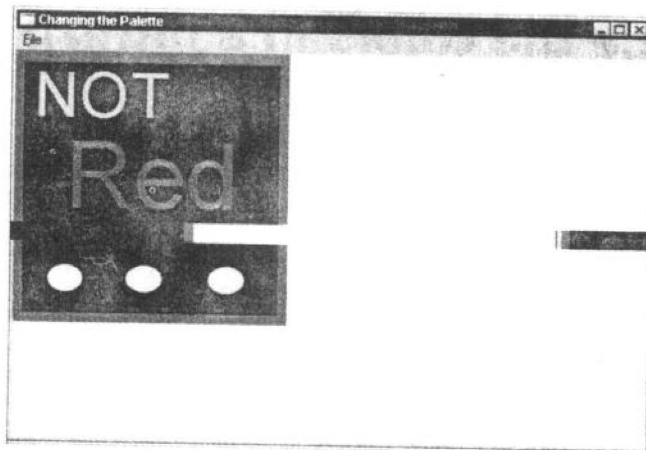


图 4-3 改变颜色前的例子程序

实现例子程序的具体步骤如下：

1. 为此例子程序创建新的目录 CHGCOLOR，用来存放所有的文件。

2. 为此例子程序生成需要的位图。使用资源编辑器创建一个 256 色的位图（大约 256×256 象素），位图中要有红色（RGB 值为 0xFF0000 或 255，0，0）、蓝色（RGB 值为 0x0000FF 或 0，0，255）以及绿色（RGB 值为 0x00FF00 或 0，255，0），因为在例子程序中要操作这些颜色，如果位图中没有这些颜色，例子程序还可以运行，但将看不到任何改变。

3. 为了将位图包含在例子程序的资源中，应该创建资源脚本。此资源脚本同时还定义菜单以及例子程序的图标。创建新文件 CHANGE.RC，在文件中键入下列资源脚本：

```
/*------------------------------------------------------------------------*/
/*                                                                        */
/* MODULE: CHANGE.RC                                                      */
/* PURPOSE: This resource script defines the menu and application         */
/*          icon associated with this application, and also includes      */
/*          a bitmap to be loaded and displayed.                          */
/*                                                                        */
/*------------------------------------------------------------------------*/
#include "change.rh"

IDM_BITMAPMENU MENU

{
  POPUP "&File"
   {
      MENUITEM "Change Color", CM_CHANGE
      MENUITEM SEPARATOR
      MENUITEM "E&xit", CM_EXIT
   }
}

IDI_APPICON ICON "change.ico"

IDBM_SAMPLEBITMAP BITMAP "sample.bmp"
```

4. 前面创建的资源脚本需要包含文件 CHANGE.RH。此文件定义了资源的标识符，用在资源脚本中和例子程序的源文件中。创建新文件 CHANGE.RH，在文件中键入下列代码：

```
#ifndef __CHANGE_RH
/*------------------------------------------------------------------------*/
/*                                                                        */
/* MODULE: CHANGE.RH                                                      */
/* PURPOSE: Defines identifiers used to address resources.                */
/*                                                                        */
/*------------------------------------------------------------------------*/
#define __CHANGE_RH
```

```
#define IDM_BITMAPMENU          301
#define CM_CHANGE               101
#define CM_EXIT                 102

#define IDI_APPICON             1
#define IDBM_SAMPLEBITMAP       3567

#endif
```

5. 创建例子程序的图标。创建一个 32×32 象素 16 色的图标，文件命名为 CHANGE.ICO。

6. 开始为例子程序创建源文件。使用文本编辑器创建新文件 CHANGE.C，在文件中添加下列代码，这段代码包含了需要的头文件，定义了预处理器符号 STRICT，表示对例子程序中使用的各种不同的类型和句柄进行严格的类型检查。

另外定义了一些常量和全局变量。MainWindowClassName 定义了例子程序的窗口类名，四个 RGBQUAD 定义了例子程序要操作的颜色，所有的全局变量都初始化为缺省值。

```
/*-------------------------------------------------------------*/
/*                                                             */
/* MODULE: CHANGE.C                                            */
/* PURPOSE: Demonstrates how you can change the colors in the */
/*          palette of a device independant bitmap.            */
/*                                                             */
/*-------------------------------------------------------------*/
#define STRICT

#include <windows.h>
#include <windowsx.h>
#include " change.rh"
// Constants.
char      *MainWindowClassName = " ColorChangeTest";

RGBQUAD   colorRed = { 0x00, 0x00, 0xFF, 0 };
RGBQUAD   colorGreen = { 0x00, 0xFF, 0x00, 0 };
RGBQUAD   colorBlue = { 0xFF, 0x00, 0x00, 0 };
RGBQUAD   colorBlack = { 0x00, 0x00, 0x00, 0 };

// Global variables.
HBITMAP       hDDBitmap = NULL;
HPALETTE      hPalette = NULL;
BITMAPINFO    *pResourceDIB = NULL;
BITMAPINFO    *pActiveDIB = NULL;
VOID          *pResourceBits = NULL;
```

```
HINSTANCE             hInstance= NULL;
BOOL                  isOriginal = TRUE;
```

7. 在同一源文件中添加函数CreateDIBPalette，代码如下。此函数使用传送来的颜色表创建一个新的逻辑调色板，如果成功则返回调色板的句柄，如果失败则返回NULL。

```
/* ----------------------------------------------------------------------- */
/* This function creates a logical palette structure from the color        */
/* table supplied with our device independant bitmap.                      */
/* ----------------------------------------------------------------------- */
HPALETTE CreateDIBPalette (BITMAPINFO *info)
{
    LOGPALETTE            *palPtr;
    DWORD                 i, numEntries;
    HPALETTE              hPal;

    /* Allocate space for a LOGPALETTE structure and array of
       entries. Using LocalAlloc with the LMEM_FIXED argument
       gives us a pointer without having to lock and unlock it. */
    if ( (numEntries = info->bmiHeader.biClrUsed) == 0)
        numEntries = 1L << info->bmiHeader.biBitCount;
    palPtr = (LOGPALETTE *) LocalAlloc (LMEM_FIXED, sizeof (LOGPALETTE) +
                                  numEntries * sizeof (PALETTEENTRY));
    if (! palPtr)
        return NULL;

    palPtr->palVersion = 0x300;
    palPtr->palNumEntries = (WORD) numEntries;

    /* Now fill in the array of palette entries from the color table */
    for (i = 0; i < numEntries; i++)
     {
        palPtr->palPalEntry [i] .peRed = info->bmiColors [i] .rgbRed;
        palPtr->palPalEntry [i] .peGreen = info->bmiColors [i] .rgbGreen;
        palPtr->palPalEntry [i] .peBlue = info->bmiColors [i] .rgbBlue;
        palPtr->palPalEntry [i] .peFlags = 0;
    }

    hPal = CreatePalette (palPtr);
    LocalFree ( (HLOCAL) palPtr);
    return hPal;
}
```

8. 添加函数ConvertDIBToDDB。此函数创建新的设备相关位图，拷贝设备无关位图中的位图数据，并完成所需要的转换。在此函数中，调用函数CreateDIBPalette来创建逻辑调色板，

接着选入内存设备描述表并实现调色板，从而保证了在转换位图时使用此调色板。

```
/*------------------------------------------------------------------------*/
/* This function converts the DIB supplied as the info argument into     */
/* a DDB ready for display in the given window. Along the way it         */
/* modifies the global variables hPalette and hDDBitmap. Returns         */
/* TRUE if successful, otherwise FALSE.                                  */
/*------------------------------------------------------------------------*/
BOOL ConvertDIBToDDB (HWND hWnd, BITMAPINFO *info, VOID *bits)
{
    HDC              hDC, hMemDC;
    HPALETTE         hOldPalette;
    HBITMAP          hOldBitmap;
    // Release the old palette.
    if (hPalette != NULL)
    {
        DeleteObject (hPalette);
        hPalette = NULL;
    }

    // Release the old bitmap.
    if (hDDBitmap != NULL)
    {
        DeleteObject (hDDBitmap);
        hDDBitmap = NULL;
    }

    /* Create a new palette with CreateDIBPalette (above), then
       create a bitmap in the device context with CreateCompatibleBitmap
       and SetDIBits. By selecting the palette into the device
       context, we ensure that the colors are correct. */
    hDC = GetDC (hWnd);
    if ( (hPalette = CreateDIBPalette (info)) != NULL)
    {
        if ( (hMemDC = CreateCompatibleDC (hDC)) != NULL)
        {
            hOldPalette = SelectPalette (hMemDC, hPalette, FALSE);
            RealizePalette (hMemDC);
            if ( (hDDBitmap = CreateCompatibleBitmap (hDC,
                                  info->bmiHeader.biWidth,
                                  info->bmiHeader.biHeight)) != NULL)
            {
                hOldBitmap = SelectObject (hMemDC, hDDBitmap);
```

```
            // Copy the bitmap data to the new bitmap (hDDBitmap).
            SetDIBits (hMemDC, hDDBitmap, 0,
                       info->bmiHeader.biHeight,
                       bits, info, DIB_RGB_COLORS);
            SelectObject (hMemDC, hOldBitmap);
        }
        SelectPalette (hMemDC, hOldPalette, FALSE);
        DeleteDC (hMemDC);
    }
    ReleaseDC (hWnd, hDC);

    // If bitmap creation failed, free the palette also.
    if (! hDDBitmap)
    {
        DeleteObject (hPalette);
        hPalette = NULL;
    }
  }
  return (hDDBitmap != NULL);
}
```

9. 在同一源文件中添加下列代码。此函数使用 FindResource 和 LoadResource 来获取位图资源的句柄，接着使用 LockResource 来获取内存中资源的指针。因为本例子程序要修改位图的颜色表，而资源是只读的，所以本段代码将计算颜色表和位图头信息的大小，并为这些信息的拷贝分配内存，例子程序便可以修改拷贝而不需要修改资源了。

```
/*-------------------------------------------------------------------------*/
/* This function opens a bitmap resource attatched to our executable      */
/* file and loads it into memory as a device independant bitmap (DIB)     */
/* The function then locks the resource and makes a copy of it using      */
/* a global memory block of the same size. The function modifies the      */
/* global pointers pResourceDIB and pActiveDIB. It returns TRUE if        */
/* successful, or FALSE if not.                                            */
/*-------------------------------------------------------------------------*/
BOOL LoadBitmapResource (HINSTANCE hInstance, WORD resId)
{
  HRSRC         hResource;
  HGLOBAL       hDib;
  DWORD         nSize, numEntries;

  if ( ( (hResource = FindResource (hInstance,
              MAKEINTRESOURCE (resId), RT_BITMAP)) != NULL) &&
               ( (hDib = LoadResource (hInstance, hResource)) != NULL))
```

```
  {
    pResourceDIB = (LPBITMAPINFO) LockResource (hDib);
    if ( (numEntries = pResourceDIB->bmiHeader.biClrUsed) == 0)
        numEntries = 1L << pResourceDIB->bmiHeader.biBitCount;
    nSize = pResourceDIB->bmiHeader.biSize +
              (numEntries * sizeof (RGBQUAD));
    pActiveDIB = (BITMAPINFO *) LocalAlloc (LMEM_FIXED, nSize);
    if (pActiveDIB != NULL)
     {
        CopyMemory (pActiveDIB, pResourceDIB, nSize);
        pResourceBits = (LPSTR) pResourceDIB + nSize;
        return TRUE;
     }
  }
  return FALSE;
}
```

10. 现在添加绘图函数。函数 Paint 在窗口需要绘制时被调用，将当前的调色板选入窗口的设备描述表并实现调色板，以便绘图时使用这些颜色，接着调用 Windows API 函数 BitBlt 将位图显示在屏幕上。

```
/*------------------------------------------------------------------*/
/* This is the actual drawing function called when WM_PAINT is     */
/* received. Because the conversion from DIB to DBB is done once    */
/* when the bitmap is loaded, this routine is a small and QUICK     */
/* Realization of the palette and BitBlt into the device context.   */
/*------------------------------------------------------------------*/
void Paint (HWND hWnd, HPALETTE hPalette, HBITMAP hBitmap,
         int BitmapWidth, int BitmapHeight)
{
  PAINTSTRUCT          ps;
  HDC                  hDC, hMemDC;
  HPALETTE             hOldPalette;
  HBITMAP              hOldBitmap;

  hDC = BeginPaint (hWnd, &ps);
  if ( (hMemDC = CreateCompatibleDC (hDC)) != NULL)
   {
     hOldPalette = SelectPalette (hDC, hPalette, FALSE);
     RealizePalette (hDC);
     hOldBitmap = SelectObject (hMemDC, hBitmap);
     BitBlt (hDC, 0, 0, BitmapWidth, BitmapHeight,
           hMemDC, 0, 0, SRCCOPY);
     SelectBitmap (hMemDC, hOldBitmap);
```

```
        SelectPalette (hDC, hOldPalette, FALSE);
        DeleteDC (hMemDC);
    }
    EndPaint (hWnd, &ps);
}
```

11. 添加函数 ChangeColor。此函数遍历由 BITMAPINFO 结构传送来的颜色表，找出所有的 RGB 值为 fromColor 的项，用 RGB 值 toColor 来替代。

```
/*------------------------------------------------------------------*/
/* This function works on the color table for a DIB. It changes all */
/* the entries in the table for fromColor to toColor. Do not use     */
/* this function on the DIB in the resource itself, as resources are */
/* read-only. Use a copy of the resource DIB.                        */
/*------------------------------------------------------------------*/
void ChangeColor (BITMAPINFO *info, RGBQUAD fromColor, RGBQUAD toColor)
{
    DWORD           i, numEntries;

    if ( (numEntries = info->bmiHeader.biClrUsed) == 0)
        numEntries = 1L << info->bmiHeader.biBitCount;

    for (i = 0; i < numEntries; i++)
        if ( (info->bmiColors [i] .rgbRed == fromColor.rgbRed) &&
             (info->bmiColors [i] .rgbGreen == fromColor.rgbGreen) &&
             (info->bmiColors [i] .rgbBlue == fromColor.rgbBlue))
        {
            info->bmiColors [i] .rgbRed = toColor.rgbRed;
            info->bmiColors [i] .rgbGreen = toColor.rgbGreen;
            info->bmiColors [i] .rgbBlue = toColor.rgbBlue;
        }
}
```

12. 添加到源文件中的下个函数为 HandleChange。当用户从菜单 File 中选择了菜单项 Change Colors 后，此函数将被调用。如果当前显示的是来自文件的原始位图，则调用函数 ChangeColor，此函数将调色板中的绿色改变成黑色，蓝色改变成红色。如果显示的是改变后的位图，此时变量 isOriginal 为 FALSE，则将从资源中再拷回颜色表，恢复原来的颜色。无论是哪一种情况，都将调用函数 ConvertDIBToDDB 来重新创建逻辑调色板和设备无关位图，接着调用函数 InvalidateRect 来更新屏幕显示。

```
/*------------------------------------------------------------------*/
/* This function responds to the CM_CHANGE message. If the flag     */
/* isOriginal is TRUE, the bitmap is the same as the resource. Two  */
/* colors are changed and the DDB recreated from the DIB. If the    */
/* flag is FALSE, this has already been done, so the original color */
/* information is copied from the resource to restore the colors.   */
```

```
/* In either case the DDB is then recreated.                                */
/* ----------------------------------------------------------------------- */
void HandleChange (HWND hWnd, BOOL isOriginal)
{
    DWORD numEntries;

    if (isOriginal)
     {
      ChangeColor (pActiveDIB, colorBlue, colorRed);
      ChangeColor (pActiveDIB, colorGreen, colorBlack);
     }
    else
     {
        if ( (numEntries = pResourceDIB->bmiHeader.biClrUsed) == 0)
            numEntries = 1L << pResourceDIB->bmiHeader.biBitCount;
        CopyMemory (pActiveDIB, pResourceDIB,
                    pResourceDIB->bmiHeader.biSize +
                    (numEntries * sizeof (RGBQUAD)));
     }
    ConvertDIBToDDB (hWnd, pActiveDIB, pResourceBits);
    InvalidateRect (hWnd, NULL, FALSE);
}
```

13. 在源文件中添加下列代码。函数 MainWndProc 是一个回调函数，用来响应传送给窗口的消息。为了响应消息 WM_CREATE，此消息在窗口第一次创建时发送，此函数装入原始位图并准备好显示。为了响应命令消息 CM_CHANGE，此函数调用函数 HandleChange 来修改位图的颜色表并重新显示位图，同时触发标志 isOrginal 的值，使得再选择菜单项可以撤消前面的改变。

```
/* ----------------------------------------------------------------------- */
/* This function is the main window callback function which will be        */
/* called by Windows to process messages for our window.                   */
/* ----------------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{

    switch (message)
     {
        case WM_CREATE:
            if (LoadBitmapResource (hInstance, IDBM_SAMPLEBITMAP))
             {
                ConvertDIBToDDB (hWnd, pActiveDIB, pResourceBits);
                InvalidateRect (hWnd, NULL, FALSE);
```

```
            isOriginal = TRUE;
        }
        break;
    case WM_COMMAND:
        switch (wParam)
        {
            case CM_CHANGE:
                HandleChange (hWnd, isOriginal);
                isOriginal = ! isOriginal;
                break;
            case CM_EXIT:
                DestroyWindow (hWnd);
                break;
            default:
                return (DefWindowProc (hWnd, message, wParam, lParam));
        }
        break;
    case WM_PALETTECHANGED:
        if ( ( (HWND) wParam ! = hWnd) && (hDDBitmap ! = NULL))
            InvalidateRect (hWnd, NULL, TRUE); /* force a repaint */
        break;
    case WM_PAINT:
        if (hDDBitmap ! = NULL)
        {
            Paint (hWnd, hPalette, hDDBitmap,
                pActiveDIB->bmiHeader.biWidth,
                pActiveDIB->bmiHeader.biHeight);
        }
        else
            return (DefWindowProc (hWnd, message, wParam, lParam));
        break;
    case WM_DESTROY:
        if (hPalette)
            DeleteObject (hPalette);
        if (hDDBitmap)
            DeleteObject (hDDBitmap);
        if (pActiveDIB)
            LocalFree ( (HLOCAL) pActiveDIB);
        PostQuitMessage (0);
        break;
    default:
        return DefWindowProc (hWnd, message, wParam, lParam);
}
```

```
    return 0;
}
```

14. 在源文件中添加函数 InitApplication。当例子程序运行第一个实例时将调用此函数，接着只有在窗口类不再存在时才调用此函数，此函数用来为例子程序的主窗口初始化并登记窗口类。

```
/*---------------------------------------------------------------------*/
/* This function initialises a WNDCLASS structure and uses it to       */
/* register a class for our main window.                               */
/*---------------------------------------------------------------------*/
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName = MAKEINTRESOURCE (IDM_BITMAPMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}
```

15. 添加函数 InitInstance。例子程序的每个实例运行时都将调用此函数，此函数创建并显示例子程序的主窗口。

```
/*---------------------------------------------------------------------*/
/* This function creates an instance of our main window. The window    */
/* is given a class name and a title, and told to display anywhere.    */
/* the nCmdShow argument passed to the program determines how the      */
/* window will be displayed.                                           */
/*---------------------------------------------------------------------*/
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst; /* Store in global variable */

hWnd = CreateWindow (MainWindowClassName," Changing the Palette",
```

```
                    WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);            /* Send a WM_PAINT message */
    return TRUE;
}
```

16. 最后添加的代码是函数 WinMain。此函数为 Windows 应用程序的入口点，用来初始化窗口类和窗口，接着循环处理消息直到例子程序结束。

```
/*--------------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if     */
/* the main window class name has already been registered, if it has */
/* not, call InitApplication to register it. Call InitInstance to     */
/* create an instance of our main window, then pump messages until    */
/* the application is closed.                                          */
/*--------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return (FALSE);
    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

17. 编译并运行此例子程序。

注释

本例子程序遍历颜色表，找出所有匹配的项并改变颜色值，如果知道颜色表中的每一颜色的确切位置，也可以直接改变颜色表中的颜色值。应该注意每次设备无关位图的颜色表被

修改，都应创建新的设备相关位图。

4.3 旋转位图

问题

有时需要旋转显示的位图。可以为应用程序准备许多不同旋转角度的位图资源，以便在需要旋转位图时使用，但这种方法是不可取的。是否还有其他的方法来实现此功能呢？

方法

没有用来旋转位图的 Windows API 函数，但是通过修改设备无关位图的数据可以简单地实现以 90 度为单位的位图旋转。在本节的例子程序中将实现此功能，同时也很好的示范了如何直接操作 DIB。

当需要旋转位图时必须为 DIB 资源作拷贝（因为资源是只读的），然后才能操作位图数据，当需要旋转的是 8 位位图（256 色）时，可以逐行来处理位图，每次拷贝一个字节。有几个非常重要的技巧需要注意：

• DIB 的每个扫描行都必须填补到 32 位的 DWORD 边界。通过检查信息头中的元素 biWidth，来确定每行象素的总位数是否是 32 位的倍数，如果不是，那么在计算源位图和目的位图的字节偏移量时，需要考虑到填补的位数。

• 当旋转位图 90 度或 270 度时，位图的高度成为宽度，反之亦然。尽管为位图分配的内存不需要改变，但需要计算填补数，以填补高度到 DWORD 边界。填补高度是因为在新位图中高度成为宽度。

• 设备无关位图存储图象从左到右，从下到上，因此储存的位图实际上是显示的图象的倒置，如图 4-4 所示，在编写直接操作位图的算法时便需要考虑到这一点。位图数据顺时针旋转 90 实际上是位图逆时针旋转 90，或者说位图顺时针旋转 270。

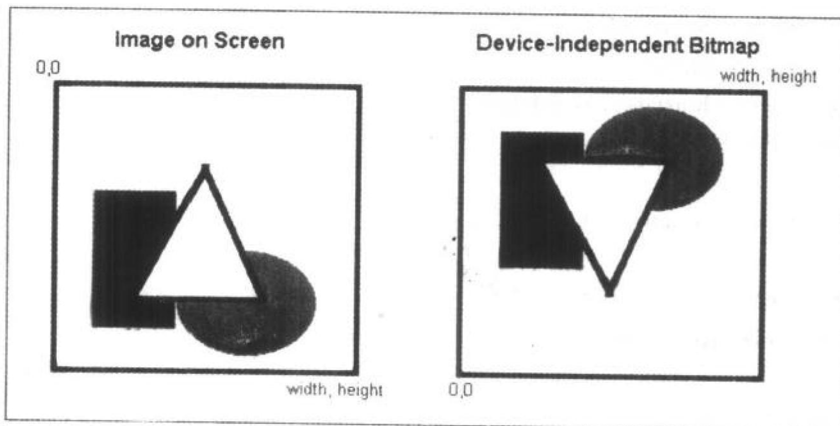


图 4-4 设备无关位图储存图象从下到上

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将打开一个窗口显示出一个小的位图，从菜单 Rotate 中选择菜单项 Normal，例子程序将从资源中装入位图并创建和显示位图，

从菜单 Rotate 中选择菜单项 Rotate 90，位图将顺时针旋转 90 度，如图 4-5 所示，选择菜单项 Rotate 180 将使位图上下倒置，而选择菜单项 Rotate 270 将使位图再旋转 90 度，即顺时针旋转 270 度，或者逆时针旋转 90 度。

在本例子程序中，使用了三个不同的函数来完成位图的旋转，尽管三个函数的框架基本上是一致的。下面来示范如何实现此例子程序。

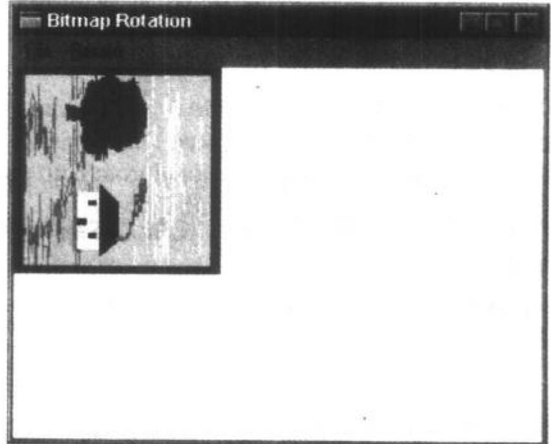


图 4-5　90 度的旋转

实现例子程序的具体步骤如下：

1. 创建新的目录 ROTATE，用来存放例子程序的所有文件。

2. 创建用来旋转的位图。使用资源编辑器创建一个 256 色的位图（因为例子程序假设位图为 256 色），大小大约为 128×128 象素，存为文件 SAMPLE. BMP。因为有的绘图软件的 RLE (run length encoding) 压缩不能同显示驱动程序兼容，所以储存文件时不要使用压缩。

3. 创建例子程序的图标。使用资源编辑器创建一个 32×32 象素 16 色的位图，并存为文件 ROTATE. ICO。

4. 创建资源文件 ROTATE. RC。添加下列资源脚本行，定义了用在例子程序中的菜单、位图和图标。

```
/* ---------------------------------------------------------------- */
/*                                                                  */
/* MODULE: ROTATE.RC                                                */
/* PURPOSE: This resource script defines the menu and application  */
/*          icon associated with this application, and also includes */
/*          a bitmap to be loaded and displayed.                    */
/*                                                                  */
/* ---------------------------------------------------------------- */
#include " rotate.rh"

IDM_BITMAPMENU MENU
{
    POPUP " &File"
    {
        MENUITEM " E&xit", CM_EXIT
    }
    POPUP " &Rotate"
    {
        MENUITEM " &Normal" CM_NORMAL CHECKED
        MENUITEM " Rotate &90" CM_90DEG
```

```
            MENUITEM " Rotate &180" CM _ 180DEG
            MENUITEM " Rotate &270" CM _ 270DEG
        }
}

IDBM _ SAMPLEBITMAP BITMAP " sample.bmp"
IDI _ APPICON ICON " rotate.ico"
```

5. 在第4步的资源脚本中包含了文件 ROTATE.RH，此文件也包含在例子程序的源文件中，定义了资源的标识符。使用文本编辑器创建新文件 ROTATE.RH，并在此文件中添加下列定义：

```
#ifndef __ROTATE _ RH
/*---------------------------------------------------------------------*/
/*                                                                     */
/* MODULE: ROTATE.RH                                                   */
/* PURPOSE: Defines identifiers used to address resources.            */
/*                                                                     */
/*---------------------------------------------------------------------*/
#define __ROTATE _ RH

#define IDM _ BITMAPMENU              301
#define CM _ NORMAL                   111
#define CM _ 90DEG                    112
#define CM _ 180DEG                   113
#define CM _ 270DEG                   114
#define CM _ EXIT                     101

#define IDI _ APPICON                 1
#define IDBM _ SAMPLEBITMAP           3567

#endif
```

6. 创建此例子程序的源文件。使用文本编辑器创建文件 ROTATE.C，在文件的顶部添加下列代码段，此段代码包含了所有需要的头文件，定义了预处理器符号 STRICT，此符号表示对 Windows 操作系统使用的各种类型和句柄进行严格的类型检查。

另外还定义了例子程序主窗口的类名和所有的全局变量，每个全局变量都初始化为缺省值，以免在程序代码中出现错误。

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/* MODULE: ROTATE.C                                                    */
/* PURPOSE: This small application demonstrates how to manipulate     */
/*          a device independant bitmap to rotate or flip an image.   */
/*                                                                     */
```

```
/*---------------------------------------------------------------*/
#define STRICT

#include <windows.h>
#include <windowsx.h>
#include " rotate.rh"

// Constants .
char *MainWindowClassName = " RotateDemoWindow";

// Instance variables .
HBITMAP          hDDBitmap = NULL;
HPALETTE         hPalette = NULL;
BITMAPINFO       *pResourceDIB = NULL;
WPARAM           currentRotation = CM _NORMAL;
HINSTANCE        hInstance = NULL;
DWORD            nResourceSize = 0;
```

7. 在源文件中添加函数 CreateDIBPalette。此函数使用设备无关位图中的颜色表来创建逻辑调色板，调色板可以在设备描述表中被实现，从而可以用于后面的操作中，此函数确保了显示的位图是其原来的颜色。指向信息头(BITMAPINFO 结构的)的指针将传送给此函数，如果创建调色板成功，则返回值为调色板的句柄，如果失败，则返回值为NULL。

```
/*---------------------------------------------------------------*/
/* This function creates a logical palette structure from the color      */
/* table supplied with our device independant bitmap. The palette is     */
/* saved into the global hPalette variable for use when converting       */
/* and displaying the image.                                             */
/*---------------------------------------------------------------*/
HPALETTE CreateDIBPalette (BITMAPINFO *info)
{
    LOGPALETTE          *palPtr;
    DWORD               i, numEntries;
    HPALETTE            hPal;

    /* Allocate space for a LOGPALETTE structure and array of
       entries. Using LocalAlloc with the LMEM _FIXED argument
       gives us a pointer without having to lock and unlock it. */
    if ( (numEntries = info->bmiHeader. biClrUsed) == 0)
        numEntries = 1L << info->bmiHeader. biBitCount;
palPtr = (NPLOGPALETTE) LocalAlloc (LMEM _FIXED, sizeof (LOGPALETTE) +
                                    numEntries * sizeof (PALETTEENTRY));
    if (! palPtr)
        return NULL;
```

```
    palPtr->palVersion = 0x300;
    palPtr->palNumEntries = (WORD) numEntries;

    // Now fill in the array of palette entries from the color table.
    for (i = 0; i < numEntries; i++)
     {
        palPtr->palPalEntry [i] .peRed = info->bmiColors [i] .rgbRed;
        palPtr->palPalEntry [i] .peGreen = info->bmiColors [i] .rgbGreen;
        palPtr->palPalEntry [i] .peBlue = info->bmiColors [i] .rgbBlue;
        palPtr->palPalEntry [i] .peFlags = 0;
    }

    hPal = CreatePalette (palPtr);
    LocalFree ( (HLOCAL) palPtr);
    return hPal;
}
```

8. 在源文件中添加函数 ConvertDIBToDDB。此函数调用 CreateDIBPalette 来创建调色板，接着将调色板选入内存设备描述表并将其实现，然后在设备描述表中创建新的（设备相关）位图，调用函数 SetDIBBits 从设备无关位图中拷贝数据到新的设备相关位图中。

```
/* ---------------------------------------------------------------------- */
/* This function converts the DIB supplied as the info argument into      */
/* a DDB ready for display in the given window. Along the way it          */
/* modifies the global variables hPalette and hDDBitmap. Returns          */
/* TRUE if successful, otherwise FALSE.                                   */
/* ---------------------------------------------------------------------- */
BOOL ConvertDIBToDDB (HWND hWnd, BITMAPINFO *info)
{
    HDC                 hDC, hMemDC;
    HPALETTE            hOldPalette;
    HBITMAP             hOldBitmap;
    DWORD               numEntries, nSize;
    VOID                *bits;

    // Free the old palette and bitmap.
    if (hPalette != NULL)
     {
        DeleteObject (hPalette);
        hPalette = NULL;
    }
    if (hDDBitmap != NULL)
     {
        DeleteObject (hDDBitmap);
```

```
        hDDBitmap = NULL;
    }
    /* Create a new palette, then create a temporary device context
       and select the palette into it. Create a bitmap in the device
       context, and use SetDIBits to convert the DIB into this DDB. */
    hDC = GetDC (hWnd);
    if ( (hPalette = CreateDIBPalette (info)) ! = NULL)
     {
        if ( (hMemDC = CreateCompatibleDC (hDC)) ! = NULL)
         {
            hOldPalette = SelectPalette (hMemDC, hPalette, FALSE);
            RealizePalette (hMemDC);
            if ( (hDDBitmap = CreateCompatibleBitmap (hDC,
                                    info->bmiHeader.biWidth,
                                    info->bmiHeader.biHeight)) ! = NULL)
             {
                if ( (numEntries = info->bmiHeader.biClrUsed) == 0)
                    numEntries = 1 << info->bmiHeader.biBitCount;
                    nSize = info->bmiHeader.biSize + (numEntries * sizeof (RGBQUAD));
                bits = (LPSTR) info + nSize;
                hOldBitmap = SelectObject (hMemDC, hDDBitmap);

                // Copy the data into the new bitmap (hDDBitmap) .
                SetDIBits (hMemDC, hDDBitmap, 0,
                          info->bmiHeader.biHeight,
                          bits, info, DIB_RGB_COLORS);
                SelectObject (hMemDC, hOldBitmap);
            }
            SelectPalette (hMemDC, hOldPalette, FALSE);
            DeleteDC (hMemDC);
        }
        ReleaseDC (hWnd, hDC);

        // If bitmap creation failed, free the palette.
        if (! hDDBitmap)
         {
            DeleteObject (hPalette);
            hPalette = NULL;
        }
    }
    return (hDDBitmap ! = NULL);
}
```

9. 在操作位图前必须先将位图装入内存，所以添加函数 LoadBitmapResource。此函数调

用 API 函数 FindResource、LoadResource 以及 LockResource 来获取内存中的位图指针，设置两个全局变量：变量 nResourceSize 用来存放位图的大小，变量 pResourceDIB 用来存放指向位图的指针。

```
/*-----------------------------------------------------------------*/
/* This function opens a bitmap resource attatched to our executable   */
/* file and loads it into memory as a device independant bitmap (DIB)  */
/*-----------------------------------------------------------------*/
BOOL LoadBitmapResource (HINSTANCE hInstance, WORD resId)
{
    HRSRC      hResource;
    HGLOBAL hDib;

    if ( ( (hResource = FindResource (hInstance,
                                   MAKEINTRESOURCE (resId),
                                   RT_BITMAP)) ! = NULL) &&
         ( (hDib = LoadResource (hInstance, hResource)) ! = NULL))
     {
        nResourceSize = SizeofResource (hInstance, hResource);
        pResourceDIB = (LPBITMAPINFO) LockResource (hDib);
        return (pResourceDIB ! = NULL);
    }
    return FALSE;
}
```

10. 在同一源文件中添加函数 Paint。每当窗口需要刷新时都将调用函数 Paint，此函数创建一个设备描述表，选入调色板并将其实现，接着调用 API 函数 BitBlt 将位图显示出来。

```
/*-----------------------------------------------------------------*/
/* This is the actual drawing function called when WM_PAINT is     */
/* received. It creates a compatible DC, and selects the Device     */
/* Dependant bitmap into it. It then uses BitBlt to copy this to the */
/* device context.                                                    */
/*-----------------------------------------------------------------*/
void Paint (HWND hWnd, HPALETTE hPalette, HBITMAP hBitmap,
        BITMAPINFO *info)
{
    PAINTSTRUCT                ps;
    HDC                        hDC, hMemDC;
    HPALETTE                   hOldPalette;
    HBITMAP                    hOldBitmap;

    hDC = BeginPaint (hWnd, &ps);
    if ( (hMemDC = CreateCompatibleDC (hDC)) ! = NULL)
     {
```

```
        hOldPalette = SelectPalette (hDC, hPalette, FALSE);
        RealizePalette (hDC);
        hOldBitmap = SelectObject (hMemDC, hBitmap);
        BitBlt (hDC, 0, 0, info->bmiHeader.biWidth,
              info->bmiHeader.biHeight, hMemDC, 0, 0, SRCCOPY);
        SelectBitmap (hMemDC, hOldBitmap);
        SelectPalette (hDC, hOldPalette, FALSE);
        DeleteDC (hMemDC);
    }
    EndPaint (hWnd, &ps);
}
```

11. 在完成了所有的准备工作后，下面将开始实现对设备无关位图的操作。函数 Rotate90Degrees 首先将分配一块内存空间，用来拷贝位图资源，由于资源是只读的，所以不能直接进行修改，任何修改资源的企图都将产生保护错误，使得应用程序失败。

一旦为新位图分配了内存，此函数将从位图资源中拷贝信息头和颜色表到新位图中，因为位图旋转了 90 度，所以交换了信息头中的高度值和宽度值。接着计算填补数，即位图中的每条扫描线末没有用到的位的数目，每条扫描线都开始于 32 位的分界处，所以应该清楚扫描线的宽度和下条扫描线的开始处的区别。函数 Rotate90Degrees 计算两个值：padWidth 为添加在资源位图中扫描线末的字节数，用来取出下一条扫描线；padHeight 为添加在目的位图中扫描线末的字节数，也是用来确定下一条扫描线。在正方形的位图中，字节数将是相同的，但在矩形的位图中，字节将是不同的。

最后遍历资源位图中每条扫描线（行），将行中的每个象素变换到目的位图中的新位置。旋转 90 度的变换公式如下：

(NewBitmapHeight x SourceColumn) + SourceRow = (OldBitmapWidth x SourceRow) + SourceColumn

256 色位图中的每一象素是 8 位，即一个字节。

```
/*---------------------------------------------------------------------------*/
/* Create a copy of a device dependant bitmap, and copy the header          */
/* Then copy bits from the source to the destination. Finally,              */
/* use CreateDDBitmap to convert the DIB to a DDB to display.               */
/* Remember that scanlines in a DIB are stored bottom to top. The           */
/* top line of the displayed image is the bottom line if the bitmap         */
/*---------------------------------------------------------------------------*/
void Rotate90Degrees (HWND hWnd, BITMAPINFO *srcInfo)
{
    HGLOBAL                     hGlobal;
    BITMAPINFOHEADER            *destBitmap, *srcBitmap;
    DWORD                       row, col, padHeight, padWidth;
    DWORD                       numEntries, nSize;
    LPSTR                       srcBits, destBits;
```

```
srcBitmap = &srcInfo.bmiHeader;
if ( (hGlobal = GlobalAlloc (GMEM_FIXED, nResourceSize)) ! = NULL)
{
    if ( (destBitmap = (BITMAPINFOHEADER ) GlobalLock (hGlobal)) ! = NULL)
    {
        // Work out how many entries there are in the color table.
        if ( (numEntries = srcBitmap->biClrUsed) == 0)
            numEntries = 1L << srcBitmap->biBitCount;

        /* Now work out the size the color table and header take up,
           and copy from the resource into our memory block.        */
        nSize = srcBitmap->biSize + (numEntries * sizeof (RGBQUAD));
        CopyMemory (destBitmap, srcBitmap, nSize);

        // Swap the width and height members in our destination.
        destBitmap->biHeight = srcBitmap->biWidth;
        destBitmap->biWidth = srcBitmap->biHeight;

        // Calculate offsets to the actual bitmap data.
        srcBits = (LPSTR) srcBitmap + nSize;
        destBits = (LPSTR) destBitmap + nSize;

        /* Work out the extra number of bytes to pad each line to a
           DWORD boundary. For a copy we would only need padWidth,
           but as we rotate, our height will be the new width */
        if ( (padHeight = (srcBitmap->biHeight % sizeof (DWORD))) ! = 0)
            padHeight = sizeof (DWORD) - padHeight;
        if ( (padWidth = (srcBitmap->biWidth % sizeof (DWORD))) ! = 0)
            padWidth = sizeof (DWORD) - padWidth;

        // Now do the actual byte by byte transformation.
        for (row = 0; row < srcBitmap->biHeight; row++)
            for (col = 0; col < srcBitmap->biWidth; col++)
                destBits [ ( (srcBitmap->biHeight + padHeight) * col) + row]
                  = srcBits [ ( (srcBitmap->biWidth + padWidth) * row) + col];
        // Finally create a DDB and tell Windows to update the display.
        ConvertDIBToDDB (hWnd, (BITMAPINFO * ) destBitmap);
        InvalidateRect (hWnd, NULL, TRUE);
        GlobalUnlock (hGlobal);
    }
    GlobalFree (hGlobal);
}
```

12. 下面添加的函数是 Rotate270Degrees。此函数顺时针旋转资源位图 270 度（或逆时针 90 度），从而生成一个新的位图，实现的步骤同前面的旋转 90 度基本相同，只有用来为每个象素计算新位置的公式不同。旋转 270 度的计算公式如下：

((NewBitmapHeight * SourceColumn) + (OldBitmapHeight - SourceRow - 1) = (OldBitmapWidth * SourceRow) + SourceColumn

```
/*-----------------------------------------------------------------------*/
/* Create a copy of a device dependant bitmap, and copy the header       */
/* Then copy bits from the source to the destination. Finally,          */
/* use CreateDDBitmap to convert the DIB to a DDB to display.           */
/* Remember that scanlines in a DIB are stored bottom to top. The       */
/* top line of the displayed image is the bottom line if the bitmap     */
/*-----------------------------------------------------------------------*/
void Rotate270Degrees (HWND hWnd, BITMAPINFO *srcInfo)
{
    HGLOBAL                     hGlobal;
    BITMAPINFOHEADER            *destBitmap *srcBitmap;
    DWORD                       row, col, padHeight, padWidth;
    DWORD                       numEntries, nSize;
    LPSTR                       srcBits, destBits;

    srcBitmap = &srcInfo->bmiHeader;
    if ( (hGlobal = GlobalAlloc (GMEM_FIXED, nResourceSize)) ! = NULL)
     {
       if ( (destBitmap = (BITMAPINFOHEADER *) GlobalLock (hGlobal)) ! = NULL)
        {
          // Work out how many entries there are in the color table.
          if ( (numEntries = srcBitmap->biClrUsed) == 0)
              numEntries = 1L << srcBitmap->biBitCount;

          /* Now work out the size the color table and header take up,
             and copy from the resource into our memory block.        */
          nSize = srcBitmap->biSize + (numEntries * sizeof (RGBQUAD));
          CopyMemory (destBitmap, srcBitmap, nSize);

          // Swap the width and height members in our destination.
          destBitmap->biHeight = srcBitmap->biWidth;
          destBitmap->biWidth = srcBitmap->biHeight;

          // Calculate offsets to the actual bitmap data.
          srcBits = (LPSTR) srcBitmap + nSize;
          destBits = (LPSTR) destBitmap + nSize;

          /* Work out the extra number of bytes to pad each line to a
```

```
        DWORD boundary. For a copy we would only need padWidth,
        but as we rotate, our height will be the new width */
      if ((padHeight = (srcBitmap->biHeight % sizeof (DWORD))) != 0)
          padHeight = sizeof (DWORD) - padHeight;
      if ((padWidth = (srcBitmap->biWidth % sizeof (DWORD))) != 0)
          padWidth = sizeof (DWORD) - padWidth;

      // Now do the actual byte by byte transformation.
      for (row = 0; row < srcBitmap->biHeight; row++)
          for (col = 0; col < srcBitmap->biWidth; col++)
              destBits [((srcBitmap->biHeight + padHeight) * col) +
                      ((srcBitmap->biHeight + padHeight) - row - 1)] =
              srcBits [((srcBitmap->biWidth + padWidth) * row) + col];

      // Finally create a DDB and tell Windows to update the display.
      ConvertDIBToDDB (hWnd, (BITMAPINFO *) destBitmap);
      InvalidateRect (hWnd, NULL, TRUE);

      GlobalUnlock (hGlobal);
    }
    GlobalFree (hGlobal);
  }
}
```

13. 同样可以实现第三个旋转函数，但是有一些细微的区别，因为旋转位图180度，所以在新的信息头中不需要交换宽度值和高度值。同样因为两个位图的填补数相同，所以只需计算一个值padWidth。变换公式如下：

(OldBitmapWidth x (OldBitmapHeight − SourceRow − 1) + (OldBitmapWidth − SourceColumn − 1) = (OldBitmapWidth * SourceRow) + SourceColumn

源位图中的象素将被拷贝到目的位图中的相反行的相反位置。例如：拷贝100×200象素的位图，位置在（3，5）的象素将被拷贝到目的位图中的（95，195）。

```
/*-------------------------------------------------------------*/
/* Create a copy of a device dependant bitmap, and copy the header   */
/* Then copy bits from the source to the destination. Finally,       */
/* use CreateDDBitmap to convert the DIB to a DDB to display.        */
/* Remember that scanlines in a DIB are stored bottom to top. The    */
/* top line of the displayed image is the bottom line of the bitmap  */
/*-------------------------------------------------------------*/
void Rotate180Degrees (HWND hWnd, BITMAPINFO *srcInfo)
{
    HGLOBAL                 hGlobal;
    BITMAPINFOHEADER        *destBitmap, *srcBitmap;
    DWORD                   row, col, padWidth;
```

```
DWORD                              numEntries, nSize;
LPSTR                              srcBits, destBits;

srcBitmap = &srcInfo->bmiHeader;
if ( (hGlobal = GlobalAlloc (GMEM_FIXED, nResourceSize)) != NULL)
 {
   if ( (destBitmap = (BITMAPINFOHEADER *) GlobalLock (hGlobal)) != NULL)
    {
      // Work out how many entries there are in the color table.
      if ( (numEntries = srcBitmap->biClrUsed) == 0)
          numEntries = 1L << srcBitmap->biBitCount;
      /* Now work out the size the color table and header take up,
         and copy from the resource into our memory block.          */
      nSize = srcBitmap->biSize + (numEntries * sizeof (RGBQUAD));
      CopyMemory (destBitmap, srcBitmap, nSize);

      // Swap the width and height members in our destination.
      destBitmap->biHeight = srcBitmap->biWidth;
      destBitmap->biWidth = srcBitmap->biHeight;

      // Calculate offsets to the actual bitmap data.
      srcBits = (LPSTR) srcBitmap + nSize;
      destBits = (LPSTR) destBitmap + nSize;

      /* Work out the extra number of bytes to pad each line to a
         DWORD boundary.    */
      if ( (padWidth = (srcBitmap->biWidth % sizeof (DWORD))) != 0)
          padWidth = sizeof (DWORD) - padWidth;

      // Now do the actual byte by byte transformation.
      for (row = 0; row < srcBitmap->biHeight; row++)
          for (col = 0; col < srcBitmap->biWidth; col++)
              destBits [ ( (srcBitmap->biWidth + padWidth) *
                         (srcBitmap->biHeight - row - 1)) +
                       (srcBitmap->biWidth - col - 1)] =
              srcBits [ ( (srcBitmap->biWidth + padWidth) * row) + col];

      // Finally create a DDB and tell Windows to update the display.
      ConvertDIBToDDB (hWnd, (BITMAPINFO *) destBitmap);
      InvalidateRect (hWnd, NULL, TRUE);

      GlobalUnlock (hGlobal);
   }
```

```
    GlobalFree (hGlobal);
  }
}
```

14. 函数 MainWndProc 是一个回调函数，用来处理传送给例子程序主窗口的消息。为了响应消息 WM _CREATE，此函数从文件中装入位图资源，初始化当前的旋转位置（到正常位置），并显示出位图。此函数也响应来自菜单的消息 WM _COMMAND，为了响应消息 CM _90DEG、CM _180DEG 以及 CM _270DEG，此函数调用适当的函数来旋转位图并刷新窗口。为了响应消息 CM _NORMAL，此函数传送资源位图给函数 ConvertDIBToDDB，从而恢复原来的位图。

```
/*-------------------------------------------------------------------*/
/* This function is the main window callback function which will be  */
/* called by Windows to process messages for our window.              */
/*-------------------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{

  switch (message)
  {
    case WM_CREATE:
      if (LoadBitmapResource (hInstance, IDBM_SAMPLEBITMAP))
      {
        currentRotation = CM_NORMAL;
        ConvertDIBToDDB (hWnd, pResourceDIB);
        InvalidateRect (hWnd, NULL, TRUE);
      }
      break;
    case WM_COMMAND:
      switch (wParam)
      {
        case CM_EXIT:
            DestroyWindow (hWnd);
            break;
        case CM_NORMAL:
        case CM_90DEG:
        case CM_180DEG:
        case CM_270DEG:
            if (currentRotation != wParam)
            {
              CheckMenuItem (GetMenu (hWnd), currentRotation,
                             MF_BYCOMMAND | MF_UNCHECKED);
```

```
                    CheckMenuItem (GetMenu (hWnd), wParam,
                                   MF_BYCOMMAND | MF_CHECKED);
                    currentRotation = wParam;
                    switch (wParam)
                      {
                        case CM_NORMAL:
                            ConvertDIBToDDB (hWnd, pResourceDIB);
                            InvalidateRect (hWnd, NULL, TRUE);
                            break;
                        case CM_90DEG:
                            Rotate90Degrees (hWnd, pResourceDIB);
                            break;
                        case CM_180DEG:
                            Rotate180Degrees (hWnd, pResourceDIB);
                            break;
                        case CM_270DEG:
                            Rotate270Degrees (hWnd, pResourceDIB);
                            break;
                      }
                }
                break;
            default:
                return (DefWindowProc (hWnd, message, wParam, lParam));
        }
        break;
    case WM_PALETTECHANGED:
        // Force a repaint.
        if ( ( (HWND) wParam ! = hWnd) && (hDDBitmap ! = NULL))
            InvalidateRect (hWnd, NULL, TRUE);
        break;
    case WM_PAINT:
        if (hDDBitmap ! = NULL)
            Paint (hWnd, hPalette, hDDBitmap, pResourceDIB);
        else
            return (DefWindowProc (hWnd, message, wParam, lParam));
        break;
    case WM_DESTROY:
        if (hPalette)
            DeleteObject (hPalette);
        if (hDDBitmap)
            DeleteObject (hDDBitmap);
```

```
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

15. 在源文件中添加两个函数。函数 InitApplication 在创建例子程序窗口时登记窗口类，此函数只有在窗口类不再存在时才被调用。相反，函数 InitInstance 在启动例子程序的每个实例时都将被调用，此函数创建并显示例子程序的主窗口。

```
/* ------------------------------------------------------------------ */
/* This function initialises a WNDCLASS structure and uses it to        */
/* register a class for our main window.                                */
/* ------------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName = MAKEINTRESOURCE (IDM_BITMAPMENU);
    wc.lpszClassName = MainWindowClassName;
    return RegisterClass (&wc);
}
/* ------------------------------------------------------------------ */
/* This function creates an instance of our main window. The window     */
/* is given a class name and a title, and told to display anywhere.     */
/* the nCmdShow argument passed to the program determines how           */
/* the window will be displayed.                                        */
/* ------------------------------------------------------------------ */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst; /* Store in global variable */
```

```
    hWnd = CreateWindow (MainWindowClassName," Bitmap Rotation",
                         WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT,

                         CW_USEDEFAULT, NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);        /* Send a WM_PAINT message */
    return TRUE;
}
```

16. 最后添加的是函数 WinMain。此函数是所有 Windows 应用程序的入口点，调用函数 InitApplication 来初始化窗口类，调用函数 InitInstance 来显示主窗口，然后处理消息直到例子程序结束。

```
/*------------------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if        */
/* the main window class name has already been registered, if it has      */
/* not, call InitApplication to register it. Call InitInstance to         */
/* create an instance of our main window, then pump messages until        */
/* the application is closed.                                             */
/*------------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;
    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

17. 编译并运行此例子程序。

注释

本节的例子程序示范了如何旋转设备无关位图，旋转的角度为 90 的倍数，可以扩充此方法，开发各种可能的应用程序。例如：如果开发一个绘图程序，可以使用此方法旋转整个位图，也可以用来旋转某一部分；修改变换公式，也可以旋转位图到任何角度。

本节的例子程序也证明了直接操作设备无关位图是多么的简单，因此可以自己编写例程来生成设备无关位图，而不需要使用 GDI。

4.4 随鼠标拖动绘制“橡皮带线”

问题

有时用户点击并拖动鼠标，希望应用程序能够绘制出一个方框或其他的一些形状，并在鼠标移动时形状也跟着改变。

方法

需要实现的功能通常被称为橡皮带线拖动，在用户移动对象、改变对象的大小以及绘图时，此功能非常实用。可以不断提供反馈给用户，而实现的方法并不困难，关键是 API 函数 SetROP2 的使用。此函数用来控制光栅运算符（ROP），ROP 在绘图到设备描述表时使用，用来定义如何来复合象素的颜色。

表 4-2 列出了常用的光栅运算符。这些运算符类似于 C 语言中的位操作运算符，例如：~（位操作的 NOT 运算符），&（位操作的 AND 运算符），以及 |（位操作的 OR 运算符）。

表 4-2 函数 SetROP2 使用的光栅运算符

运算符	助记值	最终象素
R2_BLACK	1	总为 0
R2_WHITE	16	总为 1
R2_NOP	11	保持不变
R2_NOT	6	屏幕的反相颜色
R2_COPYPEN	13	画笔颜色
R2_NOTCOPYPEN	4	画笔的反相颜色
R2_MERGEPOENNOT	14	画笔颜色与屏幕反相颜色的复合
R2_MASKPENNOT	5	画笔和屏幕反相的共同色
R2_MERGENOTPEN	12	屏幕颜色和画笔反相颜色的复合
R2_MASKNOTPEN	3	屏幕和画笔反相的共同色
R2_MERGEPEN	15	画笔颜色和屏幕颜色的复合
R2_NOTMERGEPEN	2	R2_MERGEPEN 的反相颜色
R2_MASKPEN	9	画笔和屏幕的共同色
R2_NOTMASKPEN	8	R2_MASKPEN 的反相颜色
R2_XORPEN	7	画笔和屏幕颜色的复合，但相同时除外
R2_NOTXORPEN	10	R2_XORPEN 的反相颜色

对于实现橡皮带线拖动最有用的运算符是 R2_XORPEN 和 R2_NOTXORPEN，这两

个运算符表示对画笔的颜色和屏幕的颜色进行逻辑 XOR 运算。R2_XORPEN 运算符表示 XOR 画笔的颜色和屏幕的颜色，运算结果便是象素的颜色。而 R2_NOTXORPEN 运算符表示同样的运算，但最后将运算结果取反，位值为 1 置成 0，位值为 0 置成 1。使用 XOR 运算符最大的好处在于：将绘图操作重复一次便可恢复背景的颜色，而根本不需要应用程序来存储背景色。

生成橡皮带线需要的另一技巧是鼠标动作的跟踪。在本节讨论的例子程序中，当开始拖动时，例子程序接收消息 WM_LBUTTONDOWN，而后例子程序通过处理消息 WM_MOUSEMOVE 可以跟踪鼠标移动，并在它接收到消息 WM_LBUTTONUP 后，停止橡皮带线的拖动。

步骤

按照下列步骤实现一个例子程序。此例子程序是一个非常简单的绘图程序，可以使用多种颜色绘制填充的矩形和填充的椭圆，当用户绘制图形时，例子程序将绘制出橡皮带线从而可以帮助用户确定图形的最后形状。

运行此例子程序，从菜单 Tools 中选择菜单项 Rectangle 或 Ellipse，可以绘制矩形或椭圆，从菜单 Tools 中选择菜单项 Colors，可以选取绘制图形的颜色，在窗口中按下鼠标的左键并拖动鼠标，将出现一个矩形或椭圆（即橡皮带线），移动鼠标可以改变橡皮带线的大小和形状，松开鼠标的左键，橡皮带线将被彩色的填充图形所替代。例子程序的运行如图 4-6 所示，用户正在拖动鼠标绘制矩形。

同时，本节的例子程序也示范了如何使用 XOR 光栅运算符来绘图以及如何跟踪鼠标的当前位置。

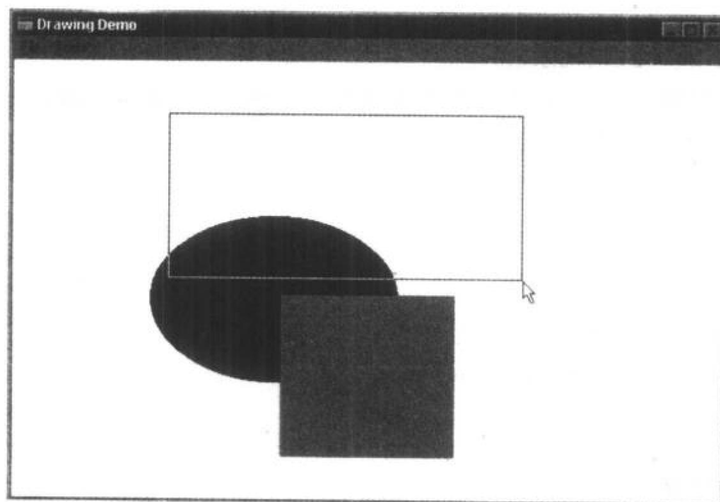


图 4-6 拖动鼠标绘制矩形

实现例子程序的具体步骤如下：

1. 创建目录 RBRBAND，用来存放例子程序的所有文件。

2. 为例子程序准备图标。使用资源编辑器创建一个 32×32 象素 16 色的图标，并存为文件 DRAWDEMO.ICO。

3. 使用文本编辑器创建新文件 DRAWDEMO.RC。在此文件中输入例子程序的资源脚

本，下列文本即为资源脚本，定义了菜单和例子程序的图标。

```
/*----------------------------------------------------------------*/
/*                                                                */
/* MODULE: DRAWDEMO.RC                                            */
/* PURPOSE: This resource script defines the menus and application */
/*          icon associated with this application.                */
/*                                                                */
/*----------------------------------------------------------------*/
#include "drawdemo.rh"

IDM_DRAWMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "E&xit", CM_EXIT
    }
    POPUP "&Tools"
    {
        MENUITEM "&Rectangle", CM_RECTANGLE, CHECKED
        MENUITEM "&Ellipse", CM_ELIPSE
        MENUITEM SEPARATOR
        MENUITEM "&Colors...", CM_COLORS
    }
}
IDI_APPICON ICON "drawdemo.ico"
```

4. 定义资源标识符，以便在资源脚本和例子程序中使用。创建新文件 DRAWDEMO.RH，在此文件中添加下列定义：

```
#ifndef __DRAWDEMO_RH
/*----------------------------------------------------------------*/
/*                                                                */
/* MODULE: DRAWDEMO.RH                                            */
/* PURPOSE: Defines identifiers used to address resources.        */
/*                                                                */
/*----------------------------------------------------------------*/
#define __DRAWDEMO_RH

#define IDM_DRAWMENU        2
#define CM_EXIT             101
#define CM_RECTANGLE        201
#define CM_ELIPSE           202
#define CM_COLORS           203
```

```
#define IDI_APPICON           1000

#endif
```

5. 现在开始编写例子程序的源代码。使用文本编辑器创建新文件 DRAWDEMO.C，在文件的顶部添加下列代码段，代码段定义了预处理器符号 STRICT，表示对在 Windows 头文件中定义的句柄和其他的类型进行严格的类型检查，还包含了所有需要的头文件，定义了所有的全局变量并进行了初始化。

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* MODULE: DRAWDEMO                                                 */
/* PURPOSE: This demonstration program uses the SetROP2             */
/*          API function in combination with a number of rectangle */
/*          and drawing functions to demonstrate rubber-banding.    */
/*                                                                  */
/*------------------------------------------------------------------*/
#define STRICT

#include <windows.h>
#include <commdlg.h>
#include <stdlib.h>
#include "drawdemo.rh"
// Constants
char *MainWindowClassName = "DrawDemoWindow";
// Instance variables.
HINSTANCE          hInstance = NULL;
COLORREF           customColors [16];
COLORREF           currentColor = 0;
BOOL               elipseOn = FALSE;
BOOL               isDrawing = FALSE;
RECT               currentRect;
POINT              startingPoint;
```

6. 在源文件中添加函数 ChangeCurrentColor。此函数显示出一个选取颜色通用对话框，用户用来选取绘图的颜色，在此函数中，初始化一个 CHOOSECOLOR 结构，接着调用 API 函数 ChooseColor，选取的颜色存放在全局变量 CurrentColor 中，全局变量 customColors 最多可以存放 16 种颜色，用来保存用户定义的颜色。

```
/*------------------------------------------------------------------*/
/* This function uses the ChooseColor common dialog box to let the  */
/* user select the drawing color. The currentColor variable is      */
/* modified if the user selects OK. The customColors array is       */
/* modified if the user defines any custom colors.                  */
/*------------------------------------------------------------------*/
void ChangeCurrentColor (HWND hWnd)
```

```
{
    CHOOSECOLOR cc;
    cc.lStructSize = sizeof (CHOOSECOLOR);
    cc.hwndOwner = hWnd;
    cc.hInstance = NULL; // No template specified, so unused.
    cc.rgbResult = currentColor;
    cc.lpCustColors = customColors;
    cc.Flags = CC_RGBINIT;
    if (ChooseColor (&cc))
        currentColor = cc.rgbResult;
}
```

7. 下面添加的函数是 MouseMove。每次处理消息 WM_MOUSEMOVE 时都将调用此函数，参数为鼠标的当前坐标，也就是此函数实现了橡皮带线的拖动。

此函数首先调用函数 GetDC 来获取窗口设备描述表的句柄，因为例子程序要直接在窗口中绘图，而不使用函数 paint。注意，使用函数 GetDC 获取到的设备描述表句柄应该使用函数 ReleaseDC 来进行释放，在此函数的最后释放了设备描述表句柄。

接着此函数设置光栅运算符，来控制如何绘图。调用函数 SetROP2，参数为 R2_NOTXORPEN，设置绘图模式为 XOR，后面的 GDI 函数将使用此模式。此函数将函数 SetROP2 的返回值存放在变量 oldROPCode 中，以便在函数的最后再调用函数 SetROP2 来恢复原来的光栅运算模式。

最后此函数利用鼠标的起始点以及当前坐标来生成一个矩形。因为矩形右下角的坐标值必须大于左上角的坐标值，所以如果用户拖动鼠标的方向相反，那么就必须交换坐标值，这样即使用户拖动鼠标到了起始点的左边或上边，程序仍能继续运行。一旦计算出矩形，便可调用 GDI 函数在窗口中绘制矩形或椭圆。由于设置了 R2_NOTXORPEN 模式，图形的轮廓线将同橡皮带线反相。

```
/*------------------------------------------------------------------*/
/* This function does the rubber banding. It tracks the current      */
/* position and the original position using currentRect. The key to */
/* drawing the rubber band is the XOR operator set with SetROP2.     */
/* This operator lets us get the restore the background by simply    */
/* redrawing the shape at the same position.                         */
/*------------------------------------------------------------------*/
void MouseMove (HWND hWnd, LONG mouseX, LONG mouseY)
{
    int oldROPCode;
    HDC hdc;

    // Draw straight into the device context.
    hdc = GetDC (hWnd);

    // Set the XOR operator to draw or erase.
```

```
oldROPCode = SetROP2 (hdc, R2_NOTXORPEN);

// Erase any old shape from the last move.
if (! IsRectEmpty (&currentRect))
    if (elipseOn)
        Ellipse (hdc, currentRect.left, currentRect.top,
                currentRect.right, currentRect.bottom);
    else
        Rectangle (hdc, currentRect.left, currentRect.top,
                currentRect.right, currentRect.bottom);
/* Assign the new coordinates to the rectangle. Note that we
   need to swap coordinates if the user moves the mouse to the
   left of or above the original starting position */
if (mouseX < startingPoint.x)
{
    currentRect.left = mouseX;
    currentRect.right = startingPoint.x;
}
else
{
    currentRect.left = startingPoint.x;
    currentRect.right = mouseX;
}
if (mouseY < startingPoint.y)
{
    currentRect.top = mouseY;
    currentRect.bottom = startingPoint.y;
}
else
{
    currentRect.top = startingPoint.y;
    currentRect.bottom = mouseY;
}
// Now draw the new rectangle or elipse.
if (elipseOn)
    Ellipse (hdc, currentRect.left, currentRect.top,
            currentRect.right, currentRect.bottom);
else
    Rectangle (hdc, currentRect.left, currentRect.top,
            currentRect.right, currentRect.bottom);

SetROP2 (hdc, oldROPCode);
ReleaseDC (hWnd, hdc);
```

```
}
```

8. 在同一源文件中添加函数 StartDrawing。当用户在用户区中按下鼠标左键时此函数将被调用，首先调用 API 函数 ClipCursor 来限制光标的拖动只能在用户区内，从而简化了鼠标管理和绘图例程。在商业应用程序中，可能不愿意把光标限制在屏幕的某一部分，如果不使用函数 ClipCursor，那么开发应用程序将需要考虑窗口是否正在激活中，应用程序挂起时是否暂停或取消当前的绘制。

使用下面的代码，在用户按下鼠标键时光标将被限制在窗口的用户区，一旦松开鼠标键，光标将可以继续任意移动。在调用函数 ClipCursor 前，此函数调用函数 GetClientRect 取回用户区的坐标，接着调用函数 ClientToScreen 将用户坐标转换为屏幕坐标，函数 ClipCursor 使用屏幕坐标。函数 Start Drawing 还储存鼠标的初始 X 值和 Y 值，用于橡皮带线拖动中以及用来重置橡皮带线。

```
/*---------------------------------------------------------------------*/
/* This function stores the starting point and starts drawing.        */
/*---------------------------------------------------------------------*/
void StartDrawing (HWND hWnd, LONG mouseX, LONG mouseY)
{
    RECT   clientRectangle;
    POINT point;
    /* Get the client rectangle, and convert it to screen
       coordinates. Convert top and left, then bottom and right. */
    GetClientRect (hWnd, &clientRectangle);
    point.x = clientRectangle.left;
    point.y = clientRectangle.top;
    ClientToScreen (hWnd, &point);
    clientRectangle.left = point.x;
    clientRectangle.top = point.y;
    point.x = clientRectangle.right;
    point.y = clientRectangle.bottom;
    ClientToScreen (hWnd, &point);
    clientRectangle.right = point.x;
    clientRectangle.bottom = point.y;

    // Keep the mouse inside the client area.
    ClipCursor (&clientRectangle);

    /* Save the coordinates of the mouse cursor. Because the
       rectangle is initially empty, set the right and bottom
       to be the same as left and top. */
    startingPoint.x = mouseX;
    startingPoint.y = mouseY;
```

```
    SetRectEmpty (&currentRect);
}
```

9. 在鼠标键被按下时调用前面的函数，而在鼠标键被松开时将调用下面的函数 FinishDrawing。当此函数被调用时，全局变量 currentRect 中已经有了矩形或椭圆形的坐标值，此函数将以选取的颜色为参数创建实体画刷，并在窗口的设备描述表中绘制填充椭圆形或填充矩形，还将调用函数 ClipCursor，参数为 NULL，用来取消对光标的限制，允许用户继续可以在整个屏幕上移动光标。

```
/*--------------------------------------------------------------------*/
/* This function ends the drawing operation by drawing an elipse or    */
/* a rectangle, stopping the rubber-banding, and releasing the          */
/* cursor to move outside the window.                                   */
/*--------------------------------------------------------------------*/
void FinishDrawing (HWND hWnd)
{
  HDC           hdc;
  HBRUSH        colorBrush, oldBrush;

    /* Draw straight into the device context. */
    hdc = GetDC (hWnd);
    colorBrush = CreateSolidBrush (currentColor);
    oldBrush = SelectObject (hdc, colorBrush);
    if (elipseOn)
        Ellipse (hdc, currentRect.left, currentRect.top,
                 currentRect.right, currentRect.bottom);
    else
        FillRect (hdc, &currentRect, colorBrush);
    SelectObject (hdc, oldBrush);
    DeleteObject (colorBrush);
    ClipCursor (NULL);
}
```

10. 下一个要添加的函数是 MainWndProc，此回调函数用来响应传送给窗口的消息。其中三个消息对于实现橡皮带线和鼠标拖动是至关重要的。当光标在窗口内，用户按下鼠标左键（如果鼠标键已经被交换则是按下鼠标的右键）时，发送消息 WM_LBUTTONDOWN 给例子程序。当鼠标键被松开时发送消息 WM_LBUTTONUP。为了响应这两个消息，此函数分别调用函数 StartDrawing 和 FinishDrawing。第三个消息是 WM_MOUSEMOVE，当鼠标在窗口内移动时发送此消息给窗口。如果用户正在绘图（标志 isDrawing 为 TRUE），则将调用函数 MouseMove 来绘制橡皮带线。

```
/*--------------------------------------------------------------------*/
/* This function is the main window callback function which will be     */
/* called by Windows to process messages for our window.                */
```

```
/*----------------------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    HMENU hMenu;

    hMenu = GetMenu (hWnd);
    switch (message)
     {
        case WM_LBUTTONDOWN:
            StartDrawing (hWnd, (LONG) LOWORD (lParam), (LONG) HIWORD (lParam));
            isDrawing = TRUE;
            break;
        case WM_MOUSEMOVE:
            if (isDrawing)
            {
                MouseMove (hWnd, (LONG) LOWORD (lParam), (LONG) HIWORD (lParam));
                break;
            }
        case WM_LBUTTONUP:
            if (isDrawing)
            {
                FinishDrawing (hWnd);
                isDrawing = FALSE;
                break;
            }
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EXIT:
                     DestroyWindow (hWnd);
                     break;
                case CM_RECTANGLE:
                    CheckMenuItem (hMenu, CM_ELIPSE, MF_BYCOMMAND |
                    MF_UNCHECKED);
                     CheckMenuItem (hMenu, CM_RECTANGLE, MF_BYCOMMAND |
                     MF_CHECKED);
                     elipseOn = FALSE;
                     break;
                case CM_ELIPSE:
                    CheckMenuItem (hMenu, CM_RECTANGLE, MF_BYCOMMAND |
```

```
MF_UNCHECKED);
                    CheckMenuItem (hMenu, CM_ELIPSE,
                                        MF_BYCOMMAND | MF_CHECKED);
                    elipseOn = TRUE;
                    break;
                case CM_COLORS:
                    ChangeCurrentColor (hWnd);
                    break;
            }
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

11. 在同一源文件中添加下列两个函数。函数 InitApplication 用来登记例子程序主窗口的窗口类，此函数只有在窗口类的实例不再存在时才被调用。函数 InitInstance 在例子程序启动时被调用，创建并显示例子程序的主窗口。

```
/*----------------------------------------------------------------------*/
/* This function initialises a WNDCLASS structure and uses it to        */
/* register a class for our main window.                                */
/*----------------------------------------------------------------------*/
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_DRAWMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
```

```
}
/*---------------------------------------------------------------*/
/* This function creates an instance of our main window. The window  */
/* is given a class name and a title, and told to display anywhere.   */
/* the nCmdShow argument passed to the program determines how         */
/* the window will be displayed.                                      */
/*---------------------------------------------------------------*/
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;        // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName," Drawing Demo",
                         WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                         NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);     // Send a WM_PAINT message.
    return TRUE;
}
```

12. 编译并运行此例子程序还需要一个函数。在同一源文件中添加函数 WinMain，此函数是例子程序的入口点，用来建立例子程序，接着处理消息直到例子程序结束。

```
/*---------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if     */
/* the main window class name has already been registered, if it has  */
/* not, call InitApplication to register it. Call InitInstance to     */
/* create an instance of our main window, then pump messages until    */
/* the application is closed.                                         */
/*---------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;
```

```
    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;
    while (GetMessage (&msg, NULL, 0, 0))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
     }
    return (msg.wParam);
}
```

注释

本节的例子程序示范了如何有效地跟踪鼠标，在 4.7 节中还将对此问题进行更加深入的讨论。

如何基于本节的例子程序开发出一个实用的应用程序呢？首先应该扩充的功能是存放用户绘制的图形。在例子程序中不响应消息 WM_PAINT，也不存放用户绘制的图形。如果某个窗口覆盖了例子程序的主窗口，当这个窗口再移开后，前面绘制的所有图形都将消失。要存放用户绘制的图形，可以拷贝用户区到一个位图中，也可以将已经绘制的图形以及图形的大小和颜色存放在一个记录中，当应用程序接收到消息 WM_PAINT 后，便可以将窗口重画。

4.5 捕捉窗口或部分屏幕

问题

有些 Windows 应用程序可以捕捉部分屏幕，并将其存放在剪贴板上或粘贴到某个绘图程序中。此功能是如何实现的呢？

方法

可以使用函数 GetDC、ReleaseDC、CreateDC 以及 DeleteDC 来存取任何窗口的设备描述表，但是需要将窗口的句柄提供给这些函数。而恰好有一些 Windows API 函数可以用来找出窗口的句柄，这些窗口可以是激活的窗口、最前面的窗口以及鼠标刚点击的窗口等等。接着应用程序便可以创建兼容的位图并将其选入内存设备描述表，调用函数 BitBlt 来实现拷贝，从屏幕设备描述表到内存设备描述表，即到位图。

当从屏幕上捕捉位图图象时，还需要存储当前的逻辑调色板，以便在位图存盘或显示时使用调色板信息。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，点击菜单 Capture 可以看到各个菜单项。菜单项 Entire Screen 被缺省选中，表示捕捉整个屏幕；菜单项 Active Window 表示捕捉整个激活的窗口（包括标题、滚动条、极大化按钮和极小化按钮）；菜单项 Active Client Area 表示捕捉激活窗口的用户区，不包括标题、滚动条以及其他的非用户区资源。

从菜单 Capture 中选取菜单项 Start Capture，将弹出一个消息框，通知用户在点击按钮 OK 五秒钟后开始捕捉屏幕。当点击按钮 OK 后，例子程序将极小化五秒钟，五秒钟后开始捕

捉屏幕，然后恢复原来的大小。图 4-7 所示的便是捕捉了整个屏幕后的主窗口。捕捉到的图象要进行缩放以便适应于主窗口。

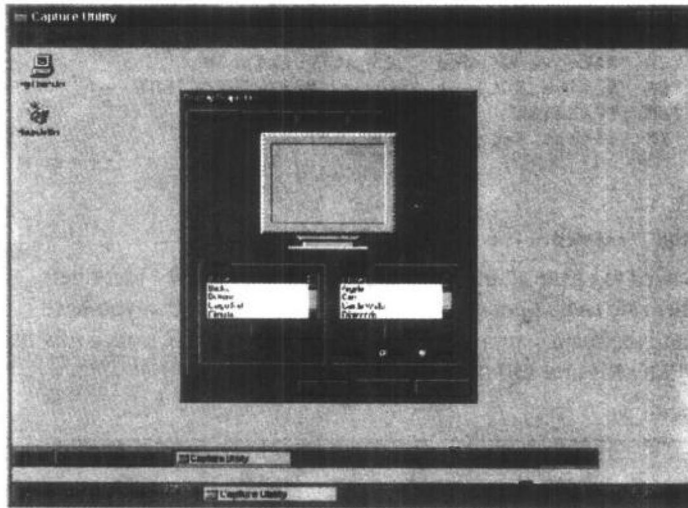


图 4-7 捕捉整个屏幕后的主窗口

实现例子程序的具体步骤如下：

1. 创建工作目录 CAPTURE，用来存放例子程序的所有文件。

2. 为例子程序准备图标。使用资源编辑器创建一个 32×32 象素 16 色的图标，并存为文件 CAPDEMO.ICO。

3. 创建资源脚本，用来包含例子程序的图标并定义例子程序的菜单。通常使用资源编辑器来创建资源脚本，但也可以使用文本编辑器来创建脚本。创建文件 CAPDEMO.RC，并在此文件中输入下列行：

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* MODULE: CAPDEMO.RC                                                */
/* PURPOSE: This resource script defines the menus and application  */
/*          icon associated with this application.                   */
/*                                                                   */
/*-------------------------------------------------------------------*/
#include "capdemo.rh"

IDM_CAPTUREMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "E&xit", CM_EXIT
    }
    POPUP "&Capture"
    {
```

```
        MENUITEM " &Entire Screen", CM _ENTIRESCREEN, CHECKED
        MENUITEM " &Active Window", CM _ACTIVEWINDOW
        MENUITEM " Active &Client Area", CM _ACTIVECLIENT
        MENUITEM SEPARATOR
        MENUITEM " &Start Capture", CM _CAPTURE
    }
}

IDI _APPICON ICON " capdemo.ico"
```

4. 创建资源头文件，用来定义资源的标识符，这些标识符既用在刚创建的资源脚本中，也用在下面将要创建的源文件中。创建新文件 CAPDEMO.RH，并在文件中添加下列代码：

```
#ifndef __CAPDEMO _RH
/*--------------------------------------------------------------------*/
/*                                                                    */
/* MODULE: CAPDEMO.RH                                                 */
/* PURPOSE: Defines identifiers used to address resources.            */
/*                                                                    */
/*--------------------------------------------------------------------*/
#define __CAPDEMO _RH

#define IDM _CAPTUREMENU          2
#define CM _EXIT                  101
#define CM _ENTIRESCREEN          201
#define CM _ACTIVEWINDOW          202
#define CM _ACTIVECLIENT          203
#define CM _CAPTURE               204

#define IDI _APPICON              1000

#endif
```

5. 现在开始实现例子程序的主要部分。使用文本编辑器创建源文件 CAPDEMO.C，在文件的开始输入下列代码段。此段代码包含了编译时所需要的 Windows 头文件，定义了预处理器符号 STRICT，此符号表示对在编程中使用的句柄类型以及其他类型进行严格的类型检查。此段代码还定义了例子程序主窗口的类名以及其他的一些全局变量，并将所有的全局变量初始化为缺省值以避免出现错误。

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* MODULE: CAPDEMO                                                    */
/* PURPOSE: This small application demonstrates how to capture a     */
/*          bitmap image from the screen, the active window, or the   */
/*          client area of the active window. The captured bitmap is  */
/*          scaled and displayed in the application's window.         */
```

```
/*                                                                    */
/*------------------------------------------------------------------*/
#define STRICT

#include <windows.h>
#include " capdemo.rh"

// Constants.
char * MainWindowClassName = " CaptureWindow";

// Instance variables.
HINSTANCE           hInstance = NULL;
HBITMAP             hBitmap = NULL;
HPALETTE            hPalette = NULL;
WORD                tickCounter = 0;
int                 captureOptions = CM_ENTIRESCREEN;
```

6. 在源文件中添加函数 CaptureImage。此函数用来完成屏幕捕捉功能，首先获取屏幕的捕捉区域，接着使用捕捉到的数据创建位图和调色板。下面稍微仔细地分析一下此函数的操作：

此函数的 switch 语句用来获取捕捉区域的屏幕坐标。如果是捕捉整个屏幕，则使用函数 GetDesktopWindows 来获取桌面句柄，接着使用函数 GetWindowRect 来获取定义桌面的矩形。如果是捕捉激活的窗口，则过程类似，使用函数 GetForegroundWindow 来获取窗口的句柄（不能使用函数 GetActiveWindow，因为此函数只返回应用程序自己的激活窗口），接着使用函数 GetWindowRect 获取坐标值。

获取用户区的坐标稍微复杂一些。首先使用函数 GetForegroundWindow 来获取窗口句柄，接着调用函数 GetClientRect 获取坐标值，但是此坐标是用户坐标，是相对于用户区的坐标，用户区左上角的坐标为（0，0）。要转换到屏幕坐标，需要调用函数 ClientToScreen，此函数将用户坐标转换为屏幕坐标。需要调用两次此函数，一次用来转换左上角，一次用来转换右下角。

得到坐标值后，调用函数 CreateDC，驱动器字符串为“DISPLAY”，其余的参数为 NULL。此函数返回屏幕设备描述表的句柄，用于拷贝位图，接着创建内存设备描述表和位图，使用函数 BitBlt 从屏幕拷贝数据到位图。

拷贝调色板是相当简单的。调用函数 GetSystemPaletteEntries 拷贝调色板的表项到内存缓冲区中，缓冲区必须足够大以便能存放得下。可以调用函数 GetDeviceCaps，参数为 SIZEPALETTE，来获取调色板的表项数。

```
/*------------------------------------------------------------------*/
/* This function captures an area of the display into a bitmap and a  */
/* logical palette. It modifies the bitmap handle and palette handle   */
/* passed as arguments. Depending upon the value of option, it         */
/* captures the entire screen (CM_ENTIRESCREEN), the active            */
/* window (CM_ACTIVEWINDOW), or the client area of the                 */
```

```
/* active window (CM_ACTIVECLIENT) .                                         */
/*---------------------------------------------------------------------------*/
void CaptureImage (int options, HBITMAP *bmPtr, HPALETTE *palPtr)
{
    HDC                 hScreenDC, hMemoryDC;
    HWND                hTopWnd;
    LONG                xRes, yRes, width, height;
    RECT                capRect;
    POINT               point;
    HBITMAP             hOldBitmap;
    int                 nColors;
    HGLOBAL             hGlobal;
    LOGPALETTE          *pPalette;
    *bmPtr = NULL;
    *palPtr = NULL;

    // Use a different rectangle depending upon options.
    switch (options)
     {
        case CM_ENTIRESCREEN:
            GetWindowRect (GetDesktopWindow (), &capRect);
            break;
        case CM_ACTIVEWINDOW:
            hTopWnd = GetForegroundWindow ();
            GetWindowRect (hTopWnd, &capRect);
            break;
        case CM_ACTIVECLIENT:
            hTopWnd = GetForegroundWindow ();
            /* Work out the client area. */
            GetClientRect (hTopWnd, &capRect);
            point.x = capRect.left;
            point.y = capRect.top;
            ClientToScreen (hTopWnd, &point);
            capRect.left = point.x;
            capRect.top = point.y;
            point.x = capRect.right;
            point.y = capRect.bottom;
            ClientToScreen (hTopWnd, &point);
            capRect.right = point.x;
            capRect.bottom = point.y;
            break;
     }
```

```
hScreenDC = CreateDC ("DISPLAY", NULL, NULL, NULL);
hMemoryDC = CreateCompatibleDC (hScreenDC);
// Make sure that the capture area is visible.
xRes = GetDeviceCaps (hScreenDC, HORZRES);
yRes = GetDeviceCaps (hScreenDC, VERTRES);
if (capRect.top < 0)
    capRect.top = 0;
if (capRect.left < 0)
    capRect.left = 0;
if (capRect.bottom > yRes)
    capRect.bottom = yRes;
if (capRect.right > xRes)
    capRect.right = xRes;
width = capRect.right - capRect.left;
height = capRect.bottom - capRect.top;

// Create a bitmap and store the image into it.
*bmPtr = CreateCompatibleBitmap (hScreenDC, width, height);
hOldBitmap = SelectObject (hMemoryDC, *bmPtr);
BitBlt (hMemoryDC, 0, 0, width, height, hScreenDC, capRect.left,
        capRect.top, SRCCOPY);
SelectObject (hMemoryDC, hOldBitmap);
DeleteDC (hMemoryDC);

// Allocate space for a LOGPALETTE structure.
nColors = GetDeviceCaps (hScreenDC, SIZEPALETTE);
if (nColors == 0)
    nColors = GetDeviceCaps (hScreenDC, NUMCOLORS);
hGlobal = GlobalAlloc (GHND, sizeof (LOGPALETTE) +
                       (nColors * sizeof (PALETTEENTRY)));
pPalette = (LOGPALETTE *) GlobalLock (hGlobal);

// Fill in the structure from the system palette.
pPalette->palVersion = 0x300;
pPalette->palNumEntries = (WORD) nColors;
GetSystemPaletteEntries (hScreenDC, 0, nColors,
                         pPalette->palPalEntry);

// Create a logical palette, and delete the LOGPALETTE structure.
*palPtr = CreatePalette (pPalette);
GlobalUnlock (hGlobal);
GlobalFree (hGlobal);
```

```
    // Delete the DC as well.
    DeleteDC (hScreenDC);
}
```

7. 添加函数Paint。此函数用来显示捕捉到的位图，并缩放位图以便适应窗口。调用函数GetObject 获取位图的宽度和高度值，接着函数 StretchBlt 使用这些值在窗口中显示位图。

```
/*-----------------------------------------------------------------------*/
/* This function displays the captured bitmap in the window. It          */
/* uses the palette of the captured bitmap so that the colors are        */
/* correct.                                                              */
/*-----------------------------------------------------------------------*/
void Paint (HWND hWnd, HPALETTE hPalette, HBITMAP hBitmap)
{
    PAINTSTRUCT          ps;
    HDC                  hDC, hMemDC;
    HPALETTE             hOldPalette;
    HBITMAP              hOldBitmap;
    BITMAP               bmInfo;

    hDC = BeginPaint (hWnd, &ps);

    GetObject (hBitmap, sizeof (BITMAP), &bmInfo);
    if ( (hMemDC = CreateCompatibleDC (hDC)) ! = NULL)
     {
        hOldPalette = SelectPalette (hDC, hPalette, FALSE);
        RealizePalette (hDC);
        hOldBitmap = SelectObject (hMemDC, hBitmap);
        StretchBlt (hDC, ps.rcPaint.left, ps.rcPaint.top,
                    (ps.rcPaint.right - ps.rcPaint.left),
                    (ps.rcPaint.bottom - ps.rcPaint.top),
                    hMemDC, 0, 0, bmInfo.bmWidth,
                    bmInfo.bmHeight, SRCCOPY);
        SelectObject (hMemDC, hOldBitmap);
        SelectPalette (hDC, hOldPalette, FALSE);
        DeleteDC (hMemDC);
    }
    EndPaint (hWnd, &ps);
}
```

8. 在源文件中输入下列代码。函数MainWndProc是一个回调函数，用来处理传送给例子程序主窗口的消息。在此函数中最重要的消息是CM _CAPTURE 和 WM _TIMER。命令消息CM _CAPTURE 在选择菜单项 Start Capture 时发送，消息 WM _TIMER 标识计时触发。

为了响应消息CM _CAPTURE，此函数显示出警告消息，通知用户五秒钟后开始捕捉屏幕。接着调用API 函数 SetTimer 来设置计时器，使得每一秒钟触发一次。API 函数 ShowWin-

dow 用来极小化例子程序主窗口，函数 SetActiveWindow 用来将主窗口的先前窗口激活，将例子程序的图标禁止，以免捕捉到的是该图标。

为了响应消息 WM_TIMER，此函数将递增计数器 tickCounter。当计数器到达 5 后，就调用前面定义的函数 Capture，用来捕捉整个屏幕、激活的窗口或激活的用户区。而后，例子程序的主窗口将恢复到原来的大小，以便显示捕捉到的图象。恢复窗口将使此函数接收到消息 WM_PAINT，从而可以将捕捉到的图象显示出来。

```
/*-----------------------------------------------------------------------*/
/* This function is the main window callback function which will be      */
/* called by Windows to process messages for our window.                  */
/*-----------------------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
     {
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EXIT:
                     DestroyWindow (hWnd);
                     break;
                case CM_ENTIRESCREEN:
                case CM_ACTIVEWINDOW:
                case CM_ACTIVECLIENT:
                     CheckMenuItem (GetMenu (hWnd), captureOptions,
                                 MF_BYCOMMAND | MF_UNCHECKED);
                     captureOptions = (int) wParam;
                     CheckMenuItem (GetMenu (hWnd), captureOptions,
                                 MF_BYCOMMAND | MF_CHECKED);
                     break;
                case CM_CAPTURE:
                     MessageBox (hWnd," The area that you have chosen\n"
                                 " wil be captured 5 seconds after\n"
                                 " you click the OK button. "," Capture",
                                 MB_ICONINFORMATION | MB_OK);
                     if (hPalette)
                         DeleteObject (hPalette);
                     hPalette = NULL;
                     if (hBitmap)
                         DeleteObject (hBitmap);
                     hBitmap = NULL;
                     tickCounter = 0;
```

```
                if (SetTimer (hWnd, 1, 1000, NULL) ! = NULL)
                 {
                    EnableMenuItem (GetMenu (hWnd), CM_CAPTURE,
                                   MF_BYCOMMAND | MF_GRAYED);
                    /* Minimize this application, and select another
                       window on the desktop. */
                  ShowWindow (hWnd, SW_MINIMIZE);
                  SetActiveWindow (GetWindow (hWnd, GW_HWNDPREV));
                }
                break;
          default:
                return DefWindowProc (hWnd, message, wParam, lParam);
      }
      break;
case WM_TIMER:
      tickCounter = tickCounter + 1;
      if (tickCounter == 5)
      {
          KillTimer (hWnd, 1);
          CaptureImage (captureOptions, &hBitmap, &hPalette);
          ShowWindow (hWnd, SW_RESTORE);
          EnableMenuItem (GetMenu (hWnd), CM_CAPTURE,
                         MF_BYCOMMAND | MF_ENABLED);
      }
      break;
case WM_PAINT:
      if ( (hBitmap) && (hPalette))
          Paint (hWnd, hPalette, hBitmap);
      else
          return DefWindowProc (hWnd, message, wParam, lParam);
      break;
case WM_SIZE:
      if ( (wParam == SIZE_RESTORED) || (wParam == SIZE_MAXIMIZED))
          InvalidateRect (hWnd, NULL, TRUE);
      break;
case WM_DESTROY:
      if (hPalette)
          DeleteObject (hPalette);
      if (hBitmap)
          DeleteObject (hBitmap);
      KillTimer (hWnd, 1);
      PostQuitMessage (0);
      break;
```

```
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

9. 在源文件中添加下列两个函数。函数 InitApplication 只有在窗口类不存在时才被调用，为主窗口登记窗口类。函数 InitInstance 用来创建并显示主窗口。

```
/*--------------------------------------------------------------------------*/
/* This function initialises a WNDCLASS structure and uses it to            */
/* register a class for our main window.                                    */
/*--------------------------------------------------------------------------*/
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName = MAKEINTRESOURCE (IDM_CAPTUREMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/*--------------------------------------------------------------------------*/
/* This function creates an instance of our main window. The window         */
/* is given a class name and a title, and told to display anywhere.         */
/* the nCmdShow argument passed to the program determines how               */
/* the window will be displayed.                                            */
/*--------------------------------------------------------------------------*/
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;    // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName," Capture Utility",
                         WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
```

```
                                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL,
NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);    // Send a WM_PAINT message.
    return TRUE;
}
```

10. 最后需要在源文件中添加的是函数 WinMain，在启动此例子程序时 Windows 95 将调用此函数。用来登记窗口类（如果此窗口类不存在），创建并显示主窗口，接着处理消息直到例子程序结束。

```
/*----------------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if       */
/* the main window class name has already been registered, if it has    */
/* not, call InitApplication to register it. Call InitInstance to       */
/* create an instance of our main window, then pump messages until      */
/* the application is closed.                                           */
/*----------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;
    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

11. 编译并运行此例子程序。

注释

本节的例子程序既不能打印位图也不能将位图存盘，但是在此例子程序的基础上，可以非常方便的实现一个实用的屏幕捕捉程序。打印函数的实现非常类似于函数 Paint。而位图的存盘，则需要将设备相关位图和调色板转换为设备无关位图，可以利用许多函数如 GetDIBits 来实现。

4.6 生成动画

问题

为了使得应用程序具有生机，有时需要在应用程序中添加一些动画。第三方软件中有的已经实现了此功能。如何使用 Windows API 函数来生成动画呢？

方法

使用 Windows API 函数可以生成一些简单的动画，只要图象不大、动画过程不太复杂，一般不会出现闪烁或速度问题。况且在 Windows 95 中，使用 32 位代码重写了许多图形设备接口函数，使得运行效率有了很大程度的提高，而在早期的 Windows 版本中，使用 16 位代码编写的图形设备接口函数使得运行效率非常低。

如果生成动画时使用的是位图图象，则需要注意以下几个问题：首先是背景的恢复。当位图图象运动经过背景的某区域时，此区域的背景则需要恢复到原状态。最简单的方法是在位图图象运动到此区域前将此区域的背景捕捉到某个位图中，位图图象离开后再恢复此区域的背景。

第二个问题是：如果运动物的形状不是正方形或矩形而是其他的形状，如何生成动画。可以使用函数 SetBkMode 来设置绘图模式为透明或不透明，但此模式不能用于位图图象的操作，要生成有透明部分的位图的唯一方法是使用屏蔽位图进行多次 BitBlt 操作。

最后一个问题是：在播放动画时如何来计时。简单的循环调用动画函数通常并不实用，会占用太多的处理器资源，而且播放的速度经常太快。一个较好的方法是使用计时器，系统计时器可以使用函数 SetTimer 进行初始化，接着应用程序便可以响应其发送来的消息 WM_TIMER。但此方法存在着两个问题：第一个问题是计时器消息的优先权太低，只有在所有的消息（除了消息 WM_PAINT）被处理后才能被处理。第二个问题是计时器的最大分辨率是 55 毫秒，也就是说应用程序每秒只能接收到 18 个消息，对于播放动画来说这是一个非常棘手的问题。在本节的例子程序中使用了多媒体计时器服务函数 timeSetEvent，并使用回调函数来实现每秒大约 25 个事件，使得足够平滑地播放动画。

步骤

按照下列步骤实现一个例子程序。此例子程序打开一个绿色背景的窗口，在窗口中一个多颜色的球碰来碰去。当窗口的大小改变时，球将重新回到窗口的左上角，然后开始新的运动。结束例子程序可以点击窗口右上角的关闭按钮。运行的例子程序如图 4-8 所示。

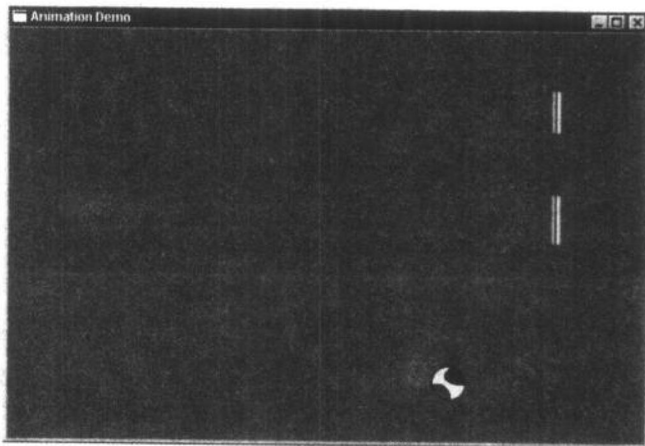


图 4-8　例子程序 ANIMATE

实现例子程序的具体步骤如下：

1. 创建目录 ANIMATE，用来存放例子程序的所有文件。

2. 创建球的位图图象。使用资源编辑器或某个绘图程序创建一个 32×32 象素 16 色（每个象素 4 位）的位图图象，并存为文件 BALL. BMP。

3. 为例子程序创建图标。使用资源编辑器创建一个 32×32 象素 16 色的图标，并存为文件 ANIMATE. ICO。

4. 下面为例子程序创建资源脚本。使用文本编辑器创建新文件 ANIMATE. RC，并输入下列文本：

```
/*---------------------------------------------------------------------------*/
/*                                                                           */
/* MODULE: ANIMATE.RC                                                        */
/* PURPOSE: This resource script defines the bitmap and application          */
/*          icon associated with this application.                           */
/*                                                                           */
/*---------------------------------------------------------------------------*/

#include " animate.rh"

IDBM_BALL BITMAP " ball.bmp"
IDI_APPICON ICON " app.ico"
```

5. 在资源脚本中包含了头文件 ANIMATE. RH，此文件为在例子程序中使用的位图和图标定义资源标识符。创建新文件 ANIMATE. RH，并在文件中输入下列定义：

```
#ifndef __ANIMATE_RH
/*---------------------------------------------------------------------------*/
/*                                                                           */
/* MODULE: ANIMATE.RH                                                        */
/* PURPOSE: Defines identifiers used to address resources.                   */
/*                                                                           */
/*---------------------------------------------------------------------------*/

#define __ANIMATE_RH
#define IDI_APPICON      1
#define IDBM_BALL        1000

#endif
```

6. 现在为例子程序创建源文件。使用文本编辑器创建新文件 ANIMATE. C，在文件的顶部添加下列代码段。此段代码定义了预处理器符号 STRICT，表示对在例子程序中使用的句柄进行严格的类型检查。常量的定义除了窗口类的名字，还有用户定义消息，用来在多媒体计时器事件发生时发送消息给例子程序的窗口（后面将详细介绍）。另外还定义了三个 RGB 值，用来对位图图象进行透明处理。

```
/*---------------------------------------------------------------------------*/
/*                                                                           */
/* MODULE: ANIMATE.C                                                         */
```

```
/* PURPOSE: This applciation demonstrates simple bitmap animation            */
/*                using a timer.                                              */
/*                                                                            */
/*--------------------------------------------------------------------------- */
#define STRICT
#include <windows.h>
#include <mmsystem.h>
#include " animate.rh"

// Constants.
char *MainWindowClassName = " AnimateWindow";
#define MYMSG_TIMER     (WM_USER + 50)
#define RGBBLACK RGB (0, 0, 0)
#define RGBWHITE RGB (255, 255, 255)
#define RGBTRANSPARENT RGBWHITE

// Instance variables.
HINSTANCE        hInstance = NULL;
HBITMAP          hBitmap = NULL;
HBITMAP          hMask = NULL;
HBITMAP          hBackground = NULL;
LONG             left, top;
LONG             width, height;
LONG             xincrement, yincrement;
UINT             timerId = 0;
BOOL             firstPaint = TRUE;
```

7. 在文件中添加第一个函数 CaptureBackground。此函数用来捕捉位图图象将要覆盖的背景区域，并将捕捉到的数据存放在某个位图中，在位图图象移开后，便可以恢复原来的背景。此函数以位图图象的大小来创建一个新的位图，或者重用已经创建好的位图，接着使用内存设备描述表，调用函数 BitBlt 来从窗口中拷贝数据到此位图中。

```
/*--------------------------------------------------------------------------- */
/* This function captures the background at a given location into            */
/* the bitmap hBackground. If the bitmap handle is NULL, a bitmap is          */
/* created to contain the image. If the bitmap is not NULL, just use          */
/* the existing bitmap.                                                       */
/*--------------------------------------------------------------------------- */
HBITMAP CaptureBackground (HWND hWnd, LONG left, LONG top,
                           LONG width, LONG height,
                           HBITMAP hBackground)
{
    HDC            hDC, hMemoryDC;
    HBITMAP        hOldBitmap;
```

```
    hDC = GetDC (hWnd);

    if (! hBackground)
        hBackground = CreateCompatibleBitmap (hDC, width, height);

    hMemoryDC = CreateCompatibleDC (hDC);
    hOldBitmap = SelectObject (hMemoryDC, hBackground);
    BitBlt (hMemoryDC, 0, 0, width, height, hDC, left, top, SRCCOPY);
    SelectObject (hMemoryDC, hOldBitmap);
    DeleteDC (hMemoryDC);
    ReleaseDC (hWnd, hDC);
    return hBackground;
}
```

8. 下面的函数用来将位图图象显示在屏幕上，并使位图图象的某些部分透明。除非运动物是填充矩形，否则便需要使位图图象的某些部分透明。Windows 没有提供函数或运算符来实现位图图象的部分透明，但可以使用单色的屏蔽位图并两次调用函数 BitBlt 来实现位图图象的部分透明。屏蔽位图的大小要同位图图象的大小相同，并且对应于位图图象的透明部分的象素值应为 1（白色），对应于不透明部分的象素值应为 0（黑色）。而位图图象需要透明的部分的象素值应为 0（黑色），但是其他部分的象素值也可为 0。下面来看看两次函数 BitBlt 调用：

第一次调用将屏蔽位图拷贝到窗口背景上，参数为 SRCAND 表示使用逻辑运算符 AND 来合并位图，从而使得在目的位图中要放置运动物的区域变为黑色，其余的区域没有变化。

第二次调用使用光栅运算符 SRCPAINT 来合并位图，即使用逻辑运算符 OR 来合并位图，所以对应于目的位图的黑色区域，位图图象的相应部分将被显示出来，而对应于位图图象的黑色区域，目的位图的相应区域则没有任何变化。在速度较慢的 PC 机上显示大的位图图象，两次函数调用之间会有明显的闪烁，可以通过再添加两个 BitBlt 函数调用来解决这个问题。第一个函数调用将窗口背景拷贝到一个内存设备描述表中，即内存中的一个位图中，接着的两个函数调用就是前面讨论的两个函数调用，最后的一个函数调用则是用来将修改过的位图拷回屏幕，这样便是在所有操作完成后再修改屏幕，所以避免了闪烁。但是因为又添加了两个函数调用，所以会影响运行的效率，如果位图图象不是很大，使用两个函数调用的方法是可以运行的相当好的。

```
/*------------------------------------------------------------------*/
/* This function performs a transparent BitBlt of the bitmap into   */
/* the chosen device context. It does this by using two BitBlt      */
/* operations with different operators.                             */
/*------------------------------------------------------------------*/
void TransparentBlt (HDC hDestDC, LONG left, LONG top,
                     LONG width, LONG height,
                     HDC hSrcDC, LONG srcleft, LONG srctop,
                     HBITMAP hMask)
{
```

```
    HDC          hMaskDC;
    HBITMAP      oldBitmap;

    hMaskDC = CreateCompatibleDC (hDestDC);
    oldBitmap = SelectObject (hMaskDC, hMask);
    /* Use SRCAND so that all but the transparent
       bits of the destination are set to black. */
    BitBlt (hDestDC, left, top, width, height,
           hMaskDC, srcleft, srctop, SRCAND);
    /* Use SRCPAINT so that all but the black bits
       of the source are placed into the destination */
    BitBlt (hDestDC, left, top, width, height,
           hSrcDC, srcleft, srctop, SRCPAINT);
    SelectObject (hMaskDC, oldBitmap);
    DeleteDC (hMaskDC);
}
```

9. 在同一源文件中添加函数 DoAnimation。此函数用来播放动画，首先调用函数 BitBlt 来恢复背景位图，接着计算位图图象的新位置，并在位图图象被放置在新位置前调用函数 CaptureBackground 来拷贝背景，然后调用函数 TransparentBlt 将位图图象放置在新位置上。

```
/*-------------------------------------------------------------------------*/
/*  This function performs the actual display of the animated bitmap.      */
/*  It follows these steps: 1. Put back the saved background               */
/*                          2. Calculate new top/left positions            */
/*                          3. Save the background                         */
/*                          4. Blit the bitmap                             */
/*-------------------------------------------------------------------------*/
void DoAnimation (HWND hWnd)
{
    HDC          hDC, hMemoryDC;
    HBITMAP      hOldBitmap;

    hDC = GetDC (hWnd);
    hMemoryDC = CreateCompatibleDC (hDC);

    /* If the background was saved, restore it */
    if (hBackground)
      {
        hOldBitmap = SelectObject (hMemoryDC, hBackground);
        BitBlt (hDC, left, top, width, height, hMemoryDC, 0, 0, SRCCOPY);
        SelectObject (hMemoryDC, hOldBitmap);
      }
```

```
    /* Calculate the new top and left coordinates */
    top += yincrement;
    left += xincrement;

    /* Now capture the background at the new location */
    hBackground = CaptureBackground (hWnd, left, top, width,
                                     height, hBackground);

    /* Finally, display the object at that location */
    if (hBitmap)
     {
        hOldBitmap = SelectObject (hMemoryDC, hBitmap);
        TransparentBlt (hDC, left, top, width, height,
                        hMemoryDC, 0, 0, hMask);
        SelectObject (hMemoryDC, hOldBitmap);
    }
    DeleteDC (hMemoryDC);
    ReleaseDC (hWnd, hDC);
}
```

10. 函数 TestForCollision 调用函数 GetClientRect 来获取当前窗口的坐标，接着判断位图图象是否同窗口碰撞。如果同窗口的上边或底边碰撞，则取反变量 yinc。如果同窗口的左右两边碰撞，则取反变量 xinc。如果碰撞到的是窗口角，则变量 xinc 和变量 yinc 都取反。

```
/*---------------------------------------------------------------------*/
/* This function tests to see if our object has collided with any of   */
/* the sides of the window's client area. If it has, the xincrement    */
/* and yincrement variables are changed.                               */
/*---------------------------------------------------------------------*/
void TestForCollision (HWND hWnd, LONG top, LONG left,
                       LONG width, LONG height,
                       LONG *xinc, LONG *yinc)
{
    RECT rcClient;

    GetClientRect (hWnd, &rcClient);
    if ( (top <= rcClient.top) ||
        ( (top + height) >= rcClient.bottom))
        *yinc *= -1;
    if ( (left <= rcClient.left) ||
        ( (left + width) >= rcClient.right))
        *xinc *= -1;
}
```

11. 现在添加函数 CreateBitmaps。此函数调用 API 函数 LoadBitmap 来将 16 色的位图图

象装入，并使用位图图象创建两个的新位图，这两个新的位图用在函数 TransparentBlt 中。在函数 TransparentBlt 中，首先需要一个单色的屏蔽位图，对应于位图图象的透明部分，此位图的相应部分的象素值为 1（白色），对应于位图图象的不透明部分，此位图的象素值为 0（黑色）。在函数 TransparentBlt 中还需要一个彩色位图，对应于屏蔽位图的白色部分，此位图的相应部分为黑色。

单色位图是通过调用函数 CreateCompatibleBitmap 在内存设备描述表中创建的。这是因为使用函数 CreateCompatibleDC 创建的设备描述表中缺省的有一个 1×1 象素的单色位图，创建与此单色位图相兼容的位图当然也是一个单色位图。接着调用函数 BitBlt 来拷贝资源位图到单色位图，此函数将自动转换所有的象素，同源 DC 中的背景色相同的象素转换为白色，其余的象素转换为黑色，使用此方法，可以快速地创建一个屏蔽位图。接着再调用函数 BitBlt 来拷贝资源位图到位图 hBitmap。第三个 BitBlt 函数调用（参数为 SRCAND）用来对屏蔽位图和位图 hBitmap 进行逻辑 AND 操作，因此在位图 hBitmap 中所有需要透明的部分都被置成了黑色。

```
/*-----------------------------------------------------------------*/
/* This function creates the bitmaps and assigns them to the bitmap    */
/* handles for this instance — hBitmap and hMask. If the screen        */
/* device supports the new transparent BitBlt mode, hBitmap is just    */
/* a copy of the resource. Otherwise, hMask is a mask bitmap and       */
/* hBitmap is a copy of the resource, modified to work with the mask.  */
/*-----------------------------------------------------------------*/
void CreateBitmaps (HWND hWnd)
{
    BITMAP          bmInfo;
    HBITMAP         hResBitmap, hOldSrc, hOldDest;
    HDC             hSrcDC, hDestDC, hDC;
    COLORREF        oldColor, oldText;

    hResBitmap = LoadBitmap (hInstance,
                          MAKEINTRESOURCE (IDBM_BALL));
    GetObject (hResBitmap, sizeof (BITMAP), &bmInfo);
    width = bmInfo.bmWidth;
    height = bmInfo.bmHeight;
    hDC = GetDC (hWnd);

    hSrcDC = CreateCompatibleDC (hDC);
    hDestDC = CreateCompatibleDC (hDC);
    /* Creates a bitmap compatible with the default bitmap
       in hMaskDC. This is a monochrome bitmap.
       Select the resource bitmap into hSrcDC and use
       BitBlt to make a mask bitmap. */
    hMask = CreateCompatibleBitmap (hDestDC, width, height);
```

```
    hOldDest = SelectObject (hDestDC, hMask);
    hOldSrc = SelectObject (hSrcDC, hResBitmap);
    oldColor = SetBkColor (hSrcDC, RGBTRANSPARENT);
    BitBlt (hDestDC, 0, 0, width, height, hSrcDC, 0, 0, SRCCOPY);
    SetBkColor (hSrcDC, oldColor);

    /* Create a Color bitmap to hold the image. Note we create
       the bitmap compatible with the screen to get a color one.
       Make a copy of the hResBitmap into hBitmap. */
    hBitmap = CreateCompatibleBitmap (hDC, width, height);
    SelectObject (hDestDC, hBitmap);
    BitBlt (hDestDC, 0, 0, width, height, hSrcDC, 0, 0, SRCCOPY);

    /* Now combine hMask with hBitmap to black out the white
         bits in hBitmap. */
    SelectObject (hSrcDC, hMask);
    SelectObject (hDestDC, hBitmap);
    oldColor = SetBkColor (hDestDC, RGBBLACK);
    oldText = SetTextColor (hDestDC, RGBWHITE);
    BitBlt (hDestDC, 0, 0, width, height, hSrcDC, 0, 0, SRCAND);
    SetTextColor (hDestDC, oldText);
    SetBkColor (hDestDC, oldColor);

    /* Restore the old bitmaps and delete DCs. */
    SelectObject (hDestDC, hOldDest);
    SelectObject (hSrcDC, hOldSrc);
    DeleteDC (hDestDC);
    DeleteDC (hSrcDC);
    ReleaseDC (hWnd, hDC);
    DeleteObject (hResBitmap);
}
```

12. 添加函数CatchMMTimer。此函数是一个回调函数，用来接收多媒体计时器的事件通知。此函数只是发送一个用户定义消息（在这里为WM_USER+50）给窗口，窗口句柄是作为用户数据传送给此函数的。

是否可以在此函数内实现动画的播放呢？这是不可以的。在计时器的回调函数中，代码应该越简单越好，一些消耗时间的操作（如位图的操作）不应该在回调函数内实现。在此函数中，仅仅实现了传送消息给窗口。此回调函数结束后，当系统再分配时间给例子程序时，消息将被处理，动画将被播放。在下面的“多媒体计时器事件的响应”中，将更加详细地介绍计时器事件的响应。

```
/* ------------------------------------------------------------------------------ */
/* This callback function receives multimedia timer events.                       */
```

```
/* Because you should not do tasks which may take a while (like         */
/* blitting a bitmap) in a timer callback, post an application          */
/* message to the window.                                               */
/*--------------------------------------------------------------------- */
void CALLBACK CatchMMTimer (UINT IDEvent,                  /* event Id */
                           UINT uReserved,                 /* unused */
                           DWORD dwUser,                   /* our hWnd */
                           DWORD dwReserved1,              /* unused */
                           DWORD dwReserved2)              /* unused  */
{
    PostMessage ( (HWND) dwUser, MYMSG_TIMER, 0, 0);
}
```

多媒体计时器事件的响应

尽管多媒体计时器事件不再象早期的 WINDOWS 版本那样是中断事件，但仍然会影响系统的效率。在计时器的回调函数中，不要执行一些消耗时间的操作（例如：BitBlt 操作）。实际上，在计时器的回调函数中，Microsoft 只建议使用下列 API 函数：

EnterCriticalSection	PostThreadMessage
LeaveCriticalSection	SetEvent
midiOutLongMsg	timeGetTime
midiOutShortMsg	timeGetSystemTime
OutputDebugString	timeKillEvent
PostMessage	timeSetEvent

13. 在源文件中添加函数 SetupAnimation。此函数使用变量 left 和 top 来设置球的初始位置，使用变量 xincrement 和 yincrement 来设置初始方向，同时还初始化多媒体计时器。调用函数 timeBeginPeriod 来设置计时器的最小分辨率，在本例子程序中为 40ms。接着调用函数 timeSetEvent 设置一个周期性的计时器，每 40ms 触发一个事件，在事件发生时，系统调用前面创建的函数 CatchMMTimer。

```
/*--------------------------------------------------------------------- */
/* Use the paint function to set up the animation. At this stage you    */
/* know that the background has been painted for the first time, so     */
/* you can safely start displaying the image.                           */
/*--------------------------------------------------------------------- */
void SetupAnimation (HWND hWnd)
{
    /* Now set up the animation increments. */
    left = 1;
    top = 1;
    xincrement = 4;
    yincrement = 4;

    /* Set up the animation timer. */
```

```
    if (timeBeginPeriod (40) == 0)
        timerId = timeSetEvent (40, 40, CatchMMTimer,
                                (DWORD) hWnd, TIME_PERIODIC);
}
```

14. 下面要添加的函数 MainWndProc 是一个回调函数，此函数用来处理传送给例子程序主窗口的消息，此函数只处理一部分信息，其余的消息交给 API 函数 DefWindowProc 来处理。有几个消息值得详细的加以讨论：

此函数响应用户定义消息 MYMSG_TIMER。每次计时器回调函数 CatchMMTimer 被调用时都将发送此消息，为了响应此消息，此函数调用函数 TestForCollision 来检测是否碰到了窗口的边，以便改变运动的方向，接着调用函数 DoAnimation 来播放动画。

此函数也响应消息 WM_PAINT，尽管只处理第一个 WM_PAINT 消息。此函数调用函数 CreateBitmaps 和 SetupAnimation 来启动动画的播放，接着清除标志 firstPaint 使得此初始化操作不再被执行。因为在例子程序接收到第一个 WM_PAINT 消息时背景色已被填充，所以在响应消息 WM_PAINT 而不是在响应消息 WM_CREATE 时进行此初始化操作。另外一个重要的消息是 WM_SIZE，在窗口被极小化、极大化、恢复或改变大小时，Windows 系统发送此消息给窗口。为了响应此消息，此函数从窗口的左上角重新开始播放动画，并强制窗口重画。

```
/*-------------------------------------------------------------------*/
/* This function is the main window callback function which will be */
/* called by Windows to process messages for our window.             */
/*-------------------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
     {
        case MYMSG_TIMER:
            TestForCollision (hWnd, top, left, width, height,
                              &xincrement, &yincrement);
            DoAnimation (hWnd);
            break;
        case WM_PAINT:
            if (firstPaint)
            {
                firstPaint = FALSE;
                CreateBitmaps (hWnd);
                SetupAnimation (hWnd);
            }
            return DefWindowProc (hWnd, message, lParam, wParam);
        case WM_SIZE:
            /* Make sure that the bitmap is visible */
```

```
            DeleteObject (hBackground);
            hBackground = NULL;
            top = 1; left = 1;
            InvalidateRect (hWnd, NULL, TRUE);
            break;
        case WM_DESTROY:
            if (hBitmap)
                DeleteObject (hBitmap);
            if (hMask)
                DeleteObject (hMask);
            if (hBackground)
                DeleteObject (hBackground);
            if (timerId)
                timeKillEvent (timerId);
            timeEndPeriod (40);
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

15. 在同一源文件中添加下面两个函数，这两个函数用来建立例子程序。函数 InitApplication 为例子程序的主窗口登记窗口类，因为类是可以被所有属于此类的窗口共享，所以此函数只有在窗口类不存在时才被调用。相反，函数 InitInstance 在每次调用例子程序时都将被调用，此函数用来创建并显示例子程序的主窗口。

```
/*-----------------------------------------------------------------------*/
/* This function initialises a WNDCLASS structure and uses it to         */
/* register a class for our main window.                                  */
/*-----------------------------------------------------------------------*/
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;
    LOGBRUSH  lb;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

```
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.lpszMenuName =  NULL;
    wc.lpszClassName = MainWindowClassName;
    lb.lbStyle = BS_SOLID;
    lb.lbColor = RGB (0, 128, 64); /* Blue-Green brush */
    wc.hbrBackground = CreateBrushIndirect (&lb);

    return RegisterClass (&wc);
}

/*----------------------------------------------------------------------*/
/* This function creates an instance of our main window. The window     */
/* is given a class name and a title, and told to display anywhere.     */
/* the nCmdShow argument passed to the program determines how           */
/* the window will be displayed.                                        */
/*----------------------------------------------------------------------*/
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst; /* Store in global variable */

    hWnd = CreateWindow (MainWindowClassName," Animation Demo",
                        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);            /* Send a WM_PAINT message */
    return TRUE;
}
```

16. **最后需要添加的函数是函数 WinMain，此函数添加在源文件末。函数 WinMain 是 Windows 应用程序的入口点，调用函数 InitApplication 来登记窗口类，调用函数 InitInstance 来创建并显示主窗口，最后处理消息直到例子程序结束。**

```
/*----------------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if       */
/* the main window class name has already been registered, if it has   */
/* not, call InitApplication to register it. Call InitInstance to       */
/* create an instance of our main window, then pump messages until      */
```

```
/* the application is closed.                                                    */
/*----------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
     }
    return (msg.wParam);
}
```

17. 编译并运行此例子程序。

注释

在本节例子程序的基础上，可以使用 Windows API 函数来实现更加复杂的动画。但是如果生成的动画相当大，或者生成动画的代码需要占用大量的处理器资源，则运行的效率会非常低，出现令人难以忍受的图象闪烁或动画播放不平滑。如果是这样，则应该尽量减少调用一些大量占用处理器资源的函数如 BitBlt，尤其是 StretchBlt、GetDIBits、SetDIBits 以及 StretchDIBits。如果仍然没有什么明显的改进，那么本节所介绍的函数将不再实用，而 DirectX API（Microsoft 的游戏软件开发包的一部分）可以用来生成快速平滑的动画以及其他的绘图操作。

4.7 实现位图的拖放

问题

有时需要在应用程序中实现拖放位图的功能，用户便可以使用鼠标选中某个位图图象并拖动此位图，从而实现了此位图的移动。如何实现此功能呢？

方法

在 Windows 95 中，广泛的使用拖放方法来操纵对象（文件，文件夹，文档，甚至几行文本）。另外，在绘图程序和字处理程序中也使用拖放方法来调整对象的位置，在窗口间拖放对象时，鼠标光标通常改变形状来表示正在拖放对象。而在窗口内拖放对象，经常遇到的是移动整个位图。本节就是用来介绍实现窗口内拖放位图的方法。

拖放位图需要两个技术，这两个技术在本章的前面几节中已经讨论过，即跟踪鼠标移动

和位图动画。在应用程序中，要响应消息 WM _LBUTTONDOWN 来开始拖放，接着响应消息 WM _MOUSEMOVE，恢复位图前面所在位置的背景，捕捉鼠标新位置处的背景，然后在新位置上显示位图，最后处理消息 WM _LBUTTONUP 来结束拖放，位图便被放置在新位置上。

步骤

按照下列步骤实现一个例子程序。此例子程序显示出一个黄色窗口，在窗口的左上角有一小块位图。用鼠标左键点击此位图块并在窗口内任意拖放，一旦鼠标键松开，位图块将被放置在新的位置上，如果位图块被部分拖放出窗口边，则只有剩余的部分被显示出来。在拖放位图块时，鼠标不能移动到窗口的用户区外，因为在例子程序中已经将鼠标限制在窗口内。在例子程序中拖放位图块如图 4-9 所示。

实现例子程序的具体步骤如下：

1. 创建工作目录 DRAGBMP，用来在开发例子程序时存放所有的源文件。

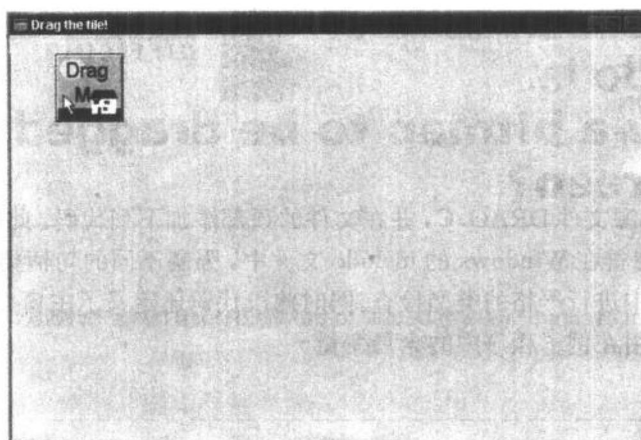


图 4-9　在例子程序 DRAG 中拖放位图块

2. 创建一个用来拖放的位图块。使用资源编辑器或某个绘图程序创建一个 64×64 象素 16 色的位图，并存为文件 IMAGE. BMP。

3. 为例子程序创建图标。使用资源编辑器创建一个 32×32 象素 16 色的图标，并存为文件 DRAG. ICO。

4. 现在为例子程序创建资源脚本。当资源脚本被编译时，位图和图标将被联编到例子程序的执行文件中。使用文本编辑器创建新文件 DRAG. RC，并在此文件中输入下列资源脚本：

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/* MODULE: DRAG.RC                                                     */
/* PURPOSE: This resource script defines the bitmap and application    */
/*          icon associated with this application.                     */
/*                                                                     */
/*---------------------------------------------------------------------*/
```

```
#include " drag.rh"

IDBM_IMAGE BITMAP " image.bmp"

IDI_APPICON ICON " app.ico"
```

5. 在资源脚本中包含了另一个文件DRAG.RH，此文件用来定义位图和图标的资源标识符，这些资源标识符也用在源文件中，所以使用了一个单独的文件来定义。创建新文件DRAG.RH，并在此文件中输入下列定义：

```
#ifndef __DRAG_RH
/*------------------------------------------------------------------*/
/*                                                                  */
/* MODULE: DRAG.RH                                                  */
/* PURPOSE: Defines identifiers used to address resources.          */
/*                                                                  */
/*------------------------------------------------------------------*/
#define __DRAG_RH
#define IDI_APPICON          1
#define IDBM_IMAGE           1000
#endif
```

6. 创建C语言源文件DRAG.C，并在文件的顶部添加下列代码。此段代码定义了预处理器符号STRICT，使得在Windows的include文件中，强制不同的句柄类型为不同类型，并在例子程序的源代码中进行严格的类型检查。同时此段代码还定义了主窗口的窗口类的类名，定义并初始化了所有用在例子程序中的全局变量。

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* MODULE: DRAG.C                                                   */
/* PURPOSE: This application demonstrates simple bitmap dragging.   */
/*                                                                  */
/*------------------------------------------------------------------*/
#define STRICT

#include <windows.h>
#include " drag.rh"

// Constants.
char *MainWindowClassName = " DragDemoWindow";

// Instance variables.
HINSTANCE          hInstance = NULL;
HBITMAP            hBitmap = NULL;
```

```
HBITMAP          hBackground = NULL;
LONG             left, top, width, height;
LONG             xOffset, yOffset;
LONG             saveLeft, saveTop, saveWidth, saveHeight;
BOOL             dragging = FALSE;
BOOL             firstPaint = TRUE;
```

7. 在源文件中添加第一个函数 CaptureBackground，在 4.6 节的例子程序中也有此函数。此函数用来拷贝部分窗口到一个位图中，如果位图不存在则调用函数 CreateCompatibleBitmap 来创建。此函数调用函数 BitBlt 来完成拷贝，从窗口位图到包含位图的内存设备描述表。

```
/*-------------------------------------------------------------*/
/* This function captures the background at a given location into   */
/* the bitmap hBackground. If the bitmap handle is NULL, a bitmap is */
/* created to contain the image. If the bitmap is not NULL, just use */
/* the existing bitmap.                                               */
/*-------------------------------------------------------------*/
HBITMAP CaptureBackground (HWND hWnd, LONG left, LONG top,
                           LONG width, LONG height,
                           HBITMAP hBackground)

{
    HDC             hDC, hMemoryDC;
    HBITMAP         hOldBitmap;

    hDC = GetDC (hWnd);

    if (! hBackground)
        hBackground = CreateCompatibleBitmap (hDC, width, height);

    hMemoryDC = CreateCompatibleDC (hDC);
    hOldBitmap = SelectObject (hMemoryDC, hBackground);
    BitBlt (hMemoryDC, 0, 0, width, height, hDC, left, top, SRCCOPY);
    SelectObject (hMemoryDC, hOldBitmap);
    DeleteDC (hMemoryDC);
    ReleaseDC (hWnd, hDC);
    return hBackground;
}
```

8. 在源文件中添加函数 HandleMouseMove，此函数是以 4.6 节中的函数 DoAnimation 为模板来实现的，假如读者并没有阅读 4.6 节中函数，那么也没有关系，可以仔细研究本节函数是如何实现的。

此函数首先检查是否先前将背景存储到了位图中。如果是，则以参数为 SRCCOPY 调用函数 BitBlt 来恢复原来的背景。假如现在能够停止例子程序的运行，则可以看到位图块从屏幕上已经消失了。接着此函数计算位图块的新位置，使用鼠标的新位置来调整位图块的左坐

标和上坐标。在变量 xoffset 和 yoffset 中存储着鼠标在位图块中的位置，变量 mouseX 和 mouseY 则用来存储鼠标在窗口中的位置。接着调整变量 saveTop、saveLeft、saveWidth 以及 saveHeight 的值，以来记录位图块的新位置。已经考虑到了部分位图超出窗口不可见的情况。

接着此函数调用函数 CaptureBackground，拷贝位图块新位置处的背景，最后调用 API 函数 BitBlt 将位图块拷贝到窗口中。应该注意，在存储背景和绘制位图时使用了坐标 saveTop、saveLeft、saveWidth 以及 saveHeight。

另外还应该注意到，在此段代码中假设了位图块是一个不透明的矩形。如果位图块有孔，或者形状是一个圆形或椭圆形，则将需要对位图进行透明处理，以保持位图下的某些背景。可以仔细阅读 4.6 节中的例子程序，其中有一个非常有用的函数 TransparentBlt。

```
/*------------------------------------------------------------------*/
/*  This function performs the actual display of the tile bitmap.   */
/*  It follows these steps: 1. Put back the saved background        */
/*                          2. Update the top/left positions        */
/*                          3. Save the background                  */
/*                          4. Blit the bitmap                      */
/*------------------------------------------------------------------*/
void HandleMouseMove (HWND hWnd, LONG mouseX, LONG mouseY)
{
    HDC             hDC, hMemoryDC;
    HBITMAP         hOldBitmap;
    RECT            rcClient;

    hDC = GetDC (hWnd);
    hMemoryDC = CreateCompatibleDC (hDC);

    // If the background was saved, restore it.
    if (hBackground)
     {
        hOldBitmap = SelectObject (hMemoryDC, hBackground);
        BitBlt (hDC, saveLeft, saveTop, saveWidth, saveHeight,
                hMemoryDC, 0, 0, SRCCOPY);
        SelectObject (hMemoryDC, hOldBitmap);
    }

    /* Calculate the new top and left coordinates.
       Adjust for where the mouse pointer is on the bitmap.
       Deal with the possibility that the bitmap extends
       outside the window client area. */
    left = mouseX - xOffset;
    top = mouseY - yOffset;
    GetClientRect (hWnd, &rcClient);
    saveTop = max (top, rcClient.top);
```

```
    saveLeft = max (left, rcClient.left);
    if (top >= rcClient.top)
      saveHeight = min (height, rcClient.bottom - saveTop);
    else
      saveHeight = height - top;
    if (left >= rcClient.left)
      saveWidth = min (width, rcClient.right - saveLeft);
    else
      saveWidth = width + left;

    // Now capture the background at the new location.
    hBackground = CaptureBackground (hWnd, saveLeft, saveTop,
                                     saveWidth, saveHeight,
                                     hBackground);

    // Finally, display the object at that location.
    hOldBitmap = SelectObject (hMemoryDC, hBitmap);
    BitBlt (hDC, saveLeft, saveTop, saveWidth, saveHeight,
          hMemoryDC, saveLeft - left, saveTop - top, SRCCOPY);
    SelectObject (hMemoryDC, hOldBitmap);

    DeleteDC (hMemoryDC);
    ReleaseDC (hWnd, hDC);
}
```

9. 当用户在窗口的用户区按下鼠标左键时，下面的函数将被调用，如果光标在图象上，则拖放操作开始。下列代码首先比较鼠标的位置（包含在参数 mouseX 和 mouseY 中）和位图图象的位置，如果鼠标在图象外点击，则返回 FALSE，如果鼠标点击在图象内，则拖放操作开始。变量 xoffset 和 yoffset 用来存储鼠标光标在图象中的偏移量。

由于本例子程序中的拖放操作只用来处理窗口内的移动，所以调用函数 ClipCursor 来限制光标在窗口内。函数 ClipCursor 需要以屏幕坐标表示的矩形，所以调用函数 GetClientRect 来获取用户区的坐标，接着调用函数 ClientToScreen 将用户坐标转换为屏幕坐标。由于函数 ClientToScreen 只能转换一个点，所以需要调用此函数两次，分别来转换矩形的左上点和右下点。

```
/*-------------------------------------------------------------------*/
/* This function checks to see if the mouse has been clicked on the   */
/* bitmap. If it has, it limits the mouse to movement within the      */
/* client area of the window, ensuring that the user does not drag    */
/* the bitmap outside the window. In a commerical application you     */
/* could allow the user to move outside, but test on each MouseMove   */
/* message and display a different pointer when the cursor is not     */
/* inside the window.                                                 */
/*-------------------------------------------------------------------*/
```

```
BOOL StartDragging (HWND hWnd, LONG mouseX, LONG mouseY)
{
    RECT        clientRectangle;
    POINT       point;

    // First check to see if the mouse was clicked in the image.
    if ( (mouseX < saveLeft) |
         (mouseX > (saveLeft + saveWidth)) ||
         (mouseY < saveTop) ||
         (mouseY > (saveTop + saveHeight)))
         return FALSE;

    xOffset = mouseX - left;
    yOffset = mouseY - top;

    /* Get the client rectangle, and convert it to screen
       coordinates. Convert top and left, then bottom and right. */
    GetClientRect (hWnd, &clientRectangle);
    point.x = clientRectangle.left;
    point.y = clientRectangle.top;
    ClientToScreen (hWnd, &point);
    clientRectangle.left = point.x;
    clientRectangle.top = point.y;
    point.x = clientRectangle.right;
    point.y = clientRectangle.bottom;
    ClientToScreen (hWnd, &point);
    clientRectangle.right = point.x;
    clientRectangle.bottom = point.y;

    // Keep the mouse inside the client area.
    ClipCursor (&clientRectangle);

    return TRUE;
}
```

10. 现在添加函数 Paint，每次绘制窗口时都将调用此函数。在窗口被改变大小、极大化、恢复或其他窗口移开时，Windows 95 都将发送消息 WM_PAINT 给窗口，此函数就是用来响应这些事件，调用函数 BitBlt 将位图块绘制在窗口的背景上。另外，在响应第一个 WM_PAINT 消息时，此函数在绘制图象前捕捉背景位图，从而保证了第一次移动位图块后可以正确的恢复背景。

```
/*-------------------------------------------------------------------*/
/* This function performs painting. It is done when not dragging.    */
```

```
/*------------------------------------------------------------------------*/
void Paint (HWND hWnd)
{
    HDC                     hDC, hMemoryDC;
    PAINTSTRUCT             ps;
    RECT                    rcClient;
    HBITMAP                 hOldBitmap;

    BeginPaint (hWnd, &ps);

    // If necessary, update the saveWidth and saveHeight.
    GetClientRect (hWnd, &rcClient);
    if (top >= rcClient.top)
      saveHeight = min (height, rcClient.bottom - saveTop);
    else
      saveHeight = height + top;
    if (left >= rcClient.left)
      saveWidth = min (width, rcClient.right - saveLeft);
    else
      saveWidth = width + left;

    // If it is the first paint, store the background.
    if (firstPaint)
     {
        hBackground = CaptureBackground (hWnd, left, top,
                                         width, height, hBackground);
        firstPaint = FALSE;
    }

    // Get a DC and draw the bitmap.
    hDC = GetDC (hWnd);
    hMemoryDC = CreateCompatibleDC (hDC);
    hOldBitmap = SelectObject (hMemoryDC, hBitmap);
    BitBlt (hDC, saveLeft, saveTop, saveWidth, saveHeight,
          hMemoryDC, saveLeft - left, saveTop - top, SRCCOPY);
    SelectObject (hMemoryDC, hOldBitmap);
    DeleteDC (hMemoryDC);
    ReleaseDC (hWnd, hDC);

    EndPaint (hWnd, &ps);
}
```

11. 下一个要添加的函数是 MainWndProc。此函数是一个回调函数，用来处理传送给窗口的消息。为了响应消息 WM_CREATE，此函数调用 API 函数 LoadBitmap 来从资源中装

入16色的位图，接着调用函数GetObject获取位图的宽度和高度信息。这些信息存放在全局变量中，在存储背景、检测鼠标光标是否在图象内以及向屏幕绘制图象时都将用到这些信息。

为了响应消息WM_LBUTTONDOWN，此消息在用户按下鼠标左键时发送，此函数调用函数StartDragging，限制了光标只能在窗口用户区内移动。相反，为了响应消息WM_LBUTTONUP，此函数将标志isDragging置为FALSE，以NULL为参数调用函数ClipCursor，使得光标在桌面上可以重新自由地移动。为了响应消息WM_DESTROY，用来存放图象和背景的位图将被释放。

```
/*----------------------------------------------------------*/
/* This function is the main window callback function which will be     */
/* called by Windows to process messages for our window.                */
/*----------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    BITMAP bmInfo;

    switch (message)
     {
        case WM_CREATE:
             hBitmap = LoadBitmap (hInstance,
                                   MAKEINTRESOURCE (IDBM_IMAGE));
             GetObject (hBitmap, sizeof (BITMAP), &bmInfo);
             saveWidth = width = bmInfo.bmWidth;
             saveHeight = height = bmInfo.bmHeight;
             saveLeft = left = 0;
             saveTop = top = 0;
             InvalidateRect (hWnd, NULL, TRUE);
             break;
        case WM_LBUTTONDOWN:
             if (! dragging)
                 dragging = StartDragging (hWnd, (LONG) LOWORD (lParam),
                                           (LONG) HIWORD (lParam));
             break;
        case WM_LBUTTONUP:
             if (dragging)
             {
                 // Set the mouse free.
                 ClipCursor (NULL);
                 dragging = FALSE;
             }
             break;
```

```
        case WM_MOUSEMOVE:
            if (dragging)
                HandleMouseMove (hWnd, (LONG) LOWORD (lParam),
                                    (LONG) HIWORD (lParam));
            break;
        case WM_PAINT:
            if (! dragging)
            {
                Paint (hWnd);
                break;
            }
            else
                return DefWindowProc (hWnd, message, wParam, lParam);
        case WM_DESTROY:
            if (hBitmap)
                DeleteObject (hBitmap);
            if (hBackground)
                DeleteObject (hBackground);
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

12. 在源文件中添加下列两个函数。函数 InitApplication 用来登记主窗口的窗口类，只有在此窗口类不存在时此函数才被调用。特别应该注意函数 CreateBrushIndirect 的调用，此函数用来创建淡黄色的画刷。此函数的参数为 LOGBRUSH 结构，在下列代码中被初始化成风格为 BS_SOLID，颜色的 RGB 值为淡黄色。在类的最后一个窗口被关闭时，函数 CreateBrushIndirect 创建的画刷将被 Windows 释放。

启动例子程序的每个实例时都将调用函数 InitInstance。此函数创建例子程序的主窗口，并调用函数 ShowWindow 和 UpdateWindow 来将窗口显示出来。

```
/*--------------------------------------------------------------------*/
/* This function initialises a WNDCLASS structure and uses it to      */
/* register a class for our main window.                              */
/*--------------------------------------------------------------------*/
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;
    LOGBRUSH  lb;
```

```
    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.lpszMenuName =  NULL;
    wc.lpszClassName = MainWindowClassName;
    lb.lbStyle = BS_SOLID;
    lb.lbColor = RGB (255, 255, 128); /* Light Yellow brush */
    wc.hbrBackground = CreateBrushIndirect (&lb);

    return RegisterClass (&wc);
}

/*------------------------------------------------------------------------*/
/* This function creates an instance of our main window. The window       */
/* is given a class name and a title, and told to display anywhere.       */
/* the nCmdShow argument passed to the program determines how             */
/* the window will be displayed.                                          */
/*------------------------------------------------------------------------*/
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;      /* Store in global variable */

    hWnd = CreateWindow (MainWindowClassName," Drag the tile!",
                        WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_
USEDEFAULT,
                        NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);    /* Send a WM_PAINT message */
    return TRUE;
}
```

13. 例子程序的最后一段代码是函数 WinMain，在源文件末添加下列代码，Windows 就是调用此函数来启动例子程序的。在函数 WinMain 中，调用函数 FindWindow 来检查是否有此类的窗口存在，如果没有，则调用函数 InitApplication 来登记窗口类，接着调用函数 InitIn-

stance 来创建主窗口并处理消息直到例子程序结束。

```
/*--------------------------------------------------------------------*/
/* The main entry point for Windows applications. Check to see if      */
/* the main window class name has already been registered, if it has   */
/* not, call InitApplication to register it. Call InitInstance to       */
/* create an instance of our main window, then pump messages until     */
/* the application is closed.                                          */
/*--------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

14. 编译并运行此例子程序。

注释

本节的例子程序示范了一个相当可靠的实现位图拖放的方法。但是有两种可能的情况没有考虑到：位图拖出了窗口（比如到了另一窗口内）和在滚动的窗口中拖放位图，这两种可能实际上是互斥的，当用户移动鼠标到了窗口外时，用户是希望拖放位图到窗口外的某一位置还是希望到滚动窗口的某一新位置，这是不容易辨别清楚的。但是无论哪一种情况，都不会再使用函数 ClipCursor 来将光标限制在用户区内，而是使用 API 函数 SetCapture 来接收鼠标的移动信息，无论鼠标移动到了什么窗口上。

第5章 对 话 框

到目前为止，Windows 95 应用程序中使用最多的便是对话框。程序员认为，对话框是从用户那里获取数据的简单方法，因此程序能进行较复杂的处理和图示“有趣的信息”。对用户来说，对话框是应用程序的前端，直接影响用户的使用，设计很好的对话框可使用户便于使用，设计差的对话框将使用户使用非常困难。

本章将介绍对话框，利用 Windows 95 的强大功能，对话框将具有出乎意料的效果。定制的字体、颜色以及三维外观都是实现复杂的用户界面所必需的，属性单构成 Windows 95 的最主要部分，也给对话框增添了新的特色。属性单是“标签式”的对话框，可以在单个对话框中向用户显示多屏信息。在本章将介绍如何学习和掌握对话框，以便运用自如。

1. 如何利用对话框作为应用程序的主窗口

通常即使编写一个简单的工具软件，也需要实现一个完整的程序，具有菜单、多文档界面、About 对话框等等，非常的繁琐，对于用户来说，为了完成一项简单的任务而要面对一系列的菜单选择也是非常不情愿的。在本节中，将介绍如何利用 Visual C＋＋的强大功能来创建以对话框作为主窗口的应用程序，应用程序的功能没有变化，只是用户界面与标准的 Windows 95 应用程序不同。一些实用工具，如拷贝、数据库查询、向导（Wizards）都可使用此方法来创建。

2. 如何改变对话框中的字体

实际上，不同的用户有不同的喜好，其中最基本的不同之一便是喜好的字体不同。有的用户喜欢在他们的显示和对话框中使用简单的打字机字体，而有的用户则喜欢使用特殊的字体（或至少有趣的字体）来装饰他们的对话框。本节将介绍如何使用户可以设置对话框中的字体，从而使得最挑剔的用户都能满意。

3. 如何改变对话框中控制的字体

在 5.2 节中，按照用户的喜好改变字体是改变对话框中显示的全部文本。那么如何改变单个静态文本或编辑控制的字体，以区别于对话框中的其他控制呢？在本节中，将介绍如何设置对话框中单个控制的字体。

4. 如何改变对话框的背景颜色

当前对话框的背景颜色流行的是使用金属灰色，轮廓分明的边框和灰色都是 Windows 95 应用程序中的标准。也许有的程序员希望用不同的色调给对话框着色，比如漂亮的樱桃色或带有淡红色的紫色。在本节中，将介绍如何随意设置对话框的颜色，从单一的颜色到彩虹的颜色。程序员可以决定对话框背景的颜色，本节将介绍如何来实现。

5. 如何激活和禁止对话框中的控制

用户希望在对话框中的操作越少越好，并且不愿意在编辑框中填写大量的数据。如果用户在 20 多个编辑域中输入数据后却发现由于选择了其他的选项而其中只有 3 个是需要的，那么还有什么事比这更令人恼火的呢？甚至更使用户为难的是在同一个对话框中可能有互斥的选项。

在本节中，将向程序员介绍如何利用 Windows 95 API 函数并根据用户在对话框中的选择来激活或禁止对话框中的控制，这种对话框的内部逻辑可以使得应用程序更加完善和专业化，并在提交数据时可以免去大多数的检查，这些检查是为了确认对话框中的数据的有效性。

6. 如何改变对话框中显示的控制

当前专业应用程序所显示的对话框可以根据用户的选择添加或删除控制，例如：初级用户可能只想查看对话框中的简单选项，而高级用户可能希望查看所有的选项。在本节中，将介绍如何只显示用户需要的控制而将当时不需要的控制隐藏起来 。

7. 如何在对话框中使用属性单

Windows 95 与 Windows 3.x 相比，最重要的添加之一便是属性单，或称为标签对话框。属性单使得在单个对话框中可以包含更多的控制，而不会使整个屏幕混乱不堪，属性单还可以使数据分成多个逻辑单元，用户同时只需考虑数据的一小部分。从而导致用户较少的操作失败，因而开发者只需提供很少的技术支持。

8. 如何在对话框中的按钮上绘制图象

当前大多数的 Windows 95 应用程序都显示包含图标的 About 框，图标一般为极小化应用程序时的图标。无论是在 Windows 3.x 还是在 Windows 95 中，都没有独立的图标控制，那么如何实现此功能呢？在本节中，将介绍如何在应用程序中实现此功能。

9. 如何在执行另一操作的同时显示“进度”对话框

如果应用程序执行某些操作而屏幕上没有任何显示，用户会感到相当的别扭，例如：文件拷贝、打印文档、查询数据库等，这些操作只需要程序执行而不需要提供用户任何反馈。用户在等待应用程序返回屏幕时可能会非常烦躁，甚至有可能认为应用程序被挂起而终止应用程序的运行。在本节中，将介绍如何在运行长时间的任务时提供用户反馈信息，从而使用户能够愉快地等待而不会终止应用程序的运行。

表 5-1 列出了本章中使用的 Windows 95 API 函数。

表 5-1　第 5 章中使用的 Windows 95 API 函数

Post Message	PostQuitMessage	GetObject
CreateFontIndirect	MakeProcInstance	FreeProcInstance
EnumChildWindows	SendMessage	FillRect
SetBkMode	CreateSolidBrush	SetCheck
EnableWindow	IsWindowVisible	ShowWindow
SetWindowText	GetWindowLong	SetWindowLong
CreateWindowEx	PeekMessage	TranslateMessage
DispatchMessage		

5.1 利用对话框作为应用程序的主窗口

问题

有的程序员准备开发一些简单的实用工具提供给用户，但又不想使用户为了运行此实用工具对话框而不得不首先打开应用程序，然后再从菜单中选择下拉菜单项，从而运行此对话框，最后用户还必须关闭此应用程序。是否有更简单的使用对话框的方法呢？许多程序员不愿意放弃使用 Visual C++和 MFC 的强大功能，但确实又不需要它们所有的附加功能。

方法

尽管大多数程序员都清楚 Visual C++（无论是在 Windows 3.x 中还是在 Windows 95 中）的强大功能，但却有许多程序员不知道在 Visual C++中实现自己的应用程序可以不使用辅助工具 AppWizard 和 ClassWizard。在 Visual C++中不使用 AppWizard 同样可以创建新的项目文件，可以将使用的代码添加到相应的文件中。

在出现程序生成器之前的许多年里，程序员使用自己的模板，这些模板是他们为解决特定的问题而开发的应用程序框架。当需要解决的问题与模板可解决的问题类似时，程序员就以此模板为框架开发新的应用程序，添加需要的代码来解决新问题。

在本节中，我们将介绍如何使用此模板方法来创建主窗口为对话框的应用程序，另外还将介绍 MFC 在内部是如何与 Windows 95 API 协调运行的。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，屏幕上将显示出一个主对话框，此对话框既没有关联的框架窗口也没有关联的菜单，如图 5-1 所示。如果要关闭对话框，可以点击按钮 Close，从而终止对话框和例子程序。

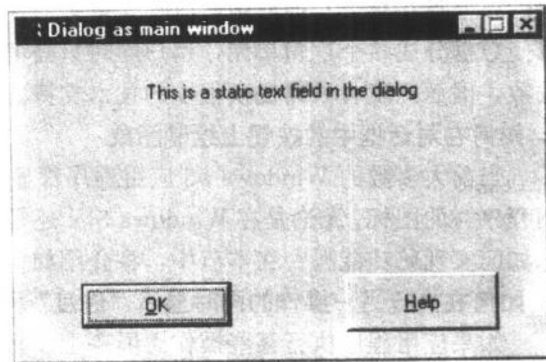


图 5-1　作为主窗口的对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，选择 File|New 来创建新的项目文件，从对话框中选择 Project 并点击按钮 OK，命名新项目文件为 CH51.MAK，点击按钮 OK，接着点击按钮 Cancel，以选择添加到此工程中的文件。

2. 点击图标 New File 或从主菜单中选择 File|New，接着从对话框中选择 Code/Text 以创建新文件。在文件中输入下列代码：

```
#ifndef __CH51_H__
#define __CH51_H__

// The dialog window class derived from CDialog.

class CDlgWin : public CDialog
{
private:

public:
    // Constructor for class

    CDlgWin ();
```

```
    // Overrides for the button handlers

   void OnOK ();
    void OnHelp ();

    // When the dialog window is closed, free the memory for
    // it since it is not the child of anything.

    void OnClose () {
        delete this;
    }

    DECLARE_MESSAGE_MAP ()

};

// The application object class

class CTheApp : public CWinApp
{
public:
    BOOL InitInstance ();
};

#endif
```

3. 保存此文件为 CH51.H。
4. 再一次选择图标 New File 或菜单 File|New，在代码编辑窗口中输入下列代码：

```
#include "afxwin.h"
#include " resource.h"
#include "ch51.h"    // Contains class definitions

// Constructor for dialog window. Simply calls the modeless create
// method to create the window based on the MAINDLG template in the
// resource file.

CDlgWin:: CDlgWin ()
{

    Create ("MAINDLG");
}
```

```
// User clicked Ok. Close the window and quit the application.

void CDlgWin:: OnOK () {
    PostMessage (WM_CLOSE);
    PostQuitMessage (0);
}

// User clicked Help. Show a message box with help text.

void CDlgWin:: OnHelp ()
{
   MessageBox ("You can display message boxes or anything else!", "Help", MB_OK );
}

BEGIN_MESSAGE_MAP (CDlgWin, CDialog)
    ON_WM_CLOSE ()
    ON_COMMAND (IDC_HELP, OnHelp)
    ON_COMMAND (IDOK, OnOK)
END_MESSAGE_MAP ()

CTheApp theApp;

// This is the entry point for the application

BOOL CTheApp:: InitInstance ()
{
    // Create a new modeless dialog box and assign
    // it to the main window pointer in the application
    // object

    m_pMainWnd = new CDlgWin ();

    // Make sure it is visible.
    m_pMainWnd_> ShowWindow (m_nCmdShow);

    // And fully updated.

    m_pMainWnd_> UpdateWindow ();
    return TRUE;
}
```

5. 保存此文件为 CH51. CPP。
6. 创建新文件并在此文件中输入下列行。保存此文件为 RESOURCE. H。

```
#ifndef _RESOURCE_H_
#define _RESOURCE_H_

#define IDC_HELP 110

#endif
```

7. 创建另外一个新文件来保存此例子程序的资源脚本，在资源文件中输入下列代码：

```
//Microsoft App Studio generated resource script.
//
#define APSTUDIO_READONLY_SYMBOLS
//////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#define APSTUDIO_HIDDEN_SYMBOLS
#include "windows.h"
#undef APSTUDIO_HIDDEN_SYMBOLS
#include "afxres.h"
#include "resource.h"

////////////////////////////////////////////////////////////////////////////////////
/////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

MAINDLG DIALOG DISCARDABLE  44, 32, 219, 122
STYLE WS_MINIMIZEBOX | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog as main window"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON      "&OK", IDOK, 26, 92, 62, 17
    PUSHBUTTON      "&Help", IDC_HELP, 136, 90, 63, 22
    LTEXT           "This is a static text field in the dialog", IDC_STATIC,
                    54, 14, 125, 15
END

#ifdef APSTUDIO_INVOKED
```

```
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resrc1.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
    "#include ""windows.h""\r\n"
    "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
    "#include ""afxres.h""\r\n"
    "#include ""resource.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

/////////////////////////////////////////////////////////////////////////////
#endif    // APSTUDIO_INVOKED

#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
// Generated from the TEXTINCLUDE 3 resource.
/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

8. 保存此文件为 CH51.RC。将这三个文件（CH51.CPP、CH51.H 和 CH51.RC）添加到工程中，并编译成可执行文件。运行此例子程序，将显示出如图 5-1 所示的对话框。

用法

当 Microsoft 基本类库（MFC）在应用程序中启动时，实例化的第一个对象是对象 Application，然后构造此对象（在本节的例子程序中为 theApp）并调用方法 InitInstance。在本节的例子程序中，方法 InitInstance 创建类 CDlgwin 的新实例，并将其赋给应用程序对象的主窗

口指针，接着在屏幕上显示并更新此对象。

在类 CWinDLg 的构造函数中，调用方法 Create 在屏幕上创建新的无模式对话框（使用 Windows API 函数 CreateDialog），函数 CreateDialog 实例化 Microsoft Windows 无模式对话框如下：

```
HWND CreateDialog (HINSTANCE hinst, LPSTR lpszDlgTemp, HWND hwndOwner, FARPROC dlgprc);
```

MFC 通过调用函数 AfxGetInstance 来获取参数 hinst，其在应用程序开始时建立。参数 lpszDlgTemp 是资源文件中的对话框模板名（在本节的例子程序中是 MAINDLG）。既然对话框为主窗口，那么也就没有属主。dlgprc 是回调函数，对话框使用此函数来进行所有处理。函数 CreateDialog 是 MFC 基类 Cdialog 的一部分。

注释

此对话框类可以作为实现某些实用程序的基础，这些实用程序提供给用户简单的交互方式来完成某些任务，另外还可作为实现独立的弹出对话框的基础，比如 Find/Replace 对话框或浮动的工具条。

5.2 改变对话框中的字体

问题

有的程序员希望能够改变对话框中的字体，以适应用户的喜好，希望对话框中所有的控制（编辑框、静态文本域、按钮以及组合框）都以选中的字体显示，而不论对话框初始定义的是什么字体。如何使用 Windows 95 API 函数来完成此目标呢？

方法

对于大多数程序员，首先想到的是在手册中查找一个函数来设置对话框中的字体，当他发现方法 SetFont 清楚地记录在基类库的文档中时，可以想象他会多么激动。但是，当他高兴地在对话框初始化函数中调用 SetFont 并发现没有任何效果时，也可以想象他是多么震惊！

MFC 使用自己的方法来设置对话框中的字体，即使用资源语句 FONT 说明的字体，或是使用 Windows 95 中缺省的系统字体来定义所有的控制，所有的控制也都是使用这种字体。那么如何超越 MFC 的这些限制，从而使得控制使用用户自己的字体呢？答案在于协调使用几个不同的 Windows 95 API 函数。

在本节中，首先介绍如何通过显示通用字体对话框来创建字体，此通用对话框提供使用 GDI 函数创建字体所需要的信息。接着，将介绍如何使用函数 EnumChildWindows 来遍历对话框中的所有控制。最后，将介绍如何使用函数 SendMessage 来改变窗口显示所用的字体。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Fonts，从下拉菜单中选择菜单项 Change Current Font，通用字体对话框将显示出来，如图 5-2 所示，选择字体名、风格（规则的、粗体的、斜体的）和字号，点击按钮 OK，选择菜单 Fonts，从下拉菜单中选择菜单项 Display Dialog，弹出的对话框将以选中的字体显示。以选中的 Roman、Regular、10 磅字体显示的对话框如图 5-3 所示。

实现例子程序的具体步骤如下：

1. 在 Viusal C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH52.MAK。

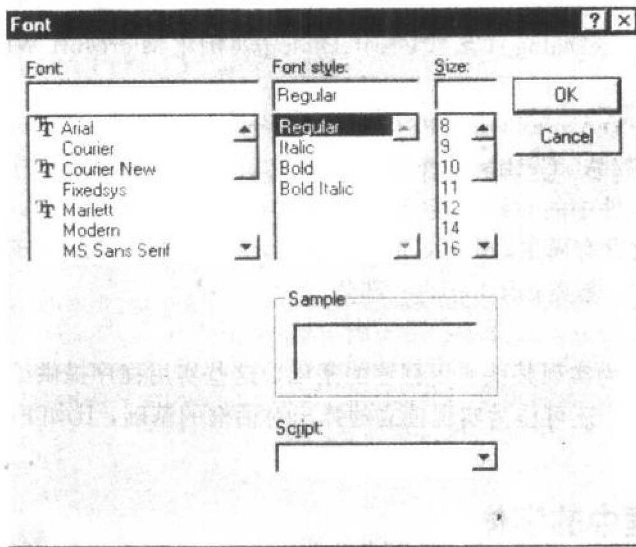


图 5-2 通用字体对话框

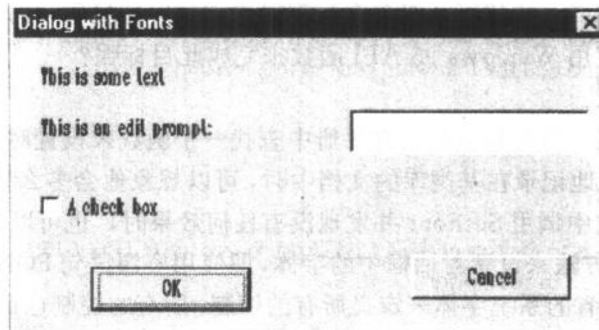


图 5-3 10 磅 Roman 字体显示的对话框

2. 选择应用程序对象的头文件 CH52.H，在头文件中添加下列行（黑体表示）：

```
class CCh52App : public CWinApp
{
private:

CFont *current_font;    // Font used for dialogs

public:
  CCh52App ();
  virtual _CCh52App ();
  CFont *GetCurrentFont (void) { return current_font; };
```

```
// Overrides
    virtual BOOL InitInstance ();
```

3. 进入 AppStudio，并从菜单列表中选择主菜单。添加新的菜单，标题为 Fonts，添加标题分别为 Change Current Font 和 Display Dialog 及标识符分别为 ID_CHANGE_FONT 和 ID_FONT_DLG 的两个下拉菜单项。

4. 进入 ClassWizard，从下拉列表中选择对象 CCh52App，从对象列表中选择标识符 ID_CHANGE_FONT，从消息列表中选 COMMAND，点击按钮 Add Function，新方法命名为 OnChangeFont。在此方法中添加下列代码：

```
void CCh52App:: OnChangeFont ()
{
  // Get the information from the existing font

  CFont * pFont = current_font;

  // Initialize font structure

  LOGFONT lf;
  memset ( &lf, 0, sizeof (lf) );

  // Get the font structure information out of the font object

  if ( pFont )
      pFont -> GetObject (sizeof (LOGFONT), &lf);

  // Display a common dialog box for fonts with this default information

  CFontDialog dlg (&lf, CF_SCREENFONTS|CF_INITTOLOGFONTSTRUCT);
  if (dlg.DoModal () ! = IDOK)
      return;
  // Create a new font based on the user request.

  CFont *tempFont = new CFont;
  if ( ! tempFont -> CreateFontIndirect ( &lf ) ) {
      delete tempFont;
    MessageBox (NULL, "Unable to create font!", "Error", MB_OK );
      return;
  }

  // If there is an existing font, delete it and replace it

  if ( current_font )
      delete current_font;
```

```
    current _ font = tempFont;

}
```

5. 在 ClassWizard 中，从下拉列表中再次选择对象 CCh52App，从对象列表中选择 ID _ FONT _ DLG，从消息列表中选择 COMMAND，点击按钮 Add Function，方法命名为 OnFontDlg。在此方法中添加下列代码：

```
void CCh52App:: OnFontDlg ()
{
    CDlgFont dlg;
    dlg. DoModal ();
}
```

6. 在 CCh52App 的源文件（CH52. CPP）的顶部添加下列行，要添加的行用黑体表示：

```
#include "ch52view. h"
#include "dlgfont. h"

#ifdef _DEBUG
#undef THIS _ FILE
static char BASED _ CODE THIS _ FILE [] = __FILE__;
#endif
//////////////////////////////////////////////////////////////////////////////
// CCh52App

BEGIN _ MESSAGE _ MAP (CCh52App, CWinApp)
    //{{AFX _ MSG _ MAP (CCh52App)
    ON _ COMMAND (ID _ APP _ ABOUT, OnAppAbout)
    ON _ COMMAND (ID _ CHANGE _ FONT, OnChangeFont).
    ON _ COMMAND (ID _ FONT _ DLG, OnFontDlg)
    //}} AFX _ MSG _ MAP
    // Standard file based document commands
    ON _ COMMAND (ID _ FILE _ NEW, CWinApp:: OnFileNew)
    ON _ COMMAND (ID _ FILE _ OPEN, CWinApp:: OnFileOpen)
    // Standard print setup command
    ON _ COMMAND (ID _ FILE _ PRINT _ SETUP, CWinApp:: OnFilePrintSetup)
END _ MESSAGE _ MAP ()

//////////////////////////////////////////////////////////////////////////////
// CCh52App construction

CCh52App:: CCh52App ()
```

```
{
  current_font = NULL;
}

CCh52App::~CCh52App()
{
  if ( current_font )
      delete current_font;
}
```

7. 进入 AppStudio 并创建新的对话框。以图 5-3 所示的对话框为模板，在对话框中添加任意的域。当完成后，保存此对话框并选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，并命名此类为 CDlgFont。

8. 在ClassWizard中，从下拉列表中选择对象CDlgFont，从对象列表中选择CDlgFont，从消息列表中选择 WM_INITDIALOG，点击按钮 Add Function，在 CDlgFont 的方法 OnInitDialog 中添加下列代码：

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

  // Make sure the window is centered on our application window
  CenterWindow();

  // Get the current defined font..

  CCh52App *app = (CCh52App *) AfxGetApp();
  CFont *font = app->GetCurrentFont();

  // If there was no font defined, just get out of here..
  if ( font == NULL )
     return TRUE;

  // Make the callback function

  FARPROC proc = MakeProcInstance( (FARPROC) SetFontProc,
                                   AfxGetInstanceHandle());

  // Loop through the child windows, setting the fonts as we go.

  EnumChildWindows(m_hWnd, (WNDENUMPROC) proc, (LPARAM) font->m_hObject );
```

```
  // Free the callback function memory

  FreeProcInstance ( proc );

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

9. 在 CDlgFont 的方法 OnInitDialog 前添加下列函数。

```
BOOL CALLBACK SetFontProc (HWND hwnd, LPARAM lParam)
{
   // Get the font handle from the lParam

   HFONT font = (HFONT) lParam;

   // Set the window font

   SendMessage ( hwnd, WM_SETFONT, (WPARAM) font, (LPARAM) MAKELONG ( (WORD)
TRUE, 0) );

   return TRUE;

}
```

10. 编译并运行此例子程序。

用法

当用户选择菜单项 Change Current Font 时，例子程序将显示出一个包含当前选中字体（如果有的话）的字体对话框，当用户从字体对话框中选择字体后，当前选中的字体就被用来创建新的 CFont 对象。这都是在成员函数 CCh52App::OnChangeFont 中实现的。

当用户选择菜单项 Display Dialog 时，例子程序将创建类 CFontDlg 的新实例。此对象从应用程序对象中获取当前的字体，接着调用函数 EnumChildWindows 来以当前的字体修改每个控制，函数 EnumChildWindows 遍历给定父窗口的每个子窗口。在本节的例子程序中，父窗口是对话框，所有的子窗口是对话框中的控制。函数 EnumChildWindows 的参数有窗口句柄（指定要遍历的父窗口）、回调函数、以及要传送给回调函数的参数。在本节的例子程序中，传送给回调函数的参数是字体句柄。

当每个窗口传送给回调函数（SetFontProc）时，就调用 SendMessage 将字体句柄传送给子窗口。在此调用中，命令是 WM_SETFONT，传送的信息是字体句柄和标志（指定控制应该立即重绘）。

当以指定的字体更新了所有的控制后，函数将控制交回 Windows 系统，从而以新字体在屏幕上绘制对话框。

5.3 改变对话框中控制的字体

问题

有的程序员希望能够随时改变对话框中控制的字体，特别地，有的程序员希望利用对话框中的单个静态文本字符串来向用户示范不同的字体。在上一节中，是在初始化整个对话框以及所有控制时改变字体，因此字体的改变是静态的。但在本节中，将介绍如何随时动态地改变控制的字体。

方法

在应用程序中，一个普遍使用的选择字体的方法是向用户显示示范文本。但有时使用通用字体对话框来实现此功能并不合适，有的用户希望查看字体的几种风格混合使用的效果，或几种不同的字体混合使用的效果。要完成此任务，就需要能够设置单个静态文本域的字体，而不影响对话框中的其他域。在本节中，将介绍如何实现此功能。

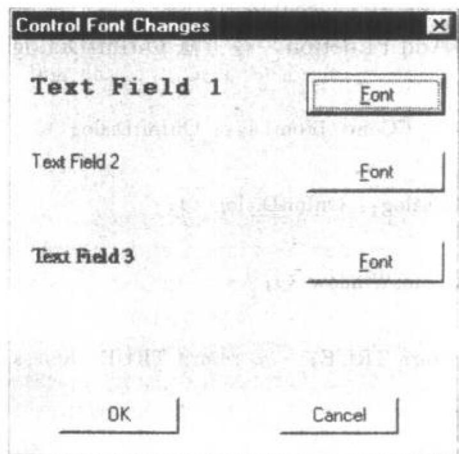


图 5-4 改变指定对话框控制的字体

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Fonts 和菜单项 Change Control Fonts，将弹出一个对话框（如图 5-4 所示），点击任一按钮 Font，在通用字体对话框中改变字体，则与按钮相对应的文本域将改变字体，以反映新的字体选择。

实现例子程序的具体步骤如下：

1. 在 Visual C＋＋中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH53.MAK。进入 AppStudio 并添加新的对话框，在对话框中添加三个静态文本域，标题分别为 Text Field 1、Text Field 2 和 Text Field 3，与文本域水平对齐添加三个按钮，每个按钮的标题都为 Font。

2. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，并命名此对话框类为 CControlFontDlg，选择对象 IDC_BUTTON1 和消息 COMMAND，点击按钮 Add Function，并命名方法为 OnTextField1。在方法 OnTextField1 中添加下列代码：

```
void CControlFontDlg:: OnTextField1 ()
{
  DoFont (1);
}
```

3. 在 ClassWizard 中，选择对象 IDC_BUTTON2 和消息 COMMAND，点击按钮 Add Function，并命名方法为 OnTextField2。在方法 OnTextField2 中添加下列代码：

```
void CControlFontDlg:: OnTextField2 ()
{
```

```
  DoFont (2);
}
```

4．在ClassWizard中，选择对象IDC_BUTTON3和消息COMMAND，点击按钮Add Function，并命名方法为OnTextField3。在方法OnTextField3中添加下列代码：

```
void CControlFontDlg::OnTextField3 ()
{
   DoFont (3);
}
```

5．在ClassWizard中，选择对象CControlFontDlg和消息WM_INITDIALOG，点击按钮Add Function，在方法OnInitDialog中输入下列代码：

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

  CenterWindow ();

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

6．最后，在CControlFontDlg的源文件（CONTROLF.CPP）中添加下列方法的定义：

```
void CControlFontDlg::DoFont ( int whichField )
{
   CFont *font = NULL;

   switch ( whichField ) {
      case 1:
         font = font1;
         break;
      case 2:
         font = font2;
         break;
      case 3:
         font = font3;
         break;
   }

   // Initialize font structure

   LOGFONT lf;
```

```
memset ( &lf, 0, sizeof (lf) );

// Get the font structure information out of the font object

if ( font )
    font -> GetObject (sizeof (LOGFONT), &lf);
// Display a common dialog box for fonts with this default information

CFontDialog dlg (&lf, CF_SCREENFONTS|CF_INITTOLOGFONTSTRUCT);
if (dlg.DoModal () != IDOK)
    return;

// Build the new font from this information

CFont *tempFont = new CFont;
if ( ! tempFont -> CreateFontIndirect ( &lf ) ) {
    delete tempFont;
    MessageBox ("Unable to create font!", "Error", MB_OK );
    return;
}

// If there is an existing font, delete it and replace it

if ( font )
    delete font;

switch ( whichField ) {
    case 1:
        font1 = tempFont;
        break;
    case 2:
        font2 = tempFont;
        break;
    case 3:
        font3 = tempFont;
        break;
}

// Now, update the text control..

GetDlgItem (IDC_TXT_FLD1+whichField-1) -> SendMessage (WM_SETFONT,
(WPARAM) tempFont -> m_hObject, (LPARAM) MAKELONG ( (WORD) TRUE, 0) );
}
```

7. 在对话框类的构造函数 CFontControlDlg::CFontControlDlg 中做下列修改（用黑体表示）。

```
CControlFontDlg::CControlFontDlg (CWnd * pParent /* =NULL */)
    : CDialog (CControlFontDlg::IDD, pParent)
{
  font1 = NULL;
  font2 = NULL;
  font3 = NULL;
    //{{AFX_DATA_INIT (CControlFontDlg)
    // NOTE: the ClassWizard will add member initialization here
    //}} AFX_DATA_INIT
}
```

8. 在头文件 CONTROLF.H 中，在类的定义中添加下列说明：

```
private:
  CFont *font1;
  CFont *font2;
  CFont *font3;
protected:
void DoFont ( int whichField );
```

9. 在 AppStudio 中，在主菜单 Fonts 中添加新的下拉菜单项，此菜单项的标题为 Change Control Fonts，标识符为 ID_FONTS_CHANGECONTROLFONTS，进入 ClassWizard，选择应用程序对象 CCh53App，点击对象标识符 ID_FONTS_CHANGECONTROLFONTS 和消息 COMMAND，点击按钮 Add Function，并命名函数为 OnChangeControlFont。在函数 OnChangeControlFont 中添加下列代码：

```
void CCh53App::OnChangeControlFont ()
{
  CControlFontDlg dlg;
  dlg.DoModal ();
}
```

10. 在文件 CH53.CPP 的顶部，include 文件 CH53DOC.H 和 CH53VIEW.H 的 include 语句的后面，添加下列行：

```
#include "controlf.h"
```

11. 编译并运行此例子程序。

用法

当用户从对话框中选择某个字体按钮时，对话框类的处理函数就使用对话框中此位置当前定义的字体（如果有的话）来创建字体对话框，字体对话框接着返回例子程序一个字体结构，以用来构造新的字体对象。例子程序像前一节一样使用此字体句柄，发送消息 WM _ SETFONT（包含标识符 WM _ SETFONT 和字体句柄）给相应的控制。

注释

此方法可以用来改变对话框中任何窗口对象的字体，在 Delphi 中能够以相当少的代码来实现同样的功能。下面便是实现例子程序的具体步骤：

1. 在 Delphi 中创建新的项目文件，或在现有的项目文件中添加新的表单，在此表单中添加三个静态文本域和三个按钮，文本域的标题分别为 TextField1、TextField2 和 TextField3，按钮的标题都为 Font，同时在表单中添加对象 Font Dialog，最后，在表单的底部添加标题为 Close 的按钮。

2. 双击第一个 Font 按钮，在处理程序中添加下列代码：

```
procedure TForm1. Button1Click (Sender: TObject);
begin
  FontDialog1. Execute;
  Label1. Font. Assign (FontDialog1. Font);
end;
```

3. 双击第二个 Font 按钮，在处理程序中添加下列代码：

```
procedure TForm1. Button2Click (Sender: Tobject);
begin
  FontDialog1. Execute;
  Label2. font. Assign (FontDialog1. Font);
end;
```

4. 双击第三个 Font 按钮，在处理程序中添加下列代码：

```
procedure TForm1. Button3Click (Sender: TObject);
begin
  FontDialog1. Execute;
  Label3. Font. Assign (FontDialog1. Font);
end;
```

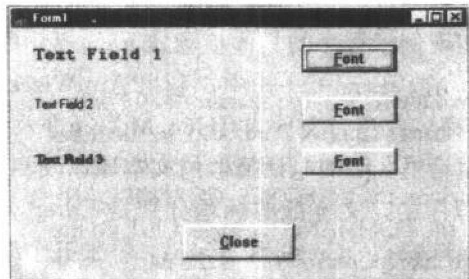


图 5-5 用来改变对话框中特定控制字体的 Delphi 表单

5. 最后，双击按钮 Close，在处理程序中添加下列代码：

```
procedure TForm1. Button4Click (Sender: TObject);
```

```
begin
  Close;
end;
```

6. 编译并运行对话框，点击按钮 Font，随意改变字体。最后的对话框应该如图 5-5 所示。

5.4 改变对话框的背景颜色

问题

尽管 Visual C++和 Windows 95 提供的颜色既好又漂亮，但有些用户总感到对话框的灰色背景不容易辨别。是否有方法改变对话框的背景颜色（比如改变为亮绿色）呢？

方法

尽管亮绿色可能会使用户的注意力分散，但它可以使任何应用程序充满生机。在本节中，将介绍如何按需求设置对话框的背景颜色。

也可以修改此方法以便在对话框的背景上显示位图。当 Windows 95 要求对话框显示窗口背景时，发送消息 WM_ERASEBKGND 给对话框，此消息提示程序员此时该显示窗口背景了，程序员就可以按自己的喜好定制对话框的背景。如何来完成这些功能呢？当然是利用 Windows 95 API。

在本节中，我们将发现 Visual C++生成的代码并非是“神圣不可侵犯的”，程序员可以不使用 ClassWizard 而直接修改消息表和重置函数。在本节的例子程序中，我们正是这么做的，这是因为 ClassWizard 不允许使用所有的窗口消息。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Colors 和菜单项 Dialog Background，将弹出一个地址簿似的对话框，如图 5-6 所示。对话框的背景是眩目的淡绿色。

图 5-6 绿色背景的对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH54.MAK。进入 AppStudio 并添加新的对话框，添加电话簿似的文本域以及相应的编辑域。在本例子程序中，文本域的标题分别是 Enter your name:、Enter your address:、City: 和 State:，添加五个编辑域，名字一个，地址两个，城市和国家各一个。

2. 选择 ClassWizard，为此模板生成新的对话框类，并命名此类为 CPhoneDlg。关闭 AppStudio 和 ClassWizard。

3. 从项目列表中选择源文件 PHONEDLG.CPP，在此文件中添加下列函数：

```
BOOL CPhoneDlg::OnEraseBkgnd (CDC * pDC)
{
    // Get the window rectangle to fill...

    CRect r;
```

```
    GetClientRect ( &r );

    // Fill in the background with the correct color

    pDC -> FillRect ( &r, &color _ brush );

    return TRUE;        // No more background painting needed
}

HBRUSH CPhoneDlg:: OnCtlColor (CDC * pDC, CWnd * pWnd, UINT nCtlColor)
{
    pDC -> SetBkMode ( TRANSPARENT );
    return m _ brush;
}
```

4. 在类的构造函数（CPhoneDlg:: CPhoneDlg）中做下列修改（用黑体表示）：

```
CPhoneDlg:: CPhoneDlg (CWnd * pParent /* =NULL */)
       : CDialog (CPhoneDlg:: IDD, pParent)
{
   m _ brush = (HBRUSH) GetStockObject ( HOLLOW _ BRUSH );
   color _ brush. CreateSolidBrush ( RGB (0, 254, 0) );
       //{{AFX _ DATA _ INIT (CPhoneDlg)
              // NOTE: the ClassWizard will add member initialization here
       //}} AFX _ DATA _ INIT
}
```

5. 通过添加下列行（黑体表示）修改类的消息映射项。

```
BEGIN _ MESSAGE _ MAP (CPhoneDlg, CDialog)
       //{{AFX _ MSG _ MAP (CPhoneDlg)
              // NOTE: the ClassWizard will add message map macros here
    ON _ WM _ ERASEBKGND ()
    ON _ WM _ CTLCOLOR ()
       //}} AFX _ MSG _ MAP
END _ MESSAGE _ MAP ()
```

6. 接着，从项目列表中打开 include 文件 PHONEDLG. H，在此文件中添加下列行（黑体表示）：

```
class CPhoneDlg : public CDialog
{
private:
```

```
  CBrush color_brush;
  HBRUSH m_brush;           // Handle of background brush
// Construction
public:
      CPhoneDlg (CWnd * pParent = NULL);      // standard constructor
// Dialog Data
      //{{AFX_DATA (CPhoneDlg)
      enum { IDD = IDD_DIALOG3 };
            // NOTE: the ClassWizard will add data members here
      //}} AFX_DATA

// Implementation
protected:
      virtual void DoDataExchange (CDataExchange * pDX);       // DDX/DDV support
      // Generated message map functions
      //{{AFX_MSG (CPhoneDlg)
      // NOTE: the ClassWizard will add member functions here
      afx_msg BOOL   OnEraseBkgnd (CDC * pDC);
      afx_msg HBRUSH OnCtlColor (CDC * pDC, CWnd * pWnd, UINT nCtlColor);
      //}} AFX_MSG
      DECLARE_MESSAGE_MAP ()
};
```

7. 重新进入 AppStudio，添加新的主菜单 Colors，在此菜单中添加新的下拉菜单项 Dialog Background，并设置此下拉菜单项的标识符为 ID_DLG_BKGD。

8. 启动 ClassWizard 并选择应用程序对象 Ch54App，从对象列表中选择对象 ID_DLG_BKGD，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名此函数为 OnDlgBkgd。在函数 OnDlgBkgd 中添加下列代码：

```
void CCh54App:: OnDlgBkgd ()
{
   CPhoneDlg dlg;
   dlg.DoModal ();
}
```

9. 在文件 CH54.CPP 中，所有其他"#include"语句的下面，添加下列 include 文件行：

```
#include "phonedlg.h"
```

10. 编译并运行此例子程序。

用法

当 Windows 95 初始化窗口并将其显示在屏幕上时，发送消息给窗口（或对话框）以绘制

它的背景。如果程序员没有重置此消息，那么就使用窗口类创建时定义的画刷来绘制背景。对于对话框，缺省的颜色通常是灰色，但对于其他窗口类，则可能是其他的颜色。

本节中的对话框只是重置了消息 WM_ERASEBKGND，此消息用来抹除对话框的背景，此消息要求返回标志给操作系统，以指示此消息是否已被处理。在本节的例子程序中，我们返回 TRUE 给操作系统。

5.5 激活和禁止对话框中的控制

问题

有的程序员希望能够根据用户的输入有选择地激活或禁止对话框中的控制。例如，如果用户选择了两个互斥选项中的一个，那么程序员就希望禁止对应于另外一个选择的所有选项。相反，如果用户改变了主意，那么程序员就需要能够重新激活这些选项并禁止对应于第一次选择的所有选项。如何以最少的精力和时间利用 Windows 95 API 来完成此目标呢?

方法

在对话框中具有选择激活和禁止控制的能力是考虑周密和有良好设计的程序所必备的，它们与只是单纯接收用户输入来执行作业的程序是有区别的。用户有可能知道也有可能不知道哪些是“互斥选择”，决定哪些选项是相容的哪些选项是不相容的是程序员的责任，而且程序员还应该防止用户选择互斥的选项。

当面对满是互斥选项的对话框时，程序员可以有两种基本方法来控制对话框的选择结果。第一种方法是：当用户要保存他的选择时，如果发现有互斥的选项选中，则显示错误。此方法有两个问题：第一，用户可能不知道为什么不能这样选择，而且也不容易找出能够相容的选项。第二，操作失败的用户会非常迷惑：“为什么应用程序知道不能这样选择而却让我选择呢?”

第二种方法是，当一个选项被选中后，禁止所有不相容的选项，从而防止用户选中互斥的选项，当然，如果用户取消此选项的选择，或选择另外一个选项，那么控制就必须更新，以反映新的选择。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Controls 和下拉菜单项 Enable and Disable，将弹出一个对话框，如图 5-7 所示。对话框顶部的核选框是整个对话框的关键，如果选中某个核选框，则相应的控制集（编辑域、组合框及核选框）被激活以允许用户输入，如果核选框未选中，则相应的控制集被禁止并且不允许用户输入。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH55.MAK。进入 AppStudio 并创建新的对话框，在此对话框中，首先在顶部添加三个核选框，标题分别为 Enable All Check Boxes、Enable All Edit Fields 和 Enable All Combo Boxes，标识符分别为 IDC_CHECKBOX1、IDC_CHECKBOX2 和 IDC_CHECKBOX3，至于风格，则选择核选框 Group 和 Tab Stop，禁选核选框 Auto。

2. 接着，在对话框中添加组框，标题为 Check Boxes。在此组框中，加入三个核选框按钮，标题分别为 First Checkbox、Second Checkbox 和 Third Checkbox，并接受缺省的标识符 ID_CHECK1、ID_CHECK2 和 ID_CHECK3。

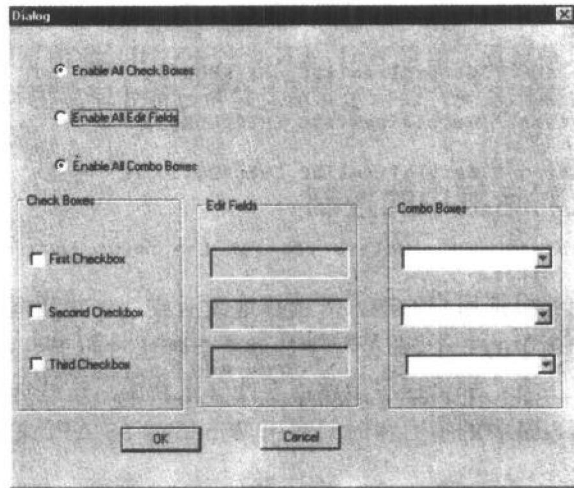


图 5-7 可以激活与禁止控制的对话框

3. 在对话框中添加标题为 Edit Fields 的组框。在此组框中，添加三个编辑域，标识符分别为 ID_EDIT1、ID_EDIT2 和 ID_EDIT3。

4. 最后添加标题为 Combo Boxes 的组框。在此组框中，添加三个组合框，标识符分别为 ID_COMBO1、ID_COMBO2 和 ID_COMBO3。选择组合框 ID_COMBO1 并点击弹出对话框右边的编辑框"Enter list choices"，在编辑框中输入下列字符串(通过按键 CTRL－ENTER 表示每个字符串的结束)："Choice A1"，"Choice A2"，"Choice A3" 和 "Choice A4"。

5. 选择第二个组合框 ID_COMBO2，并点击编辑框 "Enter list choices"，在编辑框中输入下列字符串（每个字符串的结束按键 CTRL＋ENTER)："Choice B1"，"Choice B2" 和 "Choice B3"。

6. 选择第三个组合框 ID_COMBO3，并点击编辑框 "Enter list choices"，在编辑框中输入下列字符串（每个字符串的结束按键 CTRL＋ENTER)："Choice C1" 和 "Choice C2"。

7. 选择 ClassWizard，为此模板生成新的对话框类，并命名此对话框类为 CEnableDlg。在 ClassWizard 中，从下拉列表中选择 CEnableDlg，从对象列表中选择 CEnableDlg，从消息列表中选择 WM_INITDIALOG，点击按钮 Add Function，在 CEnableDlg 的方法 OnInitDialog 中输入下列代码：

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

    CenterWindow ();

    CButton *b1 = (CButton *) GetDlgItem (IDC_CHECKBOX1);
    b1 -> SetCheck (1);
    CButton *b2 = (CButton *) GetDlgItem (IDC_CHECKBOX2);
```

```
    b2 -> SetCheck (1);
    CButton * b3 = (CButton * ) GetDlgItem (IDC _CHECKBOX3);
    b3 -> SetCheck (1);

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

8. 在 ClassWizard 中，从对象列表中选择对象 IDC _ CHECKBOX1，从消息列表中选择 BN _ CLICKED，点击按钮 Add Function，并命名方法为 OnButton1。在 CEnableDlg 的方法 OnButton1 中输入下列代码：

```
void CEnableDlg:: OnButton1 ()
{
   // Flip the button state

   if ( button1 _ flag )
       button1 _ flag = 0;
   else
       button1 _ flag = 1;

   // Reset the control to this state

   CButton * b1 = (CButton * ) GetDlgItem (IDC _ CHECKBOX1);
   b1 -> SetCheck (button1 _ flag);

   BOOL enable _ flag;

   // See if we should enable or disable the edit fields.

   if ( button1 _ flag == 1 ) // Button is set
       enable _ flag = TRUE;
   else
       enable _ flag = FALSE;

   GetDlgItem (IDC _ CHECK1) -> EnableWindow (enable _ flag);
   GetDlgItem (IDC _ CHECK2) -> EnableWindow (enable _ flag);
   GetDlgItem (IDC _ CHECK3) -> EnableWindow (enable _ flag);
}
```

9. 在 ClassWizard 中，从对象列表中选择对象 IDC _ CHECKBOX2，从消息列表中选择 BN _ CLICKED，点击按钮 Add Function，并命名方法为 OnButton2。在 CEnableDlg 的方法 OnButton2 中输入下列代码：

```
void CEnableDlg::OnButton2 ()
{
  // Flip the button state

  if ( button2_flag )
     button2_flag = 0;
  else
     button2_flag = 1;

  // Reset the control to this state

  CButton *b2 = (CButton *) GetDlgItem (IDC_CHECKBOX2);
  b2->SetCheck (button2_flag);
  BOOL enable_flag;

  // See if we should enable or disable the edit fields.

  if ( button2_flag == 1 ) // Button is set
     enable_flag = TRUE;
  else
     enable_flag = FALSE;

  GetDlgItem (IDC_EDIT1) -> EnableWindow (enable_flag);
  GetDlgItem (IDC_EDIT2) -> EnableWindow (enable_flag);
  GetDlgItem (IDC_EDIT3) -> EnableWindow (enable_flag);
}
```

10. 在ClassWizard中，从对象列表中选择IDC_CHECKBOX3，从消息列表中选择BN_CLICKED，点击按钮Add Function，并命名方法为OnButton3。在CEnableDlg的方法OnButton3中输入下列代码：

```
void CEnableDlg::OnButton3 ()
{
  // Flip the button state

  if ( button3_flag )
     button3_flag = 0;
  else
     button3_flag = 1;

  // Reset the control to this state
```

```
    CButton * b3 = (CButton * ) GetDlgItem (IDC_CHECKBOX3);
    b3 -> SetCheck (button3_flag);

    BOOL enable_flag;

    // See if we should enable or disable the combo boxes.

    if ( button3_flag == 1 ) // Button is set
        enable_flag = TRUE;
    else
        enable_flag = FALSE;

    GetDlgItem (IDC_COMBO1) -> EnableWindow (enable_flag);
    GetDlgItem (IDC_COMBO2) -> EnableWindow (enable_flag);
    GetDlgItem (IDC_COMBO3) -> EnableWindow (enable_flag);
}
```

11. 在类的构造函数 CEnableDlg::CEnableDlg 中添加下列代码：

```
button1_flag = 1;
button2_flag = 1;
button3_flag = 1;
```

12. 在类 CEnableDlg 的头文件（ENABLEDL.H）中添加下列说明：

```
int  button1_flag;
int  button2_flag;
int  button3_flag;
```

13. 返回 AppStudio 并选择菜单，添加新的主菜单 Controls 和新的菜单项 Enable and Disable，并设置下拉菜单项的标识符为 ID_CTRL_ENABLE。

14. 进入 ClassWizard，从下拉列表中选择 CCh55App，从对象列表中选择对象 ID_CTRL_ENABLE，从消息列表中选择消息 COMMAND，点击按钮 Add Function，并命名方法为 OnCtrlEnable。在 CCh55App 的方法 OnCtrlEnable 中添加下列代码：

```
void CCh55App:: OnCtrlEnable ()
{
    CEnableDlg dlg;
    dlg.DoModal ();
}
```

15. 在文件 CH55.CPP 的顶部添加下列 include 文件行：

```
#include "enabledl.h"
```

16. 编译并运行此例子程序。

用法

当用户从菜单中选择对话框时，类的方法 OnInitDialog 被调用。此方法通过发送消息 SetCheck（标志设置为 TRUE）来设置三个核选框的初始状态为“开”，控制核选框的内部程序状态变量也设置为 TRUE。

当用户从对话框中选中某个核选框时，例子程序将调用与此核选框相对应的处理程序(OnButton1、OnButton2 或 OnButton3)，此处理程序检查此核选框的内部状态标志，以确定核选框是选中的还是未选中的。如果是选中的，则设置为未选中；反之亦然。接着使用当前状态激活或禁止与此核选框相对应的控制集。

要激活一个控制，就调用此窗口的方法 Enable。在本节的例子程序中，为控制的窗口句柄实际是调用的 Windows 95 API 函数 EnableWindows。EnableWindows 有二个参数（窗口句柄和标志），标志用来指定是激活（TRUE）还是禁止（FALSE）窗口，MFC 方法 Enable 实际上是此 API 函数的调用。

注释

在 Visual C++中实现的例子程序在 Delphi 中也同样可以简单的实现。实现的步骤如下：

1. 创建新的项目文件 EDNIS.DPR。在表单中添加三个核选框，标题分别为 Enable All Check Boxes、Enable All Edit Controls 和 Enable All Combo Boxes，同时，在表单中添加三个组框，标题分别为 Check Boxes、Edit Controls 和 Combo Boxes。

2. 在组框 Check Boxes 中添加三个核选框，接受缺省的标题 Check Box1、Check Box2 和 Check Box3。

3. 在组框 Edit Controls 中添加三个编辑域，接受缺省值。

4. 在组框 Combo Boxes 中添加三个组合框，对每个组合框点击 Item property 并添加几个字符串，这是为了使组合框的激活与禁止状态区别更明显。

5. 双击核选框 Enable All Check Boxes，在方法 TForm1.Checkbox1Click 中添加下列代码：

```
procedure TForm1.CheckBox1Click (Sender: Tobject);
begin
   if CheckBox1.Checked = False Then
     begin
         CheckBox4.Enabled := False;
               CheckBox5.Enabled := False;
         CheckBox6.Enabled := False;
     end
   else
     begin
```

```
        CheckBox4.Enabled := True;
        CheckBox5.Enabled := True;
        CheckBox6.Enabled := True;
    end
end;
```

6. 双击核选框 Enable All Edit Controls Check，在方法 TForm1.Checkbox2Click 中添加下列代码：

```
procedure TForm1.CheckBox2Click (Sender: Tobject);
begin
  if CheckBox2.Checked = False Then
    begin
        Edit1.Enabled := False;
              Edit2.Enabled := False;
        Edit3.Enabled := False;
    end
  else
    begin
        Edit1.Enabled := True;
        Edit2.Enabled := True;
        Edit3.Enabled := True;
    end
end;
```

7. 双击核选框 Enable All Combo Boxes，在方法 TForm1.Checkbox3Click 中添加下列代码：

```
procedure TForm1.CheckBox2Click (Sender: Tobject);
begin
  if CheckBox2.Checked = False Then
    begin
        Edit1.Enabled := False;
              Edit2.Enabled := False;
        Edit3.Enabled := False;
    end
  else
    begin
        Edit1.Enabled := True;
        Edit2.Enabled := True;
        Edit3.Enabled := True;
    end
```

```
end;
```

8. 编译并运行此例子程序。显示的对话框如图 5-8 所示。

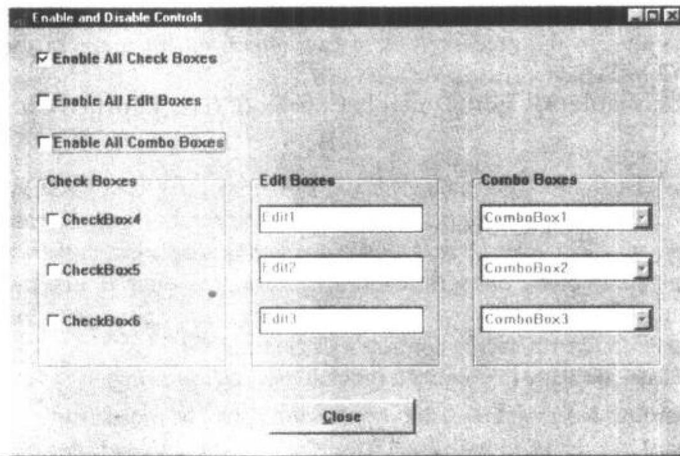


图 5-8　显示激活和禁止控制的 Delphi 表单

5.6　改变对话框中显示的控制

问题

有的程序员希望能够有时在对话框中显示某些控制，有时将这些控制隐藏起来。例如：用户只有在操作了基本选项后才会操作“高级”选项。是否有方法使得程序员利用 Windows 95 API 函数来避免每次删除和重新创建某些临时不需要的控制呢？

方法

有选择地隐藏或显示窗口或控制的功能在 Windows 95 API 中已经提供，从而使得在应用程序运行时可以不删除和重新创建窗口，API 函数 ShowWindow 就可以完成此功能。函数 ShowWindow 有两个参数（窗口句柄和标志），标志用来指定窗口是显示还是隐藏。此函数隐藏窗口只是将窗口从屏幕上删除，但不影响其内部的任何状态。

在本节中，将介绍如何在对话框中有选择地隐去或显示窗口。同时，还将介绍两个重要的 Windows 95 API 函数，ShowWindow 和 SetWindowText。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Controls 和菜单项 Show and Hide，将弹出一个对话框，如图 5-9 所示，点击对话框中的某个按钮，对话框将显示或隐藏相应的文本域，这些按钮用来切换文本域的隐藏和显示，每个按钮控制一个文本域。

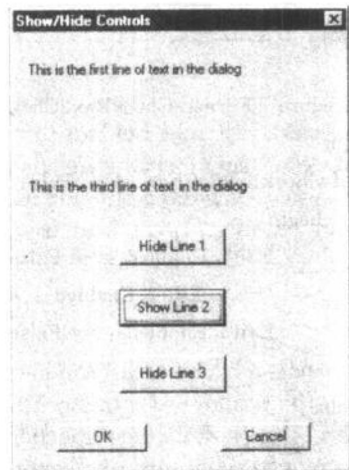


图 5-9　可显示和隐藏控制的对话框

实现例子程序的具体步骤如下：

1. 在 Visual C＋＋中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH56.MAK。进入 AppStudio 并创建新的对话框，在此对话框的顶部添加三个文本域，第一个文本域的标题为“This is the first line of text in the dialog”，标识符为 IDC _ LINE1；第二个文本域的标题为“This is the second line of text in the dialog”，标识符为 IDC _ LINE2；第三个文本域的标题为“This is the third line of text in the dialog”，标识符为 IDC _ LINE3。在对话框中添加三个按钮，标题分别为 Hide Line1、Hide Line2 和 Hide Line3。

2. 进入 ClassWizard，为此对话框模板生成新的对话框类，并命名此对话框类为 CShowControlDlg。在 ClassWizard 中，从下拉列表中选择类 CShowControlDlg，从对象列表中选择 CShowControlDlg，从消息列表中选择 WM _ INITDIALOG，点击按钮 Add Function，在 CShowContorlDlg 的方法 OnInitDialog 中输入下列代码：

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

    CenterWindow ();

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

3. 在 ClassWizard 中，从对象列表中选择对象 IDC _ BUTTON1，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名此方法为 OnButton1。在 CShowControlDlg 的方法 OnButton1 中输入下列代码：

```
void CShowControlDlg:: OnButton1 ()
{
   CStatic *text = (CStatic * ) GetDlgItem (IDC _ LINE1);
   CButton *b1    = (CButton * ) GetDlgItem (IDC _ BUTTON1);

   if ( text -> IsWindowVisible () ) {
       text -> ShowWindow (SW _ HIDE);
       b1 -> SetWindowText ( "Show Line 1");
   }
   else {
       text -> ShowWindow (SW _ SHOW);
       b1 -> SetWindowText ( "Hide Line 1");
   }
}
```

4. 在 ClassWizard 中，从对象列表中选择 IDC _ BUTTON2，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名此方法为 OnButton2。在 CShowControlDlg 的方法

OnButton2 中输入下列代码：

```
void CShowControlDlg:: OnButton2 ()
{
    CStatic *text = (CStatic *) GetDlgItem (IDC_LINE2);
    CButton *b1    = (CButton *) GetDlgItem (IDC_BUTTON2);
    if ( text -> IsWindowVisible () ) {
        text -> ShowWindow (SW_HIDE);
        b1 -> SetWindowText ( "Show Line 2");
    }
    else {
        text -> ShowWindow (SW_SHOW);
        b1 -> SetWindowText ( "Hide Line 2");
    }
}
```

5. 在 ClassWizard 中，从对象列表中选择对象 IDC_BUTTON3，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名此方法为 OnButton3。在 CShowControlDlg 的方法 OnButton3 中输入下列代码：

```
void CShowControlDlg:: OnButton3 ()
{
    CStatic *text = (CStatic *) GetDlgItem (IDC_LINE3);
    CButton *b1    = (CButton *) GetDlgItem (IDC_BUTTON3);

    if ( text -> IsWindowVisible () ) {
        text -> ShowWindow (SW_HIDE);
        b1 -> SetWindowText ( "Show Line 3");
    }
    else {
        text -> ShowWindow (SW_SHOW);
        b1 -> SetWindowText ( "Hide Line 3");
    }
}
```

6. 接着，进入 AppStudio 并添加新的主菜单，标题为 Controls。在菜单 Controls 中添加新的菜单项 Show and Hide，标识符为 ID_CONTROLS_SHOWANDHIDE。

7. 进入 ClassWizard，从下拉列表中选择对象 CCh56App，从对象列表中选择 ID_CONTROLS_SHOWANDHIDE，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名方法为 OnControlsShowandhide。在 CCh56App 的方法 OnControlsShowandhide 中输入下列代码：

```
void CCh56App:: OnControlsShowandhide ()
{
  CShowControlDlg        dlg;
  dlg.DoModal ();
}
```

8. 在文件 CH56.CPP 的顶部输入下列 include 文件行：

```
#include "showcont.h"
```

9. 编译并运行此例子程序。

用法

当创建对话框时，首先在 OnInitDialog 中调用 CenterWindow 来将对话框置于父窗口的中央。当用户按下对话框中的任一按钮时，将调用此按钮的方法并执行下列步骤：首先，处理程序检查与此按钮相连接的文本域当前是否可见的，这是通过调用 MFC 方法 IsWindowVisible 来完成的。此方法对应于 Windows API 函数 IsVisibleWindow。

在检查过窗口是否可见后，按钮处理程序接着调用 ShowWindow。如果文本域当前是可见的，则传送参数 TRUE；如果是不可见的，则传送 FALSE。函数 ShowWindow 将切换文本域的可见状态，或者使其可见，或者将其隐藏起来。

在检查并设置文本域的可见状态后，对话框处理程序接着调用函数 SetWindowsText 来更新按钮上的文本，此函数可改变屏幕上任何窗口（无论是否对话框）的窗口文本。例如，按钮的窗口文本实际上是按钮的标题，对于文本域，窗口文本实际上是静态文本控制中的文本，对于有标题条的“普通”窗口，窗口文本是标题条的标题。

这就是所有的实现，都是通过调用 Windows 95 API 来完成的。

5.7 在对话框中使用属性单

问题

新的 Windows 95 界面似乎完全建立在属性单的模式上，有的程序员希望在自己的 Windows 应用程序中也实现此模式。如何在应用程序中使用属性单（类似于 Windows 95 中的属性单）呢？

方法

属性单（以前称作标签对话框）是一个绝妙的方案，使得程序员可以使用大量的对话框，却不会用尽宝贵的屏幕资源，另外，属性单还可以避免单个对话框中有太多控制的混乱，而又不需要用户为完成某项简单任务而启动多个对话框。应用程序的数据可以集中起来分成类（称为属性页）并建立成各个单独的对话框模板，然后利用属性单组织起来。

Visual C++通过类 CPropertySheet 和 CPropertyPage 为属性单的使用提供了很好的界面，不幸的是，如果程序员使用的是 16 位的开发工具（MSVC 1.5X），那么此工具不是很适合自动生成这些类。这本节中，我们将介绍实现 16 位应用程序的方法，以便能更好地解释实现的方法，并提供一个既能用于 Windows 95 的 32 位应用程序也能用于 16 位应用程序的简

单模板。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Controls 和下拉菜单项 Property Sheet，弹出的属性单对话框如图 5-10 所示。点击属性单的第二个标签（Page Two），将显示出第二个属性页，如图 5-11 所示。

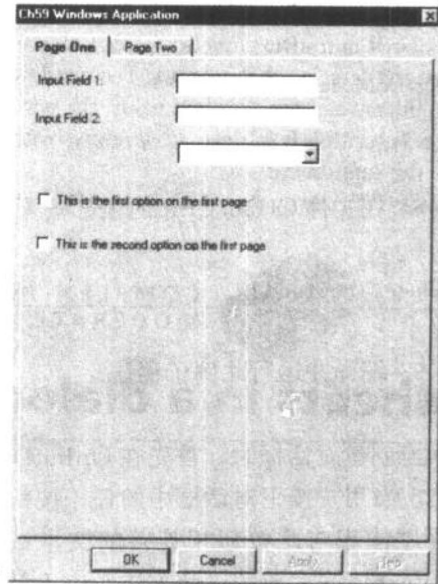


图 5-10 显示 Page one 时的属性单

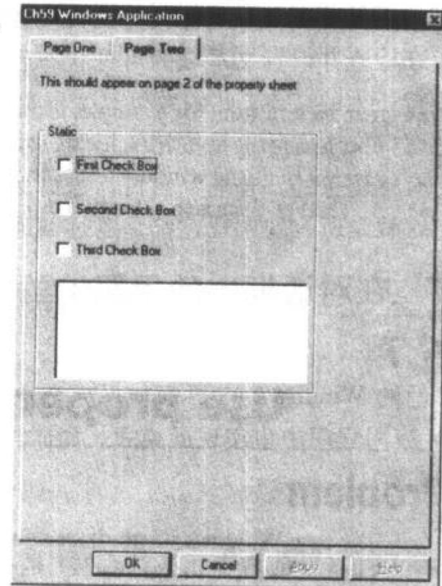


图 5-11 显示 Page two 时的属性单

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH57.MAK。进入 AppStudio 并创建两个新对话框，在创建对话框时注意它们的标识符，尽管在本例子程序中添加什么控制是无所谓的，但是应该添加尽量多的控制类型，另外要确保两个对话框的大小一样。

2. 在 AppStudio 中，双击第一个对话框中除控制以外的任何位置，从而选中此对话框并显示出它的属性对话框。从中选择 Style，从此页的组合框 Style 中选择 Child，从组合框 Border 中选择 Thin，并确保选中核选框 Disabled 和 Titlebar。在 General 中，设置第一个对话框的标题为 PageOne，设置第二个对话框的标题为 PageTwo，这些将是属性单的标签中的标题。

3. 保存对话框。从 Menus 列表中选择主菜单，添加标题为 Controls 的主菜单。在主菜单 Controls 中添加新的下拉菜单项，标题为 Property Sheet，标识符为 ID_CONTROLSPROPERTYSHEET。保存菜单并退出 AppStudio。

4. 在 Visual C++中，点击图标 New File 或从菜单 File 中选择 New，以创建新的文件。在此文件中输入下列代码并保存此文件为 PROPPGE.CPP：

```
//proppge.cpp : implementation file
//

#include "stdafx.h"
#include "proppge.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE [] = __FILE__; #endif
//////////////////////////////////////////////////////////////////////////////
```

```
// CPropPage dialog
CPropPage:: CPropPage (int id) : CPropertyPage (id)
{
  //{{AFX_DATA_INIT (CPropPage)
  //}} AFX_DATA_INIT
}
void CPropPage:: DoDataExchange (CDataExchange * pDX)
{
  CPropertyPage:: DoDataExchange (pDX);
  //{{AFX_DATA_MAP (CPropPage)
  //}} AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP (CPropPage, CPropertyPage)
  //{{AFX_MSG_MAP (CPropPage)
    // NOTE: the ClassWizard will add message map macros here
  //}} AFX_MSG_MAP
END_MESSAGE_MAP ()

/////////////////////////////////////////////////////////////////////////////
// CPropPage message handlers
```

5. 接着，在 Visual C++中创建另外一个新文件。在此文件中输入下列代码并保存此文件为 PROPPGE.H：

```
// proppge.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CPropPage dialog
class CPropPage : public CPropertyPage
{
// Construction
public:
  CPropPage (int id);
// Dialog Data
  //{{AFX_DATA (CPropPage)
  //}} AFX_DATA

// Overrides
  // ClassWizard generate virtual function overrides
  //{{AFX_VIRTUAL (CPropPage)
  protected:
```

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

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG (CPropPage)
        // NOTE: the ClassWizard will add member functions here
    //}} AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
```

6. 文件 PROPPGE.CPP 和 PROPPGE.H 是用于属性页的框架模板。在本步骤中，将创建实际的属性单对话框和页。

进入 ClassWizard，从下拉列表中选择应用程序对象 CCh57App，从对象列表中选择 ID_CONTROLSPROPERTYSHEET，从消息列表中选择 COMMAND，点击按钮 Add Function，并命名新方法为 OnPropertySheet。在 CCh57App 的方法 OnPropertySheet 中输入下列代码：

```
void CCh57App:: OnPropertySheet ()
{
    CPropertySheet prop (AFX_IDS_APP_TITLE, AfxGetMainWnd ());
    CPropPage *page1 = new CPropPage (IDD_DIALOG11);
    CPropPage *page2 = new CPropPage (IDD_DIALOG12);

    prop.AddPage ( page1 );
    prop.AddPage ( page2 );
    prop.DoModal ();
    delete page1;
    delete page2;
}
```

7. 在文件 CH57.CPP 的顶部添加下列 include 文件行：

```
#include "proppge.h"
```

8. 编译并运行此例子程序。

用法

Windows 95 的属性单对话框被 MFC 的类 CPropertySheet 所封装。每个属性单“标签”(被类 CPropertyPage 封装）是应用程序中的一个对话框，如果你希望在自己的应用程序中添加新的属性单，那么只需使用一个 CPropertySheet 对象（或自己创建的用来实现某些特殊处理的 CPropertySheet 的派生类）和添加每个页面到此对话框中，页面是 CPropertyPage 的派

生对象。在本节的例子程序中，我们使用类 CPropPage 来创建页面，此类接受对话框标识符作为对话框“页”，以用来显示该标签。

类 CPropPage 可重用在其他的应用程序中。如果要在页中添加成员变量的处理功能，只需创建类 CPropPage 的派生类并使用 MFC 的数据交换性质来获取和设置属性单中的数据。

一旦创建了属性单和页面，就可以使用 CPropertySheet 的方法 AddPage 来将每个页面同属性单关联起来。标签将使用对话框的标题作为标签的标题，并以它们添加到属性单中的顺序显示出来。一旦所有的页面都被添加到属性单中，就可以调用 CPropertySheet 的方法 DoModal 来显示对话框并允许用户在各个页面中输入数据。

当使用完属性单后，应该记住要删除为它创建的每个页面，以释放内存空间供 Windows 95 使用。不要忘记此操作，否则应用程序就可能在运行时用光内存资源。

注释

属性单模式是从用户那里获取分散但一定程度上又相关的信息的很好方法。例如：可以使用属性单来为视图设置属性，一个标签可能包含前景（文本）颜色、背景颜色和字体，另一个标签可能包含显示所有或部分数据的选项，再另一个标签可能包含打印选项或文件保存选项。

属性单已集成在 Windows 95 中，因此在应用程序中添加属性单不仅可以简化用户的操作，而且还可以使应用程序界面更加友好。

5.8 在对话框中的按钮上绘制图象

问题

Windows 95 应用程序的新界面似乎按钮上都是显示图象，而不再是简单的文本，有些按钮上还是动画，在用户点击按钮时按钮上的图象不断改变以指示某些操作正在进行。有的程序员希望能够在自己的对话框中也添加此类图象按钮，那么如何利用 API 来实现此功能呢？

方法

无论是在 Windows 3.x 中还是在 Windows 95 中（或者 Windows NT 中），要在按钮上显示图象都需要创建自绘制按钮。自绘制按钮由程序员负责处理按钮的显示和界面，而 Windows 只负责传送消息通知程序员当前所处的状态。

Visual C++ 允许使用类 CBitmapButton。此类封装自绘制按钮的功能，使得程序员可以通过在按钮上绘制图象（或位图）并为这些图象实例化类 CBitmapButton 的一个拷贝来创建按钮。在本节中，将介绍创建图象按钮所需要的步骤以及如何在用户点击按钮时改变图象。

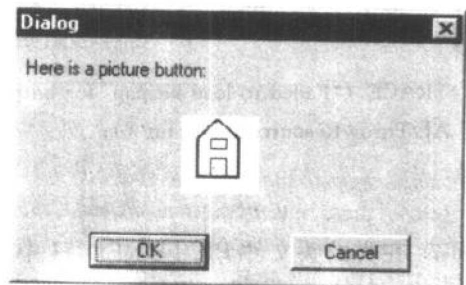


图 5-12 按钮释放时的图象对话框

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Controls 和菜单项 Picture Dialog，将弹出一个对话框，如图 5-12 所示，在显示“房子”按钮上按下鼠标左键，则按钮上的图象将改变，如图 5-13 所示。

实现例子程序的具体步骤如下：

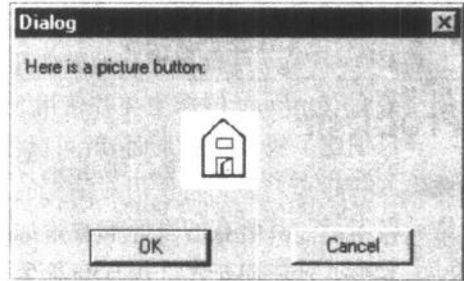


图 5-13 按钮按下时的图象对话框

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH58. MAK，进入 AppStudio 并创建新对话框。在此对话框中，添加一个静态文本域和一个按钮。

2. 设置静态文本域的标题为 “Here is a picture button”，标识符则取缺省值 IDC _ STATIC。

3. 设置按钮的标题为 DOORBUTTON，标识符为 IDC _ DOORBUTTON，按钮的属性为 Visible、Owner Draw 和 TabStop。

4. 进入 AppStudio，选择位图并创建两个新位图。第一个位图的标识符设置为 “DOORBUTTONU”，第二个位图的标识符设置为 “DOORBUTTOND”。两个标识符都包含双引号，从而使得 Visual C++不为它们生成标识符定义语句。保存位图和资源文件。

5. 在 AppStudio 中选择刚创建的对话框，选择 ClassWizard。为此对话框模板创建新的对话框类，并命名新类为 CPictureDlg。在 ClassWizard 中，从对象列表中选择对象 CPictureDlg，选择消息 WM _ INITDIALOG，在类 CPictureDlg 的方法 OnInitDialog 中输入下列代码：

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

  VERIFY (m _ Button. SubclassDlgItem (IDC _ DOORBUTTON, this));
  m _ Button. SizeToContent ();

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

6. 在类的构造函数 CPictureDlg:: CPictureDlg 中输入下列代码：

```
if (! m _ Button. LoadBitmaps ("DOORBUTTONU", "DOORBUTTOND"))
   {
     TRACE ("Failed to load bitmaps for buttons\n");
     AfxThrowResourceException ();
  }
```

7. 在类的头文件 PICTURED. H 中添加下列代码：

```
private:
  CBitmapButton m _ Button;
```

8. 接着，重新进入 AppStudio。添加新的主菜单 Controls，在此菜单中添加菜单项 Picture

Dialog，并设置此菜单项的标识符为 ID _ PICTURE _ DLG。

9. 选择 ClassWizard，从下拉列表中选择对象 CCh58App，从对象列表中选择对象 ID _ PICTURE _ DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function，并命名方法为 OnPictureDlg。在 CCh58App 的方法 OnPictureDlg 中输入下列代码：

```
void CCh58App:: OnPictureDlg ()
{
  CPictureDlg dlg;

  dlg.DoModal ();
}
```

10. 在文件 CH58.CPP 的顶部添加下列 include 文件行：

```
#include "pictured.h"
```

11. 编译并运行此例子程序。

用法

当用户选择菜单项 Picture Dialog 后，对话框对象就在应用程序对象中被构造。在实例化对话框对象时，分两个步骤执行：首先创建对象。调用对象的构造函数，此函数使用成员变量 m _ Button 来装入与按钮相关联的位图。类 CBitmapButton 包含四个位图的数组，这些图象分别在按钮 Up、Down、GetFocus 和 Disabled 时显示。这四个图象保存在类中，并在需要时（消息发送来时）使用。

接着，调用类 CPictureDlg 的方法 OnInitDialog，此方法在实例化实际窗口后和在屏幕上显示之前被调用。此方法调用按钮的 SubclassDlgItem 来子类化对话框中的控制，子类化是功能非常强大的方法，它调用 API 函数 SetWindowLong 来完成功能。

在对话框中子类化控制，将需要调用 API 函数 GetWindowLong 来获取窗口函数或回调函数，接着调用函数 SetWindowLong 来重新设置窗口的回调函数（用于消息处理）为程序员指定的函数。整个过程类似于下面所示：

```
// Get the existing window procedure

FARPROC old _ proc = (FARPROC) GetWindowLong (hWnd, GWL _ WNDPROC);

// Reset the window procedure to be our callback function.

FARPROC new _ proc = (FARPROC) MakeProcInstance ( (FARPROC) MyWindowHandler, hInst);

// This would be the callback function.

SetWindowLong (hWnd, GWL _ WNDPROC, new _ proc);
```

一旦窗口函数被替换，那么所有消息都将传送给新的窗口回调函数（在前面的例子中为new _proc)。CBitmapButton 的方法 SubclassDlgItem 完成所有这些操作，一旦此函数被调用，那么对 Paint、LeftButtonDown 等事件都将调用 CBitmapButton 的成员函数来处理。

由于按钮类将自动取消对按钮的子类化，因此在应用程序中将不需要此操作。

在按钮被子类化后，绘制函数将自动显示与按钮的“释放”状态相关联的图象。当用户在按钮上按下鼠标左键时，则按钮处于“被按下”的状态，与此状态相关联的图象将显示出来。如果用户又松开按钮，则将显示与按钮“释放”状态相关联的图象。

5.9 在执行另一操作的同时显示“进度”对话框

问题

在执行一些屏幕上无显示的操作时，有的程序员希望自己的应用程序像许多新的应用程序一样能够显示“进度”对话框。例如，假如应用程序在拷贝文件，那么程序员就希望在屏幕上显示消息以指示当前操作完成的百分比，而且允许用户在需要时取消操作的执行。

如何利用 Windows API 显示这样的对话框并允许用户在执行过程中取消操作呢？拷贝的过程是完全“密封”在小的循环中，那么用户如何来表明他们想要取消此循环呢？

方法

在 Windows 中，显示一个对话框使得用户可以取消密封循环中的操作，一直是个非常棘手的问题，状态对话框使用得相当普遍，用户对在操作开始后可以随时取消操作已经很习惯。在本节中，将介绍 Windows 消息系统是如何工作的，以及如何利用此消息系统来实现我们的目的。

此方法的基本前提是在屏幕上显示包含某种状态指示器和按钮 Cancel 的模式对话框，状态指示器用来通知用户操作的进度，按钮 Cancel 使得用户可以在操作完成前取消操作。在本节的例子程序中，状态指示器是一个进度条，进度条是添加在 Win32 中的通用控制。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Controls 和下拉菜单项 Copy File，将弹出一个对话框，如图 5-14 所示，输入有效的源文件名以及目的文件名，点击按钮 OK。要尽量选择一个大文件，否则会来不及请求取消操作。

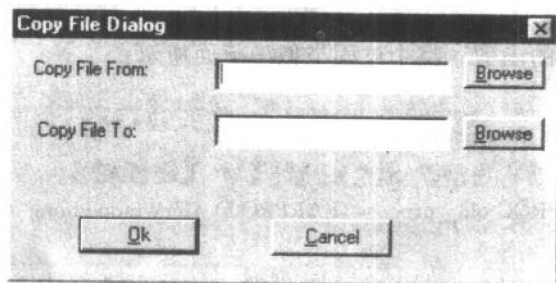


图 5-14 拷贝文件对话框

在进度对话框显示出来以后（图 5-15)，点击按钮 Close，则对话框消失和拷贝终止。而在点击按钮 Close 前，对话框将随着拷贝操作的进行而不断更新。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH59.MAK。进入 AppStudio 并创建新的对话框，在此新对话框中添加两个静态文本域，标题分别为 Copy File From: 和 Copy File To:，与静态文本域水平对齐添加两个编辑域，与编辑域水平对齐添加两个按钮，按钮的标题都设置为 &Browse。

2. 选择ClassWizard，为此对话框模板生成新的对话框类，新类命名为CCopyDlg。在Class Wizard中，点击标签Member Variables，为控制标识符IDC_EDIT1和IDC_EDIT2添加两个新的成员变量。IDC_EDIT1添加的成员变量是m_CopyFrom，IDC_EDIT2添加的成员变量是m_CopyTo。

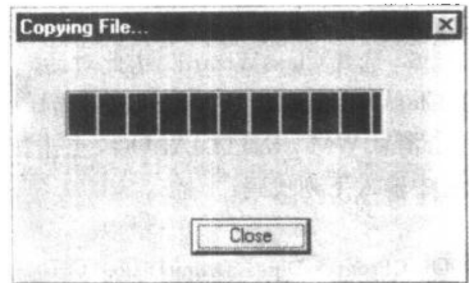


图5-15 文件拷贝的进度对话框

3. 重新进入ClassWizard，选择类CCopyDlg，选择按钮IDC_BUTTON1，从消息列表中选择消息BN_CLICKED，点击按钮Add Function，新方法命名为OnBrowseOpen。在CCopyDlg的方法OnBrowseOpen中输入下列代码：

```
void CCopyDlg:: OnBrowseOpen ()
{
   // Put up a file open dialog

   static char szFilter [] = "Text Files (*.txt) | *.TXT | All Files (*.*) | *.* ||";
   CFileDialog dlg ( TRUE, "DLL", NULL, OFN_HIDEREADONLY, szFilter, this );

   if ( dlg.DoModal () == IDOK ) {
      GetDlgItem (IDC_EDIT1) -> SetWindowText ( dlg.GetPathName () );
   }
}
```

4. 从对象列表中选择对象IDC_BUTTON2，从消息列表中选择消息BN_CLICKED，点击按钮Add Function，并命名新函数为OnBrowseSave。在CCopyDlg的函数OnBrowseSave中输入下列代码：

```
void CCopyDlg:: OnBrowseSave ()
{
   // Put up a file open dialog

   static char szFilter [] = "Text Files (*.txt) | *.TXT | All Files (*.*) | *.* ||";
   CFileDialog dlg ( FALSE, "DLL", NULL, OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
szFilter, this );

   if ( dlg.DoModal () == IDOK ) {
      GetDlgItem (IDC_EDIT2) -> SetWindowText ( dlg.GetPathName () );
   }
}
```

5. 在 AppStudio 中创建另外一个新对话框，从对话框中删除按钮OK 并将按钮Cancel 置于对话框的中央。

6. 选择 ClassWizard，为此对话框模板创建新的对话框类，并命名此类为 CProgressDlg。在 ClassWizard 中，从下拉列表中选择 CProgressDlg，从对象 ID 列表中也选择 CProgressDlg，点击消息 WM_INITDIALOG，点击按钮 Add Function，在 CProgressDlg 的方法 OnInitDialog 中输入下列代码：

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

  // Create a progress bar

  hWndPB =CreateWindowEx ( 0, PROGRESS_CLASS, NULL, WS_CHILD |
           WS_VISIBLE, 30, 30, 200, 30, m_hWnd, 0, AfxGetInstanceHandle (), NULL );
  :: SendMessage ( hWndPB, PBM_SETRANGE, 0, MAKELPARAM (0, 100) );

  CenterWindow ();
  PostMessage (WM_COMMAND, CM_DO_COPY);

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

7. 接着，在 CProgressDlg 的方法 OnInitDialog 的下面添加下列方法：

```
void CProgressDlg:: DoCopy (void)
{
   char buffer [256];

   // Open the files

   FILE *fp = fopen ( copyFrom, "r" );
   if ( fp == (FILE *) NULL ) {
      MessageBox ("Unable to open input file!", "Error", MB_OK |
                 MB_APPLMODAL | MB_ICONEXCLAMATION );
      return;
   }

   // Get File size of input file

   fseek ( fp, 0L, SEEK_END );
   long tot_bytes = ftell ( fp );
   fseek ( fp, 0L, SEEK_SET );
```

```
FILE *ofp = fopen ( copyTo, "w" );
if ( ofp == (FILE *) NULL ) {
    fclose (fp);
    MessageBox ("Unable to open output file!", "Error", MB_OK |
                MB_APPLMODAL | MB_ICONEXCLAMATION );
    return;
}

// Loop through and copy the files

long bytes = 0;
aborted   = 0;

while ( ! feof (fp) && ! aborted ) {

    // Get a line from the input file

    fgets ( buffer, 256, fp );

    // Write it to the output file

    fprintf (ofp, "%s", buffer );

    // Update progress bar

    bytes += strlen (buffer);
    int pct = (int) (100.0 * (double (bytes) / (double) tot_bytes));
    :: SendMessage ( hWndPB, PBM_SETPOS, pct, 0 );

    // Check for other messages

    MSG msg;

    if(PeekMessage((LPMSG)&msg,(HWND)NULL,(WORD)NULL,(WORD)NULL,TRUE))
     {
       TranslateMessage ( (LPMSG) &msg );
       DispatchMessage ( (LPMSG) &msg );
    }
}

fclose (fp);
```

```
  fclose (ofp);
  GetDlgItem (IDCANCEL) -> SetWindowText ( "Close" );
  :: SendMessage ( hWndPB, PBM_SETPOS, 100, 0 );

}
```

8. 在 ClassWizard 中，从对象列表中选择对象 IDCANCEL ，从消息列表中选择消息 COMMAND。点击按钮 Add Function，在 CProgressDlg 的方法 OnCancel 中输入下列代码：

```
void CProgressDlg:: OnCancel ()
{
  aborted = 1;

  CDialog:: OnCancel ();
}
```

9. 在 CProgressDlg 的构造函数 CProgressDlg:: CProgressDlg 中做下列修改：

```
CProgressDlg:: CProgressDlg (CString& file1, CString& file2, CWnd * pParent /* =NULL */)
  : CDialog (CProgressDlg:: IDD, pParent)

  copyFrom = file1;
  copyTo   = file2;
  aborted  = 0;

  //{{AFX_DATA_INIT (CProgressDlg)
    // NOTE: the ClassWizard will add member initialization here
  //}} AFX_DATA_INIT
}
```

10. 在类的头文件 PROGRESS.H 中做下列修改。注意，修改的行或添加的行用黑体表示。

```
const int CM_DO_COPY = 101;

class CProgressDlg : public CDialog
{
private:
  CString copyFrom;
  CString copyTo;
  int     aborted;
  HWND    hWndPB;

// Construction
```

```
public:
    CProgressDlg (CString& file1, CString& file2, CWnd * pParent = NULL);
// standard constructor
// Dialog Data
    //{{AFX_DATA (CProgressDlg)
    enum { IDD = IDD_DIALOG10 };
        // NOTE: the ClassWizard will add data members here
    //}} AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange (CDataExchange * pDX);      // DDX/DDV support

    // Generated message map functions
    //{{AFX_MSG (CProgressDlg)
    virtual BOOL OnInitDialog ();
    afx_msg void DoCopy (void);
    virtual void OnCancel ();
    //}} AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
```

11. 在 PROGRESS.CPP 中修改消息映射如下。改变的部分用黑体表示。

```
BEGIN_MESSAGE_MAP (CProgressDlg, CDialog)
    //{{AFX_MSG_MAP (CProgressDlg)
    ON_COMMAND (CM_DO_COPY, DoCopy)
    //}} AFX_MSG_MAP
END_MESSAGE_MAP ()
```

12. 重新进入 AppStudio 并创建新的菜单 Controls。在菜单 Controls 中添加新的菜单项 Copy File，标识符为 ID_COPY_FILE。

13. 在 ClassWizard 中，从下拉列表中选择对象 CCh59App，从对象列表中选择 ID_COPY_FILE，从消息列表中选择消息 COMMAND，点击按钮 Add Function，并命名方法为 OnCopyFile。在 CCh59App 的方法 OnCopyFile 中添加下列代码：

```
void CCh59App:: OnCopyFile ()
{
  CCopyDlg dlg;

  if ( dlg.DoModal () == IDOK ) {
      CProgressDlg pdlg (dlg.m_CopyFrom, dlg.m_CopyTo);
```

```
        pdlg.DoModal ();
    }

}
```

14. 在源文件 CH59.CPP 的顶部添加下列 include 文件行：

```
#include "copydlg.h"
#include "progress.h"
```

15. 编译并运行此例子程序。

用法

在拷贝文件对话框显示出来以后，用户输入要拷贝的源文件名和目的文件名，这些文件名接着被传送给 CProgressDialog，CProgressDialog 就使用它们开始文件的拷贝。

在初始化显示进度对话框时，对话框的方法 OnInitDialog 被调用。在此方法中，首先调用基类的方法，接着，使用 API 函数 CreateWindowsEx 和类名 PROGRESS_CLASS 来创建进度条。此方法接着又调用函数 CenterWindow 来将对话框置于屏幕的中央，并调用 API 函数 PostMessage 来传送消息给自身，此消息接着被类 CProgressDlg 的方法 DoCopy "捕获"，从而开始真正的拷贝操作。

在方法 DoCopy 中，首先打开源文件和目的文件，接着开始从源文件到目的文件一行行拷贝。在方法 DoCopy 中有意思的执行是在其最后调用 API 函数 PeekMessage、TranslateMessage 和 DispatchMessage，由于它暂时将控制权由对话框传送回 Windows，所以这称为抢占式循环。这是 API 函数 Yield 应该做但没有做的。

除了拷贝文件外，此函数还示范了如何通过发送消息PBM_SETPOS给进度条控制来更新此控制。

最后，函数关闭目的文件并结束。那么它如何处理用户的取消操作呢？很简单，对话框的方法 OnCancel 捕捉用户的任何取消事件（点击按钮 Cancel 或按 ESC 键），并在对话框中设置内部标志以表示用户已取消此操作。每次循环都检查此标志，只要设置了此标志，那么操作就终止。

第6章　编辑控制

在 Windows 用户界面中，编辑控制是最普遍使用的组件之一。在 Windows 95 中提供了大量的功能，使得程序员在开发应用程序时可以选择使用，但是，许多应用程序看起来好象仍然是为 Windows 2.0 开发的，而不是针对 Windows 新版本开发的。在本章中，将介绍如何使用编辑控制的增强功能，从而使得应用程序更加专业化。

1. 如何使编辑控制只读

在有的应用程序中，开始用户可以在屏幕上编辑数据，后来应用程序设置此数据为不可编辑，如果用户此时还要编辑此数据，则应用程序会弹出消息框提示用户。在本节中，将介绍如何从一开始就将数据保护起来，使得用户清楚此数据只能读而不能修改，同时，还要介绍如何根据数据库或其他数据源的状态来设置数据为可编辑的或只读的。

2. 如何利用编辑控制获取口令

当用户输入口令时，如果口令不在屏幕上显示出来，则应用程序会更加安全。在本节中，将介绍如何实现口令保护对话框，使得在用户输入口令时，对话框中显示星号。

3. 如何改变编辑控制的背景颜色

在本节中，将介绍如何改变编辑控制的背景颜色和文本颜色，从而使得重要的编辑控制能够从许多控制中突出出来。同时，还将介绍改变颜色的正确方法，从而使得程序员开发的用户界面一致，并且工作正常和显得美观。

4. 如何在编辑控制中替换文本

在本节中，将介绍字处理程序是如何在编辑控制中替换被选中的文本，而且不需要取回并改变编辑控制中的全部内容。本节将介绍如何简便快捷地替换用户选中的文本。

5. 如何给编辑控制添加撤消功能

利用 Windows 编辑控制的内部函数可以在应用程序中实现简单快捷的撤消功能。如果有的程序员准备开发相当规模的字处理软件或程序编辑器，那么 Windows 提供的简单撤消功能可能是不够的，但是，如果用户在选中了编辑控制中的全部内容时不小心按了 DEL 键，此时 Windows 提供的简单撤消功能会给用户很大的帮助。

6. 如何确认编辑控制中的输入有效

Windows 应用程序在用户点击对话框中的按钮 OK 之前通常不确认用户输入的有效性，其充分的理由是，因为利用 Windows API 来实现编辑控制的输入有效确认并不特别容易。在本节中，将示范如何在应用程序中实现域输入的有效确认。

7. 如何利用剪贴板进行剪切和粘贴操作

要实现用户友好的应用程序，就应该允许用户在编辑控制和剪贴板之间进行剪切、拷贝和粘贴操作。在本节中，将介绍如何在应用程序中简单快捷地实现此功能。

表 6-1 列出了本章中使用的 Windows API 函数。

表 6-1 在第 6 章中用到的 Windows API 函数

ES _ READONLY	EM _ SETREADONLY	GetWindowLong	GetDlgItem
SendMessage	ES _ PASSWORD	WM _ CTLCOLOREDIT	SetTextColor
SetBkColor	CreateSolidBrush	EM _ REPLACESEL	SendDlgItemMessage
EM _ GETSEL	EN _ CHANGE	EM _ CANUNDO	EM _ UNDO
SetWindowText	EnableWindow	CreateWindow	SetWindowLong
WM _ CHAR	MessageBeep	CallWindowProc	EM _ LIMITTEXT
WM _ CUT	WM _ COPY	WM _ PASTE	WM _ CLEAR
EnableMenuItem	IsClipboardFormatAvailable	WM _ SIZE	MoveWindow

6.1 使编辑控制只读

问题

有时在应用程序中不希望用户能够改变编辑控制中显示的信息，如何设置编辑控制为只读，从而使得用户必要时可以滚动但是不能修改数据呢？

方法

从 Windows 3.1 开始，编辑控制就有了 ES _ READONLY 风格。如果程序员将此风格添加到编辑控制的资源脚本中，则就设置了编辑控制为只读，也可以利用消息 EM _ SETREADONLY 在应用程序运行时在编辑控制中加入或删除此风格。

Windows 还提供了一种测试编辑控制状态的机制，即检查控制风格来确定是否设置了 ES _ READONLY 位。

步骤

按照下列步骤实现例子程序 READONLY.EXE。运行此例子程序，则弹出一个带菜单的窗口，选择菜单 File 中的菜单项 Test，可以测试例子程序的运行。

此时例子程序弹出一个小对话框，对话框包含编辑控制、按钮 Close 和一个用来切换编辑控制的只读或可编辑状态的按钮。如果此时编辑控制为只读，则按钮的标题为 Allow Edit，点击此按钮，将使编辑控制为可编辑的，并且按钮的标题改变为 Read—Only，用户可以来回切换编辑控制的两个状态。此例子程序显示的对话框如图 6-1 所示。

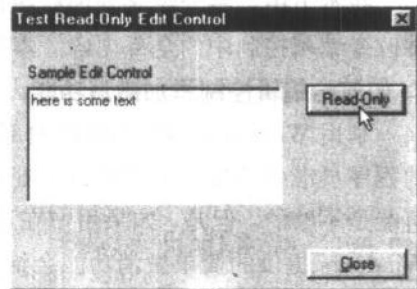


图 6-1 改变只读编辑控制的状态

实现例子程序的具体步骤如下：

1. 为例子程序创建工作目录 READONLY，用来存放例子程序的所有源文件。

2. 为例子程序创建资源脚本，此资源脚本用来定义对话框及其控制和按钮。在编辑控制中使用风格 ES _ READONLY 使得编辑控制初始是只读的，如果要使编辑控制初始为可编辑的，可以将风格 ES _ READONLY 删除，当显示对话框时，应用程序会自动检测设置哪种风格。使用文本编辑器创建新文件 READONLY.RC，并在此文件中添加下列资源脚本：

```
/* ------------------------------------------------------------------ */
```

```
/*                                                                          */
/* MODULE: READONLY.RC                                                      */
/* PURPOSE: This resource script defines the menu and test dialog          */
/*          for the sample application.                                     */
/*                                                                          */
/* ------------------------------------------------------------------------ */
#include <windows.h>
#include "readonly.rh"

IDD_EDITTEST DIALOG 6, 15, 202, 119
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU
CAPTION "Test Read-Only Edit Control"
FONT 8, "MS Sans Serif"
{
   PUSHBUTTON "&Close", IDCLOSE, 147, 100, 50, 14
   LTEXT "Sample Edit Control", -1, 8, 14, 74, 8
EDITTEXT IDC_SAMPLEEDIT, 7, 25, 127, 54, ES_MULTILINE | ES_READONLY
| ES_WANTRETURN | WS_BORDER | WS_TABSTOP
   DEFPUSHBUTTON "Read-Only", IDTOGGLE, 147, 24, 50, 14
}

IDM_TESTMENU MENU
{
   POPUP "&File"
   {
      MENUITEM "&Test", CM_TEST
      MENUITEM SEPARATOR
      MENUITEM "E&xit", CM_EXIT
   }
}
IDI_APPICON ICON "readonly.ico"
```

3. 在资源脚本中包含了例子程序的图标。可以使用资源编辑器来创建一个16色的32×32象素的图标，并保存在文件READONLY.ICO中。

4. 在资源脚本中还包含了头文件READONLY.RH，此文件为例子程序定义了资源标识符，在一个单独的文件中定义这些常量值使得例子程序的源代码也可以引用这些标识符。创建文件READONLY.RH，并在文件中添加下列定义：

```
#ifndef __READONLY_RH
/* ------------------------------------------------------------------------ */
/*                                                                          */
/* MODULE: READONLY.RH                                                      */
```

```
/* PURPOSE: This include file defines the resource identifiers used.         */
/*                                                                            */
/* -------------------------------------------------------------------------- */
#define __READONLY_RH

#define IDM_TESTMENU      200
#define CM_TEST           101
#define CM_EXIT           102

#define IDD_EDITTEST      201
#define IDC_SAMPLEEDIT    101
#define IDTOGGLE          102
#define IDCLOSE           103

#define IDI_APPICON       201

#endif
```

5. 创建新的源文件 READONLY.C，此文件是例子程序的 C 语言源文件。在文件的顶部添加下列代码，这些代码包含了编译所需的 Windows 头文件，并定义了预处理符号 STRICT，此符号表示在 C 语言环境中对 Windows 句柄进行严格的类型检查。

```
/* -------------------------------------------------------------------------- */
/*                                                                            */
/* MODULE: READONLY.C                                                         */
/* PURPOSE: This sample application demonstrates how to make an edit          */
/*          control read-only within your application, how to test            */
/*          to see if an edit control is read-only, and how to make           */
/*          a read-only edit control editable again.                          */
/*                                                                            */
/* -------------------------------------------------------------------------- */

#define STRICT
#include <windows.h>
#include <winnt.h>
#include "readonly.rh"

static char *MainWindowClassName = "ReadonlyTestWindow";
HINSTANCE hInstance = NULL;
```

6. 在同一源文件中添加函数 IsReadOnly。如果编辑控制的风格中设置了 ES_READONLY 位，则此函数返回 TRUE。函数 GetWindowlong 用来获取控制的风格字，GWL_STYLE 是在 WINDOWS.H 中定义的值，提供了从窗口句柄到风格字的偏移量。

```
/* -------------------------------------------------------------------------- */
```

```
/* This function tests to see if an edit control is readonly. It              */
/* returns TRUE if it is, or false if it is not.                              */
/* -------------------------------------------------------------------------- */
BOOL IsReadOnly (HWND hWnd)
{
   return (GetWindowLong (hWnd, GWL_STYLE) & ES_READONLY);
}
```

7. 接着，添加函数 ToggleReadonly State。此函数调用 IsReadOnly 来获取编辑控制的当前状态，然后调用 SendMessage 将消息 EM_SETREADONLY 发送给编辑控制。如果要设置 ES_READONLY 位，则 SendMessage 的参数 WPARAM 为 TRUE（非零）；反之，如果要清除 ES_READONLY 位，则参数 WPARAM 的值为 FALSE。此函数还根据编辑控制的新状态来切换按钮上的标题文本。

```
/* -------------------------------------------------------------------------- */
/* This function toggles the read-only state of the edit control.             */
/* If the edit control is read-only, it makes it editable; if it can          */
/* be edited, it makes it read-only.                                          */
/* -------------------------------------------------------------------------- */
void ToggleReadonlyState (HWND hWnd)
{
   HWND hChild;
   BOOL newStatus;
   if ( (hChild = GetDlgItem (hWnd, IDC_SAMPLEEDIT)) != NULL)
    {
      newStatus = ! IsReadOnly (hChild);
      SendMessage (hChild, EM_SETREADONLY, newStatus, 0);
      if (newStatus)
          SetDlgItemText (hWnd, IDTOGGLE,"Allow Edit");
      else
         SetDlgItemText (hWnd, IDTOGGLE,"Read-Only");
    }
}
```

8. 下一个要添加的函数是对话框的回调函数 TestDialogPro，此函数接收发送给对话框的消息。为了响应消息 WM_INITDIALOG，此函数调用前面添加的函数 IsReadOnly 来检测编辑控制的初始状态，并根据编辑控制是否只读来设置按钮的标题。此函数还响应消息 WM_COMMAND，如果接收到的是消息 IDCLOSE，则关闭对话框；如果接收到的是消息 IDTOGGLE，则切换编辑控制的状态。

```
/* -------------------------------------------------------------------------- */
/* This is the dialog function for the dialog. It handles clicks to           */
/* the readonly toggle button, and to the close button. It also sets          */
```

```
/* the initial text of the toggle button based upon the initial         */
/* state of the edit control.                                           */
/* -------------------------------------------------------------------- */
BOOL CALLBACK TestDialogProc (HWND hWnd, UINT message,
                              WPARAM wParam, LPARAM lParam)
{
   HWND hChild;

   switch (message)
   {
      case WM_INITDIALOG:
         if ( (hChild = GetDlgItem (hWnd, IDC_SAMPLEEDIT)) != NULL)
         {
            if (IsReadOnly (hChild))
               SetDlgItemText (hWnd, IDTOGGLE,"Allow Edit");
            else
               SetDlgItemText (hWnd, IDTOGGLE,"Read-Only");
         }
         break;
      case WM_COMMAND:
         if (wParam == IDTOGGLE)
            ToggleReadonlyState (hWnd);
         else
            if (wParam == IDCLOSE)
               EndDialog (hWnd, 1);
         break;
      default:
         return FALSE;
   }
   return TRUE;
}
```

9. 要添加的下一个函数是函数MainWndProc，此函数是应用程序主窗口的回调函数。当用户选择菜单项Test时，此函数用来响应菜单命令并创建测试对话框，在同一源文件READ-ONLY.C中添加此函数。

```
/* -------------------------------------------------------------------- */
/* This function is the main window callback function which will be    */
/* called by Windows to process messages for our window.                */
/* -------------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
```

```
{
    switch (message)
    {
        case WM_COMMAND:
            if (wParam == CM_TEST)
                DialogBox (hInstance, MAKEINTRESOURCE (IDD_EDITTEST),
                           hWnd, TestDialogProc);
            else
                if (wParam == CM_EXIT)
                    DestroyWindow (hWnd);
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

10. 在同一源文件的末尾添加下面三个函数。这些函数构成了本章中所有例子程序的框架，函数 InitApplication 用来注册应用程序主窗口的窗口类，函数 InitInstance 用来创建应用程序主窗口。

最后一个函数（WinMain）是应用程序的入口点。此函数调用 InitApplication 和 InitInstance 来设立应用程序，接着处理消息直到应用程序结束。添加这三个函数后，便可以编译并测试例子程序了。

```
/* ---------------------------------------------------------------- */
/* This function initialises a WNDCLASS structure and uses it to    */
/* register a class for our main window.                            */
/* ---------------------------------------------------------------- */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
```

```
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ---------------------------------------------------------------------- */
/* This function creates an instance of our main window. The window      */
/* is given a class name and a title, and told to display anywhere.      */
/* the nCmdShow argument passed to the program determines how the        */
/* window will be displayed.                                              */
/* ---------------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;      // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Edit Read-only demo",
                         WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                         NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);    // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if        */
/* the main window class name has already been registered, if it has     */
/* not, call InitApplication to register it. Call InitInstance to        */
/* create an instance of our main window, then pump messages until       */
/* the application is closed.                                             */
/* ---------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;
    if (! FindWindow (MainWindowClassName, NULL))
```

```
        if (! InitApplication (hInstance))
            return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

11. 编译并运行此例子程序。

注释

如同在例子程序的第二步中所描述的，试着在资源脚本中删除或添加风格 ES_READONLY，可以看到对话框总是以正确的按钮状态弹出的，这是因为例子程序在每次打开对话框时检查编辑控制的风格。

当用户企图改变文本内容时，使用只读编辑控制提示用户文本是不可修改的，以免用户在修改文本后才发现修改的内容不能被保存下来。

6.2 利用编辑控制获取口令

问题

有时在应用程序中需要用户输入口令。如何才能取回口令，并使用户的输入不被其他人看到呢？

方法

Windows API 提供了使编辑控制成为口令编辑控制的设施。口令编辑控制利用星号（*）来代替用户输入的字符，从而使得其他人看不到用户输入的口令，口令编辑控制如图 6-2 所示。通常在资源脚本中设置编辑控制的风格 ES_PASSWORD，使得编辑控制成为口令编辑控制。

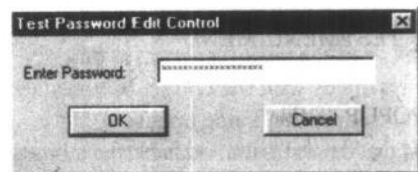


图 6-2 以 ES_PASSWORD 风格使用编辑控制

步骤

按照下列步骤实现例子程序 PASSWORD.EXE。运行此例子程序，从菜单 File 中选择菜单项 Test，则弹出一个小对话框，提示用户输入口令，在编辑控制中输入示例口令，可以看到只有星号显示出来。

例子程序并不十分安全。当用户点击按钮 OK 时，输入的口令将显示出来，以证实口令确实被存储了。因此当测试此例子程序时，不要使用你自己系统上的口令。

实现例子程序的具体步骤如下：

1. 要实现此例子程序，首先创建目录PASSWORD，以用来存储例子程序的所有源文件。

2. 现在创建例子程序的资源文件。此例子程序的关键是风格ES_PASSWORD，在下面的脚本中对编辑控制设置了此风格。使用文本编辑器输入下列脚本，并保存在文件PASSWORD.RC中。

```
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: PASSWORD.RC                                                    */
/* PURPOSE: This resource script defines the menu and test dialog         */
/*          for the sample application.                                   */
/*                                                                        */
/* ---------------------------------------------------------------------- */

#include <windows.h>
#include "password.rh"

IDD_EDITTEST DIALOG 6, 15, 202, 63
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU
CAPTION "Test Password Edit Control"
FONT 8, "MS Sans Serif"
{
   LTEXT "Enter Password:", -1, 5, 12, 55, 8
   EDITTEXT IDC_SAMPLEEDIT, 72, 10, 109, 12, ES_PASSWORD |
            WS_BORDER | WS_TABSTOP
   DEFPUSHBUTTON "OK", IDOK, 27, 32, 50, 14
   PUSHBUTTON "Cancel", IDCANCEL, 125, 32, 50, 14
}

IDM_TESTMENU MENU
{
   POPUP "&File"
    {
      MENUITEM "&Test", CM_TEST
      MENUITEM SEPARATOR
      MENUITEM "E&xit", CM_EXIT
    }
}

IDI_APPICON ICON "password.ico"
```

3. 为例子程序准备图标。使用资源编辑器创建一个16色的32×32象素的图标，并保存

在文件 PASSWORD. ICO 中。

4. 在资源文件中使用了 include 文件 PASSWD. RH，此文件为例子程序定义资源标识符。使用文本编辑器创建此文件，并在此文件中添加下列代码，在此文件中说明的资源标识符用在资源脚本中，也用在例子程序的源代码中。

```
#ifndef __PASSWORD_RH
/* ------------------------------------------------------------------------- */
/*                                                                           */
/* MODULE: PASSWORD.RH                                                       */
/* PURPOSE: This include file defines the resource identifiers used.        */
/*                                                                           */
/* ------------------------------------------------------------------------- */
#define __PASSWORD_RH

#define IDM_TESTMENU      200
#define CM_TEST           101
#define CM_EXIT           102
#define IDD_EDITTEST      201
#define IDC_SAMPLEEDIT    101

#define IDI_APPICON       203

#endif
```

5. 现在创建例子程序的源文件 PASSWORD. C，在此文件的顶部添加下列代码段，此段代码包含了 Windows 头文件以及定义资源标识符的头文件，同时还定义了主窗口的窗口类名。

```
/* ------------------------------------------------------------------------- */
/*                                                                           */
/* MODULE: PASSWORD.C                                                        */
/* PURPOSE: This small application demonstrates the use of the              */
/*          ES_PASSWORD style to hide edit control input for entry          */
/*          of a password.                                                   */
/*                                                                           */
/* ------------------------------------------------------------------------- */

#define STRICT
#include <windows.h>
#include <winnt.h>
#include "password.rh"
static char *MainWindowClassName = "PasswordTestWindow";
```

```
HINSTANCE hInstance = NULL;
```

6. 接着添加回调函数 TestDialogProc。此函数响应命令消息 IDOK 和 IDCANCEL，这些消息在用户点击对话框中的按钮时发送，当用户点击按钮 OK 时，此函数调用 GetDlgItemText 来从编辑控制中取回口令。在本节的例子程序中，口令接着被显示在消息框中，当然，在应用程序产品中是不能这样做的。

```
/* ----------------------------------------------------------------- */
/* This dialog function handles the WM_COMMAND messages, responding  */
/* to IDOK by displaying the password entered, then closing the      */
/* dialog.                                                           */
/* ----------------------------------------------------------------- */
BOOL CALLBACK TestDialogProc (HWND hWnd, UINT message,
                              WPARAM wParam, LPARAM lParam)
{
    char buf [100];

    switch (message)
    {
        case WM_INITDIALOG:
            return TRUE;
        case WM_COMMAND:
            if ( (wParam == IDOK) || (wParam == IDCANCEL))
            {
                if (wParam == IDOK)
                {
                    GetDlgItemText (hWnd, IDC_SAMPLEEDIT, buf, 100);
                    MessageBox (hWnd, buf,"Your Password",
                            MB_ICONINFORMATION | MB_OK);
                }
                EndDialog (hWnd, wParam);
            }
            break;
        default:
            return FALSE;
    }
    return TRUE;
}
```

7. 现在将函数 MainWndProc 添加到源文件中，此函数用来响应发送给例子程序主窗口的消息，特别是响应命令消息 IDTEST，此函数创建并显示对话框。

```
/* ----------------------------------------------------------------- */
/* This function is the main window callback function which will be  */
/* called by Windows to process messages for our window.             */
/* ----------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            if (wParam == CM_TEST)
                DialogBox (hInstance, MAKEINTRESOURCE (IDD_EDITTEST),
                        hWnd, TestDialogProc);
                else
                    if (wParam == CM_EXIT)
                        DestroyWindow (hWnd);
                break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

8. 下面的三个函数构成了例子程序的框架，将这三个函数添加到源文件的末尾。函数 InitApplication 用来注册例子程序主窗口的窗口类，只有同类的其他窗口不存在时才调用此函数。函数 InitInstance 用来创建并显示例子程序的主窗口，同时还保存实例句柄以便在创建对话框时使用。

最后一个函数是 WinWain，Windows 操作系统调用此函数来运行例子程序，此函数调用函数 InitApplication 和函数 InitInstance 来创建应用程序，接着处理消息直到例子程序结束。

```
/* ----------------------------------------------------------------- */
/* This function initialises a WNDCLASS structure and uses it to     */
/* register a class for our main window.                             */
/* ----------------------------------------------------------------- */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS   wc;
    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
```

```
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ---------------------------------------------------------------------- */
/* This function creates an instance of our main window. The window      */
/* is given a class name and a title, and told to display anywhere.       */
/* the nCmdShow argument passed to the program determines how the         */
/* window will be displayed.                                              */
/* ---------------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;        // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Password Demo",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);       // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if         */
/* the main window class name has already been registered, if it has      */
/* not, call InitApplication to register it. Call InitInstance to         */
/* create an instance of our main window, then pump messages until       */
/* the application is closed.                                             */
/* ---------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
```

```
                LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

9. 编译并运行此例子程序。

注释

口令编辑控制用来限制对多用户应用程序中重要部分的存取,或者在用户存取客户机/服务器数据库时用来获取用户的口令。

6.3 改变编辑控制的背景颜色

问题

有时需要使某个编辑控制显得醒目,因此有些程序员就希望能够改变控制的背景颜色,便试着以控制的句柄为参数调用函数 SetBkColor,但不能实现预期的效果。如何才能有效地改变背景颜色呢?

方法

通过处理消息 WM_CTLCOLOREDIT 可以改变编辑控制的颜色。当需要重新绘制编辑控制时,Windows 将此消息发送给应用程序。当应用程序接收此消息后,将选择控制的文本颜色和背景颜色,并返回一个画刷句柄,Windows 则使用此画刷来绘制控制。

改变编辑控制的颜色:

通常程序员为 Windows 界面的各个组件选择颜色时,都优先考虑使用标准的 Windows 颜色,即用户从属性单 Display 的标签 Appearance 中选择的颜色,这些颜色都具有标识符值,例如:COLOR_WINDOW 和 COLOR_BTNTEXT。程序员也可以通过调用函数 GetSysColor 来获取这些颜色的 RGB 值。

当程序员希望改变编辑控制的颜色时,一般不会考虑使用系统的颜色,而是使用自定义的颜色。但是,应该考虑同时改变控制的文本颜色和背景颜色,如果将背景颜色改变为绿色,而保持文本颜色为系统缺省的颜色,那么有可能界面非常漂亮,但是,如果用户将系统缺省的文本颜色也设置为绿色,那么显然就会看不到任何文本。同样,如果只将文本颜色改变为

灰色，而用户恰好也为窗口选择了灰色的背景，那么同样也是不正常的。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从菜单 File 中选择菜单项 Test，将弹出一个对话框，如图 6-3 所示，通过在黄色背景下显示黑色文本，从而突出了重要的域 Customer No.。而显示其他的编辑控制则使用了标准的颜色，即用户选择的系统颜色。

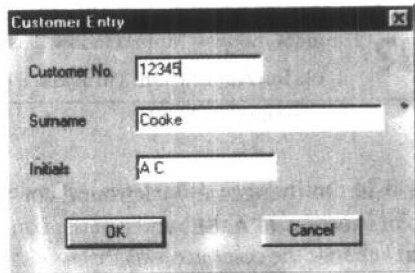


图 6-3 改变编辑控制的颜色

实现例子程序的具体步骤如下：

1. 创建目录 CTLCOLOR，用来保存例子程序的所有源文件，在后面步骤中创建的所有文件都将保存在这个目录中。

2. 为例子程序创建图标。使用资源编辑器创建一个 16 色的 32×32 象素的图标，并保存在文件 CTLCOLOR.ICO 中。

3. 现在为例子程序创建资源脚本。在此资源脚本中，定义例子程序的菜单，包含例子程序的图标，以及定义要显示的对话框。使用文本编辑器创建新文件 CTLCOLOR.RC，在此文件中输入下列文本。

```
/* ------------------------------------------------------------ */
/*                                                              */
/* MODULE: CTLCOLOR.RC                                          */
/* PURPOSE: This resource script defines the menu and test dialog */
/*          for the sample application.                         */
/*                                                              */
/* ------------------------------------------------------------ */

#include <windows.h>
#include "ctlcolor.rh"

IDD_EDITTEST DIALOG 6, 15, 202, 111
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU
CAPTION "Customer Entry"
FONT 8, "MS Sans Serif"
{
    LTEXT "Customer No.", -1, 10, 12, 45, 8
    LTEXT "Surname", -1, 10, 35, 32, 8
    LTEXT "Initials", -1, 10, 58, 22, 8
    EDITTEXT IDC_CUSTNO, 63, 10, 63, 12, WS_BORDER | WS_TABSTOP
    EDITTEXT IDC_SURNAME, 63, 33, 123, 12
    EDITTEXT IDC_INITIALS, 63, 56, 69, 12
    DEFPUSHBUTTON "OK", IDOK, 27, 85, 50, 14
    PUSHBUTTON "Cancel", IDCANCEL, 125, 85, 50, 14
```

```
}

IDM_TESTMENU MENU
{
   POPUP "&File"
     {
        MENUITEM "&Test", CM_TEST
        MENUITEM SEPARATOR
        MENUITEM "E&xit", CM_EXIT
     }
}

IDI_APPICON ICON "ctlcolor.ico"
```

4. 下一个需要创建的文件是include 文件。此文件用来定义资源标识符，在资源脚本中和C 语言源文件中都将使用此文件，创建此文件并在此文件中输入下列代码，保存此文件为CTLCOLOR.RH。

```
#ifndef __CTLCOLOR_RH
/* ------------------------------------------------------------------------ */
/*                                                                          */
/* MODULE: CTLCOLOR.RH                                                      */
/* PURPOSE: This include file defines the resource identifiers used.       */
/*                                                                          */
/* ------------------------------------------------------------------------ */
#define __CTLCOLOR_RH
#define IDM_TESTMENU      200
#define CM_TEST           101
#define CM_EXIT           102

#define IDD_EDITTEST      201
#define IDC_CUSTNO        101
#define IDC_SURNAME       102
#define IDC_INITIALS      103

#define IDI_APPICON       203

#endif
```

5. 为例子程序创建源文件CTLCOLOR.C，在此文件的顶部输入下列代码。注意全局变量hBrush，此变量用来为绘制编辑控制的背景而保存画刷。

```
/* ------------------------------------------------------------------------ */
```

```
/*                                                                    */
/* MODULE: CTLCOLOR.C                                                 */
/* PURPOSE: This application demonstrates how to handle the          */
/* WM_CTLCOLOREDIT message to change the colors of an edit           */
/* control.                                                           */
/*                                                                    */
/* ------------------------------------------------------------------ */
#define STRICT
#include <windows.h>
#include <winnt.h>
#include "ctlcolor.rh"

static char *MainWindowClassName = "ColorTestWindow";
HBRUSH     hBrush = NULL;
HINSTANCE hInstance = NULL;
```

6. 现在添加函数 SetEditColor。只要 Windows 发送消息 WM_CTLCOLOREDIT 给窗口，此函数就会被调用，在此函数中，将比较 Windows 提供的编辑控制句柄（在消息的参数 LPARAM 中）和要改变颜色的编辑控制的句柄。如果确实是要改变颜色的编辑控制，那么此函数将完成三步操作。首先，调用 SetTextColor 来设置编辑控制的文本颜色；接着，调用 SetBkColor 来设置背景的 RGB 值；最后，返回画刷句柄，以便在绘制控制时 Windows 可以使用。

```
/* ------------------------------------------------------------------ */
/* Set the colors to be used in the edit control. Returns NULL if    */
/* the control is not the right one.                                  */
/* ------------------------------------------------------------------ */
HBRUSH SetEditColor (HWND hWndOwner, HDC hDCEdit, HWND hWndEdit)
{
    if (hWndEdit == GetDlgItem (hWndOwner, IDC_CUSTNO))
    {
       SetTextColor (hDCEdit, RGB (0, 0, 0));     // black text.
       SetBkColor (hDCEdit, RGB (255, 255, 0)); // yellow background.
       return hBrush;
    }
    else
        return NULL;
}
```

7. 下一个要添加到文件的函数是 TestDialogProc，代码如下。每当有消息发送给对话框时 Windows 都将调用此回调函数，为了响应消息 WM_INITDIALOG，此函数创建画刷以用于绘制控制，此函数调用 API 函数 CreatSolidBrush 以最接近于 RGB 值的实颜色来创建画刷。

在下列代码中，利用 RGB 宏来指定画刷为明黄色。当窗口被撤消时，此函数调用 DeleteObject 来释放画刷，为了响应消息 WM_CTLCOLOREDIT，此函数调用前面编写的函数

SetEditColor，参数 LPARAM 用来指定编辑控制句柄，参数 WPARAM 用来指定设备描述表句柄。

```
/* ------------------------------------------------------------------------------- */
/* The dialog function for the dialog handles all the messages.                     */
/* It creates a brush when it receives WM_INITDIALOG, and deletes                   */
/* the brush on WM_DESTROY. It returns the brush in WM_CTLCOLOREDIT.                */
/* ------------------------------------------------------------------------------- */
BOOL CALLBACK TestDialogProc (HWND hWnd, UINT message,
                               WPARAM wParam, LPARAM lParam)
{
    HDC        hDCEdit;
    HWND       hWndEdit;

    switch (message)
    {
        case WM_INITDIALOG:
            hBrush = CreateSolidBrush (RGB (255, 255, 0));
            return TRUE;
        case WM_COMMAND:
            if ( (wParam == IDOK) || (wParam == IDCANCEL))
                EndDialog (hWnd, wParam);
            break;
        case WM_CTLCOLOREDIT:
            hWndEdit = (HWND) lParam;
            hDCEdit = (HDC) wParam;
            return (BOOL) SetEditColor (hWnd, hDCEdit, hWndEdit);
        case WM_DESTROY:
            DeleteObject (hBrush);
            break;
        default:
            return FALSE;
    }
    return TRUE;
}
```

8. 最后，在源文件中添加下列四个函数，这些函数提供了例子程序的框架。函数 MainWndProc 用来响应发送给主窗口的消息，当用户选择菜单项 Test 时，此函数显示对话框，当用户选择菜单消息 Edit 时，此函数关闭窗口和例子程序。函数 InitApplication 用来注册例子程序主窗口的窗口类，而函数 InitInstance 用来创建并显示主窗口。

最后一个函数是函数 WinWain，此函数是例子程序的入口点。在此函数中首先初始化例子程序，接着处理消息直到例子程序关闭。

```
/* ------------------------------------------------------------------------------- */
/* This function is the main window callback function which will be                */
```

```
/* called by Windows to process messages for our window.                     */
/* ------------------------------------------------------------------------ */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            if (wParam == CM_TEST)
                DialogBox (hInstance, MAKEINTRESOURCE (IDD_EDITTEST),
                           hWnd, TestDialogProc);
            else
                if (wParam == CM_EXIT)
                DestroyWindow (hWnd);
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}

/* ------------------------------------------------------------------------ */
/* This function initialises a WNDCLASS structure and uses it to            */
/* register a class for our main window.                                    */
/* ------------------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;
    return RegisterClass (&wc);
```

```
}

/* ---------------------------------------------------------------- */
/* This function creates an instance of our main window. The window  */
/* is given a class name and a title, and told to display anywhere.  */
/* the nCmdShow argument passed to the program determines how the    */
/* window will be displayed.                                          */
/* ---------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;           // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Edit Color Demo",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);             // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if    */
/* the main window class name has already been registered, if it has */
/* not, call InitApplication to register it. Call InitInstance to    */
/* create an instance of our main window, then pump messages until   */
/* the application is closed.                                         */
/* ---------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
```

```
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

9. 编译并运行此例子程序。

注释

通过响应消息 WM_CTLCOLOREDIT 来改变编辑控制的颜色，在希望突出重要的控制时是非常有用的。但是，应该尽量少用改变颜色的控制，只改变几个控制的颜色比改变所有控制的颜色效果要好的多，况且还可以尊重用户的喜好。

6.4 在编辑控制中替换文本

问题

如何在编辑控制中用其他的文本来替换选中的文本呢？

方法

Windows API 提供了消息 EM_REPLACESEL，从而使程序员可以用其他文本来替换选中的文本块，此消息假定已有文本块被选中。如果没有选中的文本块，那么替换文本将在当前插入光标所在的位置上插入。

步骤

按照下列步骤实现例子程序 REPLACE.EXE。打开并运行此例子程序，从菜单 File 中选择菜单项 Test，将显示出如图 6-4 所示的对话框。

此对话框包含两个编辑控制，在第二个编辑控制中已经包含了一些示范文本。可以在第一个编辑控制中输入一些文本，接着在第二个编辑控制中选中一些文本，然后点击按钮 Replace，则第二个编辑控制中的选中文本将被第一个编辑控制中的内容所替换。如果点击按钮 Replace 时没有选中的文本，那么替换文本将在当前插入光标所在的位置上插入。

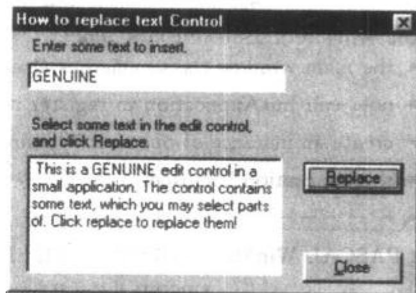


图 6-4　在编辑控制中替换文本

实现例子程序的具体步骤如下：

1. 创建目录 REPLACE，作为本例子程序的工作目录。

2. 为例子程序创建图标。使用资源编辑器创建一个 16 色的 32×32 象素的图标，并保存在文件 REPLACE.ICO 中。

3. 为例子程序创建资源脚本。此资源脚本用来定义如图 6-4 所示的对话框，并用来包含例子程序的图标。使用文本编辑器创建文件 REPLACE.RC，并在此文件中输入下列代码。

```
/* ----------------------------------------------------------------- */
/*                                                                   */
/* MODULE: REPLACE.RC                                                */
/* PURPOSE: This resource script defines the menu and test dialog    */
/*          for the sample application.                              */
/*                                                                   */
/* ----------------------------------------------------------------- */

#include <windows.h>
#include "replace.rh"

IDD_EDITTEST DIALOG 6, 15, 202, 119
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
CAPTION "How to replace text Control"
FONT 8, "MS Sans Serif"
{
  LTEXT "Enter some text to insert.", -1, 10, 2, 79, 8
  LTEXT "Select some text in the edit control, and click Replace.",
        -1, 10, 38, 112, 16
  EDITTEXT IDC_REPLACE, 8, 14, 128, 12
  EDITTEXT IDC_SAMPLEEDIT, 8, 57, 127, 54, ES_MULTILINE | ES_WANTRETURN |
        WS_BORDER | WS_TABSTOP
  PUSHBUTTON "&Replace", IDREPLACE, 146, 59, 50, 14
  PUSHBUTTON "&Close", IDCLOSE, 146, 100, 50, 14
}

IDM_TESTMENU MENU
{
  POPUP "&File"
   {
     MENUITEM "&Test", CM_TEST
     MENUITEM SEPARATOR
     MENUITEM "E&xit", CM_EXIT
   }
}

IDI_APPICON ICON "replace.ico"
```

4. 前面输入的资源脚本需要include文件REPLACE.RH。此文件用来定义对话框和图标的标识符，以便在资源脚本和例子程序代码中引用这些资源。创建此文件并在此文件中输入下列定义：。

```
#ifndef __REPLACE_RH
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: REPLACE.RH                                                     */
/* PURPOSE: This include file defines the resource identifiers used.      */
/*                                                                        */
/* ---------------------------------------------------------------------- */
#define __REPLACE_RH

#define IDM_TESTMENU     200
#define CM_TEST          101
#define CM_EXIT          102

#define IDD_EDITTEST     201
#define IDC_REPLACE      101
#define IDC_SAMPLEEDIT   102
#define IDREPLACE        103
#define IDCLOSE          104

#define IDI_APPICON       202

#endif
```

5. 现在准备为例子程序创建源文件。本例子程序是用标准C语言来编写的，但是发送给编辑控制的消息是可以在任何编程语言中发送的。使用文本编辑器创建新文件REPLACE.C，并在此文件的顶部添加下列行：

```
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: REPLACE.C                                                      */
/* PURPOSE: This sample application demonstrates how to replace           */
/*          selected text from within your application with other         */
/*          text. This application gets text from the control             */
/*          IDC_REPLACE, and uses it to replace whatever text             */
/*          is selected in the IDC_SAMPLEEDIT control.                    */
/*                                                                        */
/* ---------------------------------------------------------------------- */

#define STRICT
#include <windows.h>
#include <winnt.h>
#include "replace.rh"
```

```
static char * MainWindowClassName = "ReplaceTestWindow";
HINSTANCE hInstance = NULL;
```

6. 现在添加函数 TestDialogProc，代码如下。当对话框有消息需要处理时，Windows 将调用此回调函数。此函数在第二个编辑框中插入一些示范文本以响应消息 WM_INITDIALOG，此函数还响应消息 WM_COMMAND，该消息在用户点击对话框中的按钮时产生。为了响应命令消息 IDREPLACE，此函数首先调用函数 GetDlgItemText 来取回第一个编辑控制中的文本，接着发送消息 EM_REPLACESEL 给第二个编辑控制，此消息使得编辑控制以传送来的字符串替换选中的文本。如果没有选中的文本，则将字符串插入到当前插入光标所在的位置上。

```
/* ------------------------------------------------------------ */
/* This is the dialog function for the dialog. It returns TRUE to        */
/* the WM_INITDIALOG message to accept the default focussed control.     */
/* It handles the IDREPLACE button by using EM_REPLACESEL to replace     */
/* the selected text in one edit control with the contents of the        */
/* other. It handles IDCLOSE by calling EndDialog.                       */
/* ------------------------------------------------------------ */
BOOL CALLBACK TestDialogProc (HWND hWnd, UINT message,
                              WPARAM wParam, LPARAM lParam)
{
    char buf [100];

    switch (message)
    {
        case WM_INITDIALOG:
            SetDlgItemText (hWnd, IDC_SAMPLEEDIT," This is a sample edit control"
                                           " in a small application. The"
                                           " control contains some text,"
                                           " which you may select parts of.
"                                           " Click replace to replace them!");
            return TRUE;
        case WM_COMMAND:
            if (wParam == IDREPLACE)
            {
                GetDlgItemText (hWnd, IDC_REPLACE, buf, 100);
                SendDlgItemMessage (hWnd, IDC_SAMPLEEDIT,
                                    EM_REPLACESEL, 0, (LPARAM) buf);
            }
            else
                if (wParam == IDCLOSE)
                    EndDialog (hWnd, 1);
            break;
```

```
        default:
            return FALSE;
    }
    return TRUE;
}
```

7. 现在为例子程序主窗口添加回调函数 MainWndProc。此函数用来处理发送给例子程序主窗口的消息，以响应用户的菜单选择操作，当用户从菜单 File 中选择菜单项 Test 时，例子程序接收命令消息 CM _ TEST，为了响应此消息，此函数将显示对话框。

```
/* ---------------------------------------------------------------- */
/* This function is the main window callback function which will be */
/* called by Windows to process messages for our window.            */
/* ---------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM _ COMMAND:
            if (wParam == CM _ TEST)
                DialogBox (hInstance, MAKEINTRESOURCE (IDD _ EDITTEST),
                           hWnd, TestDialogProc);
            else
                if (wParam == CM _ EXIT)
                    DestroyWindow (hWnd);
            break;
        case WM _ DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
```

8. 在同一源文件 REPLACE.C 中添加最后三个函数，一旦添加了这三个函数，便可以准备编译并测试此例子程序了，这三个函数构成了例子程序的基本框架。当例子程序启动时，函数 WinMain 获取到控制权，此函数调用 InitApplication 来创建主窗口的窗口类，接着调用 InitInstance 来创建并显示例子程序的主窗口，最后循环处理消息直到程序结束。

```
/* ---------------------------------------------------------------- */
/* This function initialises a WNDCLASS structure and uses it to    */
/* register a class for our main window.                            */
/* ---------------------------------------------------------------- */
BOOL InitApplication (HINSTANCE hInstance)
```

```
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;
    return RegisterClass (&wc);
}

/* ---------------------------------------------------------------- */
/* This function creates an instance of our main window. The window  */
/* is given a class name and a title, and told to display anywhere.  */
/* the nCmdShow argument passed to the program determines how the    */
/* window will be displayed.                                         */
/* ---------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;        // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Edit replace demo",
               WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
               CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
               NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);        // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if    */
/* the main window class name has already been registered, if it has */
```

```
/* not, call InitApplication to register it. Call InitInstance to         */
/* create an instance of our main window, then pump messages until        */
/* the application is closed.                                             */
/* -------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}
```

9. 编译并运行此例子程序。

注释

程序员可以决定只有在用户选中文本时替换功能才有效，要确定是否有选中的文本，可以使用消息 EM_GETSEL 来获取编辑控制中选中文本块的起始字符和结束字符。如果两个值是不同的，则表示有选中的文本；如果两个值是相同的，则表示没有选中的文本，可以禁止替换操作。在 6.7 节中将进一步讨论消息 EM_GETSEL。

6.5 给编辑控制添加撤消功能

问题

有时，需要在应用程序的编辑控制中添加撤消功能。如何才能实现此功能呢？

方法

Windows API 为编辑控制实现了单层撤消缓存，因此要实现单层撤消功能便是一件非常简单的事，只需发送正确的消息给控制就可以了。

如果在应用程序中要实现单层撤消功能，则需要了解三个重要的消息。第一个消息是 EN_CHANGE，这是一个通知消息，当编辑控制中的文本改变时编辑控制发送此消息给其父窗口（对话框或其他窗口）。在本节的例子程序中，利用响应此消息作为检查撤消功能的状态的最好机会。

下一个重要的消息是 EM_CANUNDO。应用程序可以发送此消息给编辑控制，以确定此时使用撤消功能是否有用。如果此消息发送给编辑控制时没有要撤消的操作，或者此编辑

控制刚使用 SetWindowText 设置文本，则 SendMessage 将返回 FALSE，这就使得应用程序可以在撤消功能没用时禁止此功能。

第三个是消息 EM_UNDO，当应用程序发送此消息给编辑控制时，编辑控制将撤消用户的最近一次操作。此操作可能是小操作，比如：删除一个字符；也可能是大操作，比如：撤消删除控制中整个文档的操作。

步骤

按照下列步骤实现例子程序 UNDO.EXE。打开并运行此例子程序，将弹出一个有菜单的窗口，从菜单 File 中选择菜单项 Test 以显示对话框，对话框中的编辑控制初始时包含文本，而且按钮 Undo 初始时是禁止的，在编辑控制中输入文本，接着删除一部分或全部文本，然后便可以试着用按钮 Undo 来恢复前次操作时编辑控制中的内容。例子程序的运行如图 6-5 所示，此时按钮 Undo 是激活的。

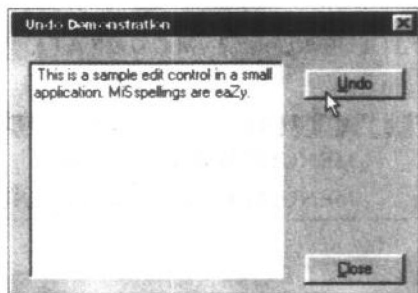


图 6-5　具有 Undo 功能的例子程序

在使用按钮 Undo 撤消一个操作以后，按钮 Undo 将仍为激活状态，如果此时再点击按钮一次，将撤消上次的撤消操作。因此此按钮也可被当作按钮 Redo。

实现例子程序的具体步骤如下：

1. 创建新目录 UNDO，作为本例子程序的工作目录，以存放例子程序的所有源代码文件。

2. 使用文本编辑器创建新文件 UNDO.RC，在此文件中输入下列资源脚本。此资源脚本定义主窗口的菜单，还定义例子程序的对话框，同时还定义多行编辑控制和按钮 Undo。

```
/* ---------------------------------------------------------- */
/*                                                            */
/* MODULE: UNDO.RC                                            */
/* PURPOSE: This resource script defines the menu and test dialog */
/*          for the sample application.                       */
/*                                                            */
/* ---------------------------------------------------------- */

#include <windows.h>
#include "undo.rh"

IDD_EDITTEST DIALOG 6, 15, 202, 119
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU
CAPTION "Undo Demonstration"
FONT 8, "MS Sans Serif"
{
   EDITTEXT IDC_SAMPLEEDIT, 8, 11, 127, 100, ES_MULTILINE | ES_WANTRETURN | WS
_BORDER | WS_TABSTOP
   PUSHBUTTON "&Undo", IDUNDO, 146, 15, 50, 14, WS_DISABLED | WS_TABSTOP
   PUSHBUTTON "&Close", IDCLOSE, 146, 100, 50, 14
```

```
}

IDM_TESTMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "&Test", CM_TEST
        MENUITEM SEPARATOR
        MENUITEM "E&xit", CM_EXIT
    }
}

IDI_APPICON ICON "undo.ico"
```

3. 创建例子程序的图标。使用资源编辑器创建一个 16 色的 32×32 象素的图标，并保存在文件 UNDO.ICO 中。

4. 在资源脚本中包含了文件 UNDO.RH，在此文件中定义了编译脚本时所需要的资源标识符。尽管这些值可以硬编码在资源脚本中，但是如果放在单独的文件中，则例子程序的源代码也可以存取它们，因此在需要添加或删除资源时，将可以减轻改变代码的工作量。创建此文件，并在文件中添加下列定义。

```
#ifndef __UNDO_RH
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: UNDO.RH                                                        */
/* PURPOSE: This include file defines the resource identifiers used.      */
/*                                                                        */
/* ---------------------------------------------------------------------- */
#define __UNDO_RH

#define IDM_TESTMENU      200
#define CM_TEST           101
#define CM_EXIT           102

#define IDD_EDITTEST      201
#define IDC_SAMPLEEDIT    102
#define IDUNDO            103
#define IDCLOSE           104
#define IDI_APPICON        202

#endif
```

5. 现在创建例子程序的C语言源文件。本例子程序是用普通C语言编写的，但消息EM_UNDO和消息EM_CANUNDO的使用可以适用于任何编程环境。使用文本编辑器创建新源文件UNDO.C，在文件的顶部添加下列语句，这些代码说明了例子程序中使用的全局变量，还包含了编译此例子程序所需的文件，另外，定义了预处理器符号STRICT，以对例子程序中使用的Windows类型进行严格的类型检查。

```
/* ------------------------------------------------------------------- */
/*                                                                     */
/* MODULE: UNDO.C                                                      */
/* PURPOSE: This sample application demonstrates how to use the        */
/*          edit control messages, EM_CANUNDO and EM_UNDO.             */
/*                                                                     */
/* ------------------------------------------------------------------- */

#include <windows.h>
#include <winnt.h>
#include "undo.rh"

static char *MainWindowClassName = "UndoTestWindow";
HINSTANCE hInstance = NULL;
```

6. 在同一源文件中添加下列代码。函数ProcessChangMessage用来响应编辑控制发送来的通知消息EN_CHANGE，此消息表示用户已经改变了编辑控制中的内容。此函数发送消息EM_CANUNDO给编辑控制，如果最近的改变可以撤消，则返回值非零（TRUE），如果改变不可撤消，则返回值为零（FALSE）。接着，此函数根据此返回值调用EnableWindow来激活或禁止按钮Undo。

```
/* ------------------------------------------------------------------- */
/* In response to an EN_CHANGE message from the edit control, this     */
/* function queries the edit control to see if the changes can be      */
/* undone. If the changes can be undone, it enables the Undo button,   */
/* otherwise it disables the Undo button.                              */
/* ------------------------------------------------------------------- */
void ProcessChangeMessage (HWND hWnd)
{
    BOOL canUndo;
    HWND hChild;

    // Find if the edit control can undo the change.
    canUndo = (BOOL) SendDlgItemMessage (hWnd, IDC_SAMPLEEDIT,
                                         EM_CANUNDO, 0, 0);

    // Get a handle to the button, and enable or disable it.
    hChild = GetDlgItem (hWnd, IDUNDO);
```

```
    if (canUndo)
        EnableWindow (hChild, TRUE);
    else
        EnableWindow (hChild, FALSE);
}
```

7. 下一个要添加的函数是 TestDialogProc，此回调函数用来处理传送给对话框的消息。为了响应消息 WM_INITDIALOG，此函数在编辑控制中设置初始文本。为了处理编辑控制发送来的通知和用户点击按钮的操作，此函数还响应消息 WM_COMMAND，为了响应命令消息 IDUNDO，此消息在点击按钮 Undo 时生成，此函数发送消息 EM_UNDO 给编辑控制。

在本例子程序中，调用 SendDlgItemMessage 来发送消息 EM_UNDO，并指定对话框句柄和编辑控制的控制 ID。也可以使用另外一种方法，即调用 GetDlgItem 获取编辑控制句柄，调用 SendMessage 发送消息 EM_UNDO。

```
/* ------------------------------------------------------------------------ */
/* This is the dialog function for the dialog. It returns TRUE to           */
/* the WM_INITDIALOG message to accept the default focussed control.        */
/* The function handles the EN_CHANGE notification by using the             */
/* EM_CANUNDO message to see if the edit control can undo changes.          */
/* If it can, the Undo button is enabled, otherwise it is grayed.           */
/* If the Undo button is clicked, the EM_UNDO message is used to            */
/* undo the changes.                                                        */
/* ------------------------------------------------------------------------ */
BOOL CALLBACK TestDialogProc (HWND hWnd, UINT message,
                              WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            SetDlgItemText (hWnd, IDC_SAMPLEEDIT,
                            " This is a sample edit control"
                            " in a small application. Make"
                            " some changes, and click Undo.");
            return TRUE;
        case WM_COMMAND:
            if (HIWORD (wParam) == EN_CHANGE)
                ProcessChangeMessage (hWnd);
            else
                if (wParam == IDUNDO)
                    SendDlgItemMessage (hWnd, IDC_SAMPLEEDIT, EM_UNDO, 0, 0);
                else
                    if (wParam == IDCLOSE)
                        EndDialog (hWnd, 1);
            break;
```

```
        default:
            return FALSE;
    }
    return TRUE;
}
```

8. 现在添加本例子程序其余的函数。在同一源文件 UNDO.C 中添加下列代码，一旦添加了这些代码，便可以编译并运行此例子程序。函数 MainWndProc 是回调函数，用来响应发送给例子程序主窗口的消息，此函数分别以显示测试对话框来响应菜单项 Test，以关闭例子程序来响应菜单项 Exit。

函数 InitAppliation 用来初始化并注册例子程序主窗口的窗口类，只有在此窗口类还没有注册时才会调用此函数。函数 InitInstance 调用函数 CreateWindow 和函数 ShowWindow 来创建并显示主窗口。最后一个函数 WinMain 是例子程序的入口点，创建例子程序并循环处理信息直到例子程序结束。

```
/* ---------------------------------------------------------------------- */
/* This function is the main window callback function which will be      */
/* called by Windows to process messages for our window.                  */
/* ---------------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            if (wParam == CM_TEST)
                DialogBox (hInstance, MAKEINTRESOURCE (IDD_EDITTEST),
                           hWnd, TestDialogProc);
            else
                if (wParam == CM_EXIT)
                    DestroyWindow (hWnd);
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
/* ---------------------------------------------------------------------- */
/* This function initialises a WNDCLASS structure and uses it to          */
/* register a class for our main window.                                  */
/* ---------------------------------------------------------------------- */
```

```
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}
/* ------------------------------------------------------------------ */
/* This function creates an instance of our main window. The window   */
/* is given a class name and a title, and told to display anywhere.   */
/* the nCmdShow argument passed to the program determines how the     */
/* window will be displayed.                                          */
/* ------------------------------------------------------------------ */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;          // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Edit Undo Demo",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);        // Send a WM_PAINT message.
    return TRUE;
}
/* ------------------------------------------------------------------ */
/* The main entry point for Windows applications. Check to see if     */
/* the main window class name has already been registered, if it has  */
```

```
/* not, call InitApplication to register it. Call InitInstance to          */
/* create an instance of our main window, then pump messages until         */
/* the application is closed.                                              */
/* ----------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

9. 编译并运行此例子程序。

注释

在编辑控制中实现单层撤消功能的方法是相当简单的，在应用程序中应该实现此功能，以使应用程序功能更加完善。但如果要使应用程序更加专业化，则需要实现多层撤消功能，此功能允许用户撤消一系列的改变，实现此功能的方法则根据应用程序的不同而不同，字处理软件的实现方法与文本编辑器的实现方法会有很大的区别，与绘图软件的实现方法也会有很大的区别。下面是在应用程序中实现多级撤消功能的几点提示：

找出在哪儿发生了改变。在简单的应用程序中，只要处理通知消息 EM _ CHANGE 就足够了，但在稍复杂的应用程序中，可能需要对各种操作（剪切，拷贝，粘贴，删除）进行分别处理，在字处理软件中，可能还需要处理格式信息。

对改变的文本进行确定。在一个大的文档中，只能对删除或修改的文本进行保存，因为同时保存整个文件的好几个拷贝是不现实的。

维护一个 FIFO（先进先出）列表或数组来保存每次的修改记录。在添加新的修改记录时可以删除旧的修改记录。

6.6 确认编辑控制中的输入有效

问题

如何确认用户在编辑控制中键入的键是有效的，从而只让正确的数据输入呢？

方法

确认数据的有效性对使用MS－DOS的程序员来说是很简单的，但对Windows程序员来说却并不简单。DOS应用程序一般具有对系统的控制权，因此可以方便地确认用户的输入是否有效，但在Windows系统中，用户输入由操作系统来管理，应用程序只在输入结束（通常是用户点击按钮OK）后才介入。

但是，在用户输入数据时在编辑控制中是可以实现一些检查的，如果需要也可以改变数据的格式，有许多窗口和编辑控制消息可以用来管理编辑控制的状态并完成适当的操作。

在本节的例子程序中，将介绍如何利用子类的概念来控制编辑控制中的输入。

什么是子类？

通常，Windows控制类（如EDIT类或BUTTON类）的所有消息都将发送给此类的窗口函数，此函数响应这些消息，从而实现控制的所有行为。

利用子类的概念可以改变个别控制的行为。如果有的程序员在应用程序中为某个控制创建了子类，则Windows就不再调用此控制的标准函数，而是调用程序员自己编写的函数，在程序员自己编写的函数中，可以为控制响应消息并实现所需要的行为。

有时，可能只需要改变控制对几个消息的响应，其他的消息则可以调用父类的处理函数来处理。

函数GetWindowLong（参数为GWL _ WNDPROC）可以用来获取父类函数的地址，函数SetWindowLong可以用来保存程序员编写的新函数，要调用原来的函数，可以使用函数CallWindowProc。注意，在撤消控制之前必须恢复原来的窗口函数。

步骤

按照下列步骤实现例子程序VALIDATE.EXE。打开并运行此例子程序，从菜单File中选择菜单项Test，弹出的对话框如图6-6所示。

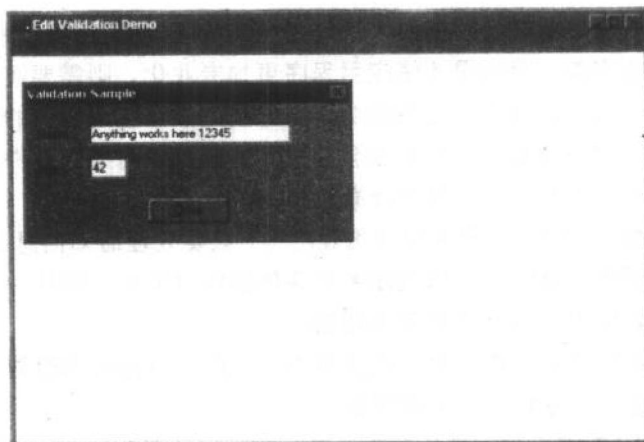


图6-6　具有确认功能的编辑控制

此对话框包含两个控制。编辑控制Name让用户输入30字符的名字，可以是任何字符的组合。编辑控制Age是进行有效确认的域，只允许输入数字字符，当用户企图插入其他字符

(如：字母）时，控制将会振铃。这是一类非常简单的有效确认，并且由于编辑控制中的内容总是有效的（或是空格或是数字字符），所以在用户离开此控制时，例子程序不需要确认控制中的内容是否有效。在用户离开控制时如何进行有效确认将在本节的末尾讨论。

实现例子程序的具体步骤如下：

1. 创建目录 VALIDATE，作为本例子程序的工作目录。

2. 创建例子程序的图标。使用资源编辑器创建一个 16 色的 32×32 象素的图标，并保存在文件 VALIDATE.ICO 中。

3. 现在创建例子程序的资源脚本。使用文本编辑器创建新文件 VALIDATE.RC，并在此文件中输入下列代码，此资源脚本定义例子程序的对话框、菜单和图标。

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/* MODULE: VALIDATE.RC                                                */
/* PURPOSE: This resource script defines the menu and test dialog    */
/*          for the sample application.                               */
/*                                                                    */
/* ------------------------------------------------------------------ */

#include <windows.h>
#include "validate.rh"

IDD_EDITTEST DIALOG 6, 15, 202, 79
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
      WS_CAPTION | WS_SYSMENU
CAPTION "Validation Sample"
FONT 8, "MS Sans Serif"
{
   LTEXT "Name", -1, 7, 13, 23, 8
   LTEXT "Age", -1, 7, 33, 18, 8
   EDITTEXT IDC_NAME, 39, 11, 126, 12
   EDITTEXT IDC_AGE, 39, 31, 24, 12
   PUSHBUTTON "&Close", IDCLOSE, 76, 55, 50, 14
}

IDM_TESTMENU MENU
{
   POPUP "&File"
   {
      MENUITEM "&Test", CM_TEST
      MENUITEM SEPARATOR
      MENUITEM "E&xit", CM_EXIT
   }
```

```
}

IDI_APPICON ICON "validate.ico"
```

4. 在资源脚本中包含了include文件VALIDATE.RH，因此要编译资源脚本便需要创建此文件，此文件用来定义资源标识符，在源代码中也包含了此文件。创建此文件并在此文件中添加下列标识符。

```
#ifndef __VALIDATE_RH
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: VALIDATE.RH                                                    */
/* PURPOSE: This include file defines the resource identifiers used.      */
/*                                                                        */
/* ---------------------------------------------------------------------- */

#define __VALIDATE_RH

#define IDM_TESTMENU      200
#define CM_TEST           101
#define CM_EXIT           102

#define IDD_EDITTEST      201
#define IDC_NAME          101
#define IDC_AGE           102
#define IDCLOSE           104

#define IDI_APPICON       202

#endif
```

5. 现在准备创建例子程序的C语言源文件。本例子程序使用标准C语言和API，以便于介绍如何使用API函数，当然在其他的任何开发环境中都可以重新实现本例子程序。创建新文件VALIDATE.C，并在此文件的顶部添加下列代码，此代码段包含了编译本例子程序所需要的文件，同时还定义了全局变量，其中最需要注意的变量是oldAgeProc，此变量用来保存原编辑控制类的函数地址。

```
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: VALIDATE.C                                                     */
/* PURPOSE: This applications demonstrateshow to subclass an edit         */
/*          control to perform validation of user input.                  */
/*                                                                        */
/* ---------------------------------------------------------------------- */
#define STRICT
```

```
#include <windows.h>
#include <winnt.h>
#include "validate.rh"

static char *MainWindowClassName = "ValidateTestWindow";
HINSTANCE hInstance = NULL;
LONG       oldAgeProc = 0;
```

6. 在源文件中添加函数 AgeWndProc。此函数是一个回调函数，用来替代编辑控制 Age 的缺省窗口函数，此函数处理发送给编辑控制的消息 WM_CHAR，只允许从 0 到 9 的字符以及 BACKSPACE 键（简称 UK_BACK），对于其他的字符则调用函数 MessageBeep 来振铃。最后，调用函数 CallWindowProc（并传送参数和函数地址）来将消息 WM_CHAR 及其他的消息传送给原编辑控制函数。

```
/* ------------------------------------------------------------ */
/* This window procedure is used to handle messages from an edit  */
/* control which is to be validated.                              */
/* ------------------------------------------------------------ */
LPARAM CALLBACK AgeWndProc (HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
    // Validate character input.
    if ( (message == WM_CHAR) && (wParam != VK_BACK) &&
         ( (wParam < '0') || (wParam > '9')))
    {
        MessageBeep (-1);
        return 0;
    }

    // For other events, call the original edit control procedure.
    return CallWindowProc ( (WNDPROC) oldAgeProc,
                           hWnd, message, wParam, lParam);
}
```

7. 下一个要添加的函数是 TestDialogProc。此回调函数处理传送给对话框的消息，特别是消息 WM_INITDIALOG 和消息 WM_DESTROY，以及用户按下按钮 Close 时生成的命令消息 IDCLOSE。为了响应消息 WM_INITDIALOG，此函数发送消息 EM_LIMITTEXT 给编辑控制 name 和 age，限制可输入的字符数分别是 30 和 3。接着以 GWL_WNDPROC 为参数调用函数 SetWindowLong 来以前面输入的函数 AgeWndProc 替代编辑控制的窗口函数，返回的原窗口函数保存在变量 OldAgeProc 中。为了响应消息 WM_DESTROY，此函数再一次调用函数 SetWindowLong，用来在撤消编辑控制前恢复原窗口函数。

```
/* ------------------------------------------------------------ */
/* This is the dialog function for the dialog. It returns TRUE to */
/* the WM_INITDIALOG message to accept the default focussed control. */
```

```
/* In addition, it subclasses the age edit control, so that messages          */
/* for that control go to AgeWndProc rather than the default class            */
/* procedure. This function responds to WM_DESTROY by removing the            */
/* subclassing, restoring the original window procedure.                      */
/* ------------------------------------------------------------------------ */
BOOL CALLBACK TestDialogProc (HWND hWnd, UINT message,
                              WPARAM wParam, LPARAM lParam)
{
   HWND hChild;

   switch (message)
   {
      case WM_INITDIALOG:
          // Set the maximum name length.
          SendDlgItemMessage (hWnd, IDC_NAME, EM_LIMITTEXT, 30, 0);
          /* Subclass the edit control — replace the default
            procedure with your own — for this control only. */
          hChild = GetDlgItem (hWnd, IDC_AGE);
          if (hChild)
          {
             SendMessage (hChild, EM_LIMITTEXT, 3, 0);
             oldAgeProc = SetWindowLong (hChild,
                                         GWL_WNDPROC,
                                          (LONG) AgeWndProc);
          }
          return TRUE;
      case WM_COMMAND:
          if (wParam == IDCLOSE)
             EndDialog (hWnd, TRUE);
          return TRUE;
      case WM_DESTROY:
          // Remove the subclassing.
          hChild = GetDlgItem (hWnd, IDC_AGE);
          SetWindowLong (hChild, GWL_WNDPROC,
                         oldAgeProc);
          return TRUE;
   }
   return FALSE;
}
```

8. 在同一源文件 VALIDATE.C 中添加下列代码。由于下面的四个函数构成了例子程序的简单框架，因此放在一块考虑。函数 MainWndProc 处理发送给例子程序主窗口的消息，在用户选择菜单项 Test 时激活测试对话框，在用户从菜单 File 中选择菜单项 Exit 时关闭例子

程序。函数 InitApplication 用来注册主窗口的窗口类，只有在此类不存在时才调用此函数。函数 InitInstance 调用 CreateWindow 和 ShowWindow 来创建并显示例子程序的主窗口。

最后一个函数 WinMain 在例子程序启动时从 Windows 系统接收控制权，也就是此函数调用前面的函数来建立并运行例子程序。

```
/* ------------------------------------------------------------------ */
/* This function is the main window callback function which will be  */
/* called by Windows to process messages for our window.              */
/* ------------------------------------------------------------------ */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            if (wParam == CM_TEST)
                DialogBox (hInstance, MAKEINTRESOURCE (IDD_EDITTEST),
                           hWnd, TestDialogProc);
            else
                if (wParam == CM_EXIT)
                    DestroyWindow (hWnd);
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            break;
        default:
            return DefWindowProc (hWnd, message, wParam, lParam);
    }
    return 0;
}
/* ------------------------------------------------------------------ */
/* This function initialises a WNDCLASS structure and uses it to      */
/* register a class for our main window.                              */
/* ------------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
```

```
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;
    return RegisterClass (&wc);
}
/* -------------------------------------------------------------------- */
/* This function creates an instance of our main window. The window     */
/* is given a class name and a title, and told to display anywhere.     */
/* the nCmdShow argument passed to the program determines how the       */
/* window will be displayed.                                            */
/* -------------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;        // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Edit Validation Demo",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);      // Send a WM_PAINT message.
    return TRUE;
}
/* -------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if       */
/* the main window class name has already been registered, if it has    */
/* not, call InitApplication to register it. Call InitInstance to       */
/* create an instance of our main window, then pump messages until      */
/* the application is closed.                                           */
/* -------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
```

```
    if (! InitApplication (hInstance))
        return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
     }
    return (msg.wParam);
}
```

9. 编译并运行此例子程序。

注释

在本节的例子程序中，以非常简单的方法示范了如何利用子类的概念来改变控制的行为以及在域一级实现有效确认。同样的技巧还可以用来实现更加复杂的域一级的有效确认，包括获取和重新格式化编辑控制的文本缓冲区。如果在应用程序中需要经常使用函数 SetWindowText 或消息 WM _SETTEXT 来改变编辑控制缓冲器，则应该发送消息 EM _SETMODIFY 给编辑控制以保持正确状态。

许多程序员特别关心在用户离开域时如何对域进行有效确认。由于此功能与界面的本质是相违背的，因此在 Windows 应用程序中是不提倡使用此功能的。有的用户可能开始时在域中输入数据，后来又去操作其他控制，这在 Windows 系统中是完全可以接受的。一般来说，只有在用户点击按钮 OK 或等同的操作后，才认为数据输入结束，大多数文本域有效确认函数强制用户停留在文本域上，直到用户输入结束，这是同 Windows 用户界面不兼容的。

如果程序员准备在 Windows 上实现一个传统的数据输入应用程序，则是可以实现域级有效确认的。消息 WM _KILLFOCUS 适合在这里使用，此消息在控制失去键盘焦点前发送给控制。但是，在处理此消息时应用程序不必显示消息框或其他的窗口，也不必设法将焦点改回控制，而应该使用 PostMessage 发送一个用户定义消息给窗口函数。当窗口函数接收此用户定义消息时，如有必要，应用程序就可以显示错误并将焦点返回编辑控制。

6.7 利用剪贴板进行剪切和粘贴操作

问题

用户可以利用 CTRL+C 来拷贝文本，利用 CTRL+V 来粘贴文本，利用 CTRL+X 来剪切文本，但是，有时程序员希望应用程序能够更方便的使用。如何在菜单中实现这些功能并使窗口中的编辑控制响应这些菜单呢?

方法

在编辑控制中剪切、拷贝或粘贴数据以及删除数据都是非常简单的。要实现这些功能，只需分别发送消息 WM _CUT、WM _COPY、WM _PASTE 和 WM _CLEAR 给编辑控制就

可以了。剪切、拷贝和删除消息是对控制中的任何选中文本进行操作，而粘贴消息是将剪贴板上的任何文本插入到编辑控制中的当前位置，并替代任何选中的文本，因此实现编辑菜单的应用程序只需发送适当的消息给控制就可以响应菜单命令了。

如果用户在编辑控制中未选中任何文本那将如何呢？如果没有选中的文本，则执行剪切、拷贝和删除操作就没有任何作用，因此菜单项 cut、copy 和 clear 有效就没有任何意义。同样，如果剪贴板上没有文本，则菜单项 paste 也就不应该有效。

本节的例子程序有两个目的。第一个目的是介绍如何发送剪切、拷贝、删除和粘贴消息给编辑控制；第二个目的是介绍如何响应消息 WM_INITMENU，以便适当地激活和禁止菜单项，有关激活和禁止菜单项的信息在 8.1 节中还将详细介绍。

步骤

按照下列步骤实现例子程序 PASTE.EXE。打开并运行此例子程序，将显示一个包含多行编辑控制的窗口，此控制最多允许键入 32K 字节的文本，而且其操作类似于 Windows 95 的应用程序 Notepad。除了一个简单的菜单 File 外，还有一个包含菜单项 Cut、Copy、Delete 和 Paste 的菜单 Edit，菜单项 Delete 实现前面介绍的 WM_CLEAR 的行为。在编辑控制中输入一些文本，选择菜单中的菜单项以查看菜单项的行为。当在编辑控制中选中一些文本后，菜单项 Cut、Copy 和 Delete 是激活的，否则是禁止的，同样，只有在剪贴板上有文本时，菜单项 Paste 才是激活的。运行的例子程序如图 6-7 所示。

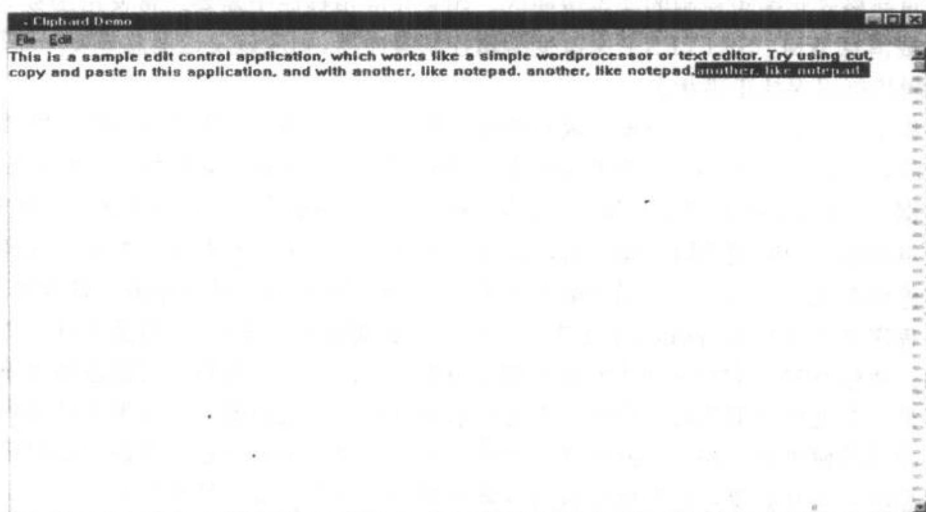


图 6-7 在编辑控制中使用剪切、拷贝和粘贴

实现例子程序的具体步骤如下：

1. 创建新目录 CUTPASTE，作为例子程序的工作目录，以存放所有的源文件。

2. 使用文本编辑器创建新文件 PASTE.RC，在文件中输入下列资源脚本。资源脚本定义应用程序中使用的菜单 File 和菜单 Edit，还包含应用程序的图标。

```
/* ----------------------------------------------------------- */
/*                                                             */
/* MODULE: PASTE.RC                                            */
/* PURPOSE: This resource script defines the menu and test dialog */
```

```
/*          for the sample application.                                  */
/*                                                                       */
/* ------------------------------------------------------------------- */

#include <windows.h>
#include "paste.rh"

IDM_TESTMENU MENU
{
    POPUP "&File"
     {
        MENUITEM "E&xit", CM_EXIT
    }
    POPUP "&Edit"
     {
        MENUITEM "Cu&t", CM_CUT
        MENUITEM "&Copy", CM_COPY
        MENUITEM "&Delete", CM_CLEAR
        MENUITEM "&Paste", CM_PASTE
    }
}
IDI_APPICON ICON "paste.ico"
```

3. 为例子程序创建图标。使用资源编辑器创建一个16色的32×32象素的图标，并保存在文件PASTE.ICO中。

4. 在资源脚本中包含了文件PASTE.RH，此文件用来定义资源标识符，在C语言源代码中也可引用这些资源。使用文本编辑器创建新文件PASTE.RH，在此文件中输入下列定义：

```
#ifndef __PASTE_RH
/* ------------------------------------------------------------------- */
/*                                                                       */
/* MODULE: PASTE.RH                                                      */
/* PURPOSE: This include file defines the resource identifiers used.    */
/*                                                                       */
/* ------------------------------------------------------------------- */
#define __PASTE_RH

#define IDM_TESTMENU     200
#define CM_EXIT          101
#define CM_CUT           102
#define CM_COPY          103
#define CM_CLEAR         104
```

```
#define CM_PASTE          105
#define IDC_SAMPLEEDIT    201

#define IDI_APPICON       202

#endif
```

5. 现在准备编写例子程序的C语言源代码。象本书的其他例子程序一样，本例子程序也是用C语言编写的，在大多数的其他应用程序开发环境中使用同样的API函数和消息也可以非常容易地实现同样的功能。

创建新文件PASTE.C，在此文件的顶部添加下列代码段。此代码段包含了编译此例子程序所需的文件，还定义了预处理器符号STRICT，以便在编译时能够发现窗口和菜单句柄错误，从而避免运行时出错，此段代码还定义了几个全局变量，其中有变量hEdit，用来保存多行编辑控制的句柄。

```
/* ------------------------------------------------------------ */
/*                                                              */
/* MODULE: PASTE.C                                              */
/* PURPOSE: This sample application demonstrates how to use the */
/*          Windows messages, WM_CUT, WM_COPY and WM_PASTE to   */
/*          to implement clipboard cut copy and paste from an edit */
/*          control.                                            */
/*                                                              */
/* ------------------------------------------------------------ */
#define STRICT
#include <windows.h>
#include <winnt.h>
#include "paste.rh"

static char *MainWindowClassName = "PasteTestWindow";
HINSTANCE hInstance = NULL;
HWND      hEdit = NULL;
```

6. 在同一源文件中添加函数CreateEditControl。此函数调用CreateWindow来创建一个EDIT类的新窗口，即编辑控制。由于例子程序在后面要响应消息WM_SIZE，以确保编辑控制能够填充整个窗口的用户区，所以在此函数中没有设置窗口的大小。

```
/* ------------------------------------------------------------ */
/* This function uses CreateWindow to create an edit contol within */
/* the main window. It adds some sample text to the edit control. */
/* ------------------------------------------------------------ */
HWND CreateEditControl (HWND hWndParent)
{
   HWND hChild;
```

```
    // Create the edit control using CreateWindow.
    hChild = CreateWindow ("EDIT", NULL,
                           WS_CHILD | WS_VISIBLE | WS_VSCROLL |
                           ES_LEFT | ES_MULTILINE | ES_AUTOVSCROLL,
                           0, 0, 0, 0, /* size will be set be WM_SIZE */
                           hWndParent, (HMENU) IDC_SAMPLEEDIT,
                           hInstance, NULL);

    // Place some sample text in the control.
    SetWindowText (hChild,"This is a sample edit control "
                          "application, which works like "
                          "a simple wordprocessor or text "
                          "editor. Try using cut, copy "
                          "and paste in this application, "
                          "and with another, like notepad. ");
    return hChild;
}
```

7. 下一个要添加的函数是 SetupEditMenu，每当例子程序接收消息 WM_INITMENU 时都将调用此函数。在用户点击菜单或下拉菜单时，消息 WM_INITMENU 被发送，在显示任何菜单之前，此消息发送给例子程序，从而提供了很好的机会可以根据例子程序的状态来激活或禁止菜单项。

在此函数中，首先发送消息 EM_GETSEL 给编辑控制，以确定是否有选中的文本。编辑控制响应此消息，返回当前选中文本的起始和结束位置，其中起始位置在返回值的低字节。结束位置在返回值的高字节。如果这两个值相等，则表明没有选中的文本，此函数就调用函数 EnableMenuItem（标志为 MF_GRAYED）来禁止菜单项；如果有选中的文本，则调用同样的函数 EnableMenuItem（标志为 MF_ENABLED）来激活菜单项。关于各种菜单项标志的讨论请见 8.1 节。

接着，此函数调用函数 IsClipboardFormatAvailable（参数为 CF_TEXT）来确定剪贴板上是否有文本，从而确定是否激活菜单项 Paste。如果剪贴板上有所要求格式的数据，则函数 IsClipboardFormatAvailable 的返回值为 TRUE；如果剪贴板上没有数据，或者数据的格式不对，则返回值为 FALSE。

```
/* ----------------------------------------------------------------- */
/* This function checks to see if any text is selected in the edit   */
/* control. If it is, then cut, copy and delete are enabled in the   */
/* menu, otherwise they are grayed. It also checks to see if there   */
/* is any compatible text in the clipboard. If there is, then the    */
/* paste option is enabled, otherwise it is grayed.                  */
/* ----------------------------------------------------------------- */
void SetupEditMenu (HMENU hMenu, HWND hEdit)
```

```
{
    DWORD selection;
    int   selStart, selEnd;

    // See if any text is selected.
    selection = SendMessage (hEdit, EM_GETSEL, 0, 0);
    selStart = LOWORD (selection);
    selEnd = HIWORD (selection);
    if (selStart != selEnd)
     {
        // Text is selected — enable menu items.
        EnableMenuItem (hMenu, CM_CUT, MF_BYCOMMAND | MF_ENABLED);
        EnableMenuItem (hMenu, CM_COPY, MF_BYCOMMAND | MF_ENABLED);
        EnableMenuItem (hMenu, CM_CLEAR, MF_BYCOMMAND | MF_ENABLED);
    }
    else
     {
        // No text selected — disable items.
        EnableMenuItem (hMenu, CM_CUT, MF_BYCOMMAND | MF_GRAYED);
        EnableMenuItem (hMenu, CM_COPY, MF_BYCOMMAND | MF_GRAYED);
        EnableMenuItem (hMenu, CM_CLEAR, MF_BYCOMMAND | MF_GRAYED);
    }

    // Now see if there is any text in the clipboard.
    if (IsClipboardFormatAvailable (CF_TEXT))
        EnableMenuItem (hMenu, CM_PASTE, MF_BYCOMMAND | MF_ENABLED);
    else
        EnableMenuItem (hMenu, CM_PASTE, MF_BYCOMMAND | MF_GRAYED);
}
```

8. 现在添加回调函数来处理传送给窗口的消息。函数 MainWndProc 响应许多消息，并调用函数 DefWindowProc 来处理其余的消息，此函数调用 CreatEditControl 来响应消息 WM_CREATE，调用 SetupEditMenu 来响应消息 WM_INITMENU。此函数还处理发送给窗口的消息 WM_SIZE，为了响应此消息，此函数调用 API 函数 MoveWindow 来重新设置编辑控制的大小，使其占有整个窗口区域。

为了响应用户选择菜单发送的消息 WM_COMMAND，此函数发送适当的消息给编辑控制。对应菜单项 Paste 发送消息 WM_PASTE，对应菜单项 Cut 发送消息 WM_COPY，对应菜单项 Delete 发送消息 WM_CLEAR。这些消息都不需要任何参数，所以传送给 SendMessage 的参数 LPARAM 和 WPARAM 都置为零。

```
/* ---------------------------------------------------------------- */
/* This function is the main window callback function which will be */
/* called by Windows to process messages for our window.            */
/* ---------------------------------------------------------------- */
```

```
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
            hEdit = CreateEditControl (hWnd);
            return 0;
        case WM_INITMENU:
            SetupEditMenu ( (HMENU) wParam, hEdit);
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_CUT:
                    SendMessage (hEdit, WM_CUT, 0, 0);
                    return 0;
                case CM_COPY:
                    SendMessage (hEdit, WM_COPY, 0, 0);
                    return 0;
                case CM_CLEAR:
                    SendMessage (hEdit, WM_CLEAR, 0, 0);
                    return 0;
                case CM_PASTE:
                    SendMessage (hEdit, WM_PASTE, 0, 0);
                    return 0;
                case CM_EXIT:
                    DestroyWindow (hWnd);
                    return 0;
            }
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            return 0;
        case WM_SETFOCUS:
            SetFocus (hEdit);
            return 0;
        case WM_SIZE:
            MoveWindow (hEdit, 0, 0, LOWORD (lParam),
                        HIWORD (lParam), TRUE);
            return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

9. 在同一源文件中添加下面三个函数，这三个函数构成了例子程序的框架，创建窗口并处理消息直到例子程序结束。函数 InitApplication 用来注册主窗口的窗口类，函数 InitInstance 用来创建并显示主窗口，函数 WinMain 是例子程序的入口点，调用 GetMessage 和 DispatchMessage 来处理消息队列。

```
/* ------------------------------------------------------------ */
/* This function initialises a WNDCLASS structure and uses it to */
/* register a class for our main window.                        */
/* ------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (NULL, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ------------------------------------------------------------ */
/* This function creates an instance of our main window. The window */
/* is given a class name and a title, and told to display anywhere. */
/* the nCmdShow argument passed to the program determines how the   */
/* window will be displayed.                                        */
/* ------------------------------------------------------------ */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;          // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName,"Clipbard Demo",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInstance, NULL);
```

```
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);           // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if              */
/* the main window class name has already been registered, if it has           */
/* not, call InitApplication to register it. Call InitInstance to              */
/* create an instance of our main window, then pump messages until             */
/* the application is closed.                                                   */
/* ---------------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return (FALSE);

    if (! InitInstance (hInstance, nCmdShow))
        return (FALSE);

    while (GetMessage (&msg, NULL, NULL, NULL))
     {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}
```

10. 编译并运行此例子程序。

注释

在有的应用程序中，可能只是简单地实现了剪切和粘贴等编辑功能，那么利用本节所介绍的激活和禁止方法可以增强应用程序的功能，从而使得应用程序更简练更专业。

第7章 列 表 框

有许多交互方式可以使得用户从特定的选项集合中进行选取操作，但列表框大概是这类交互中最自然最普遍的方式之一。在 Windows 95 中，列表框被用于各种应用环境中，并以多种方式来完成与用户的交互。从简单的单选列表（如：确定安装哪一个打印机驱动程序）到复杂的目录“树”列表，列表框成为 Windows 95 用户界面的核心。

在应用程序中，也需要同样的交互手段。从选择要处理的报告到选择应用程序的选项，都需要在应用程序中使用列表框。在本章中，将介绍如何使用 Windows 95 API 的强大功能来增强列表框的功能，从而使得用户更加容易理解和操作。

1. 如何随同列表项存储信息

在当前的应用程序中，大多数的列表框都是用来选取列表中的标识符，而标识符连接着其他的数据，例如：在有的应用程序中，显示客户名字的列表框与客户信息数据库是相连接的。在本节中，将介绍一个非常容易的方法来将这两部分信息（客户名和客户信息）连接起来，以简化程序员的工作。

2. 如何捕捉列表框中的双击

程序员对列表框非常恼火的一点是：对于用户在列表框中的双击，列表框没有相应的预定义操作。因此在许多对话框中，除了包含列表框外还需要包含按钮 OK（和任选的 Cancel），当用户从列表中选择列表项后，还必须接着点击按钮 OK。假如用户可以通过双击列表项来同时完成列表项的选取和对话框的关闭，那么应用程序会更加简洁更加易于使用。在本节中，将介绍实现此功能的方法。

3. 如何通过拖放在列表框中移动列表项

在应用程序中，经常使用列表框来选择要拷贝、打印、或进行其他操作的列表项。很多情况下，列表项的次序和列表项连接的数据同样重要。尽管有好几个方法可以用来实现列表框中列表项的选择和移动，但最容易的实现方法是：选择列表中的列表项，然后拖动到新的位置上。在本节中将介绍一个快捷方便的实现方法，只利用 Windows 95 API 的强大功能并以最少的代码来完成所需的功能。

4. 如何滚动列表框

在许多应用程序中，列表框的上面还有一个编辑框，当用户在编辑框中键入字符时，列表框则根据键入的字符做相应地滚动。在普通的列表框中，一般也具有某种自动滚动的功能，当用户在列表框中按下一个字符时，列表框自动滚动到与此字符相匹配的列表项上。但是，此功能只限于第一个字符，当用户按下第二个字符时，此字符将被看作第一个字符，列表框则滚动到相应的列表项上。在本节中，将介绍如何根据用户在编辑框中键入的字符串来自动滚动或重新定位列表框到特定的列表项上。

5. 如何实现宽列表的水平滚动

有时，在列表框中显示的信息比列表框窗口要宽，因此用户就有一部分信息看不到，而有时这些信息对用户来说又是至关重要的。但是，即使在列表框中添加上水平滚动条，在 MFC

中也没有直接的方法来实现列表框的水平滚动。在本节中，将介绍如何利用 Windows 95 API 的强大功能来实现列表框的水平滚动。

6. 如何在列表框中右对齐数字

普通的列表框只允许以左对齐的方式显示列表项，不支持其他的显示方式。但是，有些数据需要以右对齐的方式显示，例如：金额等等，许多人看金额栏目中的数字都习惯于从栏目的右边开始，而不是从左边开始，因此，有时普通的列表框就非常不适用。在本节中，将介绍如何实现右对齐方式显示的列表框。

7. 如何实现自绘制列表框

在列表框中只能显示文本数据是没有什么道理的，应该还能够显示颜色、图象和其他的数据。而被称作“自绘制”的方法功能非常强大，在应用程序中使用非常简单高效，可以使列表更易选择并提供给用户更丰富的信息。在本节中，将介绍如何创建自绘制列表框，并提供给程序员一个简单的实现框架。

8. 如何在列表框中存放更多的列表项

列表框只能拥有一定数目的列表项（64K 文本或相应数目的列表项），这个数目对于大多数应用来说是足够的，但对于大型的数据库应用程序来说则有可能太小了。在本节中，将介绍如何利用 Windows 95 API 的强大功能以及上一节介绍的自绘制概念来扩大列表框中同时可显示的列表项数。

9. 如何实现层次列表（树）

在 Windows 95 中，为程序员提供了许多新的控制类型，树列表（或树视图）便是其中之一，使得程序员可以利用可扩展和可紧缩的节点来建立列表项层次树。在本节中，将介绍如何根据需求在应用程序中实现树列表，无论是显示目录树，还是显示程序结构图，树列表都可以大大节省程序员编写应用程序的时间和精力。本节所介绍的内容只适用于 Windows 95 和 Windows NT。

表 7-1 列出了本章使用的 Windows 95 API 函数。

表 7-1　第 7 章使用的 Windows 95 API 函数

GetDlgItem	GetWindowText	SetWindowText	SetFocus
SendMessage	PostMessage	SelectObject	SetROP2
MoveTo	LineTo	SetTimer	SetCapture
KillTimer	ReleaseCapture	GetTextExtent	EnableWindow
InvalidateRect	DrawFocusRect	InvertRect	DrawText
FillRect	SetTextColor	PatBlt	SetBkColor
SetBkMode	SetCursor		

7.1 随同列表项存储信息

问题

假如某个对话中有个列表框，包含着某商人的所有客户的名字。对应于列表框中的内容，有一个内部的数据库，存储着客户的其他信息，例如：在列表中显示每个客户姓名，而在内部列表中包括的信息可能有每个客户的姓、名、地址、城市、国家以及邮编等等。

于是，每当用户从列表框中选取某个姓名，则需要遍历整个内部列表以便找到相对应的

信息。如果在列表框中以某种格式显示姓名（例如：在列表框中显示姓，逗号，名），则查询内部列表的操作会相当困难，必须首先解析此格式的姓名，然后才能进行匹配查询。

是否有更好的方法利用 Windows 95 API 来实现此功能呢？

方法

实际上确实有更好的方法利用 Windows 95 API 来实现此功能。Windows 系统中的列表框，除可见的列表内容外，还可一起存储信息。换句话说，除了置于列表框中的字符串外，还可保存一个指针，指向内部的信息，当列表框被改变时（例如：在有序列表中插入一个新的字符串），指针会自动地做些调整。

除了列表与指针自动保持同步这一优点外，此方法还有另一优点：Windows 系统维护指针，而不需要应用程序来维护。

在本节将仔细讨论此方法，为列表项创建附加信息，并将这些信息内部存储在 Windows 列表框中。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，从主菜单 Dialogs 中选择菜单项 Contact Dialog，将弹出一个对话框，如图 7-1 所示，在对话框左侧的编辑域中键入一些数据并点击按钮 Add Entry，则姓名将显示在对话框右侧的列表框中。继续输入一些商人的姓名和地址，接着，点击列表框中的某商人的姓名，则原来输入的此人的信息将填充在相应的编辑域中。

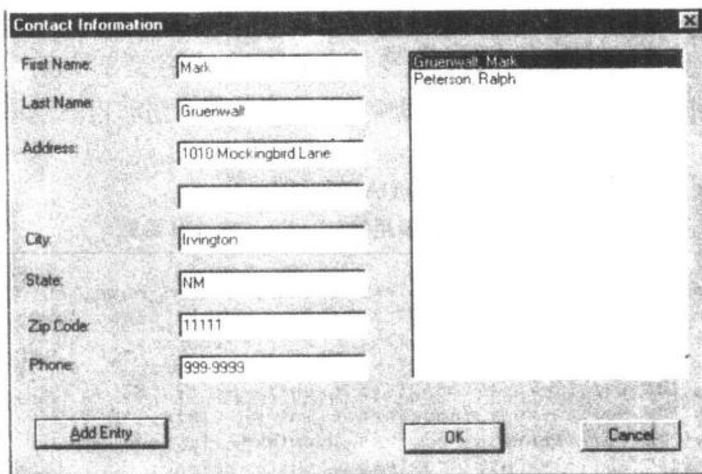


图 7-1　具有示范列表项的客户对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中利用 AppWizard 创建新的项目文件，命名为 CH71.MAK。进入 AppStudio，添加新的对话框，在新对话框中添加下列静态文本域：

First Name：

Last　Name：

Address：

City：

State:
Zip Code:
Phone:

2. 对每个静态文本域添加编辑框。在 Address: 和 City: 之间多添加一个编辑框，用于第二行地址，为编辑域接受缺省的名字 (IDC_EDIT)，并在 AppStudio 中清除编辑域中的任何文本。

3. 在对话框的底部添加一新的按钮，标题为 Add Entry，并接受缺省的名字 (IDC_BUTTON1)。

4. 在对话框的右侧添加列表框，列表框的名字为 IDC_LIST1，最后，改变对话框的标题为 Contact Information。

5. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，类名为 CContactDlg。从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 并输入新函数名为 OnAddEntry。在 CContactDlg 的方法 OnAddEntry 中输入下列代码:

```
void CContactDlg:: OnAddEntry ()
{
   // Create a new structure.

   ContactEntry *entry = new ContactEntry;

   // Load it from the dialog box

   GetDlgItem (IDC_EDIT1) ->GetWindowText ( entry->first_name, 80 );
   GetDlgItem (IDC_EDIT2) ->GetWindowText ( entry->last_name, 80 );
   GetDlgItem (IDC_EDIT3) ->GetWindowText ( entry->address_1, 80 );
   GetDlgItem (IDC_EDIT4) ->GetWindowText ( entry->address_2, 80 );
   GetDlgItem (IDC_EDIT5) ->GetWindowText ( entry->city, 80 );
   GetDlgItem (IDC_EDIT6) ->GetWindowText ( entry->state, 10 );
   GetDlgItem (IDC_EDIT7) ->GetWindowText ( entry->zip_code, 15 );
   GetDlgItem (IDC_EDIT8) ->GetWindowText ( entry->phone, 20 );

   // Add the last name + the first name to the list box

   CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);

   char buffer [256];
   sprintf (buffer, "%s, %s", entry->last_name, entry->first_name );

   int idx = list->AddString ( buffer );

   // Now, add the data for the list entry
```

```
    list->SetItemData ( idx, (DWORD) entry );

    // Clear the input fields

    // Set the fields

    GetDlgItem (IDC_EDIT1) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT2) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT3) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT4) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT5) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT6) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT7) ->SetWindowText ( "" );
    GetDlgItem (IDC_EDIT8) ->SetWindowText ( "" );

    // Reset the input focus to the first edit field

    GetDlgItem (IDC_EDIT1) ->SetFocus ();

}
```

6. 从对象列表中选择对象 IDC_LIST1，从消息列表中选择消息 LBN_SELCHANGE，点击按钮 Add Function，新函数命名为 OnListChange。在 CContactDlg 的方法 OnListChange 中输入下列代码：

```
void CContactDlg:: OnListChange ()
{
    // Get the current selection

    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    int idx = list->GetCurSel ();

    // Get the data associated with this item

    ContactEntry *entry = (ContactEntry *) list->GetItemData (idx);

    // Set the fields

    GetDlgItem (IDC_EDIT1) ->SetWindowText ( entry->first_name );
    GetDlgItem (IDC_EDIT2) ->SetWindowText ( entry->last_name );
    GetDlgItem (IDC_EDIT3) ->SetWindowText ( entry->address_1 );
    GetDlgItem (IDC_EDIT4) ->SetWindowText ( entry->address_2 );
    GetDlgItem (IDC_EDIT5) ->SetWindowText ( entry->city );
    GetDlgItem (IDC_EDIT6) ->SetWindowText ( entry->state );
```

```
  GetDlgItem (IDC_EDIT7) ->SetWindowText ( entry->zip_code );
  GetDlgItem (IDC_EDIT8) ->SetWindowText ( entry->phone );

}
```

7. 从对象列表中选择对象 IDOK，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function，新函数命名为 OnOK。在 CContactDlg 的方法 OnOK 中输入下列代码：

```
void CContactDlg:: OnOK ()
{
  CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
  for ( int i=0; i<list->GetCount (); ++i ) {
    ContactEntry *entry = (ContactEntry *) list->GetItemData (i);
    delete entry;
  }

  CDialog:: OnOK ();
}
```

8. 在 Visual C++中选择文件 CONTACTD.H，在文件的顶部添加下列结构定义：

```
typedef struct {
   char first_name [80];
   char last_name [80];
   char address_1 [80];
   char address_2 [80];
   char city [80];
   char state [10];
   char zip_code [15];
   char phone [20];
} ContactEntry;
```

9. 在 AppStudio 中，添加一新的主菜单，标题为 Dialogs，添加新的菜单项，标题为 Contact Dialog，标识符为 ID_CONTACT_DLG。

10. 在 ClassWizard 中，从下拉列表中选择应用程序对象 CCh71App，从对象列表中选择标识符 ID_CONTACT_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新函数命名为 OnContactDlg。在 CCh71App 的方法 OnContactDlg 中输入下列代码：

```
void CCh71App:: OnContactDlg ()
{
   CContactDlg dlg;
   dlg.DoModal ();
}
```

11. 最后，在文件 CH71.CPP 的顶部添加下列行：

```
#include   "contactd.h"
```

12. 编译并运行此例子程序。

用法

当用户选择了客户对话框并添加新的列表项时，例子程序为一个新的 ContactEntry 结构

分配空间，并用编辑域中的信息为此结构赋值，使用 CListBox 的 MFC 方法 SetItemData 将此结构添加到列表框的 ItemData 中。接着列表框维护一个指向此列表项数据的指针，对应的列表项用索引表示，索引通过 CListBox 的方法 AddString 来获取，方法 AddString 用来添加新字符串到列表中，并返回字符串在列表中的位置给应用程序。

如果用户从列表中选择了某项，则 CContactDlg 的方法 OnListChange 被调用。此方法取回当前选中的列表项的索引，接着调用函数 GetItemData 获取与此列表项相对应的指针，注意：由于函数 GetItemData 返回的是 DWORD（长整数）值，所以需要强制转换成确切的类型。接着将指针指向的数据填充到相应的编辑域中。

最后，当用户选择按钮 OK 后，使用同方法 OnListChange 一样的操作将每个存储的列表项从列表框中取回，接着删除指针，释放原来分配的内存空间。

注释

本节介绍了如何在列表框中存储和取回信息，这是非常有用的。以此为基础可以象本节一样实现客户信息的查询，也可实现其他的通过列表项查找信息的应用程序。为了示范真正的底层 Windows API 函数调用，下面利用 Delphi（没有相应的函数 SetItemData 和 GetItem-Data）实现同一例子程序。

运行 Delphi 实现的例子程序，将弹出如图 7-2 所示的表单。与前面的例子程序一样，在编辑域中输入一些客户信息并点击按钮 Add New Entry，则在列表框中将显示出相应的表项，继续输入一些列表项。从列表框中选取某项，则相应的数据将填充在编辑域中。

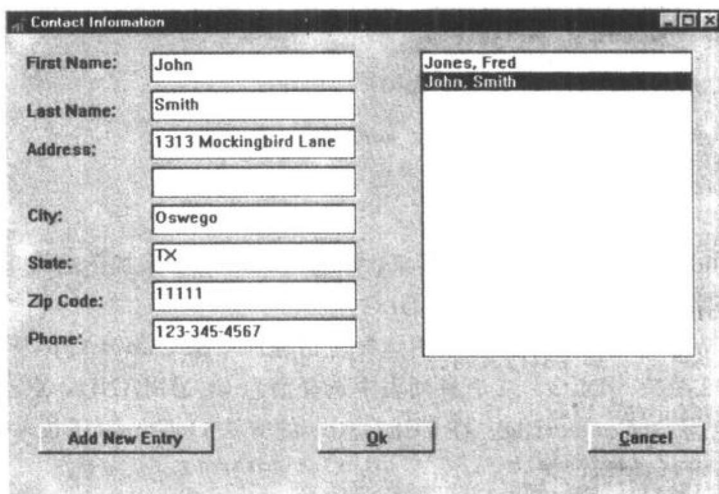


图 7-2 Delphi 实现的具有示范列表项的客户表单

1. 创建新的项目文件或在已存在的某个 Delphi 项目文件中添加新的表单。在此表单中，添加下列静态文本域：

First Name:

Last Name:

Address:

City:

State:

Zip Code:
Phone:

2. 按照顺序为每个静态文本域添加对应的编辑框。在 Address: 和 City: 之间多添加一个编辑框，用来存放地址的第二行。

3. 在表单中添加三个按钮：Add New Entry，OK 以及 Cancel。

4. 双击按钮 Add New Entry，在方法 TForm1. Button1Click 中添加下列代码：

```
procedure TForm1.Button1Click (Sender: TObject);
var
  entry: ^ ContactEntry;
  temp: String;
  idx:   Integer;
begin
   { Create a new entry variable }

   New (entry);

   { Assign the pieces of the entry variable }

   entry^.FirstName := Edit1.Text;
   entry^.LastName := Edit2.Text;
   entry^.Address1 := Edit3.Text;
   entry^.Address2  := Edit4.Text;
   entry^.City      := Edit5.Text;
   entry^.State     := Edit6.Text;
   entry^.ZipCode   := Edit7.Text;
   entry^.Phone     := Edit8.Text;

   { Put it into the list }

   temp := entry^.LastName + ', ' + entry^.FirstName;
   idx := ListBox1.Items.Add ( temp );

   { Send a message to store this information }

   SendMessage ( ListBox1.handle, LB_SETITEMDATA, idx, LongInt (entry) );

   { Clear the edit fields  }

   Edit1.Text := '';
   Edit2.Text := '';
   Edit3.Text := '';
   Edit4.Text := '';
```

```
   Edit5.Text := '';
   Edit6.Text := '';
   Edit7.Text := '';
   Edit8.Text := '';

   { Set the focus to the first edit field }

   Edit1.SetFocus;
end;
```

5. 双击列表框，在方法 TForm1.ListBox1Click 中添加下列代码：

```
procedure TForm1.ListBox1Click (Sender : TObject);
var
   idx : Integer; { Selected index in list }
   entry : ^ ContactEntry; { Entry from data }
begin
   idx : = ListBox1.ItemIndex;

   entry : = Pointer (SendMessage ( ListBox1.handle, LB_GETITEMDATA, idx, 0 ));
   Edit1.Text : = entry^ .FirstName;
   Edit2.Text : = entry^ .LastName;
   Edit3.Text : = entry^ .Address1;
   Edit4.Text : = entry^ .Address2;
   Edit5.Text : = entry^ .City;
   Edit6.Text : = entry^ .State;
   Edit7.Text : = entry^ .ZipCode;
   Edit8.Text : = entry^ .Phone;

end;
```

6. 双击按钮 OK，在方法 TForm1.Button2Click 中输入下列代码：

```
procedure  TForm1.Button2Click (Sender: TObject);
begin
   Close;
end;
```

7. 双击按钮 Cancel，在方法 TForm1.Button3Click 中输入下列代码：

```
procedure  TForm1.Button3Click (Sender: TObject);
begin
   Close;
end;
```

8. 最后，在表单的类型部分 TForm1 定义的上面输入下列代码：

```
ContactEntry = Record
     FirstName : String;
     LastName : String;
```

```
    Address1  : String;
    Address2  : String;
    City      : String;
    State     : String;
    ZipCode   : String;
    Phone     : String;
end;
```

9. 编译并运行此例子程序。

7.2 捕捉列表框中的双击

问题

有时用户在列表框中双击，希望应用程序能够捕捉到此双击事件，从而可以简单的完成许多功能。例如：如果要关闭某个对话框，此对话框只包含一个列表框和一个按钮OK，用户可以通过双击列表框中的某个列表项来完成任务。当然，有时用户更希望通过双击来选中某个列表项，而不是关闭对话框。

方法

似乎这是一个很合理的要求，非常的合理以至使人感到相当的惊讶：Windows 系统的设计者没有为程序员提供此功能。但是因为可以扩充 Windows 中已存在的控制，所以对 Windows 程序员来说几乎没有缺陷存在。利用子类的概念，可以对标准的控制进行扩充，以实现操作系统开发者所没有考虑到的功能。本节将仔细讨论此方法。

当用户在窗口中双击鼠标时，特定的 Windows 消息（WM_LBUTTONDBLCLK）将发送给窗口。在本节中，将利用 Visual C++创建自己的控制处理类，并介绍如何利用此类来处理消息 WM_LBUTTONDBLCLK 以及如何在自己的应用程序中使用此类。

步骤

. 按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Dialog，接着从下拉菜单中选择菜单项 Double Click Dialog，将弹出如图 7-3 所示的对话框，在某列表项上双击鼠标，对话框将显示出一个消息框，标示选中的列表项，关闭消息框则对话框也将关闭。

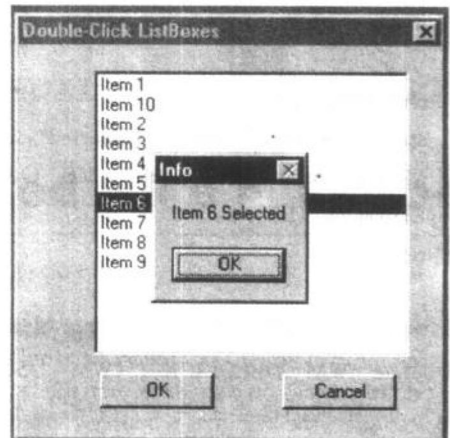


图 7-3 显示的双击对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH72.MAK。进入 AppStudio 创建新的对话框，在对话框中添加列表框。

2. 启动 ClassWizard，为刚创建的对话框模板生成对话框类，命名为 CDblClickDlg。从对象列表中选择对象 CDblClickDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function，在 CDblClickDlg 的方法 OnInitDialog 中输入下列代码：

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

  m_list.SubclassDlgItem (IDC_LIST1, this );

  // Add items to the list

  m_list.AddString ( " Item 1" );
  m_list.AddString ( " Item 2" );
  m_list.AddString ( " Item 3" );
  m_list.AddString ( " Item 4" );
  m_list.AddString ( " Item 5" );
  m_list.AddString ( " Item 6" );
  m_list.AddString ( " Item 7" );
  m_list.AddString ( " Item 8" );
  m_list.AddString ( " Item 9" );
  m_list.AddString ( " Item 10" );

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

3. 接着，在 CDblClickDlg 的消息映射中添加下列行：

```
ON_MESSAGE   ( WM_DBLCLICK, OnListDblClick )
```

4. 在类 CDblClickDlg 中添加下列方法：

```
LRESULT CDblClickDlg:: OnListDblClick (WPARAM wParam, LPARAM lParam)
{
   int idx = wParam;

   char buffer [80];
   sprintf (buffer, " Item %d Selected", idx );
   MessageBox (buffer, " Info", MB_OK );
   EndDialog (IDOK);
   return 0;
}
```

5. 在对话框类的源文件的顶部添加下列行：

```
#define  WM_DBLCLICK   (WM_USER+1)
```

6. 在对话框类的头文件中添加下列行：

```
private:
   CDblClkList   m_list;
```

7. 在对话框类的头文件中，消息映射项中添加下列行（用黑体标出）：

```
// Generated message map functions
  //{{AFX_MSG (CDblClickDlg)
```

```
virtual BOOL OnInitDialog ();
afx_msg LRESULT OnListDblClick (WPARAM, LPARAM);
//}} AFX_MSG
DECLARE_MESSAGE_MAP ()
```

8. 现在创建新的列表框类。不幸的是，在 Visual C－＋中，除非基类是 Visual C＋＋初始提供的那些类，否则便没有简单的方法来创建派生类。这里将介绍如何创建列表框派生类。首先，进入 ClassWizard 选择按钮 Add Class，类名为 CDblClkList，基类为 generic CWnd，其他的取缺省值。

9. 从项目文件列表中选取文件 DBLCLKLI.H（可能需要遍历 dependencies 来获取此文件），将所有的字符串 CWnd 改变为 CListBox，进入源文件 DBLCLKLI.CPP 做同样的操作。此时，创建的类将被 ClassWizard 认为有效的 CListBox 类，而不是 CWnd 类。

10. 进入 ClassWizard，从下拉列表中选择类 CDblClkList，从对象列表中选择对象 CDblClkList，从消息列表中选择消息 WM_LBUTTONDBLCLK，点击按钮 Add Function，在 CDblClkList 的方法 OnLButtonDblClk 中添加下列代码：

```
void CDblClkList:: OnLButtonDblClk (UINT nFlags, CPoint point)
{
  // Get the size of a line in the list box.

  int height = GetItemHeight (0);

  // Figure out if this point is on a valid item.

  int num_items = GetCount () - GetTopIndex ();
  // Number of items displayed.
  if ( point.y < 0 || point.y > height * num_items )
     return;

  CListBox:: OnLButtonDown (nFlags, point);

  int idx = GetCurSel ();

  GetParent () ->PostMessage (WM_USER+1, idx, 0 );
}
```

11. 进入 AppStudio，选择主菜单，添加主菜单 Dialog（如果不存在），在主菜单 Dialog 中添加菜单项 Double Click ListBox，ID 为 ID_DOUBLE_CLICK。

12. 进入 ClassWizard，从下拉列表中选择对象 CCh72App，接着选择对象 ID_DOUBLE_CLICK 和消息 COMMAND，点击按钮 Add Funciton，新函数命名为 OnDoubleClick。在 CCh72App 的方法 OnDoubleClick 中添加下列代码：

```
void CCh72App:: OnDoubleClick ()
{
  CDblClickDlg dlg;
```

```
dlg.DoModal ();
}
```

13. 在文件 CH7.CPP 顶部的 include 文件列表中添加下列行：

```
#include "dblclick.h"
```

14. 编译并运行此例子程序。

用法

如果用户从主菜单 Dialog 中选择了菜单项 Double Click ListBox，则在应用程序对象 CCh72App 的方法 OnDoubleClick 中创建对象 CDblClickDlg。首先调用 CDblClickDlg 的方法 OnInitDialog 来响应 Windows 发送给窗口的消息 WM_INITDIALOG，在方法 OnInitDialog 中，调用列表框方法 SubclassDlgItem，使得列表框处理程序调用程序员编写的窗口方法，而不是调用 Windows 系统中缺省的列表框方法。

在程序员编写的列表框方法中，除了双击消息外，其余的消息都是缺省操作。而双击消息由 CDblClkList 的方法 OnLButtonDblClick 来处理，此消息处理程序创建消息 WM_USER+1（指用户定义消息），接着将其传送给列表框的父窗口，即对话框，而对话框接收此消息是通过消息映射中定义的 ON_MESSAGE 处理程序 OnListDblList，此消息处理程序显示一个消息框，标示对话框已接收到消息并将选中的列表项显示出来，接着调用方法 EndDialog 关闭对话框。

注释

在本节中介绍了对象 CDblClkList 的实现，为程序员开发列表框处理程序提供了很好的起点，在本章的后面几节中使用了同样的方法来实现列表框中列表项的拖放，以及在列表框中自绘制列表项。

新列表框类可以用在对话框中向用户提供信息，另外也可以用来处理消息，以便向其他控制提供信息，例如：使用列表框存放字体名并传送这些字体名给静态文本控制以显示出来。也可以在列表框中存放目录信息并用来改变另一列表框中的文件列表。这完全由程序员来决定。

7.3 通过拖放在列表框中移动列表项

问题

在有的应用程序中，用户不仅希望能够选择要显示的列表项，而且希望能够选择显示列表项的顺序，有的程序员希望通过实现列表项的拖放功能来同时满足用户的这两个要求。但是，无论是使用 MFC 还是使用 Windows 95 API，都不能实现列表框中的拖放功能。如何以最小的代价来完成这个任务呢？

方法

在列表框中实现拖放功能的思想一直都很受欢迎，但是，就象前面所说的那样，MFC 没有提供此功能给程序员，只有程序员自己利用 Windows 95 API 函数的强大功能来扩展 MFC。

在本节中，只讨论如何在单个列表框内实现列表项的拖放功能，要实现此功能，需要利用子类的概念，即对 Windows 控制的某些已存在的方法进行替代，创建列表框的子类，以来捕捉某些 Windows 消息，并处理这些消息，从而实现列表项的拖放。

此方法可以分为下列几步：首先，用户选择列表项（希望拖放到新位置的列表项），列表项索引以及列表项字符串将被应用程序存储起来。接着，用户移动列表项到一新位置，应用程序跟踪用户的移动，并在用户到达新位置时标示出来。最后，用户松开鼠标键，表示列表项到达正确的位置。于是应用程序开始列表项的真正移动，从列表中某位置到达另一位置。

在本节的例子程序中，将绘制虚线来标示列表项在列表中的移动，如果用户此时松开鼠标键，则虚线所在的位置将为列表项要插入的位置，在用户松开鼠标键确认移动之前，列表项将一直保留在原位置上。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Dialogs 以及菜单项 Drag List Dialog，将弹出一个对话框，如图 7-4 所示，选取一个列表项（比如 Item 1）并向下移动鼠标，直到虚线显示在列表项 Item 7 的下面，然后松开鼠标键，则对话框如图 7-5 所示。

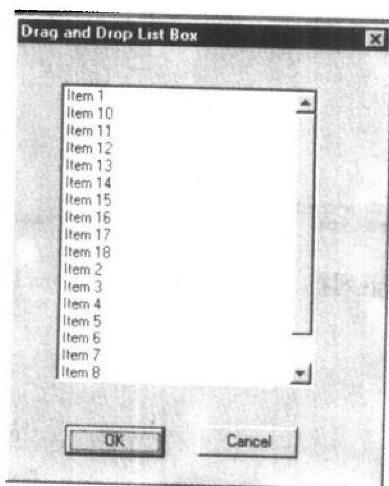


图 7-4 “拖放列表”对话框的初始状态

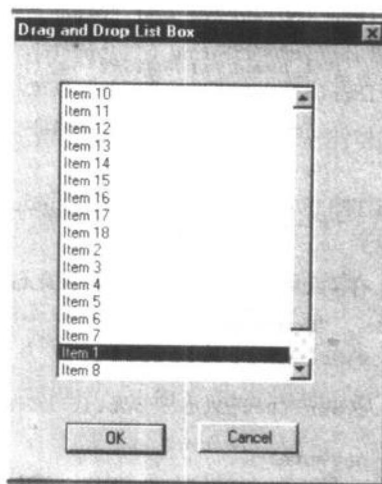


图 7-5 拖放操作后的“拖放列表”对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH73.MAK。进入 AppStudio，创建新的对话框，将按钮 OK 和 Cancel 移动到对话框的底部，并在对话框中添加一个列表框。

2. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，类名命名为 CDragDlg 并接受其他的缺省值。在 ClassWizard 中，从对象列表中选择对象 CDragDlg，从消息列表中选择消息 WM_INITDIALOG，在 CDragDlg 的方法 OnInitDialog 中输入下列代码：

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

    m_DragList.SubclassDlgItem (IDC_LIST1, this);
    m_DragList.AddString ( " Item 1" );
```

```
    m_DragList.AddString ("Item 2");
    m_DragList.AddString ("Item 3");
    m_DragList.AddString ("Item 4");
    m_DragList.AddString ("Item 5");
    m_DragList.AddString ("Item 6");
    m_DragList.AddString ("Item 7");
    m_DragList.AddString ("Item 8");
    m_DragList.AddString ("Item 9");
    m_DragList.AddString ("Item 10");
    m_DragList.AddString ("Item 11");
    m_DragList.AddString ("Item 12");
    m_DragList.AddString ("Item 13");
    m_DragList.AddString ("Item 14");
    m_DragList.AddString ("Item 15");
    m_DragList.AddString ("Item 16");
    m_DragList.AddString ("Item 17");
    m_DragList.AddString ("Item 18");

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

3. 在对话框头文件（DRAGDLG.H）的顶部添加下列行（黑体表示）：

```
#include   "dragdrop.h"

class  CDragDlg : public CDialog
{
private:
  CDragDropList  m_DragList;
```

4. 接着，返回ClassWizard。选择按钮Add Class，类名为CDragDropList，基类为generic CWnd，并接受其他的缺省值，点击按钮Create Class。

5. 在新类的源文件和头文件中，将所有的字符串“CWnd”改变成“CListBox”，重新进入ClassWizard，选择新类（CDragDropList），从对象列表中选择对象CDragDropList，从消息列表中选择消息WM_MOUSEMOVE，点击按钮Add Function，在CDragDropList的方法OnMouseMove中添加下列代码：

```
void CDragDropList::OnMouseMove (UINT nFlags, CPoint point)
{
    CRect r, wr;
    GetClientRect (&r);
    // How tall is one item?

    int height = GetItemHeight (0);

    // Figure out which one this is
```

```
    int idx = point.y / height;
    int num_items = (wr.bottom-wr.top) / height;

    CDC *dc = GetDC ();

    if ( is_tracking && idx < 0 || idx + GetTopIndex () > GetCount () ) {

       // Erase previous selection

     CPen pen (PS_DASH, 1, RGB (0, 0, 0));
     dc->SelectObject (&pen);
     dc->SetROP2 (R2_XORPEN);
     if ( last_idx ! = -1 ) {
         dc->MoveTo (0, ( (last_idx+1) * height));
         dc->LineTo (r.right, ( (last_idx+1) * height));
     }

     ReleaseDC (dc);

     return;
  }


   // If we are " dragging" don't do the normal move

  if ( ! is_tracking )
     CListBox:: OnMouseMove (nFlags, point);
  else {
     CPen pen (PS_DASH, 1, RGB (0, 0, 0));
     dc->SelectObject (&pen);
     dc->SetROP2 (R2_XORPEN);
     if ( last_idx ! = -1 ) {
         dc->MoveTo (0, ( (last_idx+1) * height));
         dc->LineTo (r.right, ( (last_idx+1) * height));
     }
     dc->MoveTo (0, ( (idx+1) * height));
     dc->LineTo (r.right, ( (idx+1) * height));
     last_idx = idx;
     }

   ReleaseDC (dc);
}
```

6. 从对象列表中选择对象 CDragDropList，从消息列表中选择消息 WM_LBUTTONDOWN，点击按钮 Add Function，在 CDragDropList 的方法 OnLButtonDown 中添加下列代码：

```
void CDragDropList:: OnLButtonDown (UINT nFlags, CPoint point)
{
  CListBox:: OnLButtonDown (nFlags, point);

  cur_sel = GetCurSel ();
  GetText (cur_sel, cur_text);
  is_tracking = TRUE;
  SetTimer ( ID_TIMER, 100, NULL );
  SetCapture ();

}
```

7. 从对象列表中选择对象 CDragDropList，从消息列表中选择消息 WM_LBUTTONUP，点击按钮 Add Function，在 CDragDropList 的方法 OnLButtonUp 中添加下列代码：

```
void CDragDropList:: OnLButtonUp (UINT nFlags, CPoint point)
{
    if ( is_tracking ) {

        // How tall is one item?

        int height = GetItemHeight (0);

        // Figure out which one this is

        int idx = GetTopIndex () + point.y / height;

        // Make sure this one is in list

        if ( idx < 0 || idx > GetCount () )
            return;

        // Delete current item

        DeleteString ( cur_sel );

        // Insert this one.

        InsertString ( idx, cur_text );

        // Clear the tracking flag
```

```
        is_tracking = FALSE;

        // And select this item...

        SetCurSel (idx);

        last_idx = -1;

        KillTimer ( ID_TIMER );

        ReleaseCapture ();
    }

  CListBox:: OnLButtonUp (nFlags, point);
}
```

8. 从对象列表中选择对象 CDragDropList，从消息列表中选择 WM_TIMER，点击按钮 Add Function，在 CDragDropList 的方法 OnTimer 中添加下列代码：

```
void CDragDropList:: OnTimer (UINT nIDEvent)
{
  POINT pt;
  GetCursorPos (&pt);

  CRect rect;
      GetClientRect ( &rect );
  ClientToScreen ( &rect );

  int TopIndex = GetTopIndex ();

  if ( pt.y < rect.top )
   {
    if ( TopIndex > 0 )
      SetTopIndex ( --TopIndex );
   }
  else if ( pt.y > rect.bottom )
   {
    if ( TopIndex < GetCount () )
      SetTopIndex ( ++TopIndex );
   }
}
```

9. 在对象 CDragDropList 的构造函数中添加下列行：

```
is_tracking = FALSE;
```

```
last _ idx = -1;
```

10. 接着选择类 CDragDropList 的头文件 DRAGDROP.H，在此文件中添加下列行：

```
private:
  int  cur _ sel;        // Currently selected item
  int  is _ tracking;    // flag for dragging and dropping
  int  last _ idx;
  CString  cur _ text;   // string for selected item.
```

11. 返回 AppStudio，选择主菜单。添加新的主菜单，标题为 Dialogs，在主菜单 Dialogs 中，添加新的菜单项，标题为 Drap List Dialog，标识符为 ID _ DRAG _ LIST _ DLG。

12. 在 ClassWizard 中，选择应用程序对象 CCh73App，从对象列表中选择对象 ID _ DRAG _ LIST _ DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function，方法名为 OnDragListDlg。在 CCh73App 的方法 OnDragListDlg 中添加下列代码：

```
void CCh73App:: OnDragListDlg ()
{
  CDragDlg dlg;
  dlg.DoModal ();
}
```

13. 在文件 CH7.CPP 顶部的 include 文件列表中添加下列行：

```
#include " dragdlg.h"
```

14. 编译并运行此例子程序。

用法

实现拖放功能的机理是相当直接的。首先应用程序等待，直到用户在某个列表项上点击鼠标左键；接着捕捉鼠标输入，以便获取鼠标的移动和点击，在鼠标移动时，应用程序跟踪鼠标的移动，并显示某种指示器或图标来标示鼠标正在拖拉；最后，用户松开鼠标键，则应用程序将列表项放置在所要求的位置上。

实现的具体细节更令人感兴趣。当用户在列表中的某项上点击鼠标左键时，则 CDragDropList 的方法 OnLButtonDown 被调用，此方法完成三件事情。首先，将用户选中的列表项的索引以及字符串存储起来；接着，在 Windows 系统中创建一个计时器对象，计时器对象只是用来以特定的时间间隔创建消息 WM _ TIMER，并将此消息发送给窗口，在本节的例子程序中，计时器的时间间隔设置为 100ms，当用户按着鼠标的左键在列表框的外面移动鼠标时，此计时器用来滚动列表；最后，调用 API 函数 SetCapture 来将所有的鼠标事件定向到窗口，也就是说，即使鼠标在列表框的外面移动，列表框也能接收到消息。

一旦用户按下鼠标左键并开始移动，则方法 OnMouseMove 被调用。此方法只是用来在鼠标当前所在的列表项的下面绘制虚线，从而提供给用户可视的反馈信息，标示拖放在实际进行，此方法进行相当多的检查，以来确认鼠标是否还在列表框内，如果在列表框的外面，则虚线的绘制将不再进行。

接着，每隔 100msWindows 就调用列表框类的方法 OnTimer 一次。此方法只是检查鼠标是否在列表框内，如果不是，则检查鼠标是否在列表框的上面，如果鼠标在列表框的上面，则列表自动向上滚动，如果鼠标在列表框的下面，则列表自动向下滚动。似乎列表框正在响应用户的请求而滚动。

最后，当用户松开鼠标键时，方法 OnLButtonUp 被调用。如果鼠标光标在列表框内，则只是指示了哪一个列表项当前在光标下，原来的列表项被删除，重新插入到了新的位置上。

注释

如果要实现在列表框内拖放列表项，则 CDragDropList 是一个很好的起点。但是此类显然是不完善的，还需要对其进行改进，例如：同时选择并拖放多个列表项。另外，可以改进函数 OnLButtonUp，以来确认是否选择了正确的列表框。一个比较简单的改进是：允许多个列表框“共享”列表项，即在多个列表框之间拖放列表项。

拖放操作并不是最简单最直接的一种用户交互手段，是否在应用程序中实现此功能应该经过反复的考虑再决定。由于许多 Windows 95 的新用户并不习惯于使用此方法，所以最好同时还提供另外一种手段来完成同样的操作。

7.4 滚动列表框

问题

有时，用户希望在列表框中实现“快速查找”的功能，使得只需在编辑框中键入几个字符（要查找的字符串的一部分），应用程序就响应这些击键操作，然后列表框自动定位要查找的字符串。在 Windows 95 的列表框中已经实现了此功能，只不过键入的是要查找字符串的第一个字符。如何实现键入多个字符而相应地滚动列表框呢？

方法

前面提到的“快速查找”方法在当前的应用程序中已经相当普遍，许多 Windows 95 通用控制都使用了此方法（打开文件对话框以及字体对话框），大量的应用程序也使用了此方法。要实现此方法，程序员便需要清楚 Windows 95 的消息和屏幕上的控制之间是如何协调工作的。有些消息是用来标示控制的改变，应用程序可以捕捉到，有些消息可以发送给 Windows 控制，使得应用程序可以改变控制的行为，就象用户在操作控制一样。

在本节中将介绍两类特别的控制消息：通知消息和操纵消息。当用户改变控制的状态时，控制发送通知消息给控制的父窗口，应用程序可以利用操纵消息来改变 Windows 控制的状态，就象用户在操作一样。

本节要特别介绍的两个消息是 EN_CHANGE 和 LB_SELECTSTRING。当用户改变编辑控制中的字符串时（例如：键入字符，删除字符，或粘贴操作），编辑控制发送消息 EN_CHANGE 给父窗口（对话框）；消息 LB_SELECTSTRING 是操纵消息，可以发送给列表框，用来查找列表框中相匹配的字符串。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Dialogs，从下拉菜单中选择菜单项 Search List，将弹出一个对话框，如图 7-6 所示。与某个列表项相匹配在编辑框中键入几个字符，则列表框中相应的列表项被加亮。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH74.MAK。进入 AppStudio 并创建新的对话框，将按钮 OK 和 Cancel 移动到对话框的底部，并在对话框中添加一个编辑框和一个列表框。

2. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类。新的对话框类命名为

CSearchDlg，并接受其他的缺省值。

3. 在 ClassWizard 中，从对象列表中选择对象 CSearchDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function，在 CSearchDlg 的方法 OnInitDialog 中添加下列代码：

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

  // Add some terms that we can search on

  CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);

  list->AddString (" Alphabet" );
  list->AddString (" BetaZoid" );
  list->AddString (" Carnival" );
  list->AddString (" Caramel" );
  list->AddString (" Country" );
  list->AddString (" Diamond" );
  list->AddString (" Elephant" );
  list->AddString (" Eeyore" );
  list->AddString (" Eyesight" );
  list->AddString (" Animal" );
  list->AddString (" Copper" );
  list->AddString (" Gold" );
  list->AddString (" Farley" );
  list->AddString (" Borrow" );
  list->AddString (" Candy" );
  list->AddString (" Ginger" );
  list->AddString (" Flint" );
  list->AddString (" New York" );
  list->AddString (" Idaho" );
  list->AddString (" Irving" );
  list->AddString (" Jenny" );
  list->AddString (" Rachel" );
  list->AddString (" Dawnna" );
  list->AddString (" Matt" );

  CenterWindow ();

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

图 7-6　显示的查找列表对话框

4. 从对象列表中选择对象 IDC _ EDIT1，从消息列表中选择通知消息 EN _ CHANGE，点击按钮 Add Function，新方法命名为 OnEditChange。在 CSearchDlg 的方法 OnEditChange 中输入下列代码：

```
void CSearchDlg:: OnEditChange ()
{
  // Get the string from the edit box

  char buffer [256];
  GetDlgItem (IDC_EDIT1) ->GetWindowText (buffer, 256);

  // Search for the string in the list box

  CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
  list->SendMessage ( LB_SELECTSTRING, -1,        (LPARAM) (LPCSTR) buffer );
}
```

5. 进入 AppStudio，从菜单列表中选择主菜单，如果不存在则添加新的主菜单，标题为 Dialogs。在主菜单 Dialogs 中添加新的菜单项，标题为 Search List，标识符为 ID _ SEARCH _ LIST。

6. 进入 ClassWizard，从下拉列表中选择应用程序对象 CCh74App，从对象列表中选择对象 ID _ SEARCH _ LIST，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新函数命名为 OnSearchList。在 CCh74App 的方法 OnSearchList 中输入下列代码：

```
void CCh74App:: OnSearchList ()
{
  CSearchDlg dlg;
  dlg.DoModal ();
}
```

7. 在文件 CH74.CPP 顶部的 include 文件列表中添加下列行：

```
#include   "searchdl.h"
```

8. 编译并运行此例子程序。

用法

当对话框弹出时，编辑框最初是空的，当用户在编辑框中键入时，每次击键 Windows 都将发送消息 EN _ CHANGE 给对话框类。对话框类则调用 API 函数 GetWindowText 来从编辑框中取回当前的文本字符串，接着将此字符串与消息 LB _ SELECTSTRING 一起发送给列表框。

消息 LB _ SELECTSTRING 从列表中查找与字符串相匹配的列表项。函数 SendMessage 的第二个参数传送的是列表项的索引，表示查找从此列表项开始，如果参数值为－1，则从第一个列表项开始查找整个列表；否则，就从参数值所指的列表项开始。接着，将传送给消息的字符串与每个列表项相比较，如果匹配，则列表重新定位，按需要滚动列表。

注释

本节介绍的方法并不仅仅限于在 Visual C＋＋中实现，实际上，此方法非常有用，所以本节还将介绍另外一种实现途径，即在 Visual Basic 中实现。用来在列表中查找字符串的 Visual

Basic 表单如图 7-7 所示。

按照下列步骤实现此例子程序：

1. 在 Visual Basic 中，打开项目文件或创建新的项目文件，并在此项目文件中添加新的表单。

2. 在表单中，添加静态文本控制，标题为 Search String:，邻接此静态文本控制，添加一个编辑域，最后，在表单中添加一个列表框。

3. 在表单的“general”部分添加下列代码：

```
DefInt A-Z
Private Declare Function SendMessage Lib " user32" Alias  " SendMessageA"
(ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal Lp As Any) As Long
Const WM_USER = &H400
Const LB_SELECTSTRING = &H18C
Const LB_SETTOPINDEX = (WM_USER + 24)
```

4. 接着，双击编辑框，在编辑框的方法“Change”中添加下列代码：

```
Private Sub Text1_Change ()
    S$ = Text1.Text
    Index = SendMessage(List1.hwnd, LB_SELECT-
STRING, -1, S$)
     Err = SendMessage (List1.hwnd, LB_SET-
TOPINDEX, Index, 0&)
End Sub
```

5. 最后，添加命令按钮，标题为 &Close，在按钮 Close 的方法“Click”中添加下列代码：

```
Private Sub Command1_Click ()
    End
End Sub
```

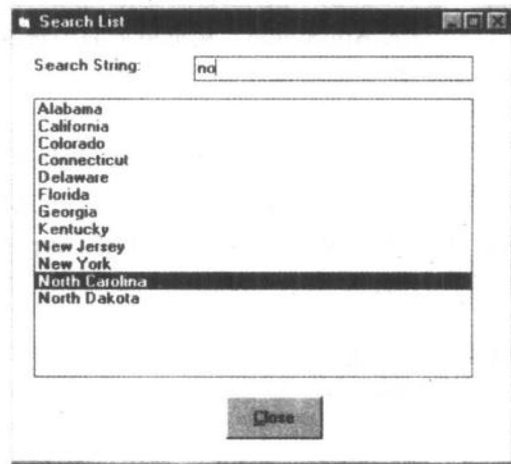


图 7-7 在 Visual Basic 中实现的查找列表项例子程序

7.5 实现宽列表的水平滚动

问题

在有的应用程序中，使用好几个列表框来显示用户信息。很多时候，需要显示的用户信息都比列表框要宽得多，当然，程序员可以扩宽列表框，但是有可能不适合在某些旧型号的显示器上显示，所以此方法有时是不可取的。

是否有方法利用 Windows API 来实现列表框的水平滚动呢？有的程序员可能试着在 AppStudio 中为对话框模板打开水平滚动标志，但是发现在应用程序运行时没有起任何作用。如何才能使列表框滚动，用来显示列表中最宽的字符串呢？

方法

Microsoft 的一个重大失误是：没有实现列表框的自动滚动，用来响应水平滚动条上的点击操作，此功能在许多应用程序中都非常有用。许多程序员都知道，只是在列表框中添加水平滚动条是不能使列表框水平滚动起来的。

实现列表框水平滚动的关键在于 Windows API 函数 SendMessage 的使用。此列表框 API 包含着一个操纵消息 LB _ SETHORIZONTALEXTENT，此消息用来设置水平滚动条的水平范围，使得列表框在用户点击水平滚动条时滚动文本字符串。

在本节中，将介绍如何找出列表框中最长的字符串，以及如何设置列表框的水平滚动因子，以便能够正确的显示出最长的字符串。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Dialogs，从下拉菜单中选择菜单项 Horizontal Scroll List，将弹出一个对话框，如图 7-8 所示。应该注意，此时在列表框中没有水平滚动条，点击按钮 Set Horizontal Scrolling On，则水平滚动条出现在列表框的底部，用户可以使用此滚动条来滚动列表，并查看列表中的长字符串。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH75. MAK。进入 AppStudio 创建新的对话框，将按钮 OK 和 Cancel 移动到对话框的底部，并在对话框中添加一个单选列表框，双击列表框，检取水平滚动核选框，在列表框的上面添加一个按钮，标题为 Set Horizontal Scrolling On。

2. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，新对话框类命名为 CHSCrollList，并接受其他的缺省值。

3. 在 ClassWizard 中，从下拉列表中选择类 CHScrollList，从对象列表中选择对象 CHScrollList，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Function，在 CHScrollList 的方法 OnInitDialog 中输入下列代码：

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

  CListBox *list = (CListBox *) GetDlgItem (IDC _ LIST1);
  list->AddString ( "This is a really really really really really really
long string" );
  list->AddString ( " This is a shorter string" );
  list->AddString ( " short string" );
  list->AddString ( " Another longer string" );
  list->AddString ( " The final String - A Very long String" );

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

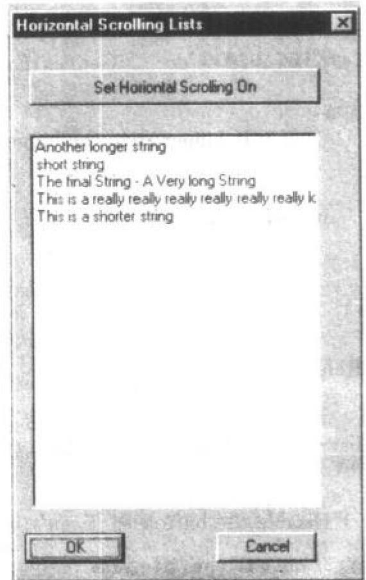


图 7-8　未显示水平滚动条的水平滚动列表对话框

4. 返回 ClassWizard，从对象列表中选择对象 IDC _ BUTTON1，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新方法命名为 OnDoHScroll。在 CHScrollList 的方法 OnDoHScroll 中添加下列代码：

```
void CHScrollList:: OnDoHScroll ()
{
  CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);

  // Get the length of the longest string.

  CDC *dc = list->GetDC ();
  char string [256];
  int  max_len = 0;

  // Loop through all strings in the list box

  for ( int i=0; i<list->GetCount (); ++i ) {

     // Get the text associated with this item

     list->GetText ( i, string );

     // Get the length in pixels of this string.

     CSize size = dc->GetTextExtent (string, strlen (string));

     // If it is bigger than the biggest, reset the biggest

     if ( size.cx > max_len )
         max_len = size.cx;
  }

  ReleaseDC (dc);

  list->SendMessage (LB_SETHORIZONTALEXTENT, max_len, 0 );

  // Disable the button so it can't be pressed again

  GetDlgItem (IDC_BUTTON1) ->EnableWindow (FALSE);
}
```

5. 进入 AppStudio，从菜单列表中选择主菜单，添加新的主菜单，标题为 Dialogs。在菜单 Dialogs 中添加新的菜单项，标题为 Horizontal Scroll List，标识符为 ID_HSCROLL_LIST。

6. 重新进入 ClassWizard，选择应用程序对象 CCh75App，从对象列表中选择对象 ID_HSCROLL_LIST，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新方法命名为 OnHscrollList。在 CCh75App 的方法 OnHscrollList 中输入下列代码：

```
void CCh75App:: OnHscrollList ()
{
  CHScrollList dlg;
  dlg.DoModal ();
}
```

7. 在文件 CH75.CPP 顶部的 include 文件列表中添加下列行:

```
#include  "hscrolll.h"
```

8. 编译并运行此例子程序。

用法

整个例子程序的关键是点击对话框中的按钮，如果点击了按钮，则调用对话框类的方法 OnDoHScroll。此方法首先获取列表框的设备描述表（DC），此设备描述表使用的文本特性与列表框的一致，接着，此方法获取列表框中的每个字符串，并调用 API 函数 GetTextExtent 来获取字符串的长度（以象素为单位），此长度与当前最长的字符串相比较，如果超过最长的字符串，则字符串长度的最大值被重置。

一旦循环结束，此方法接着调用函数 SendMessage 来发送消息 LB_SETHORIZONTALEXTENT 给列表框，并以字符串的最大长度为参数，最后，调用 API 函数 EnableWindow 来禁止按钮，使得按钮不再可用。

函数 GetTextExtent 在内存中"绘制"字符串，使用设备描述表的当前设置，返回显示字符串所需要的区域大小，绘制字符串之前确定字符串的大小，此函数是一个非常简便的方法。在有的应用中（例如：字环绕），此函数可以用来确定在给定的区域内能否放得下给定的字符串。

注释

为应用程序创建新的构件，本节的例子程序是一个很好的起点。例如：将确定字符串长度的代码提取出来，并以此为基础创建一个新的列表框类，此类可以根据添加的字符串自动地设置列表框的水平范围。

你可以尝试创建一个新列表类，并重置方法 AddString，在此方法中获取字符串的长度。一旦获取到字符串的长度，则保存在内部的成员变量中，并每次都使用此变量来设置水平文本范围，使得程序员可以不考虑水平滚动，但水平滚动在需要的时候都是有效的。

7.6 在列表框中右对齐数字

问题

有时，用户希望显示在列表框的数字是右对齐的。例如：将帐目显示在屏幕上，而会计查看钱数一般都习惯于从右边开始，所以就需要右对齐列表框中的数字，但是，似乎又没有简单的方法来实现此功能。显示静态文本是可以右对齐、左对齐、或者置中的，为什么列表框不能呢？如何解决此问题，使得会计不再烦恼呢？

方法

确实没有直接的方法来实现列表框中文本的右对齐。有些程序员可能尝试了许多方法：在风格标志中设置 DT_RIGHT 位，使用 sprintf 右对齐输出，以及其他的方法，但发现这些方法都是行不通的。

不过，有一种方法可以在列表框中实现文本右对齐，这是本书中第一个自绘制列表框的例子。自绘制列表框需要程序员编写显示函数，用来将文本显示在屏幕上，自绘制控制充分利用了 Windows API 的强大功能，为 Windows 应用程序增添了非凡的效果。

在本节中，只是简单地涉及到自绘制的概念。在本章的下一节中，将详细介绍此概念的使用，包括如何将图形和位图放入列表框中。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Dialogs，选择菜单项 Justify List，将弹出一个对话框，如图 7-9 所示，点击三个对齐按钮，可以观察列表框中文本位置的改变，点击按钮 OK 或 Cancel 可以关闭对话框。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH76.MAK。进入 AppStudio，创建新的对话框，将按钮 OK 和 Cancel 移动到对话框的底部，并在对话框的顶部添加三个按钮，在按钮的下面添加一个列表框。

2. 双击列表框，在弹出的对话框中将组合框 Owner Draw 设置为 Variable。

3. 选择 ClassWizard，为刚创建的对话框模板生成对话框类，新对话框类命名为 CJustifyList，并接受其他的缺省值。

4. 在 ClassWizard 中，选择 CJustifyList，从对象列表中选择对象 CJustifyList，从消息列表中选择消息 WM_INITDIALOG，最后，点击按钮 Add Function，在 CJustifyList 的方法 OnInitDialog 中输入下列代码：

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

  m_list.SubclassDlgItem (IDC_LIST1, this );

  // Add items to the list

  m_list.AddString ( " Item 1" );
  m_list.AddString ( " Item 2" );
  m_list.AddString ( " Item 3" );
  m_list.AddString ( " Item 4" );
  m_list.AddString ( " Item 5" );
  m_list.AddString ( " Item 6" );
  m_list.AddString ( " Item 7" );
  m_list.AddString ( " Item 8" );
  m_list.AddString ( " Item 9" );
  m_list.AddString ( " Item 10" );

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

5. 接着，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 COM-

MAND，点击按钮 Add Function，新方法命名为 OnRightJustify。在 CJustifyList 的方法 OnRightJustify 中添加下列代码：

```
void CJustifyList:: OnRightJustify ()
{
  m_list.SetJustification ( RightJustifyList );
}
```

6. 从对象列表中选择对象 IDC_BUTTON2，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新方法命名为 OnLeftJustify。在 CJustifyList 的方法 OnLeftJustify 中添加下列代码：

```
void CJustifyList:: OnLeftJustify ()
{
  m_list.SetJustification ( LeftJustifyList );
}
```

7. 从对象列表中选择对象 IDC_BUTTON3，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新方法命名为 OnCenterJustify。在 CJustifyList 的方法 OnCenterJustify 中添加下列代码：

```
void CJustifyList:: OnCenterJustify ()
{
  m_list.SetJustification ( CenterJustifyList );
}
```

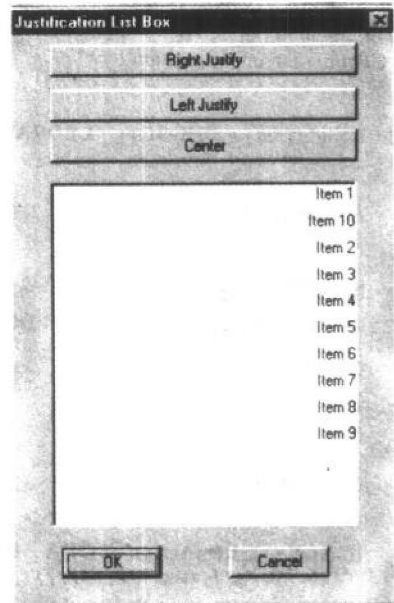


图 7-9 在列表框中右对齐

8. 再选择 ClassWizard，点击按钮 Add Class，新类命名为 CRightList，选择 generic CWnd 为基类名，并接受其他的缺省值。

9. 进入文件 RIGHTLIS.CPP，将所有的字符串“CWnd”改为字符串“CListBox”，接着选择头文件 RIGHTLIS.H，并做同样的改变，将两文件存盘。

10. 在文件 RIGHTLIS.CPP 中添加下列代码（黑体表示）：

```
CRightList:: CRightList ()
{
  mode = RightJustifyList;
}

CRightList:: ~CRightList ()
{
}

void CRightList:: SetJustification (int just)
{
  mode = just;
  InvalidateRect (NULL);
```

```
}

BEGIN_MESSAGE_MAP (CRightList, CListBox)
      // { {AFX_MSG_MAP (CRightList)
      //}} AFX_MSG_MAP
END_MESSAGE_MAP ()

/////////////////////////////////////////////////////////////////////////////
// CRightList message handlers

void CRightList::DrawItem (LPDRAWITEMSTRUCT lpdi)
{
    // Losing focus?

    if (lpdi->itemID == -1) {
        DrawFocusRect (lpdi->hDC, &lpdi->rcItem);
    return;
    }

    // Draw the while item

    if (lpdi->itemAction & ODA_DRAWENTIRE) {

        CDC dc;
        dc.m_hDC = lpdi->hDC;
        char * text = (char *) lpdi->itemData;

        int justification = DT_RIGHT;
        switch ( mode ) {
          case RightJustifyList:
              justification = DT_RIGHT;
              break;
          case LeftJustifyList:
              justification = DT_LEFT;
              break;
          case CenterJustifyList:
              justification = DT_CENTER;
              break;

        }

        dc.DrawText (text, strlen (text), & (lpdi->rcItem), justification);
```

```
        if (lpdi->itemState & ODS_SELECTED)
            InvertRect (lpdi->hDC, &lpdi->rcItem);

        if (lpdi->itemState & ODS_FOCUS)
            DrawFocusRect (lpdi->hDC, &lpdi->rcItem);

        dc.m_hDC = NULL;
    return;
    }

    // Selection made?

    if (lpdi->itemAction & ODA_SELECT) {
        InvertRect (lpdi->hDC, &lpdi->rcItem);
    return;
  }

    // Getting focus?

    if (lpdi->itemAction & ODA_FOCUS) {
        DrawFocusRect (lpdi->hDC, &lpdi->rcItem);
        return;
    }
}

int CRightList:: CompareItem (LPCOMPAREITEMSTRUCT lpCompareItemStruct)
{
   char *text1 = (char *) lpCompareItemStruct->itemData1;
   char *text2 = (char *) lpCompareItemStruct->itemData2;
   return strcmp (text1, text2);
}

void CRightList:: MeasureItem (LPMEASUREITEMSTRUCT lpMeasureItemStruct)
{
   // Get the default list height

   lpMeasureItemStruct->itemHeight = 20;
}
```

11. 在头文件 RIGHTLIS.H 中添加下列代码（同样用黑体表示）：

```
const RightJustifyList =0;
const LeftJustifyList  = 1;
const CenterJustifyList= 2;
```

```
class CRightList : public CListBox
{
private:
   int mode;

// Construction
public:
       CRightList ();

// Attributes
public:

// Operations
public:

// Implementation
public:
     virtual ~CRightList ();
     void  SetJustification (int just);

protected:
   virtual void MeasureItem (LPMEASUREITEMSTRUCT lpMIS);
   virtual void DrawItem (LPDRAWITEMSTRUCT lpDIS);
   virtual int CompareItem (LPCOMPAREITEMSTRUCT lpCIS);

      // Generated message map functions
      //{{AFX_MSG (CRightList)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP ()
};
```

12. 选择头文件 JUSTIFYL.H，在此文件中添加下列行（黑体表示）：

```
#include "rightlis.h"

class CJustifyList : public CDialog
{
  CRightList m_list;
```

13. 接着，重新进入 AppStudio 并选择主菜单，添加一个主菜单，标题为 Dialogs。在菜单 Dialogs 中，添加新的菜单项，标题为 Justify List，标识符为 ID_JUSTIFY_LIST。

14. 在 ClassWizard 中，选择应用程序对象 CCh76App，从对象列表中选择对象 ID_JUSTIFY_LIST，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新方法命名为 OnJustifyList。在 CCh76App 的方法 OnJustifyList 中输入下列代码：

```
void CCh76App:: OnJustifyList ()
```

```
{
  CJustifyList dlg;
  dlg.DoModal ();
}
```

15. 在文件 CH76.CPP 顶部的 include 列表中添加下列 include 文件行：

```
#include  " justifyl.h"
```

16. 编译并运行此例子程序。

用法

当最初显示对话框时，在对话框类的方法 OnInitDialog 中，将新列表框类 CRightList 作为列表框类的子类，类 CRightList 是自绘制列表框类，用来以适当的方式显示文本。

创建自绘制列表框类（或任何其他的自绘制控制）至少需要三个函数。首先，需要一个函数来确定列表项的高度。在 CRightList 中此函数为 MeasureItem，此方法仅将列表中所有的列表项都设置为同样的大小（在这里是 20 象素），当列表框需要绘制每个列表项时，将调用此函数来确定包围盒的大小，用来显示此列表项。

接着，列表框管理程序需要一个比较函数。此方法用来在有序列表中确定列表项的位置，在本节的例子程序中，由于列表项都是些简单文本，因此可以返回函数 strcmp 的返回值。函数 strcmp 比较两个字符串，返回的值与比较函数兼容，在本节的例子程序中，比较函数是 CompareItem。

最后，列表框的主要功能在方法 DrawItem 中实现。此方法负责将每个列表项绘制到屏幕上（以列表框中列表项的包围盒为界线），并且调用时只有一个参数，类型为 LPDRAWITEMSTRUCT。尽管此结构中有相当多的域，但最重要的几个域是：焦点设置，列表项数据，以及列表项的包围矩形。焦点设置存放在域 itemAction 中，可取下列任一值：ODA _DRAWENTIRE，ODA _SELECTED，ODA _SELECT，以及 ODA _FOCUS，这些值用来确定是否绘制此列表项，以及是否在此列表项周围绘制焦点矩形。

在方法 DrawItem 中，最重要的是处理设置 ODA _DRAWENTIRE（参数 itemAction 的设置为 ODA _DRAWENTIRE）。在此设置时，列表框调用此函数来向列表框中绘制列表项，也就是在此函数中实现列表项的对齐，当用户点击对话框中的某个按钮用来选择某个对齐设置时，则列表框的设置被修改为某个适当的值，此值（DT _RIGHT，DT _LEFT，或者 DT _CENTER）接着被传送给 API 函数 DrawText，此函数将字符串放置在适当的位置上。API 函数 DrawText 还有一个参数为包围矩形，此包围矩形曾以矩形的形式传送给方法 DrawItem。

最后一个疑问点可能是 SetJustification 中的调用 InValidateRect，此函数用来使窗口“无效”，即 Windows 重新绘制窗口。在 Windows 重新绘制窗口时，调用程序员自己编写的函数 DrawItem 用来绘制每个列表项，从而能够以新的对齐方式来重新显示文本字符串。

注释

在所有版本的 Windows 系统中，程序员都可以利用自绘制概念来实现应用程序的强大功能。但是，程序员在实现自绘制控制时都必须牢记一点：自绘制控制不是由应用程序来管理的，应用程序必须按照 Windows 系统的要求来响应消息，没有另外的消息响应方式。

以本节的例子程序为基础，可以非常容易地实现其他控制类。在本节中实现的列表框可

以用来显示钱数（右对齐），显示字符串（左对齐），或者显示其他要求置中的数据，例如：文档的页眉和页脚。

利用自绘制的强大功能，程序员还可以在列表框中绘制颜色、图形、图标、或者位图，利用可见的图形来使列表充满生机！在下一节中，将仔细介绍如何将图形显示在列表中。

7.7 实现自绘制列表框

问题

有时，用户希望能够在列表框中显示出一些具有特别属性的列表项，而这些列表项在“普通”的列表框中是不允许的。例如：在显示颜色名的同时还显示颜色框，并且以选中的颜色来显示颜色名，会使得应用程序界面更加友好。在 Windows 和 Windows 95 中，系统提供的列表框都不具有此功能。

如何“绘制”自己的列表框来实现此功能，而且不需要重新实现普通列表框所提供的功能呢？程序员不希望对排序例程、键盘输入例程、拖放例程等进行重新编写。

方法

就像上一节所讨论的，Windows 程序员可以利用自绘制列表框来实现非常强大的功能，而实现自绘制列表框的方法却是相当简单的。首先，需要应用程序能够确定列表中列表项的高度；接着，需要应用程序能够比较列表中的两个列表项；最后，需要应用程序能够将列表项绘制到列表框中。

在本节中，将介绍如何利用子类和自绘制的概念来创建一个包含文本、颜色和图形的列表框。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择主菜单 Dialogs，从下拉菜单中选择菜单项 Owner Draw Dialog，将弹出一个对话框，如图 7-10 所示，选择列表框中的任一彩色列表项，查看选中区域的移动。如果需要关闭对话框，可以点击按钮 OK 和 Cancel。

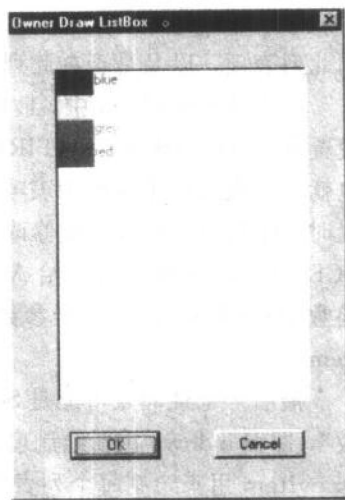


图 7-10 自绘制列表框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH77.MAK。进入 AppStudio，创建新的对话框，将按钮 OK 和 Cancel 移动到对话框的底部，并在对话框中添加一个列表框。

2. 双击列表框，在弹出的对话框中将组合框 Owner Draw 设置为 Variable。

3. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，新对话框类命名为 COwnerDrawDlg，并接受其他的缺省值。

4. 在 ClassWizard 中，选择 COwnerDrawDlg，从对象列表中选择对象 CJustifyList，从消息列表中选择消息 WM _ INITDIALOG，最后，点击按钮 Add Function，在 COwner-DragDlg 的方法 OnInitDialog 中输入下列代码：

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

  m_list.SubclassDlgItem (IDC_LIST1, this );

  // Add a nuch of items to the list

  m_list.AddString ( RGB (255, 0, 0), " red" );
  m_list.AddString ( RGB (0, 255, 0), " green" );
  m_list.AddString ( RGB (0, 0, 255), " blue" );
  m_list.AddString ( RGB (128, 128, 128), " grey" );

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

5. 在类 COwnerDrawDlg 的头文件（OWNERDRA.H）中添加下列行（黑体表示）：

```
#include " ownlis.h"

class COwnerDrawDlg : public CDialog
{
private:
   COwnerDrawList m_list;
```

6. 在 Visual C++中创建新的文件，并在此文件中输入下列代码：

```
// ownlis.cpp : implementation file
//

#include " stdafx.h"
#include " ch77.h"
#include " ownlis.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE [] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COwnerDrawList

COwnerDrawList:: COwnerDrawList ()
{
}

COwnerDrawList:: ~COwnerDrawList ()
```

```
{
}

COwnerDrawList:: AddString ( COLORREF color, char *string )
{
   ColorStruct *s = new ColorStruct;
   s->string = new char [strlen (string) +1];
   strcpy ( s->string, string );
   s->color = color;

   // Add the string to the list box

   int idx = CListBox:: AddString ( (char * ) s );

   // Save the pointer as item data so we can delete it later

   SetItemData ( idx, (DWORD) s );

   return idx;
}

BEGIN_MESSAGE_MAP (COwnerDrawList, CListBox)
  //{{AFX_MSG_MAP (COwnerDrawList)
  //}} AFX_MSG_MAP
END_MESSAGE_MAP ()

/////////////////////////////////////////////////////////////////////////////
// COwnerDrawList message handlers

void COwnerDrawList:: DrawItem (LPDRAWITEMSTRUCT lpdi)
{
    ColorStruct *s = (ColorStruct * ) lpdi->itemData;

    // Losing focus?

    if (lpdi->itemID == -1) {
         DrawFocusRect (lpdi->hDC, &lpdi->rcItem);
    return;
    }

    // Draw the while item

    if (lpdi->itemAction & ODA_DRAWENTIRE) {
```

```
        CDC dc;
        dc.m_hDC = lpdi->hDC;
        char *text = s->string;

        // First, draw the color

        COLORREF color = s->color;

        RECT r;
        r.top = lpdi->rcItem.top;
        r.bottom = lpdi->rcItem.bottom;
        r.left = lpdi->rcItem.left;
        r.right = r.left + 30;

        CBrush brush ( color );
        dc.FillRect ( &r, &brush );

        r = lpdi->rcItem;
        r.left += 30;

        dc.SetTextColor ( color );
        dc.DrawText (text, strlen (text), &r, DT_LEFT);

        if (lpdi->itemState & ODS_SELECTED)
            InvertRect (lpdi->hDC, &lpdi->rcItem);

        if (lpdi->itemState & ODS_FOCUS)
            DrawFocusRect (lpdi->hDC, &lpdi->rcItem);

        dc.m_hDC = NULL;
    return;
    }

    // Selection made?

    if (lpdi->itemAction & ODA_SELECT) {
        InvertRect (lpdi->hDC, &lpdi->rcItem);
    return;
}

    // Getting focus?
```

```
    if (lpdi->itemAction & ODA_FOCUS) {
        DrawFocusRect (lpdi->hDC, &lpdi->rcItem);
        return;
    }
}

int COwnerDrawList:: CompareItem (LPCOMPAREITEMSTRUCT lpCompareItemStruct)
{
  ColorStruct *s1 = (ColorStruct *) lpCompareItemStruct->itemData1;
  ColorStruct *s2 = (ColorStruct *) lpCompareItemStruct->itemData2;
  return strcmp (s1->string, s2->string);
}

void COwnerDrawList:: MeasureItem (LPMEASUREITEMSTRUCT lpMeasureItemStruct)
{
  // Get the default list height

  lpMeasureItemStruct->itemHeight = 20;
}
```

7. 将此文件存为 OWNLIS.CPP。接着，在 Visual C++中创建另外一个新文件，并在新文件中输入下列代码：

```
// ownlist.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// COwnerDrawList window

class COwnerDrawList : public CListBox
{
typedef struct {
  char *string;
  COLORREF color;
} ColorStruct;

// Construction
public:
  COwnerDrawList ();

// Attributes
public:

// Operations
public:
```

```
  int AddString ( COLORREF color, char * string );

// Implementation
public:
  virtual ~COwnerDrawList ();

protected:
   virtual void MeasureItem (LPMEASUREITEMSTRUCT lpMIS);
   virtual void DrawItem (LPDRAWITEMSTRUCT lpDIS);
   virtual int CompareItem (LPCOMPAREITEMSTRUCT lpCIS);

  // Generated message map functions
  // { {AFX_MSG (COwnerDrawList)
  //}} AFX_MSG
  DECLARE_MESSAGE_MAP ()
};
```

8. 将此文件存为 OWNLIS.H。

9. 进入 AppStudio，选择主菜单。添加新的主菜单，标题为 Dialogs，在菜单 Dialogs 中添加新的菜单项，此菜单项的标识符为 ID_OWNER_DRAW_DLG，标题为 Owner Draw Dialog。

10. 进入 ClassWizard，选择应用程序对象 CCh77App，从对象列表中选择对象 ID_OWNER_DRAW_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新函数命名为 OnOwnerDrawDlg。在 CCh77App 的方法 OnOwnerDragDlg 中输入下列代码：

```
void CCh77App:: OnOwnerDrawDlg ()
{
   COwnerDrawDlg dlg;
   dlg.DoModal ();

}
```

11. 在源文件 CH77.CPP 顶部的 include 文件列表中添加下列 include 文件行：

```
#include " ownerdra.h"
```

12. 编译并运行此例子程序。

用法

在上一节中，概述了自绘制列表框通常是如何工作的，也介绍清楚了自绘制的基本过程。在本节中，则主要介绍一些计算和技巧，用来在列表框中绘制列表项。

在显示对话框时，列表框开始自绘制，列表框函数调用程序员编写的方法 DrawItem 来将列表项绘制到列表框中。方法 DrawItem 将传送来的列表框数据强制转化为 ColorStruct 结构指针，并使用 ColorStruct 结构中的颜色和文本来绘制列表项。

首先，使用 ColorStruct 结构中的颜色在列表框的列表项区域绘制矩形，并使用此颜色填充矩形。接着，使用此颜色来设置文本的颜色，以便将文本显示在列表项区域中。最后，紧挨着列表项中的颜色框，将文本绘制到列表框中。

但是，如何获取ColorStruct结构中的列表项数据呢？答案在于自绘制列表框的重置方法AddString。如果仔细研究此方法的源代码，就可以发现调用此方法需要两个参数，一个是颜色（RGB值），一个是要显示的字符串。在此方法中，首先为ColorStruct结构创建一个新的实例，并将传送来的参数值分别赋给结构中的相对应的域；接着，将此结构以“字符串”的形式存放在列表框中，当此数据传送给程序员编写的处理函数时，列表框才处理此数据，所以整个实现过程恰到好处，应该注意，在比较函数中，操作传送来的数据之前，首先将这两个值强制转化成ColorStruct结构。

最后，在自绘制列表框的析构方法中，从列表框中取回每个列表项，并释放每个列表项以及列表项字符串所占的内存空间。

注释

本节例子程序创建的新列表框类可以作为构件保存在程序员的工具箱中，以便用在许多不同的应用程序中。也可以扩展此列表框类，使得此类不只可以处理一些简单的颜色，另外还可以在列表框中存放字体、数字、或任何其他需要存放的信息。

如果利用本节所介绍的技巧以及按列表项存放列表项数据的技巧，就可以在列表框中建立一个小型数据库，用来存放应用程序所需要的所有信息。

本节的例子程序是使用Visual C++来开发的，使用Delphi同样也可以创建自绘制列表框。但限于篇幅，本节将不再介绍使用Delphi开发的例子程序，读者可以自行来完成。

7.8 在列表框中存放更多的列表项

问题

在有的应用程序中，需要在列表框中存放大量的列表项，而且，此应用程序不仅需要在Windows 95上运行，有时还需要在Windows 3.x上运行，即有时需要此应用程序以16位模式运行。在Windows 95上，此应用程序一般运行得非常好，但在Windows 3.x上，当在列表框中插入过多的列表项时，会使应用程序崩溃。

因此，就需要实现新的列表框，以突破16位列表框的限制，并且在16位系统或32位系统上都能运行，最好还能够定制列表框，例如：设置背景颜色，设置文本颜色等等。如何来实现这种列表框呢？

方法

尽管本节所要实现的功能可以利用列表框控制来完成，但是实现起来比前面几节的例子程序要难得多，需要重新编写列表框控制的大部分内部代码，才能勉强实现所需的功能。在16位系统中，限制列表框只能有32767字节的数据，即最多只能有几千个列表项，要修改此限制，则凡是引用列表项的例程都需要进行修改，原因是在这些例程中，引用列表项使用的是整数值，而不是长整数值。

因此，在本节中将讨论利用Windows操作系统的另外一个功能，即创建定制控制。利用API来创建新的Windows类，以便在屏幕上显示虚拟列表框，从而可以显示出分列列表并允许水平滚动，所有这些功能都在控制内实现，与创建列表框的子类并实现任意数目的列表项添加功能相比，所需要编写的代码要少得多。

在虚拟列表框中，“虚拟”的概念是表示这么一个事实：列表框中不存放实际的列表项，而是创建“提供者”对象，用来将显示列表项所需要的信息提供给虚拟列表框。这样实现有

两个好处，第一个好处是：程序员可以改变向系统输入数据的方式，但不需要修改列表框中已显示的列表项；第二个好处是：虚拟列表框不依赖于 Windows 列表框的任何功能，所以可以不加修改地运行在 Windows 3.x、Windows 95、或 Windows NT 上。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，点击按钮 New File 或者从主菜单中选择 File|New，则弹出一个对话框，在对话框中列出两个文档类型，选择第二个文档类型并点击按钮 OK，则显示的窗口如图 7-11 所示，可以在水平或垂直方向上任意滚动这些列表项。如果要关闭窗口，可以点击关闭按钮，或者在 Windows 3.x 中双击系统菜单。

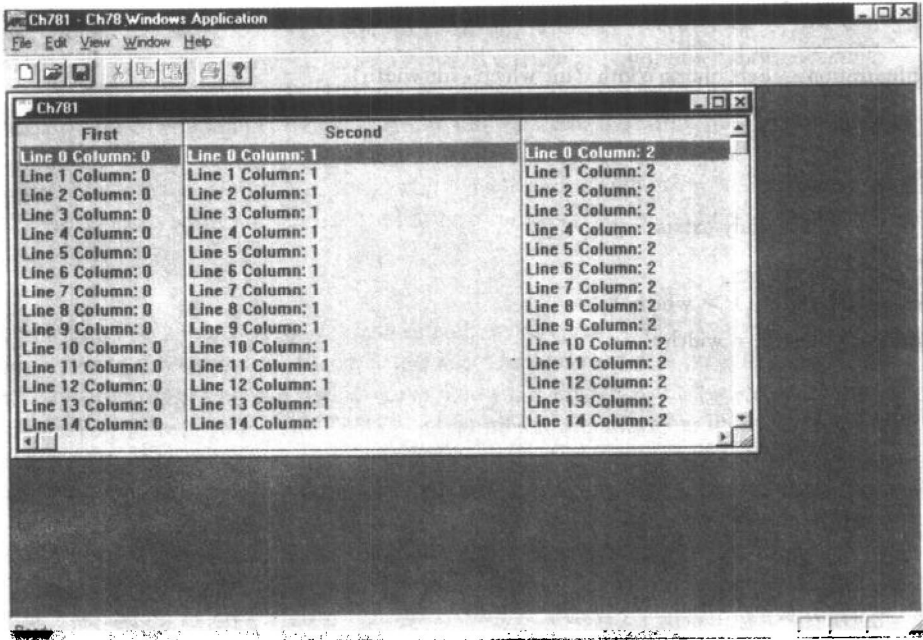


图 7-11　虚拟列表框视图

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH78.MAK。选择主菜单 File，从下拉菜单中选择菜单项 New，用来创建新的文件，并在新文件中输入下列代码：

```
#include " stdafx.h"
#include " listconf.h"

ListConfiguration:: ListConfiguration (long totalItems)
{
   totalNumberOfItems = totalItems;
   currentItem        = 0L;
}

ListConfiguration:: ~ListConfiguration (void)
```

```
{
}

// Access methods

int ListConfiguration:: AddColumn (int width, const CString& headerString)
{
   positions.Add ( width );
   headers.Add ( headerString );
   return 0;
}

int ListConfiguration:: SetColumnWidth (int which, int width)
{
   int status = 0;

   // See if this item already exists...

   if ( positions.GetSize () > which )
      positions [which] = width;
   else
      status = -1;

   return status;
}

int ListConfiguration:: GetColumnWidth (int which)
{
   if ( positions.GetSize () > which )
      return positions.GetAt (which);
   return -1;
}

int ListConfiguration:: SetColumnHeader (int which, const CString& headerString)
{
   int status = 0;

   // See if this item already exists...

   if ( headers.GetSize () > which )
      headers [which] = headerString;
   else
      status = -1;
```

```
    return status;
}

int ListConfiguration:: GetColumnHeader (int which, CString& s)
{
    if ( headers.GetSize () > which ) {
        s = headers.GetAt (which);
        return 0;
    }
    return -1;
}

// Actual list text provider methods

int ListConfiguration:: SetCurrentItemNumber (long item)
{
    int status = 0;
    if ( item >= 0 && item < totalNumberOfItems )
        currentItem = item;
    else
        status = -1;
    return status;
}

long ListConfiguration:: GetCurrentItemNumber (void)
{
    return currentItem;
}

char *ListConfiguration:: GetColumn (int which)
{
    static char buffer [80];
    sprintf (buffer, " Line %ld Column: %d", currentItem, which );
    return buffer;
}

int ListConfiguration:: NumberOfColumns ()
{
    return positions.GetSize ();
}
```

2. 将此文件存为 LISTCONF.CPP，并创建另外一个新文件，在刚创建的新文件中添加下列代码：

```
#ifndef _LCONFIG_H_
#define _LCONFIG_H_

class ListConfiguration {

private:
   CUIntArray        positions;
   CStringArray      headers;
   long              totalNumberOfItems;
   long              currentItem;

public:
   ListConfiguration (long totalItems);
   virtual ~ListConfiguration (void);

// Access methods

   int NumberOfColumns ();
   int AddColumn (int width, const CString& headerString);
   int SetColumnWidth (int which, int width);
   int GetColumnWidth (int which);
   int SetColumnHeader (int which, const CString& headerString);
   int GetColumnHeader (int which, CString& s);

// Actual list text provider methods

   long NumberOfItems (void) { return totalNumberOfItems; };
   int  SetCurrentItemNumber (long item);
   long GetCurrentItemNumber (void);
   char *GetColumn (int which);

};

#endif
```

3. 将此文件存为 LISTCONF.H，接着，创建新的文件并存为 VLISTWND.CPP，在此文件中添加下列代码：

```
// vlistwnd.cpp : implementation file
//

#include "stdafx.h"
#include "ch78.h"
#include "vlistwnd.h"
```

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

/////////////////////////////////////////////////////////////////////////////
// CVListWnd

CVListWnd:: CVListWnd (ListConfiguration *config)
{
   xoffset = 0;
   cur_row = 0;
   top_row = 0;
   last_screen_row = 8;
   top_of_list_area = 20;
   conf = config;
}

CVListWnd:: ~CVListWnd ()
{
}

BEGIN_MESSAGE_MAP (CVListWnd, CWnd)
  //{{AFX_MSG_MAP (CVListWnd)
  ON_WM_ERASEBKGND ()
  ON_WM_PAINT ()
  ON_WM_HSCROLL ()
  ON_WM_CREATE ()
  ON_WM_VSCROLL ()
  ON_WM_KEYDOWN ()
  ON_WM_LBUTTONDOWN ()
  //}} AFX_MSG_MAP
END_MESSAGE_MAP ()

/// //////////////////////////////////////////////////////////////////////
// CVListWnd message handlers

BOOL CVListWnd:: OnEraseBkgnd (CDC * pDC)
{
   COLORREF rgbBkColor = GetSysColor (COLOR_BTNFACE);
   //
   // Erase only the area needed
```

```
    //
    CRect rect;
    GetClientRect (&rect) ;

    //
    // Make a brush to erase the background.
    //
    CBrush NewBrush (rgbBkColor);

    pDC->SetBrushOrg (0, 0) ;

    CBrush * pOldBrush = (CBrush * ) pDC->SelectObject (&NewBrush);

    //
    // Paint the Background....
    //
    pDC->PatBlt (rect.left, rect.top, rect.Width (), rect.Height (), PATCOPY);
    pDC->SelectObject (pOldBrush);
    return TRUE;
}

void CVListWnd:: OnPaint ()
{
    CPaintDC dc (this); // device context for painting

      CPen text_pen (PS_SOLID, 1, GetSysColor (COLOR_GRAYTEXT));
      CPen dark_pen (PS_SOLID, 1, GetSysColor (COLOR_BTNSHADOW));
      CPen shadow_pen (PS_SOLID, 1, GetSysColor (COLOR_BTNHIGHLIGHT));
      CRect r;
      GetClientRect (&r);
      TEXTMETRIC tm;
      dc.GetTextMetrics ( &tm );

      // Next, draw the lines for the column boundaries

      int x = 0;
      int totalColumns = 0;

      for ( int i=0; i<conf->NumberOfColumns (); ++i ) {
         dc.SelectObject (&text_pen);

         int offset = (conf->GetColumnWidth (i) - xoffset) * tm.tmMaxCharWidth;
```

```
    // Do the header..

    CRect r1 (x+1, 1, offset-1, top_of_list_area-1 );
    CRect tRect = r1;

    CString s;
    conf->GetColumnHeader (i, s);

    DoColumnHeader ( dc, s, r1 );


    dc.MoveTo (offset, 0);
    dc.LineTo (offset, r.bottom);

    dc.SelectObject (&dark_pen);
    dc.MoveTo (offset-1, 0);
    dc.LineTo (offset-1, r.bottom);

    dc.SelectObject (&shadow_pen);
    dc.MoveTo (offset+1, 0);
    dc.LineTo (offset+1, r.bottom);

    x = offset + 2;
    if ( totalColumns < conf->GetColumnWidth (i) )
        totalColumns = conf->GetColumnWidth (i);
}

SetScrollRange ( SB_HORZ, 0, totalColumns+conf->NumberOfColumns () );

// Draw a line for the top of the list area

dc.SelectObject ( &dark_pen );
dc.MoveTo ( 0, top_of_list_area-1 );
dc.LineTo ( r.right, top_of_list_area-1 );

int y = top_of_list_area;
long line = top_row;
while ( y < dc.m_ps.rcPaint.bottom && line < conf->NumberOfItems () ) {
    y = DisplayLine ( &dc, line );
    if ( y < dc.m_ps.rcPaint.bottom )
        last_line = line;
    line++;
}
```

```
    ShowCursor (&dc, cur_row);
}

void CVListWnd:: OnHScroll (UINT nSBCode, UINT nPos, CScrollBar * pScrollBar)
{
  int xInc = 0;
  int min, max;

  GetScrollRange ( SB_HORZ, &min, &max );

  switch (nSBCode)
   {
    case SB_PAGEUP:
      if ( xoffset > 8 )
         xInc = -8;
      else
         xInc = -xoffset;
      break;

    case SB_PAGEDOWN:
      xInc = 8;
      break;

    case SB_LINEUP:
      if ( xoffset )
         xInc = -1;
      break;

    case SB_LINEDOWN:
      if ( xoffset < max )
         xInc = 1;
      break;

    default:
      return;

   }
  xoffset += xInc;
  SetScrollPos (SB_HORZ, xoffset, TRUE);

  CWnd:: OnHScroll (nSBCode, nPos, pScrollBar);
  if ( xInc )
```

```
    InvalidateRect ( NULL );
}

int CVListWnd:: OnCreate (LPCREATESTRUCT lpCreateStruct)
{
  if (CWnd:: OnCreate (lpCreateStruct) == -1)
    return -1;

  return 0;
}

void CVListWnd:: ShowCursor (CDC *dc, long row)
{
   // Determine the start of this row on the screen

     TEXTMETRIC tm;
     dc->GetTextMetrics ( &tm );
     int y = top_of_list_area + tm.tmHeight * (int) (row-top_row);

     // For each " field", hilight the rectangle

     int st_x = 3;
     CRect r;

     for ( int i=0; i<conf->NumberOfColumns (); ++i ) {

        int end_x = (conf->GetColumnWidth (i) -xoffset) * tm.tmMaxCharWidth;
        r.top = y;
        r.bottom = y + tm.tmHeight;
        r.left = st_x;
        if ( r.left < 3 )
            r.left = 3;
        r.right = end_x - 3;
        if ( r.right < 3 )
            r.right = 3;

        dc->InvertRect ( &r );

        st_x = end_x + 3;
     }
 }

int CVListWnd:: DisplayLine ( CDC *dc, long row )
```

```
{
// Determine the start of this row on the screen

  TEXTMETRIC tm;
  dc->GetTextMetrics ( &tm );
  int y = top_of_list_area + tm.tmHeight * (int) (row-top_row);

  // For each " field", hilight the rectangle

  int st_x = 3;
  CRect r;
  dc->SetTextColor ( GetSysColor (COLOR_BTNTEXT) );
  dc->SetBkColor ( GetSysColor ( COLOR_BTNFACE) );

  int offset = xoffset;

  for ( int i=0; i<conf->NumberOfColumns (); ++i ) {

    int end_x = (conf->GetColumnWidth (i) -xoffset) * tm.tmMaxCharWidth;

    r.top = y;
    r.bottom = y + tm.tmHeight;

    r.left = st_x;
    r.right = end_x - 3;

    conf->SetCurrentItemNumber (row);

    char *text = conf->GetColumn (i);
    if ( offset < (int) strlen (text) )
        text += offset;
    else
        text= NULL;

    if ( text )
        dc->DrawText ( text, strlen (text), &r, DT_LEFT | DT_SINGLELINE );

    offset = xoffset-conf->GetColumnWidth (i);
    if ( offset < 0 )
        offset = 0;


    st_x = end_x + 3;
```

```
        if ( st_x < 3 )
            st_x = 3;
    }

    return y + tm.tmHeight;
}

void CVListWnd::OnVScroll (UINT nSBCode, UINT nPos, CScrollBar * pScrollBar)
{
  double dpct;

  CDC *dc = GetDC ();

  // Get the text metrics used for this screen.

  TEXTMETRIC tm;
  dc->GetTextMetrics ( &tm );


  // Calculate total lines per page

  CRect r;
  GetClientRect (&r);

  int num_lines = (r.bottom - top_of_list_area) / tm.tmHeight;

  switch (nSBCode)
   {
    case SB_PAGEUP:

      // First case:   Not enough lines to " page"

      if ( top_row < num_lines ) {

         // Silly case -- top line is already first line

         if ( top_row == 0 ) {
             ReleaseDC (dc);
             return;
         }

         // Otherwise make first line top line
```

```
        top _ row = 0;
        cur _ row = top _ row;
        InvalidateRect (NULL);
    }
    else {   // Next case, just reset top line to be the old top line — " page"
        top _ row = top _ row - num _ lines + 1;
        cur _ row = top _ row;
        InvalidateRect (NULL);
    }

    break;

  case SB _ PAGEDOWN:
    // Are we already at the end?

    if ( last _ line == conf->NumberOfItems () -1 ) {
        ReleaseDC (dc);
        return;
    }

    // No.  Make the last line the first line

    top _ row = last _ line;
    cur _ row = last _ line;
    InvalidateRect (NULL);

    break;

  case SB _ LINEUP:
    if ( cur _ row > top _ row ) {
        ShowCursor (dc, cur _ row);
        cur _ row--;
        ShowCursor (dc, cur _ row);
    }
    else
        if ( top _ row ) {
            top _ row--;
            cur _ row--;
            InvalidateRect (NULL);
        }
    break;

  case SB . LINEDOWN:
```

```
        if ( cur_row < last_line) {
            ShowCursor (dc, cur_row);
            cur_row++;
            ShowCursor (dc, cur_row);
        }
        else {
            if ( cur_row < conf->NumberOfItems () -1 ) {
                top_row ++;
                cur_row ++;
                InvalidateRect (NULL);
            }
        }
        break;


      case SB_THUMBPOSITION:

        // The position of the thumb indicates what percentage of the items we are at.
        // Use it to determine where we are.

        dpct = (double) nPos / 100.0;
        top_row = (long) ( (double) conf->NumberOfItems () * dpct);
        cur_row = top_row;
        InvalidateRect (NULL);
        break;


      default:
        ReleaseDC (dc);
        return;

    }
    SetScrollRange (SB_VERT, 0, 100);
    // Determine where we are in the list
    if ( cur_row != 0 ) {
        int pct = (int) ( ( (double) (cur_row+1) / (double) conf->NumberOfItems ()) * 100.0);
        SetScrollPos (SB_VERT, pct, TRUE );
    }
    else
        SetScrollPos (SB_VERT, 0, TRUE);
    ReleaseDC (dc);
    CWnd::OnVScroll (nSBCode, nPos, pScrollBar);
}
```

```
void CVListWnd:: OnKeyDown (UINT nChar, UINT nRepCnt, UINT nFlags)
{
  WORD mScrollNotify = -1;
  WORD mScrollType = -1;
  switch (nChar)
   {
    case VK_RIGHT:
      mScrollType   = WM_HSCROLL;
      mScrollNotify = SB_LINEDOWN;
      break;

    case VK_LEFT:
      mScrollType   = WM_HSCROLL;
      mScrollNotify = SB_LINEUP;
      break;

    case VK_UP:
      mScrollType   = WM_VSCROLL;
      mScrollNotify = SB_LINEUP;
      break;

    case VK_PRIOR:
      mScrollType   = WM_VSCROLL;
      mScrollNotify = SB_PAGEUP;
      break;

    case VK_NEXT:
      mScrollType   = WM_VSCROLL;
      mScrollNotify = SB_PAGEDOWN;
      break;

    case VK_DOWN:
      mScrollType   = WM_VSCROLL;
      mScrollNotify = SB_LINEDOWN;
      break;

    case VK_HOME:
      mScrollType   = WM_VSCROLL;
      mScrollNotify = SB_TOP;
      break;

    case VK_END:
```

```
      mScrollType   = WM_VSCROLL;
      mScrollNotify = SB_BOTTOM;
      break;

    case VK_RETURN:
      return;
    default:
      break;
    }

  if (mScrollNotify != -1)
    PostMessage (mScrollType, mScrollNotify);


  CWnd::OnKeyDown (nChar, nRepCnt, nFlags);
}

void CVListWnd::OnLButtonDown (UINT nFlags, CPoint point)
{
  TEXTMETRIC tm;
  CDC *dc = GetDC ();
  dc->GetTextMetrics ( &tm );

  // Figure out which one it is...

  long which_row = top_row + ( (point.y-top_of_list_area) / tm.tmHeight);

  // If the line selected is within the current range of items, select it.

  if ( which_row < conf->NumberOfItems () ) {
    ShowCursor (dc, cur_row);
    cur_row = which_row;
    ShowCursor (dc, cur_row);
  }

  ReleaseDC (dc);

  // Life is good.  Do the default processing.

  CWnd::OnLButtonDown (nFlags, point);
}

void CVListWnd::DoColumnHeader ( CDC&dc, CString& s, CRect r )
```

```
{
    CBrush b (GetSysColor (COLOR_BTNFACE));
    dc.FillRect ( &r, &b );
    int mode = dc.GetBkMode ();

    dc.SetBkMode (TRANSPARENT);
    dc.SetBkColor ( GetSysColor (COLOR_BTNFACE) );
    dc.DrawText ( s, strlen (s), &r, DT_CENTER );

    dc.SetBkMode (mode);
}
```

4. 创建新的文件并存为 VLISTWND.H，在此文件中输入下列代码：

```
// vlistwnd.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CVListWnd window

// Include files for this object

#include " listconf.h"

class CVListWnd : public CWnd
{
private:
    int xoffset;
    int header_mode;
    int last_screen_row;
    long cur_row;
    long top_row;
    CRect draw_rect;
    int current_select_line;
    int top_of_list_area;
    long last_line;
    ListConfiguration *conf;

// Construction
public:
  CVListWnd (ListConfiguration *conf);

// Attributes
public:
```

```
// Operations
public:

// Implementation
public:
  virtual ~CVListWnd ();
  virtual void DoColumnHeader ( CDC&dc, CString& s, CRect r );

protected:
     void ShowCursor (CDC *dc, long row);
     int  DisplayLine ( CDC *dc, long row );

  // Generated message map functions
  // { {AFX_MSG (CVListWnd)
  afx_msg BOOL OnEraseBkgnd (CDC * pDC);
  afx_msg void OnPaint ();
  afx_msg void OnHScroll (UINT nSBCode, UINT nPos, CScrollBar * pScrollBar);
  afx_msg int OnCreate (LPCREATESTRUCT lpCreateStruct);
  afx_msg void OnVScroll (UINT nSBCode, UINT nPos, CScrollBar * pScrollBar);
  afx_msg void OnKeyDown (UINT nChar, UINT nRepCnt, UINT nFlags);
  afx_msg void OnLButtonDown (UINT nFlags, CPoint point);
  //}} AFX_MSG
  DECLARE_MESSAGE_MAP ()
};

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

5. 选择应用程序对象的源文件 CH78.CPP，在方法 InitInstance 中，创建对象 pDocTemplate 的代码段的正下面，添加下列代码：

```
pDocTemplate = new CMultiDocTemplate (
        IDR_CH78TYPE,
        RUNTIME_CLASS (CNewDoc),
        RUNTIME_CLASS (CMDIChildWnd),    // standard MDI child frame
        RUNTIME_CLASS (CNewView));
    AddDocTemplate (pDocTemplate);
```

6. 最后，在文件 CH78.CPP 顶部的 include 文件列表中添加下面两个 include 文件行：

```
#include " newview.h"
#include " newdoc.h"
```

7. 进入 ClassWizard，点击按钮 Add Class，新类命名为 CNewView，并选择类 CView 作为新类的基类，在 ClassWizard 中，从对象列表中选择对象 CNewView，从消息列表中选择消息 WM_CREATE，在 CNewView 的方法 OnCreate 中添加下列代码：

```
int CNewView:: OnCreate (LPCREATESTRUCT lpCreateStruct)
{
```

```
 if (CView:: OnCreate (lpCreateStruct) == -1)
   return -1;

   conf = new ListConfiguration (50);

   conf->AddColumn (10, " First");
   conf->AddColumn (30, " Second" );
   conf->AddColumn (60, " Third" );

   wnd = new CVListWnd (conf);

   CRect r1;
   GetClientRect (&r1);

   BOOL what= wnd->Create (NULL,
                 "",
                 WS_CHILD |WS_VISIBLE |WS_VSCROLL|WS_BORDER|WS_HSCROLL,
                 r1,
                 this,
                 NULL);
 return 0;
}
```

8. 在 ClassWizard 中，从对象列表中选择对象 CNewView，从消息列表中选择消息 WM_SIZE，在 CNewView 的方法 OnSize 中添加下列代码：

```
void CNewView:: OnSize (UINT nType, int cx, int cy)
{
   CView:: OnSize (nType, cx, cy);

   CRect r;
   GetClientRect (&r);
   wnd->MoveWindow ( &r );
}
```

9. 在类 CNewView 的构造方法和析构方法中添加下列行（黑体表示）：

```
CNewView:: CNewView ()
{
   wnd = NULL;
   conf = NULL;
}

CNewView:: ~CNewView ()
{
   if ( wnd )
```

```
        delete wnd;
    if ( conf )
        delete conf;
}
```

10. 在头文件 NEWVIEW.H 中添加下列代码（同样只添加黑体表示的行）：

```
class CVListWnd;
class ListConfiguration;

class CNewView : public CView
{
private:
    CVListWnd  * wnd;
    ListConfiguration  * conf;
```

11. 返回 ClassWizard，点击按钮 Add Class，新类命名为 CNewDoc，并选择 CDocument 作为新类的基类，点击按钮 Generate Class。

12. 编译并运行此例子程序。

用法

MFC 的文档/视图体系使得程序员可以非常简单方便地创建新的视图类型，并且由基类库系统来完成所有的后台处理，以便在应用程序运行时将视图添加到应用程序中，另一方面，Windows API 使得程序员可以灵活地创建新的窗口类，并且新的窗口类是以现有的 MFC 类为基类的。将 MFC 和 Windows API 联合使用，可以具有非常强大的功能，就如本节的例子程序所示范的。

当 MFC 创建视图来响应用户的 File|New 请求时，对象 CNewView 被实例化，接着，调用视图的方法 OnCreate，在视图窗口内创建新的基本窗口类型，一旦视图被接管，则相应的屏幕区域就不再由视图来负责了，所有的消息、键盘输入以及绘制操作都由程序员创建的伪列表框类来处理。应该注意对象 CVListWnd 的方法 Create 的调用，此方法完全映射 Windows API 函数 CreateWindow，窗口名为“”，即空字符串，并且窗口具有水平和垂直滚动条。

一旦创建了窗口，则所有的操作都由 CVlistWnd 的方法来完成。与本章前面几节介绍的自绘制列表框一样，CVlistWnd 的方法 OnPaint 用来响应 Windows 消息 WM _PAINT，此方法从提供者对象中获取数据，并将这些数据以有序的列显示在窗口中。

许多实际的操作是在提供者类中完成的，利用此类，使得实现真正无限制大小以及用户可以选择的列表成为可能。在本节的例子程序中，提供者类只是以列表项的行号和列号来构造所要显示的字符串。

在本节的例子程序中，有几个重要的地方需要注意，就是 Windows API 函数的使用以及它们的结果。在 CVListWnd 的方法 DoColumnHeader 中，有三个调用 Windows API 函数的好例子（在这里，间接通过 MFC 对象）。第一个是以参数 TRANSPARENT 调用函数 SetBkMode，此函数用来指示 Windows 在向屏幕绘制文本或图形时如何对待背景颜色，在本节的例子程序中，TRANSPARENT 表示绘制文本时背景不做修改。第二个是调用函数 SetBkColor，此函数用来设置背景的颜色。最后一个是调用函数 DrawText，此函数将文本实际绘制到屏幕上，一个类似的函数被用来将文本显示到列表框的每列中。

注释

要实现一个全功能的虚拟分列列表框，本节的虚拟列表框是一个很好的起点，对本节实现的虚拟列表框稍加改进，便可成为通用的构件，从而可以方便地添加到应用程序中。另外，对 SetCapture 和 WM_MOUSEMOVE API 函数稍加改进，可以实现全功能的网格工具。

这类构件最好是用在数据库或文本检索应用程序中，虚拟列表框中的列表项可以用来表示数据库中的记录或者文本检索中的文件，而列可以是字段或者文件查找中的属性。

7.9 实现层次列表（树）

问题

在有的应用程序中，需要使用“树”视图，例如：Windows 3.1 中的文件管理器，或者 Windows 95 中的资源管理器。创建树视图是否非常困难呢？如何才能实现呢？

方法

很难说清楚实现树控制(类似于 Windows 95 资源管理器中的树控制)的方法会有多么复杂，而实现树控制需要花费多长时间则更难估计，原因是在 Windows 95 中已经有了开发好的树控制，当然不会有人再考虑重新实现一个。

在本节中，将仔细讨论如何在 MFC 中创建新的视图，其中树控制是视图的基本部分。通过本节例子程序的实现，读者会发现在应用程序中创建树控制是相当简单的，而对来自于树控制的消息的响应稍微复杂一些，但利用已有的功能足以解决由于使用内部控制而引发一些问题。

注：本书的大部分例子程序都既可以在 Visual C++ 1.52 中也可以在 Visual C++ 2.1 中编译，效果都是一样的。但本节的例子程序，由于利用了 Windows 95 中的 32 位控制，所以只能在 Visual C++ 2.1 或更高版本中编译运行。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示出如图 7-12 所示的树视图，点击视图左边的加号和减号，树将相应地扩展和收缩。如果完成了对树控制的操作，可以选择 File|Exit 来终止应用程序的运行。

实现例子程序的具体步骤如下：

1. 在 Visual C++ 2.1 中，利用 AppWizard 创建新的项目文件 TREE.MAK。选择 ClassWizard，点击按钮 Add Class，创建新的视图类，新类命名为 CTreeView，并以类 CView 为基类。

2. 在 ClassWizard 中，从对象列表中选择对象 CTreeView，从消息列表中选择消息 WM_MOUSEMOVE，点击按钮 Add Function，在 CTreeView 的方法 OnMouseMove 中输入下列代码：

```
void CtreeView::OnMouseMove (UINT nFlags, CPoint point)
{
    if (m_pDragImage)
     {
        TV_HITTESTINFO tvht;
```

```
        // Fill out hit test struct with mouse pos
        tvht.pt.x = point.x;
        tvht.pt.y = point.y;

        // Check to see if an item lives under the mouse
        HTREEITEM    hTarget;
        if (NULL != (hTarget = GetTreeCtrl () .HitTest (&tvht)))
        {
            // find out what kind of item we are looking at
            TV_ITEM   tvi;    // Temporary Item
            tvi.mask = TVIF_PARAM;
            tvi.hItem = hTarget;
            GetTreeCtrl () .GetItem (&tvi);

            // if item can be dropped on ...
            if ( 0 != tvi.lParam) // can't put anything in the root
                                  // except the root
            {
                SetCursor (LoadCursor (NULL, IDC_ARROW));
            }
            else
            {
                SetCursor (LoadCursor (NULL, IDC_NO));
            }
            if ( hTarget != GetTreeCtrl () .GetDropHilightItem () )
            {
                // Hide the drag image
                ImageList_DragShowNolock (FALSE);
                // Select the item
                GetTreeCtrl () .SelectDropTarget (hTarget);
                // Show the drag image
                ImageList_DragShowNoLock (TRUE);
            }
        }
        // Do standard drag drop movement
        m_pDragImage->DragMove (point);
    }

    CView::OnMouseMove (nFlags, point);
}
```

3. 接着，在ClassWizard中选择对象CTreeView和消息WM_LBUTTONUP，点击按钮Add Function，在CTreeView的方法OnLButtonUp中输入下列代码：

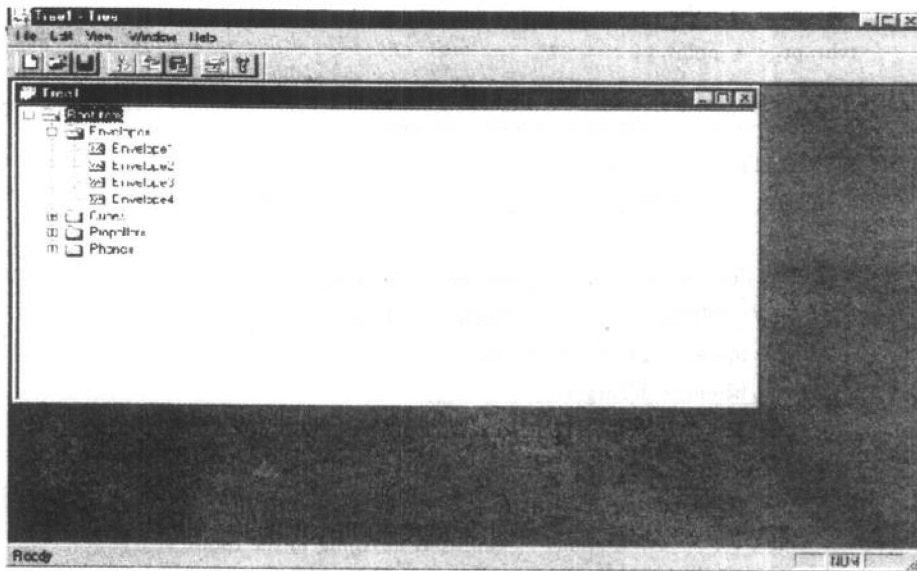


图 7-12　嵌入 MFC 视图中的树控制

```
void CTreeView:: OnLButtonUp (UINT  nFlags, CPoint  point)
{
    if (NULL != m_pDragImage)
    {
        TV_ITEM          tvi;     // Temporary Item
        TV_HITTESTINFO   tvht;

        // Fill out hit test struct with mouse pos
        tvht.pt.x = point.x;
        tvht.pt.y = point.y;
        // Check to see if an item lives under the mouse
        HTREEITEM  hTarget = GetTreeCtrl () .HitTest (&tvht);

        ImageList_EndDrag ();
        ImageList_DragLeave (m_hWnd);
        ShowCursor ( TRUE );
        SetCursor (LoadCursor (NULL, IDC_ARROW));
        ReleaseCapture ();
        m_pDragImage->DeleteImageList ();
        m_pDragImage = NULL;

        GetTreeCtrl () .SelectDropTarget (NULL);

        if (NULL != hTarget && m_hDragItem != hTarget)
            // Can't drop on nothing or itself
```

```
    {
        TV_INSERTSTRUCT   tvis;
        tvi.mask = TVIF_PARAM;
        tvi.hItem = hTarget;
        GetTreeCtrl () .GetItem (&tvi);
        if ( 0 ! = tvi.lParam && tvi.hItem ! = m_hDragItem)
        {
            // First, get the data from the object we dragged
            // and delete it from the tree
            char   buf [1024];
            tvis.item.mask = TVIF_TEXT | TVIF_IMAGE |
            TVIF_SELECTEDIMAGE | TVIF_PARAM;
            tvis.item.hItem = m_hDragItem;
            tvis.item.pszText=buf;
            tvis.item.cchTextMax = 1024;
            GetTreeCtrl () .GetItem (& (tvis.item));
            GetTreeCtrl () .DeleteItem (m_hDragItem);
            if (1 == tvi.lParam)    // it's a folder, add it to
                                    //  that folder
            {
                tvis.hParent = hTarget;
                tvis.hInsertAfter = TVI_LAST;
                GetTreeCtrl () .InsertItem (&tvis);
            }
            else  // put it after the object that's the target
            {
                tvis.hParent =GetTreeCtrl () .GetParentItem (hTarget);
                tvis.hInsertAfter = hTarget;
                GetTreeCtrl () .InsertItem (&tvis);
            }
        }
    }
    m_hDragItem = NULL;
  }

  CView:: OnLButtonUp (nFlags, point);
}
```

4. 需要添加下列函数，用来响应树控制发送来的消息。首先为类添加新方法OnChildNotify，在响应视图的子控制发送来的消息时调用此方法，在此例子程序中，树控制发送给父窗口（即视图）的任何消息都调用此方法来响应。

```
BOOL CTreeView:: OnChildNotify (UINT message, WPARAM wParam, LPARAM lParam, LRESULT *
pLResult)
```

```
{
    // Let this window handle its own notifications
    if (WM_NOTIFY == message)
    {
        switch ( ( (NMHDR * ) lParam) ->code )
        {
            case  TVN_GETDISPINFO:
                OnGetDispInfo ( (NMHDR * ) lParam);
            return TRUE;

            case  TVN_ENDLABELEDIT:
                OnEndLabelEdit ( (NMHDR * ) lParam);
            return TRUE;

            case  TVN_BEGINLABELEDIT:
                * pLResult = OnBeginLabelEdit ( (NMHDR * ) lParam);
            return TRUE;

            case  TVN_BEGINDRAG:
                OnBeginDrag ( (NMHDR * ) lParam);
            return TRUE;
        }
    }

    return CView:: OnChildNotify (message, wParam, lParam, pLResult);
}
```

5. 接着添加方法OnGetDispInfo，调用此方法来获取树中某个节点的显示信息。在本例子程序中，此方法只是根据节点是扩展的还是收缩的来确定树中要显示的是哪一个图象。

```
void  CTreeView:: OnGetDispInfo (NMHDR * pNotifyStruct)
{
    ASSERT (m_hWnd == pNotifyStruct->hwndFrom);
        TV_DISPINFO *   pDispInfo = (TV_DISPINFO * ) pNotifyStruct;

    if ( pDispInfo->item.state & TVIS_EXPANDED )
        pDispInfo->item.iSelectedImage = pDispInfo->item.iImage
                                       = imageFolderOpen;
    else
        pDispInfo->item.iSelectedImage = pDispInfo->item.iImage
                                       = imageFolder;
}
```

6. 接着，添加方法OnEndLabelEdit，使得用户可以编辑树中的标号，在响应树控制发送来的消息TVN_ENDLABELEDIT时，在OnChildNotify中调用此方法。

```
void CTreeView:: OnEndLabelEdit (NMHDR *  pNotifyStruct)
{
      ASSERT (m_hWnd == pNotifyStruct->hwndFrom);

      TV_DISPINFO * pDispInfo = (TV_DISPINFO * ) pNotifyStruct;

      // don't do anything if the mask isn't valid or if the user
      // pressed escape
      if (NULL == pDispInfo->item.pszText)     return;
      TV_ITEM  tvi;
      tvi.mask = TVIF_TEXT;
      tvi.pszText = pDispInfo->item.pszText;
      tvi.cchTextMax = lstrlen (tvi.pszText);
      tvi.hItem      = pDispInfo->item.hItem;

      GetTreeCtrl () .SetItem (&tvi);
}
```

7. 添加方法 OnBeginLabelEdit 来响应开始编辑标号的消息。此方法主要用来确定要编辑的列表项是否是合法的列表项，即是否是叶节点。

```
LRESULT   CTreeView:: OnBeginLabelEdit (NMHDR * pNotifyStruct)
{
      ASSERT (m_hWnd == pNotifyStruct->hwndFrom);

      if ( ( (TV_DISPINFO * ) pNotifyStruct) ->item.lParam > 1)    return 0;
// Can't edit folder titles or root
      else   return 1;
}
```

8. 最后一个要处理的（树发送来的）通知消息是开始拖放消息。在此方法中，需要确定拖放的节点是叶节点，而不是根节点或中间节点。如果是叶节点，则需要将此窗口设置为鼠标捕捉，以便获取列表项的新位置（在 OnLButtonUp 中实现）。

```
void   CTreeView:: OnBeginDrag (NMHDR * pNotifyStruct)
{
      ASSERT (m_hWnd == pNotifyStruct->hwndFrom);

      NM_TREEVIEW *   pNMTreeView = (NM_TREEVIEW * ) pNotifyStruct;

      // First, determine if this object is drag-and-droppable
      if (pNMTreeView->itemNew.lParam > 1)
// 0 is root, 1 is a folder, they can't be dragged
        {
            // create the drag image
            m_hDragItem = pNMTreeView->itemNew.hItem;
            m_pDragImage = GetTreeCtrl () .CreateDragImage (m_hDragItem);
```

```
        // Get the location of the item rectangle's text
        CRect   rc;
        GetTreeCtrl () .GetItemRect (m_hDragItem, &rc, TRUE);
        int  cx, cy;
        ImageList_GetIconSize (m_pDragImage->m_hImageList, &cx, &cy);
        CPoint   ptHotSpot (8, 8);    // Center it
        CPoint   ptDragSpot (pNMTreeView->ptDrag.x, pNMTreeView->ptDrag.y);
        m_pDragImage->BeginDrag (0, ptHotSpot);
        m_pDragImage->DragEnter (&GetTreeCtrl (), ptDragSpot);

        ShowCursor (TRUE);
        SetCapture ();
    }
}
```

9. 另一个被重置的方法是 PreCreateWindow。在此方法中，设置窗口的类名为树控制类，从而使得 MFC 在例子程序的视图中创建树控制，而不是在标准的窗口内。

```
BOOL  CTreeView:: PreCreateWindow (CREATESTRUCT& cs)
{
    // Force class to be a tree control...

    cs.lpszClass == WC_TREEVIEW;

    // map default CView style to default CEditView style
    cs.style |= TVS_HASLINES | TVS_EDITLABELS |
                    TVS_HASBUTTONS | TVS_LINESATROOT;

    return CView:: PreCreateWindow (cs);
}
```

10. 接着，修改视图类的方法 OnInitialUpdate，以插入节点(需要在初始视图中显示的节点)。OnInitialUpdate 只在创建视图时被调用一次，以便应用程序设置视图的初始显示状态，并初始化任何结构。

```
void CTreeView:: OnInitialUpdate ()
{
    CRect   rectClient;
    GetClientRect (&rectClient);

    // Create the image list and set it in the tree control
    m_imageList.Create (IDB_BITMAP1, 16, 1, RGB (255, 255, 255));
    GetTreeCtrl () .SetImageList (TVSIL_NORMAL, &m_imageList);

    // Build the starting tree
```

```
    HTREEITEM    hRootItem, hFolderItem;

    hRootItem = CreateTree (" Root item");
    hFolderItem =InsertTreeItem (" Envelopes", 1, hRootItem,
                I_IMAGECALLBACK);
    InsertTreeItem (" Envelope1", 2, hFolderItem, imageEnvelope);
    InsertTreeItem (" Envelope2", 2, hFolderItem, imageEnvelope);
    InsertTreeItem (" Envelope3", 2, hFolderItem, imageEnvelope);
    InsertTreeItem (" Envelope4", 2, hFolderItem, imageEnvelope);

    hFolderItem =InsertTreeItem (" Cubes", 1, hRootItem, I_IMAGECALLBACK);
    InsertTreeItem (" Cube1", 3, hFolderItem, imageCube);
    InsertTreeItem (" Cube2", 3, hFolderItem, imageCube);
    InsertTreeItem (" Cube3", 3, hFolderItem, imageCube);
    InsertTreeItem (" Cube4", 3, hFolderItem, imageCube);

    hFolderItem =InsertTreeItem (" Propellers", 1, hRootItem,
                I_IMAGECALLBACK);
    InsertTreeItem (" Propeller1", 4, hFolderItem, imagePropeller);
    InsertTreeItem (" Propeller2", 4, hFolderItem, imagePropeller);
    InsertTreeItem (" Propeller3", 4, hFolderItem, imagePropeller);
    InsertTreeItem (" Propeller4", 4, hFolderItem, imagePropeller);

    hFolderItem = InsertTreeItem (" Phones", 1, hRootItem, I_IMAGECALLBACK);
    InsertTreeItem (" Phone1", 5, hFolderItem, imagePhone);
    InsertTreeItem (" Phone2", 5, hFolderItem, imagePhone);
    InsertTreeItem (" Phone3", 5, hFolderItem, imagePhone);
    InsertTreeItem (" Phone4", 5, hFolderItem, imagePhone);

    // Expand the tree

    GetTreeCtrl () .Expand (GetTreeCtrl () .GetRootItem (), TVE_EXPAND);
}
```

11. 在类的源文件中添加下列实用函数。这两个函数实现了树控制中的两个常用操作，从而使得例子程序代码更加简洁。

```
HTREEITEM  CTreeView:: CreateTree (char *text)
{
    // Initialize tree structure

    TV_INSERTSTRUCT    tvInsertStruct;
    tvInsertStruct.item.mask = TVIF_TEXT | TVIF_IMAGE |
                TVIF_SELECTEDIMAGE | TVIF_PARAM;
    tvInsertStruct.item.stateMask = 0;
```

```
    // Set the text for the root item

    tvInsertStruct.item.pszText = text;
    tvInsertStruct.item.iSelectedImage = tvInsertStruct.item.iImage = I_IMAGECALLBACK;
    tvInsertStruct.item.cchTextMax =lstrlen (tvInsertStruct.item.pszText);
    tvInsertStruct.item.lParam = 0;
    tvInsertStruct.hParent = NULL;
    tvInsertStruct.hInsertAfter = (HTREEITEM) TVI_ROOT;
    return GetTreeCtrl () .InsertItem (&tvInsertStruct);
}

HTREEITEM  CTreeView:: InsertTreeItem (char *text, int idx,
                        HTREEITEM  parent, int image)
{
    TV_INSERTSTRUCT  tvInsertStruct;
    tvInsertStruct.item.mask = TVIF_TEXT | TVIF_IMAGE |
              TVIF_SELECTEDIMAGE | TVIF_PARAM;
    tvInsertStruct.item.stateMask = 0;
    tvInsertStruct.item.pszText = text;
    tvInsertStruct.item.iSelectedImage = tvInsertStruct.item.iImage = image;
    tvInsertStruct.item.cchTextMax = lstrlen (tvInsertStruct.item.pszText);
    tvInsertStruct.item.lParam = idx;
    tvInsertStruct.hParent = parent;
    tvInsertStruct.hInsertAfter = TVI_LAST;
    return GetTreeCtrl () .InsertItem (&tvInsertStruct);
}
```

12. 最后，需要修改类 CTreeView 的头文件。下列代码便是类 CTreeView 的说明，包括新添加方法的说明，黑体表示的行是需要添加到头文件中的代码。

```
// TreeView.h : interface of the CTreeView class
//
/////////////////////////////////////////////////////////////////////////////

enum  { imageEnvelope, imageCube, imagePropeller, imagePhone, imageFolder,
imageFolderOpen  };

class  CTreeView  :  public  CView

protected:       // create from serialization only
    CTreeView ();
    DECLARE_DYNCREATE (CTreeView)

// Attributes
```

```
public:
    CTreeDoc* GetDocument();
    CTreeCtrl &GetTreeCtrl() { return *(CTreeCtrl*)this; }
    HTREEITEM CreateTree(char *text);
    HTREEITEM InsertTreeItem(char *text, int idx,
                             HTREEITEM parent, int image);

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTreeView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual void OnInitialUpdate();
    virtual BOOL OnChildNotify(UINT message, WPARAM wParam, LPARAM lParam,
    LRESULT* pLResult);
    protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CTreeView();

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    HTREEITEM     m_hDragItem;
    CImageList    m_imageList;
    CImageList    *m_pDragImage;

// Generated message map functions
protected:
    //{{AFX_MSG(CTreeView)
    afx_msg void OnPaint();
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    //}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP ()

    // Notifications from the tree control, processed in OnChildNotify
    void  OnBeginDrag (NMHDR *   pNotifyStruct);
    void  OnGetDispInfo (NMHDR *   pNotifyStruct);
    void  OnEndLabelEdit (NMHDR * pNotifyStruct);
    LRESULT  OnBeginLabelEdit (NMHDR *   pNotifyStruct);
};

#ifndef  _DEBUG   // debug version in Treeview.cpp
inline  CTreeDoc *  CTreeView::GetDocument ()
      { return  (CTreeDoc *) m_pDocument; }
#endif

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

13. 编译并运行此例子程序。

用法

在本节的例子程序中，重要的 API 函数都是与树控制相关的。首先，在视图的方法 OnInitialUpdate 中，调用了 CImageList 对象 m_imageList 的方法 Create，此方法初始化一个图象列表以包含给定数目的图象（或位图），并将这些图象同 MFC 资源文件中的某个位图联系起来，在本节的例子程序中，使用的位图是 IDB_BITMAP1。接着，调用 API 函数 TreeView_SetImageList 来将图象列表同树控制联系起来，调用此函数是间接通过调用对象 CTreeCtrl 的 MFC 成员函数来完成的，此函数将图象列表与树控制联系起来，使得树控制可以存取这些位图，从而显示在视图中。

一旦位图被联系起来，就开始一系列的调用函数 CreateTree 和 InsertTreeItem。函数 CreateTree 首先初始化结构，以创建树的根节点，接着调用类 CTreeCtrl 的方法 InsertItem 来将根节点插入到树中，返回的句柄可用于以后的插入操作，将列表项插入到树中根节点的下面。

如果使用风格标志 TVS_EDITLABELS 来创建树，就如同函数 PreCreateWindow 中实现的那样，用户就可以编辑树中的标号，功能的实现在函数 OnBeginLabelEdit 和 OnEndLabelEdit 中完成。函数 OnGetDispInfo 则用来为树控制确定某个列表项应该显示哪个图象。当用户准备将列表项从树的某一位置拖放到另一位置时，树控制调用方法 OnBeginDrag。当用户松开鼠标键以停止拖放时，CTreeView 的方法 OnLButtonUp 被调用。

在 MFC 视图中创建树控制的所有技巧在本节中都已经介绍了，同样的方法也可以用来在对话框中创建树控制。

注释

本节的例子程序是理解并使用 Windows 95 中的树控制的一个很好的起点。由于树控制的使用相当普遍，所以作为程序员应该尽量多地了解树控制。

尽管在 Delphi 和 Visual Basic 中不能直接使用树控制（因为树控制是 32 位控制，而 Delphi 和 Visual Basic 只支持 16 位类型），但存在着许多定制控制，模拟 Windows 95 中的树控制的功能，程序员应尽量考虑在应用程序中使用这些控制，以使用户能够熟悉这类控制。

第8章 菜　单

本章中，将介绍一些在应用程序中如何对菜单进行操作的方法。菜单是 Windows 中最常用的界面组件，就连一些最简单的应用程序也需要有菜单。

本章所使用的函数，如表 8-1 所示，这些函数使得程序员可以通过编程修改菜单。例如：根据激活的窗口或应用程序中的部件不同而显示不同的选项，以及把一些在一定时间内不可选的菜单项禁止（这时菜单项变为灰色）等。使用这些函数来设计菜单，将会使应用程序变得更直观、漂亮。

表 8-1　第 8 章中使用的 Windows 95 API 函数

GetMenu	EnableMenuItem
WM_INITMENU	DrawMenuBar
GetMenuState	GetSubMenu
AppendMenu	InsertMenu
DeleteMenu	RemoveMenu
CheckMenuItem	SetMenuItemBitmaps
GetSystemMetrics	GetSysColor
GetSystemMenu	ExitWindowsEx
CreatePopupMenu	LoadMenu
TrackPopupMenu	MYMSG_CONTEXTMENU

1. 如何激活和禁止菜单项

本节介绍如何根据应用程序中的状态来激活和禁止菜单项。这里提出了解决这个问题的两个方法。

2. 如何添加和删除菜单项

本节讨论在现有菜单的末尾添加新的菜单项或在指定位置添加菜单项的方法，并介绍如何确定某菜单项是否存在以及如何删除不再需要的菜单项的方法。

3. 如何为菜单项添加核选标记

在菜单项的旁边利用核选标记，可向用户提供反馈，指出此选项当前是否处于被选中状态，也可用于模仿单选按钮的行为，即在一组按钮中，只能一个被选中，而其余不被选中。本节介绍如何在应用程序中使用这两种方法。

4. 如何利用自己定义的核选标记

本节建立在 8.3 节的基础上，介绍如何根据自己应用程序的实际情况来选用更贴切的核选标记代替 Windows 中标准的核选标记。例如，可用某种颜色或某种符号做核选标记。

5. 如何为系统菜单添加选项

程序员除可以修改应用程序中自己的菜单之外，也可利用相同的方法来修改应用程序的系统菜单或应用程序中其他窗口的系统菜单。本节示范了在应用程序主窗口的系统菜单里如

何添加“Shutdown”菜单项的方法。

6. 如何设计点击鼠标右键后弹出的菜单

在 Windows 95 中出现的最新用户界面工具中，有一个是与上下文有关的弹出式菜单。当在几乎任一应用程序的用户区中按下鼠标右键时，将会在窗口上出现一个上浮弹出式菜单，菜单上包含一些适用于应用程序当前所处状态或应用程序被点击的部分所处状态的选项。本节介绍如何在应用程序中编入这些特性。

8.1 激活和禁止菜单项

问题

人们总希望应用程序的菜单能更准确地反映出当前应用程序所处的状态。例如，在编辑菜单中，未选择文本时，菜单项 Cut 或 Copy 处于激活状态是毫无意义的，因为此时根本没有 Cut 或 Copy 所能执行的对象。这时，如果让 Cut 或 Copy 处于禁止状态，也就是处于不可选状态，会更实际、更有意义些。那么，如何能做到激活或禁止整个菜单项呢?

方法

在设计应用程序时，禁止菜单项的方法就是把资源脚本中该菜单项的类型设置为 GRAYED 或在资源编辑器中选择相应的方框。在运行应用程序时，改变菜单项或菜单的状态也不是什么难做到的事情。

无论改变菜单的何种状态或属性，都需要首先知道该菜单的句柄。API 的函数 GetMenu 以某个窗口的句柄作为参数，用来返回该窗口中菜单的句柄，其类型为 HWND（系统定义的类型)。需要注意的是，对某一子窗口来调用 GetMenu 函数是无法返回有效的菜单句柄的，因为子窗口不会有菜单。

获得了菜单句柄后，就可以调用 API 函数 EnableMenuItem。虽然这个函数的名字为激活菜单项，实际上利用它，既可激活也可变灰菜单项（变灰，就是禁止的意思，关于变灰与禁止的比较，参见下面的注释）。调用 EnableMenuItem 时，需传递几个参数。首先，传递要修改的菜单的句柄，另外还要传递要修改的菜单项的 ID（标识符）或它在菜单中的位置值，最后，还要传递一些标志，这些标志用来确定是按 ID 查找还是按位置查找，是激活该菜单项还是禁止该菜单项。这些标志有：MF _ BYCOMMAND，这一标志通知 Windows 按菜单项的 ID 来查找该菜单项，标志 MF _ BYPOSITION 通知 Windows 按菜单项的位置来查找该菜单项；标志 MF _ ENABLED 用来激活菜单项，标志 MF _ GRAYED 用来变灰即禁止菜单项。

我们既可以在对事件的响应中激活或变灰菜单项，也可以在菜单被下拉前，利用 Windows 发送的消息 WM _ INITMENU 来激活或变灰菜单项。当应用程序收到消息 WM _ INITMENU 后，将根据当前应用程序的状态激活或变灰菜单项。后一种方法常常能设计出简单、可扩充的应用程序，因为它把激活与变灰菜单项的代码集中地保存在一个地方。

关于“变灰”与“禁止”的说明

在 Windows 早期的版本中，应用程序没有使不能被选择的菜单项变灰，而是看起来与操作起来都同一般可被选择的菜单项相同，只是选择后不产生任何反应。实际上，这个菜单项已利用标志 MF _ DISABLED 被禁止了。标志 MF _ DISABLED 阻止了 Windows 把 WM _ COMMAND 消息传送到应用程序的那个菜单项上，只不过是没有改变该菜单项的外观表现

而已。

现在的应用程序利用标志 MF _ GRAYED 来代替标志 MF _ DISABLED，以便用户能够分辨出某个菜单项是否已处于不能被选择的状态，用户也就不再为“为什么一些菜单项选择后不执行任何工作”而迷惑了。

因此，虽然在谈到菜单项时利用激活和禁止是比较简单的，但本书中一般使用激活（enabled）与变灰（grayed）这两个术语，以免在应用程序中设置标志时产生任何混淆。

步骤

按照如下步骤生成的例子程序 ENABLE. EXE 运行时显示如图 8-1 所示的窗口。该例子程序有两个菜单：菜单 File 与 菜单 Options。图中显示的菜单 Options 是被激活的，但它可通过从菜单 File 中选择菜单项 Enable Menu 或菜单项 Disable Menu 来激活或变灰。在此例显示的图中，菜单项 Disable Menu 正处于激活状态而菜单项 Enable Menu 处于变灰状态，这正是因为菜单 Options 此时是处于激活状态的。

图 8-1　例子程序 ENABLE 的运行结果

下面的步骤指导如何生成这一例子程序。

1. 首先创建一个保存例子程序所需资源文件的工作目录。在这里我们创建一个目录，并把该目录命名为 ENABLE。

2. 为例子程序创建资源文件，该文件将定义此例子程序里用到的菜单以及应用程序图标。利用文本编辑器创建新文件，并把此新文件命名为 ENABLE. RC。在此文件中输入下面的资源脚本。

```
/* ---------------------------------------------------------------- */
/*                                                                  */
/* MODULE: ENABLE.RC                                                */
/* PURPOSE: This resource script defines the base menu for this    */
/*          example application.                                    */
/*                                                                  */
```

```
/*-----------------------------------------------------------------------*/

#include <windows.h>
#include "enable.rh"

IDM_TESTMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "&Enable Menu", CM_ENABLE
        MENUITEM "&Disable Menu", CM_DISABLE
        MENUITEM SEPARATOR
        MENUITEM "E&xit", CM_EXIT
    }
    POPUP "&Options"
    {
        MENUITEM "&Configure...", CM_CONFIGURE
        MENUITEM "&Directories...", CM_DIRECTORIES
        MENUITEM "&Save Options", CM_SAVEOPTIONS
    }
}

IDI_APPICON ICON "app.ico"
```

3. 创建另一个新文件并插入以下源码，把此文件保存在ENABLE.RH中。此文件定义菜单和菜单项的资源标识符，这些标识符在资源脚本与C源代码中都将被引用到。

```
#ifndef __ENABLE_RH
/*-----------------------------------------------------------------------*/
/*                                                                       */
/* MODULE: ENABLE.RH                                                     */
 * PURPOSE: This include file defines the resource identifiers used.    */
/*                                                                       */
/*-----------------------------------------------------------------------*/
#define __ENABLE_RH

#define IDM_TESTMENU        200
#define CM_ENABLE           102
#define CM_DISABLE          101
#define CM_EXIT             103
#define CM_CONFIGURE        111
#define CM_DIRECTORIES      112
#define CM_SAVEOPTIONS      113

#define IDI_APPICON         202
```

```
#endif
```

4. 在资源脚本中还引用了一个图标文件 APP.ICO。用户可以利用资源编辑器来创建一个标准的 16 色或少于 16 色的 32×32 象素的图标文件。创建后，把图标文件 APP.ICO 保存在前面所创建的工作目录中。

5. 接下来，再创建一个新文件，此文件包含了该例子程序的 C 源代码，把此文件命名为 ENABLE.C。把下面的注释与全局变量添加到此源文件中，这些代码包括所用的 Windows 头文件以及刚刚建立的资源头文件，并且还定义了该例子程序所用窗口类的名字。

```
/*----------------------------------------------------------------------*/
/*                                                                      */
/* MODULE: ENABLE.C                                                     */
/* PURPOSE: This sample application demonstrates how to use the         */
/*          EnableMenuItem function to enable and disable menu items    */
/*          and Menus. It also demonstrates how to respond to the       */
/*          WM_INITMENU message, which signals that the menu is         */
/*          about to be displayed. This is a good time to change        */
/*          menu state depending upon some condition.                   */
/*                                                                      */
/*----------------------------------------------------------------------*/

#define STRICT
#include <windows.h>
#include <winnt.h>
#include "enable.rh"

static char *MainWindowClassName = "EnableMenuWindow";
```

6. 把下面的函数添加到同一源文件 ENABLE.C 中。函数 ChangeOptionsMenuState 用来明确地激活与禁止菜单选项，它是利用 GetMenu 来获取菜单句柄，然后用 EnableMenuItem 来改变菜单的状态。需注意的是，这里必须使用标志 MF_BYPOISTION 而且提供菜单在菜单条上的位置。这是因为弹出式菜单不像菜单项一样有 ID 值。菜单的位置以 0 为初始位置，所以菜单 Options 的位置为 1。

```
/*----------------------------------------------------------------------*/
/* This function gets the handle of the main menu from the window,      */
/* and uses EnableMenuItem to change the state of the Options menu.     */
/* If the command is CM_ENABLE then the menu will be enabled; if it     */
/* is CM_DISABLE, then it will be grayed.                               */
/*----------------------------------------------------------------------*/
void ChangeOptionsMenuState (HWND hWnd, UINT command)
{
    HMENU hMenu;

    // Get a handle to the main menu.
```

```
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    /* Test the value of the command and enable or disable
       the Options menu. Note that we use the position of the
       menu (1) as a popup menu does not have an ID. */
    if (command == CM_ENABLE)
        EnableMenuItem (hMenu, 1, MF_BYPOSITION | MF_ENABLED);
    else
        EnableMenuItem (hMenu, 1, MF_BYPOSITION | MF_GRAYED);

    // Refresh the menu bar.
    DrawMenuBar (hWnd);
```

7. 在菜单 File 中添加激活与变灰菜单项的函数，此函数响应消息 WM_INITMENU 被调用。当用户开始下拉某一菜单时 Windows 就把此消息发送到窗口，由于此消息只被发送一次，所以它会影响所有的菜单，因此用户能沿着菜单条在菜单间移动而不产生更多消息。只有当菜单被关闭以及用户重新选择菜单条时，另外一个 WM_INITMENU 消息才会产生。

此段代码调用 GetMenuState 来返回菜单 Options 的状态，根据这一状态，来激活与变灰菜单 File 上的菜单项 Enable Menu 与菜单项 Disable menu。GetMenuState 返回一组标志。如果返回值是 MF_GRAYED，代码就利用 EnableMenuItem 激活 Enable Menu 选项，而且变灰 Disable menu 选项；如果返回值不是 MF_GRAYED，则变灰 Enable Menu 选项，而激活 Disable menu 选项。

```
/* ------------------------------------------------------------------- */
/* This function uses GetMenuState to test whether the Options menu     */
/* is enabled or grayed. If the menu is enabled, then the Enable        */
/* Menu item needs to be grayed, and the Disable Menu item needs to     */
/* be enabled. If the menu is grayed, then the item states are          */
/* reversed.                                                            */
/* ------------------------------------------------------------------- */
void MenuPulledDown (HWND hWnd)

    HMENU hMenu;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    // Get the Options menu state, and see if it is grayed.
    if (GetMenuState (hMenu, 1, MF_BYPOSITION) & MF_GRAYED)
```

```
    {
        EnableMenuItem (hMenu, CM_ENABLE,
                        MF_BYCOMMAND | MF_ENABLED);
        EnableMenuItem (hMenu, CM_DISABLE,
                        MF_BYCOMMAND | MF_GRAYED);
    }
    else
    {
        EnableMenuItem (hMenu, CM_ENABLE,
                        MF_BYCOMMAND | MF_GRAYED);
        EnableMenuItem (hMenu, CM_DISABLE,
                        MF_BYCOMMAND | MF_ENABLED);
    }
}
```

8. 前面的函数已把一些功能添加到了例子程序中，目前，需要一个窗口程序来处理从Windows 发来的消息并调用那些函数。把下面的代码添加到源文件中，需要注意的是，这段代码是响应 Windows 发送的消息 WM_INITMENU 而做出的反应，在显示菜单之前，调用函数 MenuPulledDown 来激活或变灰菜单 File 中的菜单项。这个窗口程序也处理由菜单产生的 WM_COMMAND 消息，如果是消息 CM_EXIT，则使窗口关闭，并退出例子程序；如果是消息 CM_ENABLE 和 CM_DISABLE，则调用前面所添加的函数 ChangeOptionsMenuState.

```
/* ---------------------------------------------------------------- */
/* This function is the main window callback function which will be */
/* called by Windows to process messages for the window.              */
/* ---------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EXIT:
                    DestroyWindow (hWnd);
                    return 0;
                case CM_ENABLE:
                case CM_DISABLE:
                    ChangeOptionsMenuState (hWnd, (UINT) wParam);
                    return 0;
            }
            break;
```

```
        case WM_INITMENU:
            MenuPulledDown (hWnd);
            return 0;
        case WM_DESTROY:
            PostQuitMessage (0);
            return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

9. 下面列出的三个函数做为一个程序块也要添加到同一源文件 ENABLE.C 中。这三个函数 InitApplication，InitInstance 和 WinMain 提供了本例子程序的框架。当启动例子程序时，WinMain 作为接口程序被 Windows 调用，WinMain 首先检测该例子程序窗口类名是否已被注册，如果没有，则调用函数 InitApplication 来创建和注册窗口类，包括创建例子程序的菜单与图标。WinMain 还调用函数 InitInstance，该函数用来创建主窗口并使应用程序运行。接着 WinMain 开始循环，不断检索消息队列中的消息，并把这些消息送到窗口程序进行处理，直到例子程序结束。

```
/* ------------------------------------------------------------------ */
/* This function initializes a WNDCLASS structure and uses it to      */
/* register a class for our main window.                              */
/* ------------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS   wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ------------------------------------------------------------------ */
/* This function creates an instance of our main window. The window   */
/* is given a class name and a title, and told to display anywhere.   */
```

```
/* The nCmdShow argument passed to the program determines how the          */
/* window will be displayed.                                                */
/* ------------------------------------------------------------------------ */
BOOL InitInstance (HANDLE hInst, int nCmdShow)
{
    HWND hWnd;

    hWnd = CreateWindow (MainWindowClassName," Enable/Gray Menu Demo",
            WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
            CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
            NULL, NULL, hInst, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);            // Send a WM_PAINT message.
    return TRUE;
}

/* ------------------------------------------------------------------------ */
/* The main entry point for Windows applications. Check to see if          */
/* the main window class name has already been registered, if it has       */
/* not, call InitApplication to register it. Call InitInstance to          */
/* create an instance of our main window, then pump messages until         */
/* the application is closed.                                               */
/* ------------------------------------------------------------------------ */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
```

```
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

10. 编译与运行此例子程序

注释

如同例子程序所示，可以利用 EnableMenuItem 来激活或变灰菜单项和子菜单。但要在主菜单下变灰子菜单中的菜单项时，会出现一种特殊情况。如果按照本节中的代码，利用菜单项所在菜单中的位置 0 或 1 来变灰子菜单项时，就不会得到预想的结果，变灰的不是所要求的子菜单，而是其高层的菜单。此时需要首先利用函数 GetSubMenu 来获 取该菜单包含的所要变灰的子菜单的句柄，然后再利用此句柄而非主菜单的句柄来调用函数 EnableMenuItem。

8.2 添加和删除菜单项

问题

有时需要增加或删除应用程序的菜单项。例如，当用户想要定制菜单项时就是如此。那么，如何为菜单增加菜单项，又如何从菜单中删除菜单项呢?

方法

Windows API 提供一些以不同方式操作菜单的函数。本节就介绍如何利用函数 AppendMenu，InsertMenu 以及 DeleteMenu，把菜单项添加到菜单的末尾、在菜单的任一位置上插入菜单项以及从菜单中删除菜单项的操作。

AppendMenu 函数用来在已有的菜单中添加菜单项或子菜单。当利用此函数往菜单中添加菜单项或子菜单时，需要知道该菜单的句柄，把菜单项添加到由 GetMenu 返回的菜单上后，再将此菜单加到菜单条上。往下拉式菜单上添加菜单项时，需通过调用函数 GetSubMenu 首先获得此菜单的句柄。

函数 InsertMenu 的工作方式与 AppendMenu 相同，但它还需要现有的菜单项的 ID 或位置值，新的菜单项将被添加到此菜单项的前面或上面。

最后，函数 DeleteMenu 用于从菜单中移去菜单项或子菜单并且把他们删除。如果想在以后再用某一子菜单，要注意利用函数 RemoveMenu 来代替函数 DeleteMenu，函数 RemoveMenu 只是把菜单项或菜单从他们的父菜单中移去，而不把它删除。

步骤

下面，我们将介绍一个例子程序 ADDITEM.EXE，此例子程序可通过以下步骤得到。运行此例子程序可以看到它有两个菜单：菜单 File 与菜单 Test。菜单 Test 初始时只包括一个菜单项 Empty，这个菜单项在整个应用程序中保持变灰状态，即不可选，它唯一的目的是让用户下拉菜单 Test，以便查看其他菜单项。菜单 File 包括四个菜单项：菜单项 Inserte Item、菜单项 Append Item、菜单项 Exit 以及一个允许从菜单中删除菜单项的子菜单。

从菜单 File 中选择菜单项 Append Item 后，下拉菜单 Test，这时将发现一个称为 Appended 的菜单项被添加到了 菜单 Test 的末尾。如果再次选择菜单项 Append Item，将会看到一个消息框，此消息框用于例子程序检测菜单项是否已被追加上。再次查看一下菜单 File，会发现菜单项 Remove Item 被增亮。菜单项 Remove Item 可打开一个有两个选项 Appended

Item 与 Inserted Item 的子菜单，如果选择选项 Appended Item，标有 Appended 的菜单项就从菜单 Test 中删除。

现在从菜单 File 中选择 Insert Item，再次查看一下菜单 Test，将会发现一个新的菜单项被加到了菜单的顶端，在菜单项 Empty 之上，此菜单项被称为 Inserted。图 8-2 显示了具有菜单项 Appended 与 Inserted 的例子程序 ADDITEM。

根据下面的步骤，建立此应用程序。

1. 为例子程序创建新的目录 ADDITEM。工作时，把例子程序的所有源文件都存放在此目录中。

2. 利用文本编辑器创建此例子程序的资源脚本，把资源脚本文件命名为 ADDITEM.RC，并插入下面的资源脚本代码。除包括此例子程序需要用到的图标外，这段代码定义了测试例子程序时所显示的初始菜单。需注意的是，这段代码在初始状态时除菜单 Test 中虚设的菜单项 Empty 外，所有菜单项都没有被变灰，变灰子菜单项 RemoveItem 的工作是由例子程序自己来处理的。

```
/* ------------------------------------------------------------ */
/*                                                              */
/* MODULE: ADDITEM.RC                                           */
/* PURPOSE: This resource script defines the base menu for this */
/*          example application.                                */
/*                                                              */
/* ------------------------------------------------------------ */

#include <windows.h>
#include " additem.rh"

IDM_TESTMENU MENU
{
    POPUP " &File"
    {
        MENUITEM " &Insert Item", CM_INSERTITEM
        MENUITEM " &Append Item", CM_APPENDITEM
        POPUP " &Remove Item"
        {
            MENUITEM " Inserted Item", CM_DELINSERTED
            MENUITEM " Appended Item", CM_DELAPPENDED
        }
        MENUITEM SEPARATOR
        MENUITEM " E&xit", CM_EXIT
    }
    POPUP " &Test"
    {
        MENUITEM " Empty", CM_DONOTHING, GRAYED
```

```
    }
}

IDI_APPICON ICON " app.ico"
```

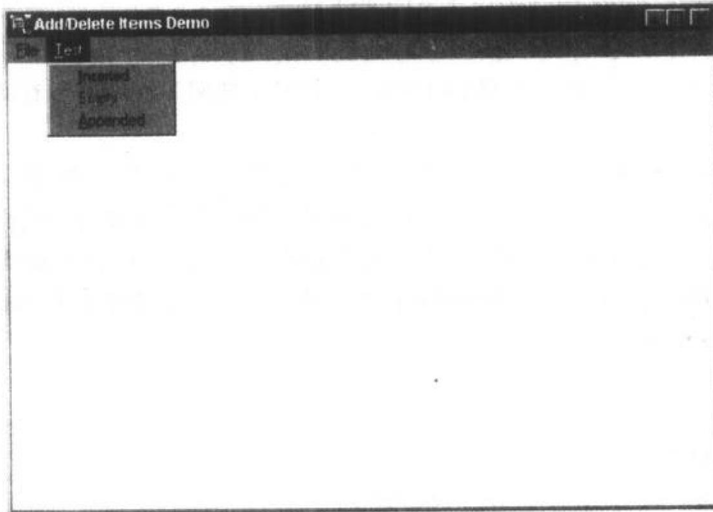


图 8-2 在例子程序 ADDITEM 中的菜单项 Appended 和 Inserted

3. 创建 include 文件，此文件为在资源文件中用到的标识符作了定义。此文件还将被C 源代码用于菜单项的引用，创建 include 文件并插入下面的定义，然后把此文件保存为 ADDITEM.RH 。

```
#ifndef __ADDITEM_RH
/* ---------------------------------------------------------------- */
/*                                                                  */
/* MODULE: ADDITEM.RH                                               */
/* PURPOSE: This include file defines the resource identifiers used. */
/*                                                                  */
/* ---------------------------------------------------------------- */

#define __ADDITEM_RH

#define IDM_TESTMENU        200
#define CM_INSERTITEM       101
#define CM_APPENDITEM       102
#define CM_DELINSERTED      103
#define CM_DELAPPENDED      104
#define CM_EXIT             105

#define CM_DONOTHING        110
```

```
#define CM_APPENDED       120
#define CM_INSERTED       121

#define IDI_APPICON       202

#endif
```

4. 下一步，创建C源代码文件ADDITEM.C。在此文件的顶端，添加下面的注释与代码。在包括例子程序窗口所需的头文件之前，定义预处理符STRICT，用来保证对所有句柄进行严格的类型检查。这样，就能避免一些句柄误用的现象，例如由于偶然的因素把应该传递菜单句柄的地方传递了窗口句柄。这段程序还定义了一个标识符ITEMVALID，此标识符在例子程序的后面将用来判定菜单项是否存在。如果被访问的菜单或菜单项不存在，则调用函数GetMenuState时，返回的值为-1或0xFFFFFFFF。

```
/* ---------------------------------------------------------------- */
/*                                                                  */
/* MODULE: ADDITEM.C                                                */
/* PURPOSE: This sample application demonstrates how to use the     */
/*          InsertMenu and AppendMenu API functions to append items */
/*          to and insert into existing menus. It also shows how to */
/*          use the DeleteMenu API function to delete items, and how */
/*          you can use GetSubMenu to get submenu handles and       */
/*          GetMenuState to find out if an item exists.             */
/*                                                                  */
/* ---------------------------------------------------------------- */

#define STRICT
#include <windows.h>
#include <winnt.h>
#include " additem.rh"

#define ITEMINVALID 0xFFFFFFFF

static char * MainWindowClassName = " AddItemWindow";
HINSTANCE hInstance = NULL;
```

5. 添加下面所列的函数AppendMenuItem。此函数包含的代码其功能在于获取菜单Test的句柄以及利用API函数AppendMenu在菜单的末尾增加一个菜单项，函数AppendMenu需要四个参数，第一个参数是菜单项所要添加到的菜单或子菜单的句柄，如果想要把一个新菜单添加到菜单条上，就需要在该参数中利用函数GetMenu返回的菜单句柄。既然这样，如果想要把菜单项添加到菜单Test中，就需要利用函数GetSubMenu来得到菜单Test的句柄。

第二个参数是为所加的菜单或菜单项指定标志。例如，当为所加的菜单项指定标志为MF_STRING时，则此菜单项将以字符串的方式被显示出来，MF_STRING标志也是缺省的标志。另外，可指定的标志还有MF_BITMAP与MF_POPUP，标志MF_STRING说明所加

的菜单项将以图象的方式被显示，标志 MF _ POPUP 说明所加的将是子菜单。

第三个参数是所加菜单项的 ID 值，如果第二个参数的标志指定的是 MF _ POPUP 的话，此参数将被忽略，因为弹出式菜单没有 ID 值。

最后一个参数是要显示的字符串或位图。在本节介绍的例子中，插入的菜单项被指定为以字符串的方式显示，并利用 & 字符为菜单项生成一个具有下划线的快捷键。

```
/* ---------------------------------------------------------------- */
/* This function uses AppendMenu to add a new item to the second    */
/* menu (the Test menu) . The item is called " Appended" .          */
/* ---------------------------------------------------------------- */
void AppendMenuItem (HWND hWnd)
{
    HMENU hMenu, hTestMenu;

    /* Get a handle to the main menu. Leave the
       function if the main menu does not exist. */
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    /* Get a handle to the test menu. Leave the
       function if the test menu does not exist. */
    hTestMenu = GetSubMenu (hMenu, 1);
    if (! hTestMenu)
        return;

    /* Add an item to the END of the test menu.
       Give it the ID CM _ APPENDED, and the name " Appended" . */
    AppendMenu (hTestMenu, MF _STRING, CM _ APPENDED," &Appended");

    /* Refresh the menu bar. */
    DrawMenuBar (hWnd);
}
```

6. 添加函数 InsertMenuItem 的下列代码，此函数示范了第二个菜单修改函数 Insert-Menu 的用法。利用函数 InsertMenu，只要知道菜单的句柄及其想要插入的位置，或在插入到某菜单项之前该菜单项的 ID 值，便可在此菜单的任何位置插入子菜单或菜单项。在本节例子中，一个称为 Inserted 的菜单项，被添加到菜单 Test 中所有已存在的菜单项之前 0 的位置。在用到的函数 InsertMenu 中，参数 0 用来指定菜单项所插入的位置，标志 MF _ BYPOSITION 用来通知 Windows，0 是代表位置，而不是一个菜单项的 ID 值，CM _ INSERTED 是新菜单项的 ID 值，最后一个参数是菜单项显示时的字符串。

```
/* ---------------------------------------------------------------- */
```

```
/* This function uses InsertMenu to add a new item into the second                 */
/* menu (the Test menu) . The item is called " Inserted" .                          */
/* ------------------------------------------------------------------------------- */
void InsertMenuItem (HWND hWnd)
{
    HMENU hMenu, hTestMenu;

    /* Get a handle to the main menu. Leave the
       function if the main menu does not exist. */
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    /* Get a handle to the test menu. Leave the
       function if the test menu does not exist. */
    hTestMenu = GetSubMenu (hMenu, 1);
    if (! hTestMenu)
        return;

    /* Add an item to the START of the test menu (in front
       of item number 0 by position) .
       Give it the ID CM_INSERTED, and the name " Inserted" . */
    InsertMenu (hTestMenu, 0, MF_BYPOSITION | MF_STRING,
              CM_INSERTED," &Inserted");

    /* Refresh the menu bar. */
    DrawMenuBar (hWnd);
}
```

7. 继续添加下列代码，函数 DeleteMenuItem 说明如何调用 API 函数 DeleteMeun 来删除菜单项。下面的代码中同时利用函数 GetMenuState 来确定所需的菜单项是否存在，函数 GetMenuState 返回的结果通常是描述所指定菜单项的标志的组合（如 MF_GRAYED，MF_ENABLED，MF_CHECKED 等），如果此菜单项不存在，则函数返回－1，在此例子程序中，将用标识符 ITEMINVALID 来表示。

注：利用主菜单，函数 GetMenuState 与 DeleteMenu 都能操作菜单项，这是因为当搜索现有的菜单项时，便自动搜索所指定菜单的全部子菜单。如果要插入或追加新菜单项，或者要对没有 ID 的子菜单进行一定的操作，那么只要指定一个确切的菜单句柄就可以了。在这里，必须指定确切的菜单句柄并传递菜单项的位置而不是它的 ID 值。

```
/* ------------------------------------------------------------------------------- */
/* This function deletes the menu item specified from the test menu                 */
/* ------------------------------------------------------------------------------- */
```

```
void DeleteMenuItem (HWND hWnd, UINT command)
{
    HMENU hMenu;

    /* Get a handle to the main menu. Leave the
       function if the main menu does not exist. */
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    /* See if the Edit menu exists - return if it doesn't. */
    if (GetMenuState (hMenu, command, MF_BYCOMMAND) == ITEMINVALID)
        return;

    /* Delete the menu item */
    DeleteMenu (hMenu, command, MF_BYCOMMAND);

    /* Refresh the menu bar. */
    DrawMenuBar (hWnd);
}
```

8. 把下面的函数 EnableMenuItem 添加到源文件中。此函数响应消息 WM_INITMENU 被调用，在显示菜单前 Windows 将发送此消息。用户每当访问下拉式菜单时，此消息就被处理，此时给出了测试条件的合适时间并根据条件把菜单项设置为核选、取消核选、激活、变灰等状态。本例中，函数 GetMenuState 用于确定在菜单 Test 中是否存在菜单项 Appended 与 Inserted，如果其中的任何一项存在，就激活相应的 Remove Item 子菜单项，否则，就变灰。菜单项的变灰或激活实际上是通过调用 API 函数 EnableMenuItem 来完成的，关于如何激活与变灰菜单项的详细讨论，请参阅 8.1 节。

```
/* ---------------------------------------------------------------- */
/* This function uses GetMenuState to see if the appended and/or    */
/* inserted menu items exist. If one or other exists, the           */
/* corresponding Remove item is enabled, otherwise it is grayed.    */
/* ---------------------------------------------------------------- */
void EnableMenuItems (HWND hWnd)
{
    HMENU hMenu;

    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;
```

```
    /* Test to see if the " Appended" menu item exists.
       If it does, enable the Remove function for it.  */
    if (GetMenuState (hMenu, CM_APPENDED,
                      MF_BYCOMMAND) ! = ITEMINVALID)
        EnableMenuItem (hMenu, CM_DELAPPENDED,
                        MF_BYCOMMAND | MF_ENABLED);
    else
        EnableMenuItem (hMenu, CM_DELAPPENDED,
                        MF_BYCOMMAND | MF_GRAYED);

    /* Test to see if the " Inserted" menu item exists.
       If it does, enable the Remove function for it.  */
    if (GetMenuState (hMenu, CM_INSERTED,
                      MF_BYCOMMAND) ! = ITEMINVALID)
        EnableMenuItem (hMenu, CM_DELINSERTED,
                        MF_BYCOMMAND | MF_ENABLED);
    else
        EnableMenuItem (hMenu, CM_DELINSERTED,
                        MF_BYCOMMAND | MF_GRAYED);
}
```

9. 下面列出的函数 MainWndProc 是该例子程序的主窗口过程。当注册例子程序的主窗口类时，Windows 把它作为回调函数注册，所有的窗口消息（诸如 WN_INITNENU，WM_CREATE，WM_DESTROYD 等）都将发送给此函数，而来自于菜单的所有命令消息（WM_COMMAND 消息诸如 CM_EXIT，CM_APPENDITEM ）也是如此。此函数相应于用户的动作调用函数 AppendMenuItem、InsertMenuItem 以及 DeleteMenuItem，当接收到消息 WM_INITMENU 时，便通过调用函数 EnableMenuItems 来激活相应的菜单项。

```
/* ------------------------------------------------------------------- */
/* This function is the main window callback function which will be    */
/* called by Windows to process messages for the window.               */
/* ------------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EXIT:
                    DestroyWindow (hWnd);
                    return 0;
```

```
            case CM_APPENDITEM:
                AppendMenuItem (hWnd);
                return 0;
            case CM_INSERTITEM:
                InsertMenuItem (hWnd);
                return 0;
            case CM_DELAPPENDED:
                DeleteMenuItem (hWnd, CM_APPENDED);
                return 0;
            case CM_DELINSERTED:
                DeleteMenuItem (hWnd, CM_INSERTED);
                return 0;
        }
        break;
    case WM_INITMENU:
        EnableMenuItems (hWnd);
        return 0;
    case WM_DESTROY:
        PostQuitMessage (0);
        return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

10. 添加下面三个函数与构成源文件的其余部分结合在一起。这三个函数是该例子程序的框架，在本章的所有例子中，几乎不用修改就能利用这些函数。函数 InitApplication 为此例子程序的主窗口注册一个窗口类，并利用标识符 MainWindowClassName 来指定该窗口类的类名，此标识符在源文件的开始部分已被定义过。函数 InitApplication 仅当此名字的窗口类没有被注册时调用，相反，在创建每个主窗口实例时，函数 InitInstance 就被调用一次，因此，如果同时有例子程序的多个复制程序在运行，则函数 InitApplication 只被调用一次，而函数 InitInstance 每次都被调用。函数 InitInstance 用于创建与显示主窗口。

最后，当例子程序开始运行时，函数 WinMain 便被 Windows 调用。此函数首先检测窗口类是否已被注册，然后根据情况来调用函数 InitApplication 与 InitInstance，继而开始循环往复地接收并处理消息。

```
/* ------------------------------------------------------------------- */
/* This function initializes a WNDCLASS structure and uses it to       */
/* register a class for our main window.                               */
/* ------------------------------------------------------------------- */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;
```

```
    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ---------------------------------------------------------------------- */
/* This function creates an instance of our main window. The window      */
/* is given a class name and a title, and told to display anywhere.       */
/* The nCmdShow argument passed to the program determines how the         */
/* window will be displayed.                                              */
/* ---------------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;          // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName," Add/Delete Items Demo",
                        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                        NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);        // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if         */
```

```
/* the main window class name has already been registered, if it has       */
/* not, call InitApplication to register it. Call InitInstance to          */
/* create an instance of our main window, then pump messages until        */
/* the application is closed.                                              */
/* ----------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

11. 编译与运行此例子程序。

注释

用户将上述有关增加、插入、删除菜单项的相同方法可以运用于所有的菜单与子菜单。根据在应用程序中打开的窗口或菜单的配置，可以利用以上介绍的方法把菜单项添置在菜单上。

8.3 为菜单项添加核选标记

问题

如果想通过在菜单项旁添加核选标记来向用户显示当前菜单项的选择状况，则如何为菜单项添加核选标记呢？

方法

当需要在菜单中核选或取消核选任一菜单项时，可利用 API 函数 CheckMenuItem 来完成，另外还可以利用函数 GetMenuState 来测试菜单项是否被核选。本节所讨论的例子程序说明了如何利用以上两个函数在菜单项旁添加与删除核选标记。

此例子程序还介绍了如何“同使用一组单选按钮一样”的方式来使用一组菜单项，即在任一时刻，只能有一个菜单项被选中。要达到这一点是很容易的，我们所需作的只是利用一个简单变量来存放当前被选中的菜单项，而其他的工作是由 Windows 来完成的。

步骤

按照下面的步骤将会生成一个例子程序CHECK.EXE，当运行此例子程序时，会看到它有两个菜单：菜单File与菜单Options。菜单File仅包含一个菜单项Exit，而菜单Options包含三种文本颜色（黑，红，绿）以及一个菜单项Allow Undo。这些选项不像真正的应用程序那样执行一些实际的任务，事实上他们不执行任何工作，而只是为了演示如何核选或取消核选菜单项。

当最初启动此例子程序时，在菜单项Black Text的旁边有一个核选标记，当从菜单中选择了另外颜色时，比如说选择了菜单项Red Test，此时，将会看到核选标记从 菜单项Black Text的旁边删掉而转到菜单项Red Test的旁边，这就是一个在一组菜单项中以单选按钮的方式使用核选标记的例子，也就是说在每一时刻，一组菜单项中只能有一个菜单项被选中。

顺着菜单下移，当选择菜单项Allow Undo时，在此菜单项旁边便会添加一个核选标记，而此时菜单项Red Test旁边的核选标记并不受影响，也没有消失。重新选择菜单项Allow Undo，此时该菜单项旁的核选标记就会消失，这 就是一个开关式处理核选标记的例子。试着把其他颜色的菜单项与菜单项Allow Undo不同结合地进行操作，会发现这两个菜单项确实完全无关。图8-3显示了核选择菜单项Red Test与菜单项Allow Undo时，例子程序的外观形式。

图8-3　在菜单中使用核选标记

下面的步骤将指导如何实现上面所讨论的例子程序。

1. 创建新目录CHECK，利用此目录作为此例子程序的工作目录，所有例子程序的源文件都将存放在此目录中。

2. 接下来，创建此例子程序所用的菜单资源文件。创建文本文件并输入下面资源脚本代码，把此文件保存为CHECK.RC。仔细观察第二个菜单，即菜单Options，可以看到菜单项Black Text有附加格式CHECKED，而其他的菜单没有，CHECKED意味着菜单项Black Text在初始状态时就显示核选标记。

```
/*-----------------------------------------------------------*/
```

```
/*                                                              */
/* MODULE: CHECK.RC                                             */
/* PURPOSE: This resource script defines the base menu for this */
/*          example application.                                */
/*                                                              */
/* ------------------------------------------------------------ */

#include <windows.h>
#include "check.rh"

IDM_TESTMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "E&xit", CM_EXIT
    }
    POPUP "&Options"
    {
        MENUITEM "&Black Text", CM_TEXTBLACK, CHECKED
        MENUITEM "&Red Text", CM_TEXTRED
        MENUITEM "&Green Text", CM_TEXTGREEN
        MENUITEM SEPARATOR
        MENUITEM "&Allow Undo", CM_ALLOWUNDO
    }
}

IDI_APPICON ICON "app.ico"
```

3. 现在创建 include 文件，此文件将定义在菜单中使用的标识符。步骤 2 创建的资源脚本以及 C 源码文件都将用到此 include 文件，这样也保证了在资源脚本中定义的菜单项与例子程序源文件中相应的菜单项有相同的值。为下面的定义创建新文件 CHECK.RH。

```
#ifndef __CHECK_RH
/* ------------------------------------------------------------ */
/*                                                              */
/* MODULE: CHECK.RH                                             */
/* PURPOSE: This include file defines the resource identifiers used. */
/*                                                              */
/* ------------------------------------------------------------ */

#define __CHECK_RH

#define IDM_TESTMENU        200
#define CM_EXIT             101
```

```
#define CM_TEXTBLACK        111
#define CM_TEXTRED          112
#define CM_TEXTGREEN        113
#define CM_ALLOWUNDO        114

#define IDI_APPICON         202

#endif
```

4. 创建C源码文件CHECK.C。运行此例子程序以及控制菜单的代码将全部被放置在此文件中，在此源文件头部插入下面的注释与说明。这里需要注意一下在包含Windows的文件之前标识符STRICT的用法，此标识符表示对句柄进行严格的类型检查，从而使得窗口程序对于不同的句柄要求使用不同的类型。例如，当需要传递HWND类型句柄的地方传递了HMENU类型的句柄时，C编译器便提出警告。

在下面的程序中，需要强调说明的变量是colorGroup，此UINT值将被用来存放当前被选中的菜单项。这里，它将被初始化为CM_TEXTBLACK，因为Black Text菜单项在初始时处于核选状态。

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/* MODULE: CHECK.C                                                 */
/* PURPOSE: This sample application demonstrates how to use the    */
/*          CheckMenuItem API function to check and uncheck menu   */
/*          items, and also demonstrates how you can use a set of  */
/*          menu items like a set of radio buttons, with only one  */
/*          member of the set being checked at any one time.       */
/*                                                                 */
/*-----------------------------------------------------------------*/
#define STRICT
#include <windows.h>
#include <winnt.h>
#include "check.rh"

static char *MainWindowClassName = "CheckItemWindow";
HINSTANCE    hInstance = NULL;

// colorGroup stores the menu ID which has the checkmark.
UINT         colorGroup = CM_TEXTBLACK;
```

5. 添加下面列出的函数ToggleState，此函数用来改变菜单项Allow Undo的核选标记。它调用函数GetMeunState来返回此菜单项的当前状态，接着用逻辑和操作符&来测试此菜单项是否已设置为标志MF_CHECKED，然后根据情况调用API函数CheckMenuItem来改变此菜单项的状态。

函数CheckMenuItem需三个参数，第一个参数是要修改的菜单的句柄，第二个参数是要

修改的菜单项的ID值或此菜单项在菜单中以0为基点的位置值，最后，是标志参数，它用来指出第二个参数是ID值（MF _ COMMAND），还是位置值（MF _ BYPOISTION）；此参数还用来说明此菜单项应该被核选（MF _ CHECKED），还是应该被取消核选（MF _ UNCHECKED）。

```
/* ----------------------------------------------------------------------- */
/* This function can be used to turn a checkmark on or off for any         */
/* single menu item, based on its previous state.                           */
/* ----------------------------------------------------------------------- */
void ToggleState (HWND hWnd, UINT command)
{
    HMENU hMenu;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    /* Look at the state of the menu item. If it is already
       checked, uncheck it, otherwise check it. */
    if (GetMenuState (hMenu, command, MF_BYCOMMAND) & MF_CHECKED)
        CheckMenuItem (hMenu, command, MF_BYCOMMAND | MF_UNCHECKED);
    else
        CheckMenuItem (hMenu, command, MF_BYCOMMAND | MF_CHECKED);
}
```

6. 把下面的函数添加到源文件中。当用户以单选按钮的方式从一组菜单项中选择某项时，窗口过程就调用函数CheckGroupItem。函数CheckGroupItem需要接收两个变量，一个是用于存放新选菜单项的变量command，另一个是用于存放当前所选的菜单项的变量variale。此函数首先通过调用函数CheckMenuItem（利用标志MF _ UNCHECKED）来删除现有菜单项旁边的核选标记，然后再次通过调用函数CheckMenuItem（利用标志MF _ CHECKED）为新选的菜单项添加核选标记，最后，把新选菜单项的ID值存放在变量variable中。

```
/* ----------------------------------------------------------------------- */
/* This function checks one menu item in a group, and unchecks the         */
/* other items in the group. The function uses a single variable to        */
/* store the ID of the currently checked menu item. You determine          */
/* which items belong to the group programmatically; by calling this       */
/* function for any item which you want to participate.                    */
/* ----------------------------------------------------------------------- */
void CheckGroupItem (HWND hWnd, UINT command, UINT *variable)
{
    HMENU hMenu;
```

```
    // Nothing to do if this is the same item.
    if (command == *variable)
        return;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    // Uncheck the previous command.
    CheckMenuItem (hMenu, *variable, MF_BYCOMMAND | MF_UNCHECKED);

    // Check the new command, and store the result.
    CheckMenuItem (hMenu, command, MF_BYCOMMAND | MF_CHECKED);
    *variable = command;
}
```

7. 下面的代码实现一个回调函数。只要有消息要求处理时，Windows 便会调用此回调函数。此函数响应消息 WM_COMMAND，处理菜单的选取；当窗口关闭时，响应消息 WM_DESTROY，关闭此例子程序；响应菜单项 Allow Undo 的选取（CM_ALLOWUNDO），该函数调用前面键入的函数 ToggleState；响应文本颜色的选取（CM_TEXTBLACK，CM_TEXTRED，CM_TEXTGREEN），该函数通过调用函数 CheckGroupItem 来处理把核选标记从一个菜单项移到另 一个菜单项。

```
/*---------------------------------------------------------------------*/
/* This function is the main window callback function which will be   */
/* called by Windows to process messages for the window.               */
/*---------------------------------------------------------------------*/
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EXIT:
                    DestroyWindow (hWnd);
                    return 0;
                case CM_TEXTBLACK:
                case CM_TEXTRED:
                case CM_TEXTGREEN:
                    CheckGroupItem (hWnd, (UINT) wParam, &colorGroup);
                    return 0;
```

```
            case CM_ALLOWUNDO:
                ToggleState (hWnd, (UINT) wParam);
                return 0;
        }
        break;
    case WM_DESTROY:
        PostQuitMessage (0);
        return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

8. 最后，把下面三个函数的代码添加到源文件中。当窗口类没有被注册时，函数 InitApplication 用来注册窗口类。对于不同的应用程序，需要利用不同的窗口类，这是因为不同的应用程序需要用到不同的菜单与窗口过程。当创建与显示应用程序的主窗口时，调用函数 InitInstance。函数 WinMain 调用上面的两个函数，它是此例子程序的主入口点，除调用其他函数来创建此例子程序外，函数 WinMain 也执行 GetMessage、TranslateMessage 以及 DispatchMessage 等过程使应用程序接收与处理消息。

```
/* ------------------------------------------------------------------------ */
/* This function initializes a WNDCLASS structure and uses it to            */
/* register a class for our main window.                                    */
/* ------------------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ------------------------------------------------------------------------ */
/* This function creates an instance of our main window. The window         */
/* is given a class name and a title, and told to display anywhere.         */
```

```
/* The nCmdShow argument passed to the program determines how the          */
/* window will be displayed.                                               */
/* ----------------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;         // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName," Check Menu Items Demo",
             WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
             CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
             NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);        // Send a WM_PAINT message.
    return TRUE;
}

/* ----------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if          */
/* the main window class name has already been registered, if it has       */
/* not, call InitApplication to register it. Call InitInstance to          */
/* create an instance of our main window, then pump messages until         */
/* the application is closed.                                              */
/* ----------------------------------------------------------------------- */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
```

```
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

9. 编译与运行此例子程序前，需要添加在资源脚本中引用到的应用程序图标文件 APP.ICO。利用资源编辑器来创建此 32×32 象素的图标文件。

注释

利用菜单项旁的核选标记向用户提供当前菜单选择状态的反馈消息是一种被广泛采用的方法。显然，在诸如 Open、Close 、Save 以及 Print 等具有较强动作性的命令式菜单项旁添加核选标记是不合适的，而在诸如 Ruler、Show Toolbars、Allow Undo 以及其他一些有关设置的选项或描述某种状态的菜单项旁使用核选标记却十分有意义。

8.4 利用自己定义的核选标记

问题

有时候，需要利用在某种意义上更适合于某菜单项的小图形，放在此菜单项的旁边做为核选标记，以便使用户更清楚地了解此选项的状态及用途。例如：在颜色菜单项的旁边显示颜色标记，在其他菜单项旁使用不同的符号。那么，如何来改变核选标记，使得当菜单项被选中时，在此菜单项旁显示的是自己所要的图形而非缺省的核选标记呢?

方法

在 8-3 节中，介绍了如何在菜单中使用缺省的核选标记，程序员可以利用核选标记来表明某菜单项是开着的还是关闭的（即处于选中状态还是处于非选中状态），也可以在一组选项中以单选按钮的方式在菜单中使用核选标记。Windows 实际上对每个菜单项都使用了两个核选标记位图，一个是菜单项的状态被设置为 MF_CHECKED 时所显示的位图（标准的核选标记），另一个是菜单项取消核选时所显示的位图，缺省时，取消核选时的位图显示出的图形是空白。

利用函数 SetMenuItemBitmaps，可以改变菜单中任一菜单项的其中一个或两个核选标记位图，此函数需用到显示菜单项时的两个位图句柄（HBITMAP)。如果把两个位图句柄中的任意一个设置为 NULL，则会得到其相应的缺省的 Windows 位图（一个是未被核选时的空白图象，一个是被核选时标准的核选标记）。在改变菜单项核选标记的位图时，还需考虑另外几个问题。

第一个问题是此位图应该有多大? Windows API 文档并不清楚位图的大小，其原因在于在绘菜单之前不知道位图到底有多大。菜单项的大小是由系统所用屏幕的分辨率以及用户为菜单选用的字体来决定的。通过调用 API 函数 GetSystemMetrics 传递 SM_CXMENUCHECK 来获取菜单核选标记位图的宽度，传递 SM_CYMENUCHECK 来获取其高度，从而确定出位图应有的大小。

第二个问题是此图象应是何种颜色？显然，核选标记的前景颜色是由应用程序的设计者自己决定的，而菜单的背景却是能被用户通过屏幕属性单来改变的。一些用户可能习惯用 Windows 3.1 中的白色做背景，另外一些用户可能更喜欢用 Windows 95 中的灰色，还有一些用户可能喜欢使用他们自己设置的一些混合颜色做背景。因此，做为应用程序的设计者来说，

是无法确定选择何种颜色模式来定制核选标记会与用户选择的背景颜色能较好搭配的，但至少可以保证图象的背景颜色是同菜单的背景颜色相同的。通过调用函数 GetSysColor，利用 COLOR _MENU 变量来获取菜单背景的 RGB 值，然后在显示前改变位图的颜色表，以保证与背景颜色的匹配。本节所介绍的例子程序利用了设备无关位图，以便能直接地访问位图的颜色表。关于修改位图颜色表的一些技巧，请参阅 4.2 节。

本节讨论的例子将执行上述的所有操作。

步骤

如果阅读了 8.3 节“如何为菜单项添加核选标记”，便会发现本节中讨论的例子与 8.3 节的例子是同一应用程序。按下面的步骤生成的例子程序 CHECKBMP.EXE 运行时产生两个菜单，一个是仅包含菜单项 Exit 的菜单 File，另一个是包含三个文本颜色选项（Black text，Red Text ，Green Text）与一个 Allow Undo 选项的菜单 Options 。在前面的例子程序中，菜单项 Black Text 初始时是被核选的，所有其他的菜单项是未被核选的。在本例子程序中，在菜单项 Black Text 的左边，有一个小的黑方块，而在其他颜色的菜单项旁没有核选标记，在菜单项 Allow Undo 旁有一个中间有斜线的圆圈，即国际符号“No”的标记。如图 8-4 所示。

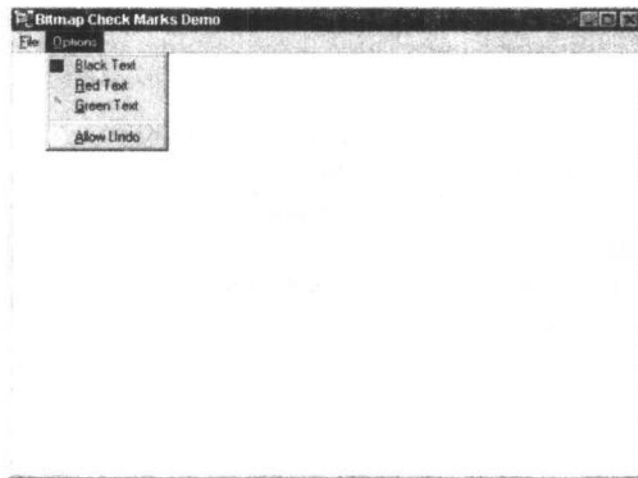


图 8-4　在菜单中使用定义核选标记

从菜单 Options 中选择另一种颜色项，如果选择了菜单项 Red Text，一个红方块将在此菜单项的旁边出现，而菜单项 Black Text 旁的黑方块消失；同样地，选择了菜单项 Green Text 时，在菜单项 Green Text 旁会出现一个绿方块而红方块将被删除。

当选择菜单项 Allow Undo 时，圆形的“No”符号消失取而代之的是一个绿色的核选标记。

按照下面的步骤，便能实现上述的功能。

1. 创建目录并把它命名为 CHECKBMP，　把这个目录做为此项目文件的工作目录。

2. 创建文件 CHECKBMP.RC，输入下面的资源脚本。如果阅读过 8.3 节，会发现此资源脚本与 8.3 节的资源脚本基本相同。菜单结构是完全相同的，唯一添加上去的是用于生成核选标记的位图。这里，共有五个位图，分别为：文本颜色选项所需的三个彩色方块、菜单

项 Allow Undo 所需的一个红“No”符号以及此菜单项被核选时用到的绿色核选标记。

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/* MODULE: CHECKBMP.RC                                                */
/* PURPOSE: This resource script defines the base menu for this       */
/*          example application.                                      */
/*                                                                    */
/* ------------------------------------------------------------------ */

#include <windows.h>
#include "checkbmp.rh"

IDM_TESTMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "E&xit", CM_EXIT
    }
    POPUP "&Options"
    {
        MENUITEM "&Black Text", CM_TEXTBLACK, CHECKED
        MENUITEM "&Red Text", CM_TEXTRED
        MENUITEM "&Green Text", CM_TEXTGREEN
        MENUITEM SEPARATOR
        MENUITEM "&Allow Undo", CM_ALLOWUNDO
    }
}

/* Application Icon */
IDI_APPICON ICON "app.ico"

/* Menu Checkmark Bitmaps */
IDBM_BLACKBOX BITMAP "black.bmp"
IDBM_REDBOX BITMAP "red.bmp"
IDBM_GREENBOX BITMAP "green.bmp"
IDBM_CHECK BITMAP "check.bmp"
IDBM_UNCHECK BITMAP "uncheck.bmp"
```

3. 在此资源脚本编译之前，需要创建五个位图与应用程序图标。通过资源编辑器来创建 24×24 象素、16 种颜色的位图与应用程序图标，并把他们保存到如表 8-2 所列的文件中。

表 8-2 本例子程序的位图与图标文件名

文件名	内容
APP.ICO	应用程序图标

（续）

文件名	内容
BLACK.BMP	黑方块位图
RED.BMP	红方块位图
GREEN.BMP	绿方块位图
CHECK.BMP	绿核选标记位图
UNCHECK.BMP	红色"No"符号位图

4. 现创建include文件，此文件定义了一些标识符。利用这些标识符，可用来装入位图与引用菜单项。把此文件保存为CHECKBMP.RH，它将被资源脚本与C源文件共同享用。CHECKBMP.RH的内容如下：

```
#ifndef __CHECKBMP_RH
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* MODULE: CHECKBMP.RH                                                     */
/* PURPOSE: This include file defines the resource identifiers used.       */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#define __CHECKBMP_RH

#define IDM_TESTMENU        200
#define CM_EXIT             101

#define CM_TEXTBLACK        111
#define CM_TEXTRED          112
#define CM_TEXTGREEN        113
#define CM_ALLOWUNDO        114

#define IDBM_BLACKBOX       301
#define IDBM_REDBOX         302
#define IDBM_GREENBOX       303
#define IDBM_CHECK          304
#define IDBM_UNCHECK        305

#define IDI_APPICON         202

#endif
```

5. 现创建C源文件，并学习如何替换核选标记。创建文件CHECKBMP.C，并在其中添加下面的注释与说明。利用预处理符STRICT的定义，使得在句柄或其他数据类型发生混用时，C编译器提出警告。例如，当向Windows函数传递参数时，如果在要求HMENU类型的变量中传递了HWND类型的参数，编译器便会提出警告。在程序中，RGBLIGHTGRAY被

定义为存放浅灰色的RGB值，它用来把核选标记位图的背景色替换为菜单条的背景色。变量colorGroup用于存放当前所选文本颜色菜单项的ID值（初始时是Black Text），而各个位图句柄用于装入与删除核选标记位图。

```
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: CHECKBMP.C                                                     */
/* PURPOSE: This example application demonstrates the use of the         */
/*          GetMenuCheckMarkDimensions and SetMenuItemBitmaps API         */
/*          functions to set the checked and unchecked bitmaps for        */
/*          selected menu items. It also demonstrates how to load         */
/*          bitmaps as device independant bitmaps, and change the         */
/*          colors of the bitmap to match the background before          */
/*          display.                                                      */
/*                                                                        */
/* ---------------------------------------------------------------------- */

#define STRICT
#include <windows.h>
#include <winnt.h>
#include " checkbmp.rh"

static char     *MainWindowClassName = " BitmapCheckWindow";
HINSTANCE          hInstance = NULL;

// colorGroup stores the menu ID which has the checkmark.
UINT            colorGroup = CM_TEXTBLACK;
static RGBQUAD  RGBLIGHTGRAY = { 192, 192, 192, 0 };

// The following bitmap handles store the checkmarks.
HBITMAP         blackBox = NULL;
HBITMAP         redBox = NULL;
HBITMAP         greenBox = NULL;
HBITMAP         checkOn = NULL;
HBITMAP         checkOff = NULL;
```

6. 把下面的程序添加到已创建的源文件中。函数LoadBitmapResource代替API函数LoadBitmap用于从资源中检索位图，此函数是从第四章的例子中摘取过来的，它被用于装入与设备无关的位图，而不像函数LoadBitmap那样装入与设备有关的位图。这些位图都包含一个可改变的颜色表，以便修改位图的背景色。

```
/* ---------------------------------------------------------------------- */
/* This function uses LoadResource to get a bitmap resource as a DIB;     */
/* it takes a copy of the DIB and returns a pointer to the copy.          */
/* Free the pointer with LocalFree ( (HLOCAL) pointer) when done.         */
```

```
/* ---------------------------------------------------------------- */
BITMAPINFO *LoadBitmapResource (HINSTANCE hInstance, WORD resId)
{
    HRSRC        hResource;
    HGLOBAL      hDib;
    int          nSize, numEntries;
    BITMAPINFO *pResourceDIB, *pCopyDIB;

    if ( ( (hResource = FindResource (hInstance, MAKEINTRESOURCE (resId),
                                      RT_BITMAP)) ! = NULL) &&
        ( (hDib = LoadResource (hInstance, hResource)) ! = NULL))
    {
        pResourceDIB = (LPBITMAPINFO) LockResource (hDib);
        if ( (numEntries = pResourceDIB->bmiHeader.biClrUsed) == 0)
            numEntries = 1L << pResourceDIB->bmiHeader.biBitCount;
        nSize = pResourceDIB->bmiHeader.biSize +
                (numEntries * sizeof (RGBQUAD)) +
                pResourceDIB->bmiHeader.biSizeImage;
        if ( (pCopyDIB = (BITMAPINFO NEAR *) LocalAlloc (LMEM_FIXED,
                                              nSize)) ! = NULL)
        {
            CopyMemory (pCopyDIB, pResourceDIB, nSize);
            return pCopyDIB;
        }
    }
    return NULL;
}
```

7. 下面的函数也是从4.2节的颜色例子中摘取来的，此函数被用于扫描设备无关位图的颜色表（如利用上述的函数 LoadBitmapResource 装入的颜色表），并且把一种颜色的所有具体值改变为另一种颜色的所有具体值。

```
/* ---------------------------------------------------------------- */
/* This function works on the color table for a DIB. It changes all  */
/* the entries in the table for fromColor to toColor. Do not use     */
/* this function on the DIB in the resource itself, as resources are */
/* read-only. Use a copy of the resource DIB.                        */
/* ---------------------------------------------------------------- */

void ChangeColor (BITMAPINFO *info, RGBQUAD fromColor, RGBQUAD toColor)
{
    int i, numEntries;

    if ( (numEntries = info->bmiHeader.biClrUsed) == 0)
```

```
    numEntries = 1L << info->bmiHeader.biBitCount;

for (i = 0; i < numEntries; i++)
    if ( (info->bmiColors [i] .rgbRed == fromColor.rgbRed) &&
         (info->bmiColors [i] .rgbGreen == fromColor.rgbGreen) &&
         (info->bmiColors [i] .rgbBlue == fromColor.rgbBlue))
    {
        info->bmiColors [i] .rgbRed = toColor.rgbRed;
        info->bmiColors [i] .rgbGreen = toColor.rgbGreen;
        info->bmiColors [i] .rgbBlue = toColor.rgbBlue;
    }
}
```

8. 第三个函数取自于4.2节，此函数把设备无关位图转化为设备有关位图。在该节中，原始函数是实现一个逻辑调色板，在变换之前，用来使位图与设备描述表相匹配，这就保证了位图中位的正确性，通过与调色板的运用，可以显示256种颜色的位图。而在本函数中，调色板代码已被删除，这是因为我们不必去画位图，Windows将会利用缺省的系统调色板来显示菜单中的核选标记。

把下面的代码添加到源文件中。这段程序生成与窗口DC相兼容的储存器DC，然后创建一个合适大小的位图，并把此位图放入DC中，最后，此函数利用函数StretchDIBits把DIB（设备无关位图）拷贝到DC（成为新的位图），然后根据需要对图象进行缩放。

```
/* ------------------------------------------------------------------------- */
/* This function converts the DIB supplied as the info argument into          */
/* a DDB ready for display in the given window. It returns the                */
/* HBITMAP handle of the bitmap.                                              */
/* ------------------------------------------------------------------------- */
HBITMAP ConvertDIBToDDB (HWND hWnd, BITMAPINFO *info,
                         int width, int height)
{
    HDC      hDC, hMemDC;
    HBITMAP hBitmap, hOldBitmap;
    int      numEntries, bytesToSkip;
    void     *bits;

    /* Work out where the bitmap data itself starts. Point
       bits to this location. */
    if ( (numEntries = info->bmiHeader.biClrUsed) == 0)
        numEntries = 1L << info->bmiHeader.biBitCount;
    bytesToSkip = info->bmiHeader.biSize +
                        (numEntries * sizeof (RGBQUAD));
    bits = (LPCSTR) info + bytesToSkip;

    /* Get a handle to a device context */
```

```
    hDC = GetDC (hWnd);

    /* Create a memory DC and a bitmap of the right size.
       Select the bitmap into the DC, then use
       StretchDIBits to convert the DIB into the bitmap. */
    if ( (hMemDC = CreateCompatibleDC (hDC)) ! = NULL)
    {
        if ( (hBitmap = CreateCompatibleBitmap (hDC,
                                    width, height)) ! = NULL)
        {
            hOldBitmap = SelectObject (hMemDC, hBitmap);
            StretchDIBits (hMemDC,
                           0, 0, width, height,
                           0, 0, info->bmiHeader.biHeight,
                           info->bmiHeader.biWidth,
                           bits, info, DIB_RGB_COLORS, SRCCOPY);
            SelectObject (hMemDC, hOldBitmap);
        }
        DeleteDC (hMemDC);
    }
    ReleaseDC (hWnd, hDC);

    return hBitmap;
}
```

9. 把下面的函数添加到源文件中。函数 LoadCheckMark 执行如下几步：从资源中装入位图，改变背景颜色使之与菜单匹配，把图象缩放为适当的大小，转换为用于显示的设备相关位图。代码中调用 API 函数 GetSystemMetrics 来查找菜单项需要的宽度（利用 SM_CXMENUCHECK）和高度（利用 SM_CYMENUCHECK）；调用函数 GetSysColor (COLOR_MENU) 来返回菜单背景的 RGB 值，然后装入位图资源并且通过调用函数 ChangeColor 把资源中的浅灰颜色转化为菜单的颜色，最后，调用前面添加的函数 ConvertDIBToDDB 来转换位图，并把它缩放为合适的大小。函数 LoadCheckMark 的返回值为已建成的核选标记位图的句柄。

```
/* ---------------------------------------------------------- */
/* This function loads a single bitmap from the resource file, for   */
/* use as a menu checkmark. The bitmap is loaded as a Device         */
/* Independent bitmap (DIB) and the light gray color replaced with   */
/* the menu background color (COLOR_MENU) before conversion to a     */
/* bitmap and stretching to the correct dimensions.                  */
/* ---------------------------------------------------------- */
HBITMAP LoadCheckMark (HINSTANCE hInstance, HWND hWnd, WORD resourceId)
{
    BITMAPINFO      *pDIB;
```

```
    HBITMAP         hBitmap;
    int             width, height;
    LONG            dimensions;
    RGBQUAD         menuColor;
    DWORD           mColor;

    // Find out the correct size for menu checkmarks.
    dimensions = GetMenuCheckMarkDimensions ();
    width = LOWORD (dimensions);
    height = HIWORD (dimensions);

    // Load a DIB from the resource.
    if ( (pDIB = LoadBitmapResource (hInstance, resourceId)) ! = NULL)
    {
        // Get the menu color, and convert to a RGBQUAD.
        mColor = GetSysColor (COLOR_MENU);
        menuColor.rgbRed = LOBYTE (LOWORD (mColor));
        menuColor.rgbGreen = HIBYTE (LOWORD (mColor));
        menuColor.rgbBlue = LOBYTE (HIWORD (mColor));
        menuColor.rgbReserved = 0;
        // Change the background to menu color.
        ChangeColor (pDIB, RGBLIGHTGRAY, menuColor);

        // Convert to a DDB and scale to the correct size.
        hBitmap = ConvertDIBToDDB (hWnd, pDIB, width, height);
        LocalFree ( (HLOCAL) pDIB);
        return hBitmap;
    }
    return NULL;
}
```

10. 如下所示的函数 SetupCheckMarks 通过调用函数 LoadCheckMark 为此例子程序装入每个核选标记。函数 LoadCheckMark 根据需要重新设置每个位图的大小和颜色，所以此函数所要做的只是存储函数 LoadCheckMar 返回的句柄，并利用函数 SetMenuItemBitmaps 为每个菜单项设置位图。核选标记位图要设置在每个菜单项上，也就是说不能把单个核选标记赋给整个子菜单来使用。在调用函数 SetMenuItemBitmaps 设置菜单项的核选标记时，利用了菜单项的 ID 值，因此在每个调用的标志中都指定了 MF_BYCOMMAND。值得注意的是，在最先的三次调用中，在位图句柄之前有一个参数 NULL，此处的 NULL 是当菜单项未被核选时所用的位图句柄，利用 NULL 做为句柄是为了通知 Windows 利用缺省的图象做为核选标记，在这里，缺省图象为空白。最后一次调用函数 SetMenuItemBitmaps 时，传递了两个位图句柄：一个是菜单项未被核选时的“No”符号，另一个是被核选时的绿色核选标记。

```
/* ------------------------------------------------------------------------ */
/* This function loads the bitmaps for each checkmark. The bitmap           */
```

```
/* handles are stored in variables global to this instance, because     */
/* you need to free them before the window closes.                       */
/* -------------------------------------------------------------------- */
void SetupCheckMarks (HINSTANCE hInstance, HWND hWnd)
{
    HMENU hMenu;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    // Load the bitmap for the black text.
    blackBox = LoadCheckMark (hInstance, hWnd, IDBM_BLACKBOX);
    if (blackBox)
        SetMenuItemBitmaps (hMenu, CM_TEXTBLACK, MF_BYCOMMAND,
                            NULL, blackBox);

    // Load the bitmap for the red text.
    redBox = LoadCheckMark (hInstance, hWnd, IDBM_REDBOX);
    if (redBox)
        SetMenuItemBitmaps (hMenu, CM_TEXTRED, MF_BYCOMMAND,
                            NULL, redBox);

    // Load the bitmap for the green text.
    greenBox = LoadCheckMark (hInstance, hWnd, IDBM_GREENBOX);
    if (greenBox)
        SetMenuItemBitmaps (hMenu, CM_TEXTGREEN, MF_BYCOMMAND,
                            NULL, greenBox);

    // Load the two bitmaps for the Allow Undo checkmarks.
    checkOn = LoadCheckMark (hInstance, hWnd, IDBM_CHECK);
    checkOff = LoadCheckMark (hInstance, hWnd, IDBM_UNCHECK);
    if ( (checkOn) && (checkOff))
        SetMenuItemBitmaps (hMenu, CM_ALLOWUNDO, MF_BYCOMMAND,
                            checkOff, checkOn);
}
```

11. 把下列代码添加到源文件中。函数 RemoveCheckMarks 用于把菜单项的核选标记改回到系统的缺省状态，并且释放位图所占用的内存空间。当关闭应用程序或用户改变屏幕属性时，函数 RemoveCheckMarks 便被调用，因为此时菜单项的背景色或字体大小都可能改变。函数 RemoveCheckMarks 为每个要修改的菜单项调用函数 SetMenuItemBitmaps，并且为每个位图句柄传递 NULL 值，这就使得核选标记返回到系统缺省的核选标记，最后，通过调用

函数 DeleteObject 释放每个位图，并把相应句柄设置为 NULL。

```
/* ---------------------------------------------------------------- */
/* This function removes the user-defined checkmarks by setting     */
/* the menus back to the Windows defaults. It then destroys the     */
/* bitmaps and sets the handles to NULL.                            */
/* ---------------------------------------------------------------- */
void RemoveCheckMarks (HWND hWnd)
{
    HMENU hMenu;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    // Release the bitmap for the black text.
    if (blackBox)
    {
        SetMenuItemBitmaps (hMenu, CM_TEXTBLACK, MF_BYCOMMAND,
                            NULL, NULL);
        DeleteObject (blackBox);
        blackBox = NULL;
    }

    // Release the bitmap for the red text.
    if (redBox)
    {
        SetMenuItemBitmaps (hMenu, CM_TEXTRED, MF_BYCOMMAND,
                            NULL, NULL);
        DeleteObject (redBox);
        redBox = NULL;
    }

    // Release the bitmap for the green text.
    if (greenBox)
    {
        SetMenuItemBitmaps (hMenu, CM_TEXTGREEN, MF_BYCOMMAND,
                            NULL, NULL);
        DeleteObject (greenBox);
        greenBox = NULL;
    }

    if ( (checkOn) || (checkOff))
```

```
    {
        SetMenuItemBitmaps (hMenu, CM_ALLOWUNDO, MF_BYCOMMAND,
                             NULL, NULL);
        if (checkOn)
            DeleteObject (checkOn);
        if (checkOff)
            DeleteObject (checkOn);
        checkOn = checkOff = NULL;
    }
}
```

12. 现在，添加用于打开与关闭核选标记的函数。如果读者已经阅读过8.3节，便会发现这些函数与8.3节中所用的函数完全相同，即根据用户的开关打开或关闭核选标记，这些函数不是用来专门处理定制的核选标记，只要是利用函数SetMenuItemBitmaps设置的核选标记，Windows就会处理他们。把下面两个函数添加到源文件中，其中函数ToggleState用来根据需要打开或关闭菜单项Allow Undo的核选标记；函数CheckGroupItem用于从文本颜色菜单项中核选某菜单项，并删除先前所选定的菜单项的核选标记，从而使得一组菜单项具有如同单选按钮那样的工作方式。

```
/* ------------------------------------------------------------------- */
/* This function can be used to turn a checkmark on or off for any     */
/* single menu item, based on its previous state.                      */
/* ------------------------------------------------------------------- */
void ToggleState (HWND hWnd, UINT command)
{
    HMENU hMenu;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    /* Look at the state of the menu item. If it is already
       checked, uncheck it, otherwise check it. */
    if (GetMenuState (hMenu, command, MF_BYCOMMAND) & MF_CHECKED)
        CheckMenuItem (hMenu, command, MF_BYCOMMAND | MF_UNCHECKED);
    else
        CheckMenuItem (hMenu, command, MF_BYCOMMAND | MF_CHECKED);
}

/* ------------------------------------------------------------------- */
/* This function checks one menu item in a group, and unchecks the     */
/* other items in the group. The function uses a single variable to   */
/* store the ID of the currently checked menu item. You determine     */
```

```
/* which items belong to the group programmatically; by calling this          */
/* function for any item which you want to participate.                       */
/* -------------------------------------------------------------------------- */
void CheckGroupItem (HWND hWnd, UINT command, UINT *variable)
{
    HMENU hMenu;

    // Nothing to do if this is the same item.
    if (command == *variable)
        return;

    // Get a handle to the main menu.
    hMenu = GetMenu (hWnd);
    if (! hMenu)
        return;

    // Uncheck the previous command.
    CheckMenuItem (hMenu, *variable, MF_BYCOMMAND | MF_UNCHECKED);

    // Check the new command, and store the result.
    CheckMenuItem (hMenu, command, MF_BYCOMMAND | MF_CHECKED);
    *variable = command;
}
```

13. 下面的函数是此例子程序的主窗口过程。每当主窗口处理消息时，便会调用此函数，此函数响应若干窗口消息，并响应用户选择菜单项时所产生的消息 WM_COMMAND。响应创建窗口时发出的消息 WM_CREATE，此函数调用函数 SetupCheckMarks；响应消息 WM_DESTROY，此函数调用函数 RemoveCheckMarks。当例子程序接收到消息 WM_SYSCOLORCHANGE 时，特殊情况就会发生，此消息表明用户已修改了屏幕的属性，可能改变了分辨率、颜色或菜单上的字体等。为了保证菜单行为的一致性，该函数首先调用函数 RemoveCheckMarks 删除核选标记，然后调用函数 SetupCheckMarks 根据新的设置来重新创建核选标记位图。把下面的代码添加到同一源文件 CHECKBMP.C 中。

```
/* -------------------------------------------------------------------------- */
/* This function is the main window callback function which will be           */
/* called by Windows to process messages for the window.                      */
/* -------------------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
            SetupCheckMarks (hInstance, hWnd);
```

```
            return 0;
        case WM_SYSCOLORCHANGE:
            RemoveCheckMarks (hWnd);
            SetupCheckMarks (hInstance, hWnd);
            return 0;
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EXIT:
                    DestroyWindow (hWnd);
                    return 0;
                case CM_TEXTBLACK:
                case CM_TEXTRED:
                case CM_TEXTGREEN:
                    CheckGroupItem (hWnd, (UINT) wParam, &colorGroup);
                    return 0;
                case CM_ALLOWUNDO:
                    ToggleState (hWnd, (UINT) wParam);
                    return 0;
            }
            break;
        case WM_DESTROY:
            RemoveCheckMarks (hWnd);
            PostQuitMessage (0);
            return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

14. 最后，把下面三个函数添加到源文件 CHECKBMP.C 中。这三个函数提供了例子程序的框架。函数 InitApplication 用于为此例子程序注册唯一名字的窗口类，只有在该窗口类不存在时，此函数才被调用，函数 InitInstance 用来为例子程序创建和显示主窗口，函数 WinMain 执行这两个函数并不断收集消息使例子程序能正常运行。

```
/* ------------------------------------------------------------------------- */
/* This function initializes a WNDCLASS structure and uses it to             */
/* register a class for our main window.                                     */
/* ------------------------------------------------------------------------- */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
```

```
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ---------------------------------------------------------------------- */
/* This function creates an instance of our main window. The window       */
/* is given a class name and a title, and told to display anywhere.       */
/* The nCmdShow argument passed to the program determines how the         */
/* window will be displayed.                                               */
/* ---------------------------------------------------------------------- */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;        // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName," Bitmap Check Marks Demo",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInstance, NULL);
    if (! hWnd)
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);      // Send a WM_PAINT message.
    return TRUE;
}

/* ---------------------------------------------------------------------- */
/* The main entry point for Windows applications. Check to see if         */
/* the main window class name has already been registered, if it has      */
/* not, call InitApplication to register it. Call InitInstance to          */
/* create an instance of our main window, then pump messages until        */
/* the application is closed.                                              */
/* ---------------------------------------------------------------------- */
```

```
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

15. 编译并运行此例子程序。

注释

本节所讨论的例子比本章其他部分的例子较复杂一些，但增加的大部分复杂性与函数 SetMenuItemBitmaps 和 GetSystemMetrics 没有直接关系，而与调用函数 SetMenuItem-Bitmaps 前位图的准备有很大关系。位图的准备是很重要的，这是因为在 800×600 屏幕上看起来很漂亮的应用程序也应能够在具有完全不同字体与颜色表的 640×400 屏幕上较好地显示。

8.5 为系统菜单添加选项

问题

有时需要在应用程序窗口的系统菜单中添加选项，那么能做到这一点吗？系统菜单只是可读的吗？

方法

系统菜单既是只读的又不是只读的。为什么这么说呢？当使用类型 WS_SYSMENU 第一次创建窗口时，利用的是 Windows 提供的缺省系统菜单，因此 Windows 监控着菜单上的菜单项。比如，在极大化或极小化窗口之前，使菜单项 Restore 一直呈现为变灰状态，即不可选状态；当窗口为图标时，使菜单项 Minimize 变灰，处于不可选状态等等。用户不能或者至少说不必改变这种缺省的系统菜单，因为这种缺省的系统菜单是为许多应用程序所共用的。

不过，Windows 也的确允许用户以一种简易的方法来修改用户自己窗口的系统菜单。当调用函数 GetSystemMenu 时，Windows 便为应用程序窗口创建一个系统菜单的拷贝，并且返回此拷贝的句柄，这样，就可以用各种方式来修改系统菜单。在随后对函数 GetSystemMenu 调用时，就能请求系统或者返回窗口现有的系统菜单的拷贝，或者放弃对系统菜单所做的一

切修改而返回一个新的系统菜单拷贝的句柄。

从下面的例子程序可以看出，为系统菜单添加菜单项是非常容易的。

步骤

按如下步骤生成的例子程序 ADDSYS.EXE 运行时打开一个简单的窗口，此窗口除在标题条左上角的系统菜单外，没有任何其他菜单。

当下拉系统菜单时，便会注意到有个附加的菜单项 Shutdown System 被添加到菜单 Close 的下面，当选择此菜单项时，便会使 Windows 关闭，如同从 Start 菜单中选择 Shutdown 菜单项一样。这就是一个为应用程序系统菜单添加菜单项的例子。图 8-5 显示了在系统菜单中添加菜单项时的情况。

通过选择系统菜单中的菜单项 Close 或点击标题条最右端的关闭图标来关闭例子程序。下面的步骤将指导如何实现这一简单的例子程序。

1. 为此例子程序创建工作目录 ADDSYS，把此例子程序的所有源文件存放到此目录中。

2. 为此例子程序创建资源脚本。创建文本文件 ADDSYS.RC，并添加下面的程序，由于此例子程序只有一个图标，所以这是个很简单的资源文件。

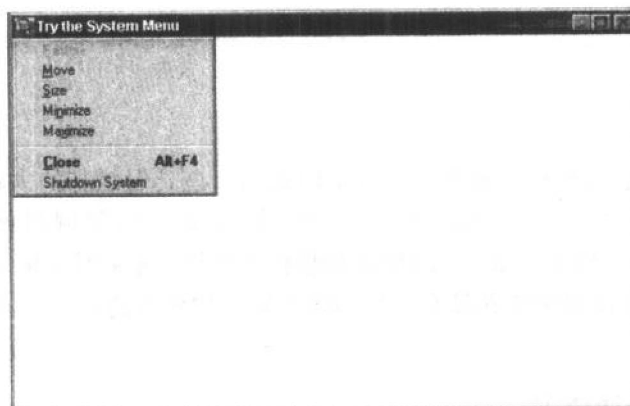


图 8-5 在系统菜单中添加菜单项

```
/* ---------------------------------------------------------------------- */
/*                                                                        */
/* MODULE: ADDSYS.RC                                                      */
/* PURPOSE: This resource script defines the application icon for         */
/*          this application.                                             */
/*                                                                        */
/* ---------------------------------------------------------------------- */

#include " addsys.rh"

IDI_APPICON ICON " app.ico"
```

3. 在资源脚本编译之前，需要把图标文件添加到工作目录中，此图标文件可根据前面的介绍自行创建。

4. 创建步骤 2 中用到的 include 文件 ADDSYS. RH，此文件将包含一些定义。这些定义把 C 源码中用到的资源 ID 与资源脚本中的资源连接起来。本例中，只有应用程序图标。

```
#ifndef __ADDSYS_RH
/* ---------------------------------------------------------------- */
/*                                                                  */
/* MODULE: ADDSYS.RH                                                */
/* PURPOSE: This include file defines the resource identifiers used. */
/*                                                                  */
/* ---------------------------------------------------------------- */
#define __ADDSYS_RH

#define IDI_APPICON     202

#endif
```

5. 现创建例子程序的 C 源文件。此文件中代码的功能在于：创建例子程序，在例子程序的系统菜单里插入菜单项 Shutdown System。另外，它还根据用户的要求，处理菜单项 Shutdown System 的消息，关闭系统运行。

创建新文件 ADDSYS. C，把下面的代码添加到此文件的头部。这些代码用来包括窗口文件并定义 STRICT，定义 STRICT 的目的是为了使编译系统进行严格的类型检查，从而让用户小心使用 HWND、HMENU 之类的 Windows 类型。可以注意到，代码中还定义了一个标识符 SC_SHUTDOWN，此标识符将做为菜单项 Shutdown System 的 ID。

```
/* ---------------------------------------------------------------- */
/*                                                                  */
/* MODULE: ADDSYS.C                                                 */
/* PURPOSE: This sample application demonstrates how you can use    */
/*          GetSystemMenu to get a handle to a copy of the system   */
/*          menu for a Window, and then modify that menu using the  */
/*          AppendMenu API function.                                */
/*                                                                  */
/* ---------------------------------------------------------------- */

#define STRICT
#include <windows.h>
#include <winnt.h>
#include " addsys.rh"

#define SC_SHUTDOWN    1000

static char *MainWindowClassName = " AddSystemWindow";
```

6. 添加下面的函数 AddSystemShutdownItem。这是一个了解如何修改系统菜单的关键函数，此函数调用函数 GetSystemMenu 来获取窗口系统菜单的 HMENU 句柄，如果窗口刚被打开，它利用的是 Windows 提供的缺省系统菜单。在这种情况下，Windows 就为此菜单创

建一个拷贝并且返回此拷贝的句柄。如果在应用程序的后边再次调用函数 GetSystemMenu，此时可能就有一些代码对该菜单作了修改，在这种情况下，有两种选择，一种选择是进一步修改现有的菜单，第二种选择是从 Windows 中获取一个原始（未作任何修改）系统菜单的拷贝重新开始。函数 GetSystemMenu 中的第二个参数就是让用户做以上选择的，如果第二个参数选择的是值 FALSE，就返回现有的、可能被修改过的菜单句柄；如果是 TRUE 值，就返回一个原始的、未做任何修改的系统菜单的拷贝。在本例中，利用的是 FALSE，这是因为当添加新菜单项时，只有不破坏对该菜单所作的任何修改才有意义。当获得了菜单句柄后，就能像修改普通菜单那样随心所欲地对其进行修改了。下面的函数 AddSystemShutdownItem 通过调用函数 AppendMenu，在系统菜单的末尾增加了一个 ID 值为 SC _ SHUTDOWN，显示的字符串为 Shutdown System 的新菜单项。

```
/* ------------------------------------------------------------------------ */
/* This function gets a handle to a copy of the system menu for this        */
/* window. It then uses AppendMenu to add the " Shutdown System" item       */
/* to the menu.                                                             */
/* ------------------------------------------------------------------------ */
void AddSystemShutdownItem (HWND hWnd)
{
    HMENU hMenu;

    /* Get a handle to the system menu for this window. If
       this Window's menu is no different from the default
       system menu, Windows will make a copy of the system
       menu first, and give us the handle to that. */
    hMenu = GetSystemMenu (hWnd, FALSE);
    if (! hMenu)
        return;

    /* Add an item to the end of the system menu.
       Give it the ID SC _SHUTDOWN. */
    AppendMenu (hMenu, MF _STRING, SC _SHUTDOWN," Shutdown System");
}
```

7. 把下面的程序添加到源文件中。当用户从例子程序的系统菜单里选择菜单项 Shutdown System 时，函数 StartSystemShutdown 将被调用。此函数利用一个消息框来确认用户是否确实要关闭系统，如果确认，系统将因调用函数 ExitWindowsEx 而停止运行。在调用函数 ExitWindowsEx 时，利用的是参数 EWX _ SHUTDOWN，此参数使得系统停止运行，并准备关机。下面的这段程序采用的较好做法是：如同从菜单 Start 中选择菜单项 Shutdown 时产生的对话框那样，为用户提供了一个重新启动系统或以 MS _ DOS 的方式启动系统的选项。

```
/* ------------------------------------------------------------------------ */
/* This function calls ExitWindowsEx with the EWX _SHUTDOWN argument        */
/* to shutdown the system in preparation for turning off the PC.           */
/* ------------------------------------------------------------------------ */
```

```
void StartSystemShutdown (HWND hWnd)
{
    if (MessageBox (hWnd," Shutdown the system?",
                    " Confirm Shutdown",
                    MB_OKCANCEL | MB_ICONQUESTION) == IDOK)
    {
        ExitWindowsEx (EWX_SHUTDOWN, 0);
    }
}
```

8．下面所列的函数是此例子程序的主回调函数，通过此函数，处理例子程序窗口的所有消息。响应消息 WM_CREATE（此消息在窗口建立后，但是在显示前被发送），此函数调用前面所编写的函数 AddSystemShutdown。它把菜单项 Shutdown System 添加到系统菜单中。当用户从菜单中选择此菜单项时，一个消息便被发送到此窗口过程，此消息与一般菜单项发送的消息不同。一般菜单项发送的消息是 WM_COMMAND，在 WPARAM 中有消息的 ID 值；而系统菜单项发送的是 WM_SYSCOMMAND 消息，此消息的 ID 值也在 WPARAM 参数中。下面的代码将处理此事件，并调用以前输入的函数 StartSystemShutdown。

```
/* ---------------------------------------------------------------- */
/* This function is the main window callback function which will be */
/* called by Windows to process messages for the window.            */
/* ---------------------------------------------------------------- */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
            AddSystemShutdownItem (hWnd);
            return 0;
        case WM_SYSCOMMAND:
            if (wParam == SC_SHUTDOWN)
            {
                StartSystemShutdown (hWnd);
                return 0;
            }
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

9．下面的三个函数组成此例子程序的框架。事实上，本章中的所有程序都利用到与此类

似的三个函数。把下面的代码添加到源文件的末端。函数 InitApplication 被此应用程序用来注册窗口类，此函数只有当未注册窗口类时才调用；函数 InitInstance 用于创建和显示应用程序的主窗口；函数 WinMain 用作此例子程序的入口点，它用来注册并建立窗口，然后利用函数 GetMessage、TranslateMessage 以及 DispatchMessage 来获取并处理消息，以便例子程序能正常运行。

```
/* ------------------------------------------------------------------------ */
/* This function initializes a WNDCLASS structure and uses it to            */
/* register a class for our main window.                                    */
/* ------------------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ------------------------------------------------------------------------ */
/* This function creates an instance of our main window. The window         */
/* is given a class name and a title, and told to display anywhere.         */
/* The nCmdShow argument passed to the program determines how the           */
/* window will be displayed.                                                */
/* ------------------------------------------------------------------------ */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hWnd = CreateWindow (MainWindowClassName," Try the System Menu",
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                NULL, NULL, hInst, NULL);
    if (! hWnd)
```

```
        return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);          // Send a WM_PAINT message.
    return TRUE;
}

/* ------------------------------------------------------------------------ */
/* The main entry point for Windows applications. Check to see if           */
/* the main window class name has already been registered, if it has        */
/* not, call InitApplication to register it. Call InitInstance to           */
/* create an instance of our main window, then pump messages until          */
/* the application is closed.                                               */
/* ------------------------------------------------------------------------ */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

10. 编译与运行此例子程序。

注释

如果获得了系统菜单的句柄，把菜单项添加到系统菜单则是一件十分简单的事情。当往系统菜单中添加菜单项时，要小心一点。不过，大多数用户一般都不利用系统菜单，而是利用应用程序的菜单。实际上，由于 Windows 95 在标题条的右端有一个专门的 close 按钮，因此用户或许从来不去访问系统菜单。除了诸如停止运行系统这样的实用函数外，大多数菜单操作应该是针对一般应用程序菜单进行的。

8.6 设计点击鼠标右键后弹出的菜单

问题

如果想要实现一个与上下文相关的菜单，当用户在应用程序的某一区域上按下鼠标右键时，此菜单就会弹出。如何才能实现这样的菜单呢?

方法

显示浮动式弹出菜单有两个需解决的问题。第一个问题是何时显示菜单。本节中所讨论的例子程序响应鼠标右键的点击而显示菜单，然而，鼠标右键的点击并没有被发送到对话框或窗口，而是被发送到鼠标所在的控制。为了获取这些消息，必须子类化控制，用自己的消息处理函数来替换类中定义的消息处理函数。在此函数中，当有事件发生时，便查找消息 WM_RBUTTONDOWN 并把它通知到父窗口，而所有其他消息将传送给原来的控制处理程序。

另一个问题是如何显示弹出式菜单。用户既可以通过拷贝现有的应用程序菜单来创建菜单，也可以通过利用函数 CreatPopupMenu 来创建菜单，还可以通过调用函数 LoadMenu 装入来自资源的菜单。当获得要显示的菜单的句柄时，便可通过调用函数 TrackPopupMenu 显示菜单并跟踪用户的选择，函数 TrackPopupMenu 返回后，实际选择菜单项的消息便被发送到应用程序的窗口。

步骤

按照如下步骤生成的例子程序 POPUP.EXE 运行时产生一个标准的窗口和一个菜单 File，菜单 File 仅包含菜单项 Exit。如果此例子程序是文字处理器，菜单 File 可能还会包含其他选项，诸如 New、Open、Save 、Save As 等。除此之外，文字处理器可能还会有一个包含 Cut、Copy 以及 Paste 等选项的 Edit 菜单，一个 Format 菜单以及一个 Help 菜单。

所有的菜单都是有很多项的，这意味着用户必须记住他们所要用的命令在哪个菜单上。现在，把鼠标指针置于窗口里的某一位置，点击鼠标右键，一个与上下文相关的菜单就会弹出。此菜单中可能就包含在文字处理器中所希望找到的命令，这样，用户就不必或至少不需太频繁地搜寻所有的菜单来寻找所需的命令了，适合用户当前工作的一些常用命令在弹出式菜单中都是可以得到的。图 8-6 表示当显示弹出式菜单时，应用程序的外观形式。

在本例所显示的菜单中，包含有菜单项 Cut、Copy 以及 Paste 等，这些菜单项在文字处理器或文本处理器中都可能找到。另外还包含菜单项 Help，这在现代程序设计环境中经常可以看到。这里的菜单项 Help 仅显示一个消息对话框，其他菜单项可以正常工作。此外，菜单项 Cut 与 Copy 只有在选择一些文本后才被激活，菜单项 Paste 只有在裁剪板中有被粘贴的文本时才被激活。当点击菜单外的任一地方时，此菜单就会消失，而且不做任何选择。

下面的步骤指导如何实现这一简单例子。从这些步骤中，可以学习到如何利用资源装入菜单并作为弹出式菜单把它显示出来。关于利用这类菜单的更多讨论，请参看本节末尾的注释部分。

1. 在开始为此例子程序编制代码之前，创建存储源码的工作目录 POPUP。

2. 创建存放菜单的资源文件。利用文本编辑器创建下面的资源脚本，保存该文件为 POPUP.RC。注意此处定义了两个不同的菜单，一个是为主菜单定义的 IDM_TESTMENU，另一个是为弹出式菜单定义的 IDM_POPMENU。

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/* MODULE: POPUP.RC                                                   */
/* PURPOSE: This resource script defines the base menu for this       */
```

```
/*              example application.                                  */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include <windows.h>
#include "popup.rh"

IDM_TESTMENU MENU
{
    POPUP "&File"
    {
        MENUITEM "E&xit", CM_EXIT
    }
}

IDM_POPUPMENU MENU
{
    POPUP "&Not used"
    {
        MENUITEM "Cu&t", CM_EDITCUT
        MENUITEM "&Copy", CM_EDITCOPY
        MENUITEM "&Paste", CM_EDITPASTE
        MENUITEM SEPARATOR
        MENUITEM "Go to &help topic", CM_GOTOHELP
    }
}

IDI_APPICON ICON "app.ico"
```

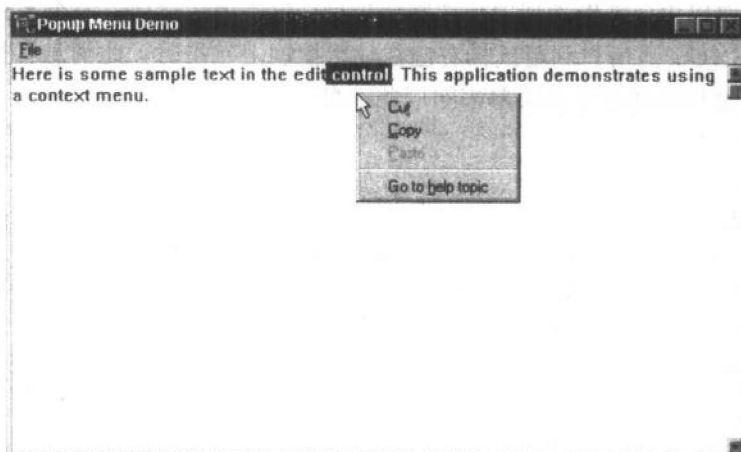


图 8-6　显示弹出式菜单

3. 创建 include 文件，此文件将被资源脚本与 C 源码文件利用。下面的程序代码定义了此例子程序所用的资源标识符，把此文件保存为 POPUP. RC。

```
#ifndef __POPUP_RH
/* ------------------------------------------------------------------ */
/*                                                                    */
/* MODULE: POPUP.RH                                                   */
/* PURPOSE: This include file defines the resource identifiers used.  */
/*                                                                    */
/* ------------------------------------------------------------------ */
#define __POPUP_RH

#define IDM_TESTMENU        200
#define CM_EXIT             103

#define IDM_POPUPMENU       203
#define CM_EDITCUT          111
#define CM_EDITCOPY         112
#define CM_EDITPASTE        113
#define CM_GOTOHELP         115

#define IDI_APPICON         202

#endif
```

4. 在编译资源之前，需要一个包含应用程序图标的文件。利用资源编辑器创建一个 16 色的、32×32 象素的图标文件，并把此文件保存为 APP. ICO。

5. 接下来，创建此例子程序的源码文件。创建新文件 POPUP. C，把下面的注释与定义添加到此文件。需要注意的是，程序中说明了一个编辑控制的句柄，此编辑控制将在例子程序窗口中显示。另外，还说明了一个 LONG 类型的变量，此变量用来存放缺省的处理程序的地址，此处理程序用来处理发送到编辑控制的消息。程序中还定义了一个客户消息 MYMSG_CONTEXTMENU，当用户利用鼠标右键在编辑控制上点击时，此消息就被送到主窗口的回调函数，窗口过程将通过显示菜单来响应此消息。

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/* MODULE: POPUP.C                                                    */
/* PURPOSE: This example application demonstrates how you can catch   */
/*          the right mouse button click over an area of your window  */
/*          and use TrackPopupMenu to display a context-sensitive     */
/*          popup menu.                                               */
/*                                                                    */
/* ------------------------------------------------------------------ */

#define STRICT
```

```
#include <windows.h>
#include <winnt.h>
#include " popup.rh"

static char *MainWindowClassName = " PopupMenuWindow";
HINSTANCE hInstance = NULL;
HWND       hEdit = NULL;
LONG       oldEditProc = 0;

#define MYMSG_CONTEXTMENU (WM_USER + 400)
```

6. 现在，把下面的函数添加到源文件中。函数 EditWndProc 是编辑控制的置换窗口过程，当点击鼠标右键时，此函数通过把消息 MYMSG_CONTEXTMENU 发送到父窗口来响应鼠标右键的点击（消息 WM_RBUTTONDOWN）；所有其他的消息是通过利用函数 CallWindowProc 调用存储在 oldEditProc 中原有的窗口过程来处理的。

```
/* ------------------------------------------------------------------------ */
/* This window procedure is used to handle messages from an edit            */
/* control. It catches right mouse clicks and creates context menu          */
/* messages to be sent to the owner.                                        */
/* ------------------------------------------------------------------------ */
LPARAM CALLBACK EditWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
   HWND hOwner;

   // Grab right mouse button clicks, send a MYMSG_CONTEXTMENU message.
   if (message == WM_RBUTTONDOWN)
   {
      hOwner = GetParent (hWnd);
      PostMessage (hOwner, MYMSG_CONTEXTMENU, (WPARAM) hWnd, lParam);
      return 0;
   }

   // For other events, call the original edit control procedure.
   return CallWindowProc ( (WNDPROC) oldEditProc, hWnd, message, wParam, lParam);
}
```

7. 把下面的函数 CreateEditWindow 添加到同一源文件 POPUP.C 中。此函数利用函数 CreateWindow 来创建具有“EDIT”类的新窗口，也就是编辑控制。函数中没有设置此编辑控制在应用程序窗口中的大小和位置，关于编辑控制的大小和位置，是在编辑控制显示之前，响应消息 WM_SIZE 时加以设置的。此段程序还调用了函数 SetWindowLong 来修改与编辑控制有关的 GWL_WNDPROC 长整数，它利用前面定义的 EditWndProc 处理程序替换窗口类的处理程序，旧的值将被 SetWindowLong 返回并存储在变量 oldEditProc 中，旧的处理程序在编辑控制删除前必须被恢复。

```
/*----------------------------------------------------------------------*/
/* This function uses CreateWindow to create an edit contol within      */
/* the main window. It adds some sample text to the edit control.       */
/*----------------------------------------------------------------------*/
HWND CreateEditControl (HWND hWndParent)
{
    HWND hChild;

    // Create the edit control using CreateWindow.
    hChild = CreateWindow ("EDIT", NULL,
                          WS_CHILD | WS_VISIBLE | WS_VSCROLL |
                          ES_LEFT | ES_MULTILINE | ES_AUTOVSCROLL,
                          0, 0, 0, 0, // size will be set be WM_SIZE.
                          hWndParent, NULL, hInstance, NULL);

    // Subclass the control — set up a handler for messages.
    oldEditProc = SetWindowLong (hChild, GWL_WNDPROC, (LONG) EditWndProc);

    return hChild;
}
```

8. 把下面的函数 SetupEditMenu 添加到源文件中。函数 SetupEditMenu 负责激活与变灰弹出式菜单上的菜单项 Cut、Copy 以及 Paste。在一个完整的应用程序中，可利用这里的同段代码来处理菜单条上的主 Edit 菜单，只要把要操作的相应菜单句柄传递过去即可。这段程序利用 EM_GETSEL 来检索在编辑控制中，当前所选择的开始与结束位置，如果从 DWORD 中获取的开始与结束位置相同，说明没有文本被选择，所以 Cut 与 Copy 的菜单项就被变灰。

确定菜单项 Paste 是否应该被激活的方法与菜单项 Cut、Copy 略有不同。此程序调用函数 IsClipboardFormatAvailable 来确定具有标准的 CF_TEXT 格式的内容在剪贴板中是否有效，如果有正确格式的内容，此程序就利用函数 EnableMenuItem 来激活菜单项 Paste；否则就禁止此菜单项。

```
/*----------------------------------------------------------------------*/
/* This function checks to see if any text is selected in the edit      */
/* control. If it is, then cut and copy are enabled in the menu,        */
/* otherwise they are grayed. It also checks to see if there is any     */
/* compatible text in the clipboard. If there is, then the paste        */
/* option is enabled, otherwise it is grayed.                           */
/*----------------------------------------------------------------------*/
void SetupEditMenu (HMENU hMenu, HWND hEdit)
{
    DWORD selection;
    int   selStart, selEnd;

    // See if any text is selected.
```

```
    selection = SendMessage (hEdit, EM_GETSEL, 0, 0);
    selStart = LOWORD (selection);
    selEnd = HIWORD (selection);
    if (selStart != selEnd)
    {
        // Text is selected — enable menu items.
        EnableMenuItem (hMenu, CM_EDITCUT, MF_BYCOMMAND | MF_ENABLED);
        EnableMenuItem (hMenu, CM_EDITCOPY, MF_BYCOMMAND | MF_ENABLED);
    }
    else
    {
        // No text selected — disable items.
        EnableMenuItem (hMenu, CM_EDITCUT, MF_BYCOMMAND | MF_GRAYED);
        EnableMenuItem (hMenu, CM_EDITCOPY, MF_BYCOMMAND | MF_GRAYED);
    }

    // Now see if there is any text in the clipboard.
    if (IsClipboardFormatAvailable (CF_TEXT))
        EnableMenuItem (hMenu, CM_EDITPASTE, MF_BYCOMMAND | MF_ENABLED);
    else
        EnableMenuItem (hMenu, CM_EDITPASTE, MF_BYCOMMAND | MF_GRAYED);
}
```

9. 添加下面的函数。此函数利用 API 函数 LoadMenu 负责从资源脚本中装入弹出式菜单，并利用函数 TrackPopupMenu 把此菜单显示出来。下面，我们就来看一下此函数显示菜单的具体步骤。

首先，此函数调用 LoadMenu 装入资源。由于应用程序的资源标识符是整型值，所以利用宏 MAKEINTRESOURCE，如果利用字符串来标识资源，就不必利用宏 MAKEINTRESOURCE，因为此时 LoadMenu 需要的是字符串，不过，利用整数型会使应用程序变得较小而且速度较快。函数 LoadMenu 返回的句柄是 HMENU，但它并不正是所需要的，还需要利用此句柄，通过调用函数 GetSubMenu 来检索要显示的子菜单的句柄。利用这个方法，就能从菜单资源中获取弹出式菜单，并利用函数 GetSubMenu 来检索要显示的相应菜单的句柄。

最后，把鼠标坐标转换为屏幕坐标，并用来把菜单定位在鼠标所在位置，当用户从菜单中选择菜单项，或通过点击取消菜单时，函数 TrackPopupMenu 就会返回。

```
/* ------------------------------------------------------------ */
/* This function uses LoadMenu to load the resource of the popup */
/* menu. It then decides where to place the menu, and uses the   */
/* TrackPopupMenu function to display the menu. It destroys the  */
/* menu when done, using DestroyMenu to release resources.       */
/* This function returns TRUE if the menu was displayed.         */
/* ------------------------------------------------------------ */
BOOL HandlePopupMenu (HINSTANCE hInstance, HWND hWnd,
                      LONG mouseX, LONG mouseY, HWND hEdit)
```

```
HMENU hResourceMenu, hPopupMenu;
POINT mousePos;
RECT  rcClient;

/* Get the bounds of the client area, and make sure
   that the mouse was inside it when the button was
   clicked. If it wasn't, return straight away. */
GetClientRect (hWnd, &rcClient);
mousePos.x = mouseX;
mousePos.y = mouseY;
if (! PtInRect (&rcClient, mousePos))
    return FALSE;

// Load the Popup menu from the resource.
hResourceMenu = LoadMenu (hInstance,
                    MAKEINTRESOURCE (IDM_POPUPMENU));
if (! hResourceMenu)
    return FALSE;

// Get the popup menu from the resource menu.
hPopupMenu = GetSubMenu (hResourceMenu, 0);

// Enable or disable menu items.
SetupEditMenu (hPopupMenu, hEdit);

/* Convert the mouse coordinates to screen coordinates
   for use by TrackPopupMenu. */
ClientToScreen (hWnd, &mousePos);

/* Track the context-sensitive popup menu. Destroy
   it when done. Use TPM_LEFTALIGN so that the menu
   pops up to the right of the mouse, and use
   TPM_RIGHTBUTTON so the user can use the right mouse
   button to select items from the menu. */
TrackPopupMenu (hPopupMenu, TPM_LEFTALIGN | TPM_RIGHTBUTTON,
            mousePos.x, mousePos.y, 0, hWnd, NULL);

DestroyMenu (hResourceMenu);

return TRUE;
}
```

10. 把下面的代码添加到源文件中。MainWndProc 是窗口的消息处理函数，此函数从

Windows 中接收消息并处理它们。下面的代码演示了如何捕获窗口的鼠标右键事件并显示菜单。当用户按下鼠标右键时，例子程序就会从编辑控制的处理程序中接收到消息 MYMSG_CONTEXTMENU，为了响应此消息，下面的程序调用了刚刚输入的函数 HandlePopupMenu 来装入并显示菜单，因为参数 LPARAM 包含有鼠标的坐标（从原始消息 WM_RBUTTONDOWN 中拷贝来的），因此该函数利用 LOWORD（lParam）检索鼠标在屏幕上的 X 位置值，利用 HIWORD（lParam）来检索鼠标在屏幕上的 Y 位置值。

输入下面的代码时可以看到：当接收消息 WM_CREATE 时，编辑控制通过调用函数 CreateEditControl 而创建。在本程序段的末尾，响应消息 WM_DESTROY（在删除窗口与控制前），把编辑控制的处理过程替换为旧的处理过程（原始的处理过程）。此程序还响应任何 WM_SIZE 消息，并总保证编辑控制占据着窗口的整个用户区。

```
/* ------------------------------------------------------------------ */
/* This is the main window procedure for the application. This        */
/* procedure responds to the WM_RBUTTONDOWN message by displaying      */
/* the popup menu.                                                     */
/* ------------------------------------------------------------------ */
LPARAM CALLBACK MainWndProc (HWND hWnd, UINT message,
                             WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
            hEdit = CreateEditControl (hWnd);
            return 0;
        case WM_COMMAND:
            switch (wParam)
            {
                case CM_EDITCUT:
                    SendMessage (hEdit, WM_CUT, 0, 0);
                    return 0;
                case CM_EDITCOPY:
                    SendMessage (hEdit, WM_COPY, 0, 0);
                    return 0;
                case CM_EDITPASTE:
                    SendMessage (hEdit, WM_PASTE, 0, 0);
                    return 0;
                case CM_GOTOHELP:
                    MessageBox (hWnd," Help item selected. ",
                                " Popup Response",
                                MB_ICONINFORMATION | MB_OK);
                    return 0;
                case CM_EXIT:
                    DestroyWindow (hWnd);
```

```
                return 0;
            }
            break;
        case MYMSG_CONTEXTMENU:
            // Handle right clicks.
            HandlePopupMenu (hInstance, hWnd, LOWORD (lParam),
                             HIWORD (lParam), hEdit);
            return 0;
        case WM_SIZE:
            // Resize the edit control to the entire window.
            MoveWindow (hEdit, 0, 0, LOWORD (lParam),
                        HIWORD (lParam), TRUE);
            return 0;
        case WM_SETFOCUS:
            // Set the focus to the edit control.
            SetFocus (hEdit);
            return 0;
        case WM_DESTROY:
            SetWindowLong (hEdit, GWL_WNDPROC, oldEditProc);
            PostQuitMessage (0);
            return 0;
    }
    return DefWindowProc (hWnd, message, wParam, lParam);
}
```

11. 下面所示的三个函数组成此例子程序的框架。把这些函数添加到源文件 POPUP.C 中,并保存此文件。首先出现的函数 InitApplication 用来为 Windows 的应用程序注册窗口类,此函数仅当窗口类名还未被注册时,才被主程序调用。函数 InitInstance 用来创建例子程序的主窗口,并利用函数 ShowWindow 来确保主窗口的显示。函数 WinMain 是此例子程序的入口点,此函数调用另外两个函数,并循环往复地收集消息以便例子程序正常运行。

```
/* ------------------------------------------------------------ */
/* This function initializes a WNDCLASS structure and uses it to */
/* register a class for our main window.                          */
/* ------------------------------------------------------------ */
BOOL InitApplication (HINSTANCE hInstance)
{
    WNDCLASS  wc;

    wc.style = 0;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
```

```
    wc.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_APPICON));
    wc.hCursor = LoadCursor (NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
    wc.lpszMenuName =  MAKEINTRESOURCE (IDM_TESTMENU);
    wc.lpszClassName = MainWindowClassName;

    return RegisterClass (&wc);
}

/* ------------------------------------------------------------------------ */
/* This function creates an instance of our main window. The window        */
/* is given a class name and a title, and told to display anywhere.         */
/* The nCmdShow argument passed to the program determines how the           */
/* window will be displayed.                                                */
/* ------------------------------------------------------------------------ */
BOOL InitInstance (HINSTANCE hInst, int nCmdShow)
{
    HWND hWnd;

    hInstance = hInst;        // Store in global variable.

    hWnd = CreateWindow (MainWindowClassName," Popup Menu Demo",
              WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
              CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
              NULL, NULL, hInstance, NULL);
    if (! hWnd)
       return FALSE;

    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);        // Send a WM_PAINT message.
    return TRUE;
}

/* ------------------------------------------------------------------------ */
/* The main entry point for Windows applications. Check to see if           */
/* the main window class name has already been registered, if it has        */
/* not, call InitApplication to register it. Call InitInstance to           */
/* create an instance of our main window, then pump messages until          */
/* the application is closed.                                               */
/* ------------------------------------------------------------------------ */
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
```

```
    MSG msg;

    if (! FindWindow (MainWindowClassName, NULL))
        if (! InitApplication (hInstance))
            return FALSE;

    if (! InitInstance (hInstance, nCmdShow))
        return FALSE;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

12. 编译并运行此例子程序。

注释

本节所讨论的例子程序介绍了在应用程序中显示弹出式菜单的基本方法。程序开发人员要想使应用程序设计得出类拔萃，必须使上下文相关菜单中的外观特征能够随时改变，以便适应用户当前的工作环境与需求。例如，如果在现阶段拼写检验是非常必要的，则可以在菜单中添加一个用来检验拼写的菜单项。如果用户在不同的窗口或控制上点击鼠标右键，则所显示的弹出式菜单也应该有所不同，因为对应用程序的不同组件来说，用户所要进行的操作可能不同，因此，弹出式菜单应能反映这一点，而不能千篇一律。

利用菜单的句柄虽然能直接修改菜单项，但可以发现：如果在资源中有多个弹出式菜单会更方便一些。通过调用函数 LoadMenu 并设置不同的资源 ID，就能装入不同的菜单；或者也可装入一个包含全部选项作为子菜单的菜单（在这种情况下，可以根据位置利用函数 GetSubMenu 检索相应的子菜单），以上两种方法都是十分有效的。

第9章　文档和编辑器

计算机已经改变了商业界的工作方式。以前曾经利用打字机打印以及复印机复印的文档，如今可利用文字处理器与激光打印机来完成，在许多办公室里，数据库已经代替了塞得满满的书柜。当然，“无纸张”的办公室目前虽然还没有实现，但我们会向着这个目标而逐年迈进。然而，虽然世界在发展，技术在进步，但有些东西还是保持不变的，人们依旧需要把数据输入到表单、信件以及其他一些文档中。除了图形表示外，程序员依旧需要查看所列出的实际源码，秘书依旧需要利用录入技能来生成文档。

文档与编辑器是大多数计算机应用的主要环节。无论是简单地从文本表示中查看数据，还是通过一个复杂的表单应用程序输入数据，都含有一定的编辑成分。本章中，将着重讨论文档编辑器并介绍利用 Windows API 函数为用户进行编辑而改进应用程序的一些方法。另外，还将讨论显示文本、接收表单数据以及其他一些细节性问题，例如使光标以自己想要的方式而非固有的设计方式闪烁等。

除标准的 Windows 控制外，Windows95 还引入了供程序员使用的一些新的控制。本章中，将讨论一个新的 Windows95 控制——Rich Text Edit 控制，此控制代替了标准 Windows 的编辑控制，它可以处理较大的文件并能够显示文本信息的属性。

最后，还将讨论如何利用标准的 Windows 控制来扩充文档与编辑器的功能。程序员可以利用 Windows API 查找或替换在屏幕上所显示文档中的文本，也可以简单而快速地在应用程序窗口中显示文件。此外，将讨论如何在应用程序的无模式对话框中利用建立在对话框中的表单功能。

1. 如何自动打开 MDI 子窗口

本节将介绍在 MDI 框架中自动创建 MDI 子窗口的方法。所有 Visual C++的程序员都知道，在应用程序对象中可以调用函数 OnFileOpen，但极少有人知道可以选用另外的方法，本节，将介绍这些方法中的一个。

2. 如何在 MDI 窗口中利用无模式对话框

当在屏幕上显示 MDI 子窗口时，最理想的是显示类似对话框模样的窗口，而且不把用户局限于该窗口。本节将介绍如何利用 MDI 子窗口做为表单窗口并向用户显示无模式对话框。

3. 如何随 MDI 子窗口调整列表框或编辑控制的大小

MDI 窗口的通常用法是：向用户显示一个全屏编辑器或列表选择框。利用包含诸如列表框、编辑控制等 Windows 控制的窗口，存在的问题之一是：当父 MDI 窗口框架变化时，其中的编辑控制或列表框不能随之调整大小，本节将介绍处理此问题需要了解的 Windows 信息与通知，并为用户自动调整控制的大小。这里也将从 Visual Basic 的观点来考察这一问题，并显示出一个能够自动调整控制大小的 VB 表单。

4. 如何创建简单的文件阅览器

有时，程序员需要显示一个小的文件来让用户阅读，比如 readme 文件或报告输出文件等。无论是在 Visual C++还是在 Delphi 中，都提供了无需修改的显示文件让用户阅读的简单方法。本节中，将介绍如何完成这一任务。

5. 如何查找和替换文本

如果在应用程序中没有增加代码的任何额外开销的情况下能够自动查找与替换文档中的简单文本串，岂不是一件非常好的事情，这一特定的任务已经以多种不同的方法实现，但是没有一种方法象本节介绍的如此简单。这种方法仅调用了几个 Windows API 函数，就能实现一个完整的、能够在文档中查找文本然后进行替换的 Find/Replace 对话框。

6. 如何显示大于 64KB 的文件

Windows 95 为显示大量文本提供了称为 Rich Text Edit 的新控制。本节中，将讨论如何利用此控制显示大于 64KB 字节的文件（一般的 Window 编辑控制，如 Windows 3.1，限制显示文件的大小在 32KB 与 64KB 之间），沿着这一方法，这里还将讨论如何利用 Rich Text 控制显示不同格式及字体的文本。

7. 如何改变编辑控制中插入光标的类型

“对于同一事物来说，不是每个人都会喜欢”，这是一个简单的事实，许多程序员在编辑器中可能宁愿使用不标准的各种不同类型的插入光标。本节中，将讨论如何改变应用程序中插入光标的大小与形状。

8. 如何改变插入光标的闪烁速度

一些用户或许不能忍受编辑器中光标闪烁的速度，另一些用户可能希望闪烁速度加快以便他们能较容易地发现光标，要解决这些问题，关键在于能够让 Windows 的用户可以选取他们想要的速度。本节中，将介绍如何改变插入光标的闪烁速度。

表 9-1 列出了本章中使用的 Windows 95 API 函数。

表 9-1 第 9 章中使用的 Windows 95 API 函数

CreateDialog	MoveWindow	GetClientRect
GetDlgltem	SwtWindowText	GetWindowText
SendMessage	RegisterWindowMessage	CreateCaret
ShowCaret	DestroyCaret	SetCaretBlinkTime

9.1 自动打开 MDI 子窗口

问题

有时需要使用 Visual C++创建应用程序，并且要在屏幕上打开新的文档窗口。Visual C++的资料说明指出：如果要打开新窗口，调用方法 OnFileNew；如果要打开包含有文件的窗口，则调用方法 OnFileOpen。不过，当要显示多种类型的窗口时，利用方法 OnFileNew 或 OnFileOpen，MFC 便总会显示一个引人注目的对话框，向用户询问要打开何种类型的文档。

如何通过 Windows API，利用 MFC 中的基本功能，直接打开所要的文档窗口而不依赖于 Visual C++呢？

方法

当在父 MDI 框架窗口上打开新的 MDI（Multiple Document Interface）窗口时，实际上

是向父框架窗口发送了一个简单的 Windows 消息，此消息就是消息 WM _ MDICREATE。在应用程序中实现上述过程是相当简单的，只需利用消息 WM _ MDICREATE 调用 Windows API 函数 SendMessage 并设置向父框架窗口发送的参数即可。

不幸的是，我们还不能够利用 MFC 中一般创建窗口的方法，由于没有一个消息映射函数能正确工作，窗口的菜单也不能被正确刷新。总之，事情不能如愿以偿。

不过，绕过 MFC 函数 OnFileNew 与 OnFileOpen 而直接编码来创建窗口是完全可能的。如果利用此方法，MFC 将会继续正常工作并且所有其他功能也保持完整无缺。本节中，将介绍如何利用此基本功能。

步骤

打开与运行按照如下步骤生成的例子程序 CH91. MAK。选择主菜单 Dialog 中的菜单项 Create New MDI Window。此时，便会显示如图 9-1 所示的 MDI 子窗口。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 App Wizard 创建新的项目文件 CH91. MAK。进入 AppStudio，从菜单列表中选择主菜单，添加标题为 Dialog 的主层菜单，在此主菜单上，添加标题为 Create New MDI Window、标识符为 ID _ NEW _ MDI _ WINDOW 的新的下拉菜单项。保存此资源文件，退出 AppStudio。

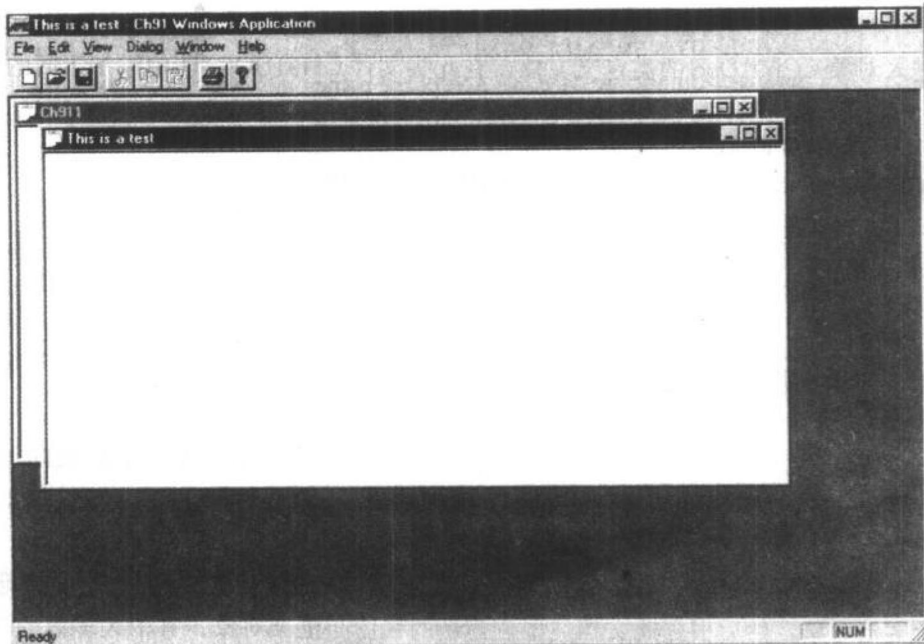


图 9-1 新的 MDI 子窗口

2. 在此例子程序对象的头文件中，把下面标记为黑体的行添加到类定义的顶端。

```
class CC91App: public CWinApp
{
private:
    CMultiDocTemplate * pDocTemplate;
```

3. 从项目列表中选择文件 CH91.CPP 并编辑它。对CCh91App 的方法 InitInstance 作如下改变。

```
//Register the application's document templates. Document templates
//serve as the connection between documents, frame windows and views.
// * * * Remove the declaration of pDocTemplate here * * *
pDocTemplate=new CMultiDocTemplate (
    IDR_CH91TYPE,
    RUNTIME_CLASS (CCh91Doc),
    RUNTIME_CLASS (CMDIChildWnd),                //standardMDIchildframe
    RUNTIME_CLASS (CCh91View));
AddDocTemplate (pDocTemplate);
```

4. 进入 ClassWizard，从下拉列表中选择对象 CCh91App，从对象列表中选择对象 ID_NEW_MDI_WINDOW，从消息列表中选择消息 COMMAND，接受名字 OnNewMdiWindow 作为新函数的名字，把下面的代码添加到 CCh91App 的方法 OnNewMdiWindow 中。

```
void CCh91App:: OnNewMdiWindow ()
{
    CCh91Doc * doc = new CCh91Doc (" Thisisatest");
    CFrameWnd * wnd = pDocTemplate->CreateNewFrame (doc, NULL);
    pDocTemplate->InitialUpdateFrame (wnd, doc, TRUE);
}
```

5. 进入对象 CCh91Doc 的类定义文件，在此头文件 CH91DOC.H 中添加下面的说明。

```
public:
    CCh91Doc (char * title);
```

6. 接着，把下面的代码添加到源文件 CH91DOC.CPP 中。

```
CCh91Doc:: CCh91Doc (char * title)
{
    SetTitle (title);
}
```

7. 编译与运行此例子程序。

用法

Visual C++的文档模板系统生成一个允许 MFC 由模板定义实现完整的 MDI 窗口的单联编。在模板对象（类 CMultiDocTemplate）中，存放有文档对象、MDI 框架对象以及视图对象的类型。

如果在模板中给定信息，MFC 就能创建新的 MDI 子窗口。要做到这一点，需要创建指定类型的新文档对象，然后根据此文档类型创建新的视图与框架对象。其中大多数的功能是在基对象 CWinApp 的方法 OnFileNew 中被实现的。为了绕过这一函数，程序员需要作一些自己的工作。

首先要做的是，在文档对象中添加公有的构造函数。这一步是必要的，因为所生成的文档构造函数是保护的，因此不能被另一个类调用，而我们需要在框架之外构造文档，所以这里需要一个能够被调用的构造函数。本例中的构造函数只是接受一个用来做为 MDI 窗口标题的字符串。

下一步要做的事情是：具体创建新的文档，这是通过方法 OnNewMdiWindow 来完成的。在 OnNewMdiWindow 中首先调用方法 CCh91Doc；接着构造函数，将调用文档模板的方法 CreateNewFrame，此方法用来创建新的框架与视图对象并初始化窗口句柄；最后，方法 OnNewMdiWindow 将调用模板的方法 InitialUpdateFrame，从而把 MDI 子窗口以及它的框架放在 MDI 父窗口上。此方法也负责初始化所有文档与视图对象的内部参数。

当所有这些完成之后，常规的 MFC 处理就会发生。当用户通过在系统菜单或关闭框上双击时，对象便被删除。该对象正常响应它所知的菜单命令以及在应用程序中建立的快捷键，并也将在菜单条的 Windows 菜单上显示出来。

注释

此方法对于创建新的 MDI 子窗口的方法 OnFileNew 或 OnFileOpen 来说是一个巨大进步，它给了程序员而不是 MFC 确定显示何种类型窗口的能力，并使程序具有创建多种类型窗口的灵活性。

如果想扩充创建 MDI 子窗口的功能，应讨论如何处理消息 WM _ MDICREATE。因此，可以首先考察对象 CMainFrame 的方法 OnClientCreate，然后讨论传递的参数；有时还需要以不同的方法来重置创建过程。例如，当每次创建子窗口时，就把它极大化，这一点可以利用方法 OnClientCreate 来实现。

9.2 在 MDI 窗口中利用无模式对话框

问题

如果想在应用程序的屏幕上显示表单，利用对话框固然可以达到这一目的，但如何也能使表单窗口具有利用菜单和快捷键的能力呢？

人们一般较喜欢让 MDI 父窗口上的所有窗口看起来相同，因此不想直接地显示一个无模式对话框。那么，有能够在 MDI 子窗口中嵌入无模式对话框的方法吗？

方法

解决这一问题的方法很多。一种方法是：在 MDI 窗口中创建一个无模式对话框作为控制窗口。但会发现这种方法也存在一些问题，无模式对话框实际上并没有父窗口，而只有所有者，这就使得无模式对话框不能正确显示并会在错误的位置出现。

另一种方法是：根据对象 CDialog，创建一个新的 MDI 子窗口类。这种方法当然有效，但会涉及大量的工作，程序员不得不去处理按键、快捷键以及菜单项命令等繁琐的工作。

幸运的是，这些方法并不是必要的，MFC 提供了一个类，该类在 MFC 子窗口中封装了类 CDialog 的功能，它被称为类 CFormView，并且 ClassWizard、AppStudio 以及 MFC 的组件都（或多或少）直接支持它。

这里，没有直接调用 API 函数，但注意一下此过程中利用 WindowsAPI 的方法是很有意义的。关于 MFC 所使用的方法，将在本节“用法”中作更详细的讨论。

步骤

打开与运行按照如下步骤生成的例子程序 CH92. MAK。选择主菜单 Dialog 中的菜单项 Diaplay Form Window。此时，便会显示如图 9-2 所示的 MDI 子窗口。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH92. MAK。进入 AppStudio，

创建新的对话框，在此对话框中，添加几个静态文本域、编辑框以及单选按钮。

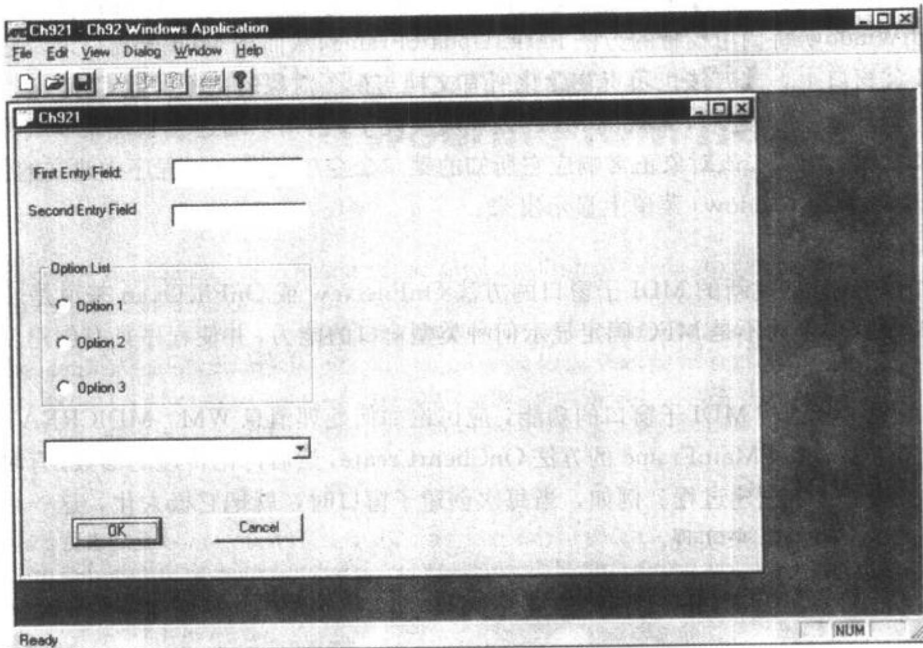


图 9-2 新的表单窗口

2. 双击对话框，选择选项 Styles。把风格组合框设置为 Child，把边界组合框设置为 None，确保不选择核选框 Titlebar 与 Visable。

3. 进入 ClassWizard，为刚创建的模板生成新的对话框类。把此类命名为 CDialogView，并从下拉列表中选择类 CFormView 作为它的基类，接受缺省的名字作为此类的文件名并保存此新类。

4. 选择类 CDialogView 的源文件 DIALOGVI. CPP，把下面标记为黑体的行添加到文件的消息映射中。

```
BEGIN_MESSAGE_MAP (CDialogView, CFormView)
    //{{AFX_MSG_MAP (CDialogView)
        ON_COMMAND (IDOK, OnOK)
    //}} AFX_MSG_MAP
END_MESSAGE_MAP ()
```

5. 把下面的方法添加到类 CDialogView 中。

```
void CDialogView:: OnOK ()
{
    GetParent () ->PostMessage (WM_CLOSE);
}
```

6. 把下面的说明添加到类 CDialogView 的头文件 DIALOGVI. H 中，注意这里也只是添加标记为黑体的行。

```
//Generated message map functions
```

```
//{{AFX_MSG (CDialogView)
virtual void OnOK ();
//}} AFX_MSG
DECLARE_MESSAGE_MAP ()
```

7. 从工程列表中选择对象 CH92.CPP。在 CCh92App 的方法 InitInstance 中添加下面的代码。

```
pDocTemplate1=new CMultiDocTemplate (
        IDR_CH92TYPE,
        RUNTIME_CLASS (CCh92Doc),
        RUNTIME_CLASS (CMDIChildWnd),          //standard MDI child frame
        RUNTIME_CLASS (CCh92View));
        AddDocTemplate (pDocTemplate1);
```

8. 接着，重新进入 AppStudio。从菜单列表中选择主菜单对象，并添加标题为 Dialog 的菜单，在此主菜单上添加标题为 Display Form Window、标识符为 ID_DISPLAY_FORM_WINDOW 的菜单项，保存此菜单，退出 AppStudio。

9. 在 ClassWizard 中，从下拉列表中选择对象 CCh92App。从对象列表中选择对象 ID_DISPLAY_FORM_WINDOW，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnDisplayFormWindow，把下面的代码添加到 CCh92App 的方法 OnDisplayFormWindow 中。

```
void CCh92App:: OnDisplayFormWindow ()
{
    CCh92Doc *doc =new CCh92Doc (" FormWindow");
    CFrameWnd *wnd= pDocTemplate1->CreateNewFrame (doc, NULL);
    pDocTemplate1->InitialUpdateFrame (wnd, doc, TRUE);
}
```

10. 把下面的 include 文件行添加到源文件 CH92.CPP 的顶端。

```
#include " dialogvi.h"
```

11. 在文件 CH92DOC.H 中输入对象 CCh92Doc 的类定义，把下面的说明添加到此头文件中。

```
public:
    CCh92Doc (char *title);
```

12. 把下面的代码添加到源文件 CH92DOC.CPP 中。

```
CCh92Doc:: CCh92Doc (char *title)
{
    SetTitle (title);
}
```

13. 最后，从项目列表中选择 include 文件 CH92.H，把下面的说明添加到类 CCh92App 的类定义中。

```
CMultiDocTemplate * pDocTemplate1;
```

14. 编译与运行此例子程序。

用法

如果考察MFC的源代码，特别是类CFormView的源代码，就会发现Windows API在创建类CFormView的过程中被重点考虑。CFormView利用Windows API函数CreateDialog来创建新的无模式对话框窗口，然后，在父窗口CView的区域内移动此窗口，以便对话框在视图中出现。

无模式对话框通过调用函数CreateDialog被创建，此Windows API函数的调用格式如下：

```
HWND CreateDialog (hinst, lpszDlgTemp, hwndOwner, dlgprc)
HINSTANCE hinst;
LPCSTR lpszDlgTemp;
HWND hwndOwner;
DLGPROC dlgprc;
```

在CFormView的情况下，参数hinst是应用程序的句柄，参数lpszDlgTemp是对话框模板的名字，在CFormView的继承类CDilogView中被定义为IDD_DIALOG3；参数hwndOwner指定此窗口，它“拥有”无模式对话框并从该对话框发送消息，这里也就是创建对话框的表单视图窗口；最后一个参数dlgprc用来指定对话框的过程，MFC对该参数传递NULL，以使所有对话框函数被送到标准对话框过程。

当有了在窗口中创建对话框的基础后，剩下的就很容易了。无形地创建对话框窗口并移动到拥有它的视图的用户区，这是由函数SetWindowPos来完成的，最后，对快捷键及菜单命令进行处理，其处理方法与第五章的无模式对话框中利用快捷键的方法相同。

注释

重复做别人所做过的事情往往意义不大，但是，当需要利用现有的MFC来做些工作时其意义是很大的。譬如，首先分析实现任务的基本Windows API调用，而后查找支持这些Windows API调用的MFC类，若这样做的话，就会使程序在API内运行起来更平稳，并使程序在不同平台间的移植性更好。

9.3 随MDI子窗口调整列表框或编辑控制的大小

问题

有时在应用程序中需要这样一种功能，随着父窗口大小的改变自动缩放MDI子窗口中的列表框或编辑控制。实现这种功能是非常有用的，例如：可以利用这种功能，来创建运行时可以向用户显示或隐藏的编辑或列表窗口，而无需利用模式对话框（继续前必须撤消的对话框）。

方法

这是存在于许多应用程序中的一个很共同的问题。用户经常希望在应用程序的窗口或视图的某一框格中显示不变的列表框，并允许用户任何时刻都能对列表框进行选择，这就意味着既不能简单地利用模式对话框（由于这种对话框把用户只限制于该对话框），也不能利用无模式对话框（由于这种对话框看起来不在屏幕的适当位置而且不在窗口列表中显示）。解决的办法是：在视图窗口中显示列表框，利用普通的Windows处理来负责更新与绘制区域。

这里的方法，就是需要列表框对用户来说应是“透明的”，用户不必担心列表框太大还是太小，这就意味着列表框应根据其嵌入的窗口进行自我调整。

本节利用了几个 Windows API 函数，但最值得注意的一个函数是 MoveWindow。此函数需要一个窗口句柄(即 MFC 中的窗口对象)以及一个利用结构 RECT 或坐标值表示的矩形做为参数，如果有父窗口的话，坐标就用于把此窗口映射到它的父窗口，因此，这里的方法就是获取窗口大小的消息（WM _ SIZE)，并把列表框调整为视图的用户窗口的大小。在这里，也将学习到如何在视图窗口的范围内创建控制。

步骤

打开与运行按照如下步骤生成的例子程序 CH93. MAK。选择主菜单 Dialog 中的下拉菜单项 Resize List。此时，便会显示如图 9-3 所示的 MDI 子窗口。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件。进入 ClassWizard，点击按钮 Add Class 添加新类，把新添加的类命名为 CResizeView，选择类 CView 做为新类的基类，然后保存此类。

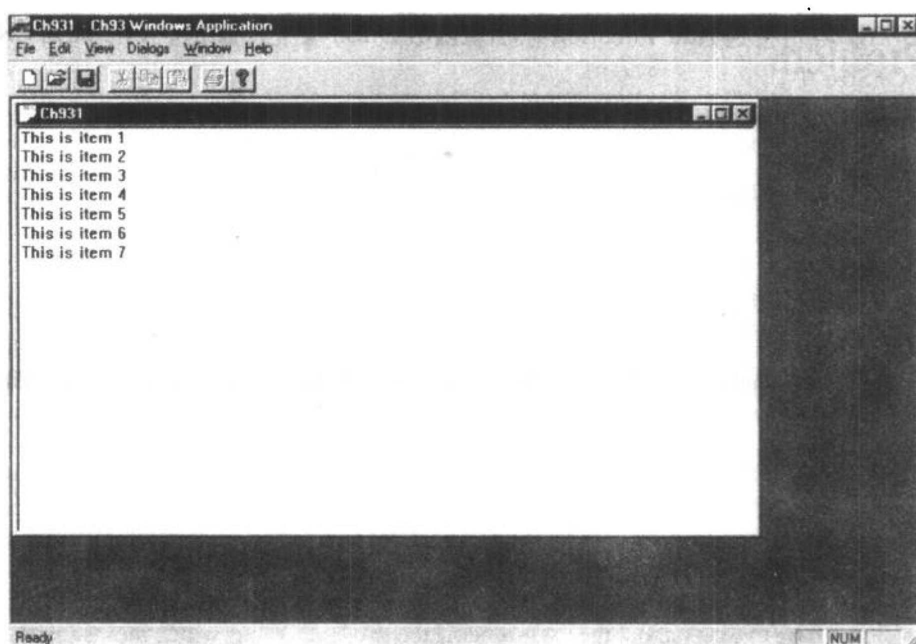


图 9-3 调整列表窗口

2. 在 ClassWizard 中，从下拉列表中选择类 CResizeView。从对象列表中选择 CResize-View，从消息列表中选择消息 WM _CREATE，点击按钮 Add Function 添加新函数 OnCreate。把下面的代码添加到 CResizeView 的方法 OnCreate 中。

```
int CResizeView:: OnCreate (LPCREATESTRUCT lpCreateStruct)
{
  if (CView:: OnCreate (lpCreateStruct) ==-1)
    return-1;
    //Get the size of the client rectangle
    CRect r;
```

```
    GetClientRect (&r);
    // Create a list box object with this view
    list = new CListBox;
    list->Create (WS_CHILD|WS_VISIBLE|WS_VSCROLL, r, this, 1001);
    //Add some items to the list
    list->AddString (" Thisisitem1");
    list->AddString (" Thisisitem2");
    list->AddString (" Thisisitem3");
    list->AddString (" Thisisitem4");
    list->AddString (" Thisisitem5");
    list->AddString (" Thisisitem6");
    list->AddString (" Thisisitem7");
    return 0;
}
```

3. 在ClassWizard中，从对象列表中再次选择对象CResizeView，从消息列表中选择消息WM_SIZE，点击按钮Add Function添加新函数OnSize。把下面的代码添加到CResizeView的方法OnSize中。

```
void CResizeView:: OnSize (UINT nType, int cx, int cy)
{
  CView:: OnSize (nType, cx, cy);
  CRect r;
  GetClientRect (&r);
  list->MoveWindow (&r);
}
```

4. 最后，把下面代码添加到类CResizeView的析构函数CResizeView::~CResizeView中。

```
CResizeView::~CResizeView
{
    delete list;
}
```

5. 把下面几行添加到类CResizeView的头文件RESIZEVI.H中。

```
private:
    CListBox * list;
```

6. 把下面几行添加到应用程序对象CCh93App的方法InitInstance中。

```
pResizeTemplate=new CMultiDocTemplate (
        IDR_CH93TYPE,
        RUNTIME_CLASS (CCh93Doc),
        RUNTIME_CLASS (CMDIChildWnd),          //standard MDI child frame
        RUNTIME_CLASS (CResizeView));
        AddDocTemplate (pResizeTemplate);
```

7. 进入AppStudio。从菜单列表中选择主菜单对象，添加标题为Dialogs的菜单，在此主菜单上，添加标题为Resize List Window、标识符为ID_RESIZE_DLG的下拉菜单项。

8. 进入 ClassWizard，从对象列表中选择应用程序对象 CCh93App。从对象列表中选择对象 ID_RESIZE_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnResizeDlg。把下面的代码添加到 CCh93App 的方法 OnResizeDlg 中。

```
void CCh93App:: OnResizeDlg ()
{
    CCh93Doc * doc=new CCh93Doc (" Resizable List Window");
    CFrameWnd * wnd=pResizeTemplate->CreateNewFrame (doc, NULL);
    pResizeTemplate->InitialUpdateFrame (wnd, doc, TRUE);
}
```

9. 把下面的 include 文件行添加到源文件 CH93.CPP 的顶端。

```
#include" resizevi.h"
```

10. 把下面两行添加到头文件 CH93.H 中。

```
private:
    CMultiDocTemplate * pResizeTemplate;
```

11. 在 CH93DOC.H 中输入对象 CCh93Doc 的类定义，把下面的说明添加到此头文件中。

```
public:
    CCh93Doc (char * title);
```

12. 把下面的代码添加到源文件 CH93DOC.CPP 中。

```
CCh93Doc:: CCh93Doc (char * title)
{
        SetTitle (title);
}
```

13. 编译与运行此例子程序。

用法

当创建视图对象时，在它的边界内创建了一个列表框，此列表框作为视图的子对象被创建，并且在视图内被设定为是可见的。由于列表框是子对象，所以它的边界框是相对于视图窗口的用户区的，因此，只要在视图对象的方法 OnCreat 中通过获取用户区域，并把窗口移动到这些坐标处，列表框便会在整个视图窗口中显现出来。

当调整视图的大小时，视图对象便收到消息 WM_SIZE，此消息将被视图对象的方法 OnSize 捕获，在对象 CResizeView 的方法 OnSize 中，列表只是被移动到视图的新的用户区并把它的大小设置为此用户窗口的大小。

在应用程序对象中，只是通过方法 CreateNewFrame，利用文档模板中标准的文档类实例化一个新的视图类，然后利用文档模板，在屏幕上实例化一个新的视图的备份。

注释

在视图的编辑框中可以利用同样的方法，它也是用来创建全屏幕编辑器的方法，此方法包括：在视图中创建一个多行编辑框，然后根据视图中的大小消息来缩放编辑框。

此方法可以用于视图边界内的任何一个控制上，例如，属性页就可以在视图内被创建，并根据用户的需要来调整其大小。

利用 VisualBasic 也可以编写出同样的程序，下面是一个 VisualBasic 表单的例子，此表单允许用户自动地调整其控制的大小。

1. 首先，创建新的项目文件或在现有的项目文件中添加表单，把新的项目文件命名为SIZE.MAK。在表单中添加名为List1的列表框。

2. 把下面的代码添加到表单的方法Form _ Load中。

```
SubForm _ Load ()
    List1.Top=20;
    List1.Height=Form1.Height
    List.Left=0;
    List1.Width=Form1.Width
    For I=1 To 20
      List1.AddItem" Item" +I
    Next I
End Sub
```

3. 把下面的代码添加到表单的方法Form _ Resize中。

```
Sub Form _ Size ()
    List1.Top=0;
    List1.Height=Form1.ScaleHeight
    List1.Left=0
    List1.Width=Form1.ScaleWidth
EndSub
```

4. 保存并运行此项目文件。

9.4 创建简单的文件阅览器

问题

有时需要在应用程序的对话框中创建一个简单的文件阅读器用来显示不是很大（主要由版本信息组成）而且没有任何特别格式的文件，但又不愿为此编写需要处理滚动、绘制等工作的窗口对象。

那么，利用Windows 95 API，是否有以最少的编码量来创建一个简单文件阅读器的办法呢?

方法

有人认为，如果提供了MFC的文档或视图结构以及Windows或Windows 95的编辑器风格，此刻定会有人创建出文件阅览器。遗憾的是，通读Windows API指南或MFC指南都找不到文件阅览器API调用的例子，也找不到文件阅览器类。

幸运的是，Windows 95与Windows 3.x中，具有显示小文件（数据小于32KB）能力的多行编辑框，编辑框的功能在多数Windows应用程序中都未能很充分地描述，这里将进一步着重说明。通过利用通用对话框的某些功能以及多行编辑器，就能在几分钟之内创建出具备所有功能的文件阅览器对话框。

步骤

打开与运行按照如下步骤生成的例子程序CH94.MAK。从主菜单Dialogs中选择菜单项File Viewer Dialog，此时，便会显示如图9-4所示的对话框。点击按钮Browse，并选择要查看的文本文件，当选择文件后，点击按钮OK，便在对话框顶端的编辑框中显示出所选择的文件的名字，点击按钮Load，编辑框便立即显示出此文件。

用户可以标记或拷贝编辑框中的文本，试着执行一下会发现，还可以对编辑框中的文本进行编辑。点击核选框 Set Read Only 后，此功能便被删除，不过，依旧可以查看、标记或拷贝编辑框中的文本，只是不能修改而已。

为了实现上述功能，请按如下步骤进行。

图 9-4 文件阅览器对话框

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH94. MAK。进入 AppStudio，创建新的对话框，在对话框的顶端，添加标题为 View File 的静态文本域，与此静态文本域相邻的右边，添加标识符为“IDC_EDIT2”的单行编辑框，然后，添加与这两个控制并排的按钮，它的标题为 &Browse、标识符为 IDC_BUTTON1。

2. 在刚添加的控制的下面，即对话框的中央，添加一个编辑框。从对话框中删除按钮 OK，并把按钮 Cancel 的标题设定为 Close，把按钮 Close 移到对话框底部的右下角，为多行编辑域设置如下的风格。首先，设置其标识符为 IDC_EDIT1；接着，把风格标志设置为“multiline”与“no hide sel”（点击这些核选框）。

3. 进入 ClassWizard，为此模板生成新的对话框类。把此类命名为 CFileViewDlg，接受该类的所有其他缺省值并保存此类。

4. 退出 AppStudio，保存资源文件。进入 ClassWizard，从下拉列表中选择 CFileViewDlg，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 添加新函数 OnBrowse。在 CFileViewDlg 的方法 OnBrowse 中添加下面的代码。

```
void CFileViewDlg:: OnBrowse ()
```

```
{
   CFileDialog dlg (TRUE," txt"," *.txt");
   if (dlg.DoModal () ==IDOK) {
     CEdit * edit= (CEdit * ) GetDlgItem (IDC _ EDIT2);
     edit->SetWindowText (dlg.GetPathName ());
   }
}
```

5. 从对象列表中选择对象 IDC _ BUTTON2，从消息列表中选择消息 BN _ CLICKED，点击按钮 Add Function 添加新函数 OnLoad，在 CFileViewDlg 的方法 OnLoad 中添加下面的代码。

```
const MAX _ BUFFER _ LEN=256;
void CFileViewDlg:: OnLoad ()
{
   char fileName [_ MAX _ PATH];
   char buffer [MAX _ BUFFER _ LEN+1];
   CEdit * edit= (CEdit * ) GetDlgItem (IDC _ EDIT1);
   //Get the file to load
   GetDlgItem (IDC _ EDIT2) ->GetWindowText (fileName, _MAX _ PATH);
   //Load the file into the edit box
   FILE * fp=fopen (fileName," r");
   if (fp== (FILE * ) NULL) {
     MessageBox (" Unable to open file"," Error", MB _ OK);
     return;
   }
   // Clear out the edit box
   edit->SetWindowText ("");
   while (! feof (fp)) {
     //Get a line from the input file
     if (fgets (buffer, MAX _ BUFFER _ LEN, fp) ==NULL)
     break;
     //Clear off the carriage return at end of line
     if (strlen (buffer))
     buffer [strlen (buffer) -1] =0;
     //Append CR/LF to make it appear nice
     strcat (buffer," \r\n");
     edit->SetFocus ();
     //Set the caret to be at the end of the edit box
     int ndx=edit->GetWindowTextLength ();
     edit->SendMessage (EM _ SETSEL, ndx, ndx);
     //Append the text to the edit box
```

```
    edit->SendMessage (EM_REPLACESEL, 0, (LPARAM) (LPCSTR) buffer);
  }
  //Close the file
  fclose (fp);
}
```

6. 从对象列表中选择对象 IDC_CHECK1，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnSetReadOnly，在 CFileViewDlg 的方法 OnSetReadOnly 中添加下面的代码。

```
void CFileViewDlg:: OnSetReadOnly ()
{
  CButton * b= (CButton * ) GetDlgItem (IDC_CHECK1);
  GetDlgItem (IDC_EDIT1) ->SendMessage (EM_SETREADONLY, b->GetCheck (), 0);
}
```

7. 再次进入 AppStudio。添加标题为 Dialogs 的主菜单，在菜单 Dialogs 上，添加标题为 FileViewer Dialog、标识符为 ID_FILE_VIEW_DLG 的菜单项，保存此菜单，然后退出 AppStudio。

8. 在 ClassWizard 中，从下拉列表中选择对象 CCh94App。从对象列表中选择对象 ID_FILE_VIEW_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnFileViewDlg，在 CCh94App 的方法 OnFileViewDlg 中添加下面的代码。

```
void CCh94App:: OnFileViewDlg ()
{
  CFileViewDlg dlg;
  dlg.DoModal ();
}
```

9. 把下面的 include 行添加到源文件 CH94.CPP 的顶端。

```
#include" fileview.h"
```

10. 编译与运行此例子程序。

用法

当最初显示对话框时，在显示文件名的编辑框以及多行编辑框中都没有任何东西。当用户选择按钮 Browse 时，便显示出一个对话框（通用文件打开对话框）。当用户从通用文件打开对话框中选择了某文件并点击按钮 OK 后，所选择的文件名便显示在对话框顶端的编辑框中。

当用户选择按钮 Load 时，便查询显示文件名的编辑框，并把指定的文件打开，此文件将逐行被读入到输入缓存，然后追加到编辑框中。这项工作，是通过向编辑框发送命令 EM_SETSEL（设置选择的范围），并设置范围为（-1，-1）来选择编辑框中最后一个字符的位置来完成的。获取了编辑文本的末端位置后，通过发送命令 EM_REPLACESEL 来替换当前的选择，这里，只是把文本追加到编辑域的末端。

核选框 Set Read Only 用来控制编辑框的“只读”设置，当选择它后，该域的核选标志被设置为 True，因此将传递 OnSetReadOnly 中的消息 EM_SETREADONLY 为标志 True，如

果不选择它，标志将设置为 False，这样也就关闭了“只读”方式。

9.5 查找和替换文本

问题

想要在应用程序的编辑框中为用户提供查找与替换文本的功能，最理想的做法是：使这个过程对程序员来讲只需要少量的工作甚至极少量的代码，而尽可能多地利用 Windows 的标准组件。

Windows API 提供了在应用程序中实现查找与替换文本的必要功能呢，还是需要从头编写整个程序呢？如果有一些在程序中能够应用的查找与替换的函数，则必定有人已经解决了这一普通的问题。

方法

实际上，这个问题早已被众多的程序员以数百种方式解决，但是，没有一个能够象本节所介绍的那样，利用如此少的代码来解决这一问题。

在 Windows 95 API 中，编辑框是一个了不起组件，它集灵活性与强有力的功能性于一体。本节中，将讨论可发送到编辑框用来定位与替换编辑文本中字符串的一些消息，另外，还将讨论如何创建标准的 Windows 通用查找对话框用来获取用户的输入，从而对所查找的文本进行修改。

步骤

打开与运行按照如下步骤生成的例子程序 CH95.MAK。选择主菜单 Dialogs 中的菜单项 Find and Replace，此时，便会看到如图 9-5 所示的对话框，点击按钮 Find and Replace 并在通用查找对话框中输入需要查找与替换的检索词，当输入了两个检索词后，点击按钮 Replace，如果此时查找到有需要替换的检索词，则编辑框将立即用给定的检索词对其进行替换。

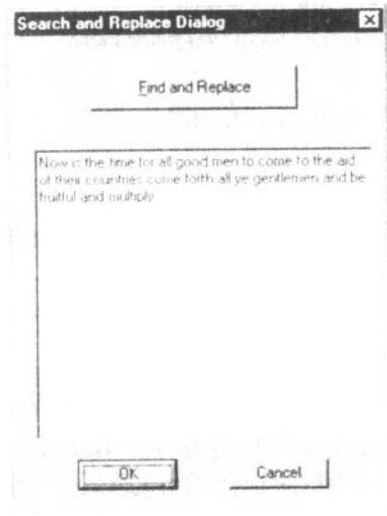


图 9-5 查找与替换对话框

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH95.MAK。进入 AppStudio，创建新的对话框，在对话框中，添加标题为 Find and Replace 的按钮，接受其缺省标识符 IDC_BUTTON1。

2. 在对话框中，添加风格设置为多行并核选 No Hide Sel 的编辑框，接受其缺省标识符 IDC_EDIT1 做为此编辑框的名字。

3. 进入 ClassWizard，为刚创建的模板生成一个新的对话框类。把此类命名为 CSearchAndReplaceDlg，从对象列表中选择对象 CSearchAndReplaceDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function 添加新函数 OnInitDialog，在 CSearchAndReplaceDlg 的方法 OnInitDialog 中添加下面的代码。

```
BOOL CSearchAndReplaceDlg::OnInitDialog ()
{
```

```
    CDialog:: OnInitDialog ();
        //Add some text to the edit list
        CEdit * edit= (CEdit * ) GetDlgItem (IDC_EDIT1);
        edit->SetWindowText (" Now is the time for all good men to come to
                                the aid of their countries");
        CString s;
        edit->GetWindowText (s);
        int length=s.GetLength ();
        edit->SetSel (length, length);
        edit->ReplaceSel (" comeforth all ye gentlemen and be fruitful
                            and multiply");
    return TRUE; //return TRUE unless you set the focus to a control
}
```

4. 从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnFindAndReplace，在 CSearchAndReplaceDlg 的方法 OnFindAndReplace 中添加下面的代码。

```
void CSearchAndReplaceDlg:: OnFindAndReplace ()
{
    dlg=new CFindReplaceDialog;
    dlg->Create (FALSE,"", NULL, FR_DOWN, this);
}
```

5. 把下面一行添加到对话框的构造函数 CSearchAndReplaceDlg:: CSearchAndReplaceDlg 中：

```
WM_FINDREPLACE=:: RegisterWindowMessage (FINDMSGSTRING);
```

同时，把下面一行添加到源文件的顶端：

```
static UINT WM_FINDREPLACE;
```

6. 接下来，把下面一行添加到类 CSearchAndReplaceDlg 的消息映射中。

```
ON_REGISTERED_MESSAGE (WM_FINDREPLACE, OnFileReplace)
```

7. 把下面两个函数添加到类 CSearchAndReplaceDlg 的源文件中。

```
void CSearchAndReplaceDlg:: ReplaceString (const char * string, const char * replace)
{
    CString text;
    CEdit * edit= (CEdit * ) GetDlgItem (IDC_EDIT1);
    edit->GetWindowText (text);
    int idx=text.Find (string);
    if (idx==-1) {//Not found
      MessageBox (" Not Found!"," Error", MB_OK|MB_ICONEXCLAMATION);
      return;
    }
    //Otherwise, set that selection point in the edit box
    edit->SetSel (idx, idx+strlen (string));
    //Now, replace that text with the replacement string
```

```
    edit->ReplaceSel (replace);
}
LRESULT CSearchAndReplaceDlg::OnFindReplace (WPARAM wParam, LPARAM lParam)
{
    //If they don't want to terminate the dialog, check search parameters
    if (! dlg->IsTerminating ()) {
      //See if they just want replace current item
      if (dlg->ReplaceCurrent ()) {
      ReplaceString (dlg->GetFindString (), dlg->GetReplaceString ());
    }
}
return 0L;
}
```

8. 在此类的头文件 SEARCHAN. H 中，添加下面标记为黑体部分的行。

```
class CSearchAndReplaceDlg: public CDialog
{
private:
        CFindReplaceDialog * dlg;
//Construction
public:
    CSearchAndReplaceDlg (CWnd * pParent=NULL); //standard constructor
    void ReplaceString (const char * string, constchar * replace);
//Dialog Data
    //{{AFX_DATA (CSearchAndReplaceDlg)
    enum {IDD=IDD_DIALOG1};
        //NOTE: the ClassWizard will add data members here
    //}}AFX_DATA
//Implementation
protected:
    virtual void DoDataExchange (CDataExchange * pDX); //DDX/DDV support
    //Generated message map functions
    //{{AFX_MSG (CSearchAndReplaceDlg)
    virtual BOOL OnInitDialog ();
    afx_msg void OnFindAndReplace ();
    afx_msg LRESULT OnFindReplace (WPARAM wParam, LPARAM lParam);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
```

9. 接着，重新进入 AppStudio。从菜单列表中选择主菜单对象并添加标题为 Dialog 的菜单，在此菜单上添加标题为 Find and Replace、标识符为 ID_FIND_REPLACE_DLG 的新的菜单项，保存此菜单，退出 AppStudio。

10. 在 ClassWizard 中，从下拉列表中选择对象 CCh95App。从对象列表中选择对象 ID_

FIND _ REPLACE _ DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnFindReplaceDlg，把下面的代码添加到 CCh95App 的方法 OnFindReplaceDlg 中。

```
void CCh95App::OnFindReplaceDlg ()
{
    CSearchAndReplaceDlg dlg;
    dlg.DoModal ();
}
```

11. 把下面的 include 文件行添加到源文件 CH95.CPP 的顶端。

```
#include" searchan.h"
```

12. 编译与运行此例子程序。

用法

在 30 行代码之内就实现了一个完整的查找与替换系统。此例子真正地体现了 Windows API 内在的强大能力并向程序员提供了强有力的功能，以前程序员需要花费几个星期进行设计、开发以及调试的工作，如今已省略成可通过使用 API 函数以及消息系统就能实现的事情。

本节的例子利用编辑控制的消息系统做了大量的工作。当用户选择按钮 Find and Replace 时，便显示一个通用对话框用来提示用户输入要查找与替换的内容，此通用对话框是建立在 Windows 之中并且是十分常见的对话框，程序员是完全不必在这方面花费功夫的。

当用户选择了要查找与替换的字符串后，Windows API 的“魔块”便开始发挥作用了，此时通用对话框向应用程序的对话框发送一个消息，此消息表明要对文本进行查找与替换。那么，是如何完成这一工作，又是如何“知道”要对文本执行一些替换呢？答案在于类 CSearchAndReplaceDlg 的构造函数中调用的函数 RegisterWindowMessage，此 API 函数需要一个消息字符串做为参数并用唯一的号码表示，当此消息被窗口发出后，Windows 将返回此号码。消息映射中的宏 ON _ REGISTERED _ MESSAGE 将“捕捉”此消息，然后便调用方法 OnFindReplace。

在方法 OnFindReplace 中，通用对话框控制的各种选项被核查，本例中，需查找的只是按钮 Replace。当然，这里也可以核查用户是否选择了“区分大小写”以及是否“全字匹配”，不过，为了使程序简单，本例中省去了这些选项。如果对话框没有通知关闭（函数 IsTerminating 用来判定是否关闭对话框），就要检查用户是否选择了按钮 Replace，如果选择了按钮 Replace，则把查找与替换的字符串传送给对话框中的另一个函数 ReplaceString。需要注意的是：在这里，检查要查找的字符串中是否有内容是没有必要的，这是因为通用对话框在用户没有输入所要查找的字符串之前是不允许选择按钮 Replace 的。

在方法 ReplaceString 中，只是把从多行编辑控制中取出的文本存放到一个 CString 变量里。类 CString 包含从文本中查找字符串的方法 Find，通过此方法，确定出字符串在文本中的位置并把它记录下来，然后，此位置用来在利用消息 EM _ SETSEL（在 MFC 中，它被封装在 CEdit 的函数 SetSel 里）的编辑框控制中标记选择区域，接着，以替换的字符串为参数调用封装在函数 ReplaceSel 中的消息 EX _ REPLACESEL。

这就是查找与替换的整个过程。

注释

这种十分有用的方法决不仅限于在 Visual C++ 中使用，下面，就是在 Delphi 中的一个

相同的例子。

1. 创建新的项目文件或在现有的项目文件中添加新的表单，在表单中，添加两个静态文本域与并排的两个编辑域，设定两个静态文本域的标题分别为 Text To Find 与 Text to Replace。

2. 在表单的中央，添加一个编辑框。

3. 在表单的底部，添加两个按钮，设定第一个按钮的标题为 Replace、第二个按钮的标题为 Close。

4. 双击表单，把下面的代码添加到方法 FormCreate 中。

```
procedure TForm1.FormCreate (Sender: TObject);
var
    i: Integer;
    Str: String;
begin
    for i: =1 to 100 do
    begin
      Str: ='This is line'+IntToStr (i);
      Memo1.Lines.Add (Str);
    end;
end;
```

5. 接着，双击标题为 Replace 的按钮，把下面的代码添加到方法 Button1.Click 中（如果它不叫做 Button1 的话，需改动代码中相应的引用）。

```
procedure TForm1.Button1Click (Sender: TObject);
var
flag: Boolean;
position: Integer;
begin
     {Assume best case}
    flag: =True;
     {See if the find string is filled in}
    if (Length (Edit1.Text) =0) then
          begin
            MessageBox (0, 'Must enter a string to find! ', 'Error', 'MB_OK');
            flag: =False;
        end;
    if (flagand (Length (Edit2.Text) =0)) then
        begin
          MessageBox (0, 'Must enter a string to replace! ', 'Error', 'MB_OK');
          flag: =False;
        end;
    if (flag) then
        begin
           {Find the string in the memo field}
```

```
        position: =Pos (Edit1.Text, Memo1.Text);
         {See if it was found}
        if (position=0) then
          begin
            MessageBox (0, 'Notfound', 'Error', 'MB_OK');
            flag: =False;
          end;
      end;
       {We now have the position in position}
      if (flag=True) then
          begin
            Memo1.SelStart: =position-1;
            Memo1.SelLength: =Length (Edit1.Text);
            Memo1.SelText: =Edit2.Text;
          end;
    end;
```

6. 最后，双击表单上的按钮 Close，把下面的代码添加到方法 Button2.Click 中。

```
procedure TForm1.Button2Click (Sender: TObject);
begin
    Close;
end;
```

9.6 显示大于 64KB 的文件

问题

当向用户显示一些非常大的文件时，便会出现一些问题，这是因为 Windows 的编辑控制在未用尽存储空间之前只允许利用大约 32KB 的数据，不过有一些办法可以使得编辑控制能够显示大约 64KB 的数据，但这已经是能够显示的极限了。

利用 Windows 95 中新的 32 位控制，有突破此极限从而显示更大文本文件的方法吗？如果有的话，如何在 MDI 应用程序中利用它呢？

方法

如上所述，有一些方法允许在 Windows 编辑控制中显示较大的文件。但是，其中没有一个方法能显示特别大的文件，也没有一个方法允许把数据的格式编排为一种“很恰当”的模式。

随着 Windows 95 中新的 32 位控制的出现，Microsoft 在 API 的可用控制库中添加了一些新的控制，在这些控制中有一个被称为 RTE (Rich Text Edit) 的控制，此控制提供了在编辑控制中为每一个字符或段落定义格式信息的功能，它可以控制黑体、斜体或为文本添加下划线（以及删除下划线）并可以控制段落的居右、居左或居中的方式。

此控制另一方面作用是：它虽然表现起来同普通编辑控制基本相同，但却有一个重要的差别，RTE 控制比普通文本编辑控制能够容纳大得多的信息，因此它显示一个非常大的文件是毫无问题的。

本节中，将基于新的 RTE 控制创建一个 MFC 视图，并介绍如何利用此视图来允许用户

通过标记文本与选择菜单项，随意地修改编辑控制中的格式信息。

步骤

打开与运行按照如下步骤生成的例子程序 WORDRET.MAK。此时，一个新的 MDI 窗口便会在例子程序中出现，在此 MDI 窗口中输入文本直到有足够的文本用于测试时为止。利用鼠标从输入的文本中选择一部分并把它标记下来，接着从主菜单中选择菜单项 Format，并从其下拉菜单列表选项中选择菜单项 Character，此菜单便产生出含有几个选项的另一个菜单，选择其中的菜单项 Italic，便会看到被选择的文本以斜体的方式显示出来，如图 9-6 所示。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visusl C++中，创建项目文件 WORDRTE.MAK，利用 CRTFDoc 与 CRTFView 作为文档与视图的名字，保存此项目文件的 make 文件。

2. 在 Visusl C++中创建新的文件，把下面的代码添加到此文件中，保存此文件为 RTFVIEW.CPP，它将作为 RTE 控制视图的新的基视图类，在其他应用程序中，可能会经常用到它。

```
#include " stdafx.h"
#include " rtfview.h"
#include<ctype.h>
#ifdef AFX_CORE4_SEG
#pragma code_seg (AFX_CORE4_SEG)
#endif
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE [] = _FILE_;
#endif
////////////////////////////////////////////////////////////////////////////////////////////////
//CRTFEditView
#define new DEBUG_NEW
static const UINT nMsgFindReplace = :: RegisterWindowMessage (FINDMSGSTRING);
#ifdef _UNICODE
staticHFONT hUnicodeFont;
struct _AFX_EDITVIEW_TERM
{
    ~_AFX_EDITVIEW_TERM ()
    {
        AfxDeleteObject ( (HGDIOBJ * ) &hUnicodeFont);
    }
};
static const _AFX_EDITVIEW_TERM editviewTerm;
#endif
BEGIN_MESSAGE_MAP (CRTFEditView, CView)
    //{{AFX_MSG_MAP (CRTFEditView)
    ON_WM_CREATE ()
```

```
	ON_WM_PAINT()
	ON_EN_CHANGE(AFX_IDW_PANE_FIRST, OnEditChange)
	ON_UPDATE_COMMAND_UI(ID_EDIT_CUT, OnUpdateNeedSel)
	ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateNeedClip)
	ON_UPDATE_COMMAND_UI(ID_EDIT_SELECT_ALL, OnUpdateNeedText)
	ON_UPDATE_COMMAND_UI(ID_EDIT_UNDO, OnUpdateEditUndo)
	ON_COMMAND(ID_EDIT_CUT, OnEditCut)
	ON_COMMAND(ID_EDIT_COPY, OnEditCopy)
	ON_COMMAND(ID_EDIT_PASTE, OnEditPaste)
	ON_COMMAND(ID_EDIT_CLEAR, OnEditClear)
	ON_COMMAND(ID_EDIT_UNDO, OnEditUndo)
	ON_COMMAND(ID_EDIT_SELECT_ALL, OnEditSelectAll)
	ON_UPDATE_COMMAND_UI(ID_EDIT_FIND, OnUpdateNeedText)
	ON_UPDATE_COMMAND_UI(ID_EDIT_REPLACE, OnUpdateNeedText)
	ON_COMMAND(ID_EDIT_FIND, OnEditFind)
	ON_COMMAND(ID_EDIT_REPLACE, OnEditReplace)
	ON_UPDATE_COMMAND_UI(ID_EDIT_REPEAT, OnUpdateNeedFind)
	ON_COMMAND(ID_EDIT_REPEAT, OnEditRepeat)
	ON_UPDATE_COMMAND_UI(ID_EDIT_COPY, OnUpdateNeedSel)
	ON_UPDATE_COMMAND_UI(ID_EDIT_CLEAR, OnUpdateNeedSel)
	//}} AFX_MSG_MAP
	ON_REGISTERED_MESSAGE(nMsgFindReplace, OnFindReplaceCmd)
	//Standard Print commands (print only--not preview)
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
END_MESSAGE_MAP()
const AFX_DATADEF DWORD CRTFEditView::dwStyleDefault=
	AFX_WS_DEFAULT_VIEW|
	WS_HSCROLL|WS_VSCROLL|
	ES_AUTOHSCROLL|ES_AUTOVSCROLL|
	ES_MULTILINE|ES_NOHIDESEL;
	const AFX_DATADEF UINT CRTFEditView::nMaxSize=64*1024;
//class name for control creation
static const TCHAR szClassName[]=_T("RICHEDIT");
/////////////////////////////////////////////////////////////////////////////
//AFX_EDIT_STATE implementation
AFX_EDIT_STATE::AFX_EDIT_STATE()
{
	//Note: it is only necessary to intialize non-zero data.
	bNext=TRUE;
}
AFX_EDIT_STATE::~AFX_EDIT_STATE()
{
}
```

```
/////////////////////////////////////////////////////////////////////////////
//CRTFEditView construction/destruction
CRTFEditView:: CRTFEditView ()
{
    m_pShadowBuffer=NULL;
    m_nShadowSize=0;
}
CRTFEditView:: ~CRTFEditView ()
{
    ASSERT (m_hWnd==NULL);
    ASSERT (m_pShadowBuffer==NULL||afxData.bWin32s);
    delete m_pShadowBuffer;
}
BOOL CRTFEditView:: PreCreateWindow (CREATESTRUCT& cs)
{
    ASSERT (cs.lpszClass==NULL);
    cs.lpszClass=szClassName;
    //map default CView style to default CRTFEditView style
    if (cs.style==AFX_WS_DEFAULT_VIEW)
        cs.style=dwStyleDefault;
    return CView:: PreCreateWindow (cs);
}
int CRTFEditView:: OnCreate (LPCREATESTRUCT lpcs)
{
    if (CView:: OnCreate (lpcs)! =0)
      return-1;
    GetEditCtrl () .LimitText (nMaxSize);
    return 0;
}
//EDIT controls always turn off WS_BORDER and draw it themselves
void CRTFEditView:: CalcWindowRect (LPRECT lpClientRect, UINT nAdjustType)
{
    if (nAdjustType! =0)
    {
        //default behavior for in-place editing handless crollbars
        DWORD dwStyle=GetStyle ();
        if (dwStyle & WS_VSCROLL)
            lpClientRect->right-=afxData.cxVScroll-CX_BORDER;
        if (dwStyle&WS_HSCROLL)
            lpClientRect->bottom+=afxData.cyHScroll-CY_BORDER;
    return;
    }
    :: AdjustWindowRectEx (lpClientRect, GetStyle () WS_BORDER, FALSE,
```

```
        GetExStyle () &~ (WS_EX_CLIENTEDGE));
}
//////////////////////////////////////////////////////////////////////////////
//CRTFEditView document like functions
CRTFEdit& CRTFEditView:: GetEditCtrl () const
{
    return * (CRTFEdit * ) this;
}
void CRTFEditView:: DeleteContents ()
{
    ASSERT_VALID (this);
    ASSERT (m_hWnd! =NULL);
    SetWindowText (NULL);
    ASSERT_VALID (this);
}
//////////////////////////////////////////////////////////////////////////////
//CRTFEditView drawing
void CRTFEditView:: OnPaint ()
{
    //do not call CView:: OnPaint since it will call OnDraw
    CWnd:: OnPaint ();
}
void CRTFEditView:: OnDraw (CDC * )
{
  //do nothing here since CWnd:: OnPaint () will repaint the EDIT control
}
//////////////////////////////////////////////////////////////////////////////
//CRTFEditView Pringting Suppor+
//////////////////////////////////////////////////////////////////////////////
//CRTFEditViewcommands
void CRTFEditView:: OnUpdateNeedSel (CCmdUI * pCmdUI)
{
    ASSERT_VALID (this);
    int nStartChar, nEndChar;
    GetEditCtrl () .GetSel (nStartChar, nEndChar);
    pCmdUI-->Enable (nStartChar! =nEndChar);
    ASSERT_VALID (this);
}
void CRTFEditView:: OnUpdateNeedClip (CCmdUI * pCmdUI)
{
    ASSERT_VALID (this);
    pCmdUI->Enable (:: IsClipboardFormatAvailable (CF_TEXT));
    ASSERT_VALID (this);
```

```
}
void CRTFEditView::OnUpdateNeedText (CCmdUI * pCmdUI)
{
      ASSERT_VALID (this);
      pCmdUI->Enable (GetWindowTextLength ()! =0);
      ASSERT_VALID (this);
}
void CRTFEditView::OnUpdateNeedFind (CCmdUI * pCmdUI)
{
      ASSERT_VALID (this);
      AFX_EDIT_STATE * pEditState=AfxGetEditState ();
      pCmdUI->Enable (GetWindowTextLength ()! =0&&
          ! pEditState->strFind.IsEmpty ());
      ASSERT_VALID (this);
}
void CRTFEditView::OnUpdateEditUndo (CCmdUI * pCmdUI)
{
      ASSERT_VALID (this);
      pCmdUI->Enable (GetEditCtrl () .CanUndo ());
      ASSERT_VALID (this);
}
void CRTFEditView::OnEditChange ()
{
      ASSERT_VALID (this);
      GetDocument () ->SetModifiedFlag ();
      ASSERT_VALID (this);
}
void CRTFEditView::OnEditCut ()
{
      ASSERT_VALID (this);
      GetEditCtrl () .Cut ();
      ASSERT_VALID (this);
}
void CRTFEditView::OnEditCopy ()
{
      ASSERT_VALID (this);
      GetEditCtrl () .Copy ();
      ASSERT_VALID (this);
}
void CRTFEditView::OnEditPaste ()
{
      ASSERT_VALID (this);
      GetEditCtrl () .Paste ();
```

```
	ASSERT_VALID (this);
}
void CRTFEditView::OnEditClear ()
{
	ASSERT_VALID (this);
	GetEditCtrl () .Clear ();
	ASSERT_VALID (this);
}
void CRTFEditView::OnEditUndo ()
{
	ASSERT_VALID (this);
	GetEditCtrl () .Undo ();
	ASSERT_VALID (this);
}
void CRTFEditView::OnEditSelectAll ()
{
	ASSERT_VALID (this);
	GetEditCtrl () .SetSel (0, -1);
	ASSERT_VALID (this);
}
//////////////////////////////////////////////////////////////////////////////////////////
//CRTFEditView attributes
LPCTSTR CRTFEditView::LockBuffer () const
{
	ASSERT_VALID (this);
	ASSERT (m_hWnd! =NULL);
	if (afxData. bWin32s)
	{
	  //under Win32s, it is necessary to maintain a shadow buffer.
	  //It is only updated when the control contents have been changed.
	  if (m_pShadowBuffer==NULL||GetEditCtrl () .GetModify ())
	   {
		ASSERT (m_pShadowBuffer! =NULL||m_nShadowSize==0);
		UINT nSize=GetWindowTextLength () +1;
		if (nSize>m_nShadowSize)
		 {
		  //need more room for shadow buffer
		  CRTFEditView * pThis= (CRTFEditView * ) this;
		  delete m_pShadowBuffer;
		  pThis->m_pShadowBuffer=NULL;
		  pThis->m_pShadowBuffer=new TCHAR [nSize];
		  pThis->m_nShadowSize=nSize;
		}
```

```
            //update the shadow buffer with GetWindowText
            ASSERT (m_nShadowSize>=nSize);
            ASSERT (m_pShadowBuffer != NULL);
            GetWindowText (m_pShadowBuffer, nSize);
            //turn off edit control's modify bit
            GetEditCtrl () .SetModify (FALSE);
        }
        return m_pShadowBuffer;
    }
    //else--running under non-subset Win32 system
    HLOCAL hLocal=GetEditCtrl () .GetHandle ();
    ASSERT (hLocal! =NULL);
    LPCTSTR lpszText= (LPCTSTR) LocalLock (hLocal);
    ASSERT (lpszText! =NULL);
    ASSERT_VALID (this);
    return lpszText;
}
void CRTFEditView:: UnlockBuffer () const
{
    ASSERT_VALID (this);
    ASSERT (m_hWnd! =NULL);
    if (afxData.bWin32s)
        return;
    HLOCAL hLocal=GetEditCtrl () .GetHandle ();
    ASSERT (hLocal! =NULL);
    LocalUnlock (hLocal);
}
//this function returns the length in characters
UINT CRTFEditView:: GetBufferLength () const
{
    ASSERT_VALID (this);
    ASSERT (m_hWnd! =NULL);
    LPCTSTR lpszText=LockBuffer ();
    UINT nLen=lstrlen (lpszText);
    UnlockBuffer ();
    return nLen;
}
void CRTFEditView:: GetSelectedText (CString& strResult) const
{
    ASSERT_VALID (this);
    LPSTR lpszText;
    GetEditCtrl () .GetSelText (lpszText);
    strResult=lpszText;
```

```
        ASSERT_VALID (this);
}
///////////////////////////////////////////////////////////////////////////////
//CRTFEditView Find & Replace
void CRTFEditView:: OnEditFind ()
{
   ASSERT_VALID (this);
        OnEditFindReplace (TRUE);
        ASSERT_VALID (this);
    }
void CRTFEditView:: OnEditReplace ()
{
        ASSERT_VALID (this);
        OnEditFindReplace (FALSE);
        ASSERT_VALID (this);
}
void CRTFEditView:: OnEditRepeat ()
{
        ASSERT_VALID (this);
        AFX_EDIT_STATE * pEditState=AfxGetEditState ();
        if (! FindText (pEditState->strFind,
              pEditState->bNext,
              pEditState->bCase))
       {
              OnTextNotFound (pEditState->strFind);
        }
        ASSERT_VALID (this);
}
void CRTFEditView:: OnEditFindReplace (BOOL bFindOnly)
{
        ASSERT_VALID (this);
        AFX_EDIT_STATE * pEditState=AfxGetEditState ();
        if (pEditState->pFindReplaceDlg! =NULL)
        {
          if (pEditState->bFindOnly= =bFindOnly)
           {
            pEditState->pFindReplaceDlg->SetActiveWindow ();
            pEditState->pFindReplaceDlg->ShowWindow (SW_SHOW);
            return;
          }
          else
           {
            ASSERT (pEditState->bFindOnly! =bFindOnly);
```

```
            pEditState->pFindReplaceDlg->SendMessage (WM_CLOSE);
            ASSERT (pEditState->pFindReplaceDlg==NULL);
            ASSERT_VALID (this);
          }
        }
        CString strFind;
        GetSelectedText (strFind);
        if (strFind.IsEmpty ())
            strFind=pEditState->strFind;
        CString strReplace=pEditState->strReplace;
        pEditState->pFindReplaceDlg=new CFindReplaceDialog;
        ASSERT (pEditState->pFindReplaceDlg! =NULL);
        DWORD dwFlags=FR_HIDEWHOLEWORD;
        if (pEditState->bNext)
          dwFlags|=FR_DOWN;
        if (pEditState->bCase)
          dwFlags|=FR_MATCHCASE;
        if (! pEditState->pFindReplaceDlg->Create (bFindOnly, strFind,
          strReplace, dwFlags, this))
         {
          pEditState->pFindReplaceDlg=NULL;
          ASSERT_VALID (this);
          return;
        }
        ASSERT (pEditState->pFindReplaceDlg! =NULL);
        pEditState->bFindOnly=bFindOnly;
        ASSERT_VALID (this);
}
void CRTFEditView:: OnFindNext (LPCTSTR lpszFind, BOOL bNext, BOOL bCase)
{
        ASSERT_VALID (this);
        AFX_EDIT_STATE * pEditState=AfxGetEditState ();
        pEditState->strFind=lpszFind;
        pEditState->bCase=bCase;
        pEditState->bNext=bNext;
        if (! FindText (pEditState->strFind, bNext, bCase))
          OnTextNotFound (pEditState->strFind);
        ASSERT_VALID (this);
}
void CRTFEditView:: OnReplaceSel (LPCTSTR lpszFind, BOOL bNext, BOOL bCase,
        LPCTSTR lpszReplace)
{
        ASSERT_VALID (this);
```

```
  AFX_EDIT_STATE * pEditState=AfxGetEditState ();
      pEditState->strFind=lpszFind;
      pEditState->strReplace=lpszReplace;
      pEditState->bCase=bCase;
      pEditState->bNext=bNext;
      if (! InitializeReplace ())
          return;
      GetEditCtrl () .ReplaceSel (pEditState->strReplace);
      FindText (pEditState->strFind, bNext, bCase);
      ASSERT_VALID (this);
}
void CRTFEditView:: OnReplaceAll (LPCTSTR lpszFind, LPCTSTR lpszReplace, BOOL bCase)
{
      ASSERT_VALID (this);
      AFX_EDIT_STATE * pEditState=AfxGetEditState ();
      pEditState->strFind=lpszFind;
      pEditState->strReplace=lpszReplace;
      pEditState->bCase=bCase;
      pEditState->bNext=TRUE;
      if (! InitializeReplace () &&
          ! SameAsSelected (pEditState->strFind, pEditState->bCase))
      {
          //initial find was not successful
          return;
      }
      do
       {
          GetEditCtrl () .ReplaceSel (pEditState->strReplace);
      } while (FindText (pEditState->strFind, 1, bCase));
      ASSERT_VALID (this);
}
BOOL CRTFEditView:: InitializeReplace ()
          //helper to do find first if no selection
{
          ASSERT_VALID (this);
          AFX_EDIT_STATE * pEditState=AfxGetEditState ();
          //do find next if no selection
          int nStartChar, nEndChar;
          Get EditCtrl () .GetSel (nStartChar, nEndChar);
          if (nStartChar==nEndChar)
           {
                if (! FindText (pEditState->strFind, pEditState->bNext,
                        pEditState->bCase))
```

```
            {
                //text not found
                OnTextNotFound (pEditState->strFind);
            }
            returnFALSE;
    }
    if (! SameAsSelected (pEditState->strFind, pEditState->bCase))
{
            if (! FindText (pEditState->strFind, pEditState->bNext,
            pEditState->bCase))
            {
            //text not found
            OnTextNotFound (pEditState->strFind);
            }
            return FALSE;
    }
    ASSERT_VALID (this);
    return TRUE;
}
LRESULT CRTFEditView:: OnFindReplaceCmd (WPARAM, LPARAM lParam)
{
    ASSERT_VALID (this);
    AFX_EDIT_STATE * pEditState=AfxGetEditState ();
    CFindReplaceDialog * pDialog=CFindReplaceDialog:: GetNotifier (lParam);
    ASSERT (pDialog! =NULL);
    ASSERT (pDialog==pEditState->pFindReplaceDlg);
    if (pDialog->IsTerminating ())
     {
        pEditState->pFindReplaceDlg=NULL;
    }
    else if (pDialog->FindNext ())
     {
        OnFindNext (pDialog->GetFindString (),
            pDialog->SearchDown (), pDialog->MatchCase ());
    }
    else if (pDialog->ReplaceCurrent ())
     {
        ASSERT (! pEditState->bFindOnly);
        OnReplaceSel (pDialog->GetFindString (),
            pDialog->SearchDown (), pDialog->MatchCase (),
            pDialog->GetReplaceString ());
    }
    else if (pDialog->ReplaceAll ())
```

```
        {
          ASSERT (! pEditState->bFindOnly);
          OnReplaceAll (pDialog->GetFindString (), pDialog->GetReplaceString (),
          pDialog->MatchCase ());
        }
        ASSERT _ VALID (this);
        return0;
}
typedef int (WINAPI * AFX _ COMPARE _ PROC) (LPCTSTR str1, LPCTSTR str2);
BOOL CRTFEditView:: SameAsSelected (LPCTSTR lpszCompare, BOOL bCase)
{
        //check length first
        size _ t nLen=lstrlen (lpszCompare);
        int nStartChar, nEndChar;
        GetEditCtrl () .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 CRTFEditView:: FindText (LPCTSTR lpszFind, BOOL bNext, BOOL bCase)
{
        ASSERT _ VALID (this);
        ASSERT (lpszFind! =NULL);
        ASSERT (* lpszFind! ='\0');
        UINT nLen=GetBufferLength ();
        int nStartChar, nEndChar;
        GetEditCtrl () .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;
        BeginWaitCursor ();
        LPCTSTR lpszText=LockBuffer ();
        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_tnLenFind=lstrlen (lpszFind);
    if (nStart+nLenFind-1>=nLen)
     {
      if (iDir<0 && nLen>=nLenFind)
       {
        if (_afxDBCS)
         {
          //walk back to previous character n times
          nStart=nLen;
          int n=nLenFind;
          while (n--)
           {
              nStart-= (lpszText+nStart) -
                        _tcsdec (lpszText, lpszText+nStart);
            }
          }
          else
             {
                //single-byte character set is easy and fast
                nStart=nLen-nLenFind;
             }
    ASSERT (nStart+nLenFind-1<=nLen);
    }
   else
   {
       UnlockBuffer ();
       EndWaitCursor ();
       return FALSE;
   }
   }
//start the search at nStart
LPCTSTR lpsz=lpszText+nStart;
   AFX_COMPARE_PROC pfnCompare=bCase? lstrcmp: lstrcmpi;
if (_afxDBCS)
```

```
{
    //double-byte string search
    LPCTSTR lpszStop;
    if (iDir>0)
    {
            //start at current and find _first _occurrance
            lpszStop=lpszText+nLen-nLenFind+1;
    }
    else
    {
            //start at top and find _last _occurrance
            lpszStop=lpsz;
            lpsz=lpszText;
    }
    LPCTSTR lpszFound=NULL;
    while (lpsz<=lpszStop)
    {
         if (*lpsz==*lpszFind &&
              (! _istlead (*lpsz) ||lpsz [1] ==lpszFind [1]))
        {
              LPTSTR lpch= (LPTSTR) (lpsz+nLenFind);
              TCHAR chSave= *lpch;
        *lpch='\0';
        int nResult= (*pfnCompare) (lpsz, lpszFind);
        *lpch=chSave;
        if (nResult==0)
        {
                lpszFound=lpsz;
                if (iDir>0)
                  break;
            }
    }
    lpsz=_tcsinc (lpsz);
}
UnlockBuffer ();
    if (lpszFound! =NULL)
    {
         int n= (int) (lpszFound-lpszText);
         GetEditCtrl () .SetSel (n, n+nLenFind);
         EndWaitCursor ();
         return TRUE;
    }
}
```

```
else
{
    //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);
            GetEditCtrl () .SetSel (n, n+nLenFind);
            ASSERT_VALID (this);
            EndWaitCursor ();
            return TRUE;
        }
        //restore character at end of search
        *lpch=chSave;
        //move on to next substring
        nCompare--;
        lpsz+=iDir;
        }
        UnlockBuffer ();
    }
    ASSERT_VALID (this);
    EndWaitCursor ();
    return FALSE;
}
void CRTFEditView:: OnTextNotFound (LPCTSTR)
{
ASSERT_VALID (this);
MessageBeep (0);
    }
```

```
#undef new
#ifdef AFX_INIT_SEG
#pragma code_seg (AFX_INIT_SEG)
#endif
#ifndef _WINDLL
AFX_DATADEF AFX_EDIT_STATE _afxEditState;
#endif
IMPLEMENT_DYNCREATE (CRTFEditView, CView)
////////////////////////////////////////////////////////////////////////////////
```

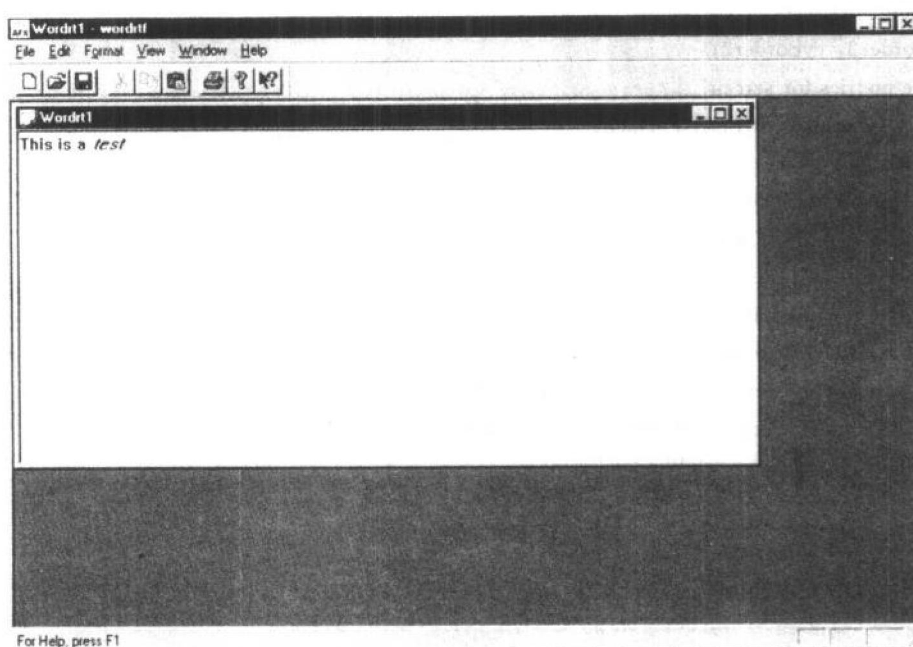


图 9-6 显示在 RTE 控制窗口中的斜体文本

3. 在 Visusl C++中创建新的文件，把下面的代码添加到此文件中。保存此文件为 RTFVIEW.H。

```
//rtfview.h—header file for CRTFEditView
#include" crtfedit.h"
#ifndef __RTFVIEW_H__
#define __RTFVIEW_H__
#undef AFX_DATA
#define AFX_DATA AFX_CORE_DATA
#define CX_BORDER 1
#define CY_BORDER 1
#defineWS_EX_CLIENTEDGE              0x00000200L
//UNICODE/MBCS abstractions
#ifdef _MBCS
```

```
        extern const BOOL _afxDBCS;
    #else
        #define _afxDBCS FALSE
    #endif
////////////////////////////////////////////////////////////////////////////
//Auxiliary System/Screen metrics
struct AUX_DATA
{
    //system metrics
    int cxVScroll, cyHScroll;
    int cxIcon, cyIcon;
    int cxBorder2, cyBorder2;
    //device metrics for screen
    int cxPixelsPerInch, cyPixelsPerInch;
    int cySysFont;
    //solid brushes with convenient gray colors and system colors
    HBRUSH hbrLtGray, hbrDkGray;
    HBRUSH hbrBtnHilite, hbrBtnFace, hbrBtnShadow;
    HBRUSH hbrWindowFrame;
    HPEN hpenBtnHilite, hpenBtnShadow, hpenBtnText;
    //color values of system colors used for CToolBar
    COLORREF clrBtnFace, clrBtnShadow, clrBtnHilite;
    COLORREF clrBtnText, clrWindowFrame;
    //standard cursors
    HCURSOR hcurWait;
    HCURSOR hcurArrow;
    HCURSOR hcurHelp;            //cursor usedin Shift+F1 help
    //special GDI objects allocated on demand
    HFONT hStatusFont;
    HFONT hToolTipsFont;
    HBITMAP hbmMenuDot;
    //othersysteminformation
    UINT  nWinVer; //Major.Minor version numbers
    BOOL  bWin32s; //TRUE if Win32s (or Windows 95)
    BOOL  bWin4; //TRUE if Windows 4.0
    BOOL  bNotWin4; //TRUE if not Windows 4.0
    BOOL  bSmCaption; //TRUE if WS_EX_SMCAPTION is supported
    BOOL  bWin31; //TRUE if actuallyWin32 son Windows 3.1
    BOOL  bMarked4; //TRUE if marked as 4.0
    //special Windows API entry points
    int (WINAPI * pfnSetScrollInfo) (HWND, int, LPCSCROLLINFO, BOOL);
    BOOL (WINAPI * pfnGetScrollInfo) (HWND, int, LPSCROLLINFO);
//Implementation
```

```
    AUX_DATA ();~AUX_DATA ();
    void UpdateSysColors ();
    void UpdateSysMetrics ();
};
    extern AFX_DATA AUX_DATA afxData;
    void AFXAPI AfxDeleteObject (HGDIOBJ * pObject);
/////////////////////////////////////////////////////////////////////////////
//CRTFEditView-simple text editor view
class CRTFEditView: public CView
{
    DECLARE_DYNCREATE (CRTFEditView)
//Construction
public:
    CRTFEditView ();
    static AFX_DATA const DWORD dwStyleDefault;
//Attributes
public:
    //CRTFEdit control access
    CRTFEdit& GetEditCtrl () const;
    //other attributes
    void GetSelectedText (CString& strResult) const;
//Operations
public:
    BOOL FindText (LPCTSTR lpszFind, BOOL bNext=TRUE, BOOL bCase=TRUE);
//Overrideables
protected:
    virtual void OnFindNext (LPCTSTR lpszFind, BOOL bNext, BOOL bCase);
    virtual void OnReplaceSel (LPCTSTR lpszFind, BOOL bNext, BOOL bCase,
              LPCTSTR lpszReplace);
    virtual void OnReplaceAll (LPCTSTR lpszFind, LPCTSTR lpszReplace,
              BOOL bCase);
    virtual void OnTextNotFound (LPCTSTR lpszFind);
//Implementation
public:
    virtual~CRTFEditView ();
    virtual void OnDraw (CDC * pDC);
    virtual void DeleteContents ();
    static AFX_DATA const UINT nMaxSize;
            //maximum number of characters supported
protected:
    LPTSTR m_pShadowBuffer;          //special shadow buffer only used in Win32s
    UINT m_nShadowSize;
    CUIntArray m_aPageStart;       //array of starting pages
```

```
    //construction
    virtual BOOL PreCreateWindow (CREATESTRUCT& cs);
    //find & replace support
    void OnEditFindReplace (BOOL bFindOnly);
    BOOL InitializeReplace ();
    BOOL SameAsSelected (LPCTSTR lpszCompare, BOOL bCase);
    //buffer access
    LPCTSTR LockBuffer () const;
    void UnlockBuffer () const;
    UINT GetBufferLength () const;
    //special overrides for implementation
    virtual void CalcWindowRect (LPRECT lpClientRect,
        UINT nAdjustType=adjustBorder);
    //{{AFX_MSG (CRTFEditView)
    afx_msg int OnCreate (LPCREATESTRUCTlpCreateStruct);
    afx_msg void OnPaint ();
    afx_msg void OnUpdateNeedSel (CCmdUI * p CmdUI);
    afx_msg void OnUpdateNeedClip (CCmdUI * p CmdUI);
    afx_msg void OnUpdateNeedText (CCmdUI * p CmdUI);
    afx_msg void OnUpdateNeedFind (CCmdUI * p CmdUI);
    afx_msg void OnUpdateEditUndo (CCmdUI * p CmdUI);
    afx_msg void OnEditChange ();
    afx_msg void OnEditCut ();
    afx_msg void OnEditCopy ();
    afx_msg void OnEditPaste ();
    afx_msg void OnEditClear ();
    afx_msg void OnEditUndo ();
    afx_msg void OnEditSelectAll ();
    afx_msg void OnEditFind ();
    afx_msg void OnEditReplace ();
    afx_msg void OnEditRepeat ();
    afx_msg LRESULT OnFindReplaceCmd (WPARAM wParam, LPARAM lParam);
    //}} AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
#endif//__RTFEDIT_H_
```

4. 接下来，选择CRTFView的源文件WORDRVW.CPP，并把下面的代码添加到此文件的底部。

```
void CRTFView:: OnInitialUpdate ()
{
    CRTFEditView:: OnInitialUpdate ();
    //Force normal page size
    HDC hDC;
```

```
    CPrintDialog prtDlg (FALSE, PD_ALLPAGES|PD_USEDEVMODECOPIES|
                        PD_NOPAGENUMS|PD_HIDEPRINTTOFILE|
                        PD_NOSELECTION, NULL);
    prtDlg.GetDefaults ();
    hDC=prtDlg.m_pd.hDC;
    if (! hDC)
    {
        //Printer DC Failed, Create one from the Display
        hDC=CreateCompatibleDC (NULL);
    }
    if (hDC)
    {
        if (! GetEditCtrl () .SetTargetDevice (hDC, ( (14400*85) /10)))
        {
            DeleteDC (hDC);
        }
        else
        {
            m_hTargetDC=hDC;
        }
    }
    //Enable Notification Messages
    DWORD dwMask;
    dwMask=GetEditCtrl () .GetEventMask ();
    dwMask|=ENM_CHANGE|ENM_SELCHANGE;
    GetEditCtrl () .SetEventMask (dwMask);
}
```

5. 进入视图类的头文件 WORDRVW.H，把下面的代码添加到此文件中。

```
virtual void OnInitialUpdate ();
```

6. 进入 AppStudio，从菜单列表中选择 MDI 菜单，在此菜单中添加标题为 Format 的菜单，在此菜单中添加两个标题分别为 Character 和 Paragraph 的菜单项，把 Character 与 Paragraph 设定为下拉菜单。在 Character 菜单中，添加标题分别为 Bold、Italic 与 Underline 的三个新的下拉菜单项，把其标识符分别设定为 ID_FORMAT_BOLD、ID_FORMAT_ITALIC 与 ID_FORMAT_UNDERLINE。在 Paragraph 菜单中，添加标题分别为 Right、Left、Center 与 Bullet 的四个新的菜单项，把其标识符分别设定为 ID_FORMAT_RIGHT、ID_FORMAT_LEFT、ID_FORMAT_CENTER 与 ID_FORMAT_BULLET。保存此菜单，然后退出 AppStudio。

7. 进入 ClassWizard，从下拉列表中选择对象 CRTFView，从对象列表中选择对象 ID_FORMAT_BOLD，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的方法 OnFormatBold。把下面的代码添加到 CRTFView 的方法 OnFormatBold 中。

```
void CRTFView:: OnFormatBold ()
{
```

```
    CHARFORMAT cf;
    cf.cbSize=sizeof (cf);
    GetEditCtrl () .GetCharFormat (TRUE, &cf);
    if (cf.dwMask&CFM _ BOLD)
          cf.dwEffects^ =CFE _ BOLD;
    else
          cf.dwEffects|=CFE _ BOLD;
    //Only look at BOLD bit
    cf.dwMask=CFM _ BOLD;
    //Set the status for the selected text
    GetEditCtrl () .SetCharFormat (SCF _ SELECTION, &cf);
}
```

8. 进入 ClassWizard，从下拉列表中选择对象 CRTFView，从对象列表中选择对象 ID_FORMAT_ITALIC，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的方法 OnFormatItalic。把下面的代码添加到 CRTFView 的方法 OnFormatItalic 中。

```
void CRTFView:: OnFormatItalic ()
{
    CHARFORMAT cf;
    cf.cbSize=sizeof (cf);
    GetEditCtrl () .GetCharFormat (TRUE, &cf);
    if (cf.dwMask&CFM _ ITALIC)
          cf.dwEffects^ =CFE _ ITALIC;
    else
          cf.dwEffects|=CFE _ ITALIC;
    cf.dwMask=CFM _ ITALIC;
    GetEditCtrl () .SetCharFormat (SCF _ SELECTION, &cf);
}
```

9. 进入 ClassWizard，从下拉列表中选择对象 CRTFView，从对象列表中选择对象 ID_FORMAT_UNDERLINE，从消息列表中选择消息 COMMAND，点击按钮 AddFunction 添加新的方法 OnFormatUnderline。把下面的代码添加到 CRTFView 的方法 OnFormatUnderline 中。

```
voidCRTFView:: OnFormatUnderline ()
{
    CHARFORMAT cf;
    cf.cbSize=sizeof (cf);
    GetEditCtrl () .GetCharFormat (TRUE, &cf);
    if (cf.dwMask & CFM _ UNDERLINE)
          cf.dwEffects^ =CFE _ UNDERLINE;
    else
          cf.dwEffects|=CFE _ UNDERLINE;
    cf.dwMask=CFM _ UNDERLINE;
    GetEditCtrl () .SetCharFormat (SCF _ SELECTION, &cf);
```

}

10. 进入ClassWizard，从下拉列表中选择对象CRTFView，从对象列表中选择对象ID _ FORMAT _ RIGHT，从消息列表中选择消息COMMAND，点击按钮Add Function添加新的方法OnFormatRight。把下面的代码添加到CRTFView的方法OnFormatRight中。

```
void CRTFView:: OnFormatRight ()
{
PARAFORMAT pf;
pf.cbSize=sizeof (pf);
pf.dwMask=PFM _ ALIGNMENT;
pf.wAlignment=PFA _ RIGHT;
GetEditCtrl () .SetParaFormat (&pf);
}
```

11. 进入ClassWizard，从下拉列表中选择对象CRTFView，从对象列表中选择对象ID _ FORMAT _ LEFT，从消息列表中选择消息COMMAND，点击按钮Add Function添加新的方法OnFormatLeft。把下面的代码添加到CRTFView的方法OnFormatLeft中。

```
void CRTFView:: OnFormatLeft ()
{
    PARAFORMAT pf;
    pf.cbSize=sizeof (pf);
    pf.dwMask=PFM _ ALIGNMENT;
    pf.wAlignment=PFA _ LEFT;
    GetEditCtrl () .SetParaFormat (&pf);
}
```

12. 进入ClassWizard，从下拉列表中选择对象CRTFView，从对象列表中选择对象ID _ FORMAT _ CENTER，从消息列表中选择消息COMMAND，点击按钮Add Function添加新的方法OnFormatCenter。把下面的代码添加到CRTFView的方法OnFormatCenter中。

```
void CRTFView:: OnFormatCenter ()
{
    PARAFORMAT pf;
    pf.cbSize=sizeof (pf);
    pf.dwMask=PFM _ ALIGNMENT;
    pf.wAlignment=PFA _ CENTER;
    GetEditCtrl () .SetParaFormat (&pf);
}
```

13. 进入ClassWizard，从下拉列表中选择对象CRTFView，从对象列表中选择对象ID _ FORMAT _ BULLET，从消息列表中选择消息COMMAND，点击按钮Add Function添加新的方法OnFormatBullet。把下面的代码添加到CRTFView的方法OnFormatBullet中。

```
void CRTFView:: OnFormatBullet ()
{
    PARAFORMAT pf;
    pf.cbSize=sizeof (pf);
```

```
    GetEditCtrl () .GetParaFormat (&pf);
    if (pf.dwMask & PFM_NUMBERING)
        pf.wNumbering^ =PFN_BULLET;
    else
        pf.wNumbering =PFN_BULLET;
    //Only bulleted bit
    pf.dwMask=PFM_NUMBERING;
    GetEditCtrl () .SetParaFormat (&pf);
}
```

14. 编译与运行此例子程序。

用法

解释类 CRTFEditView 的用法已超出本书讨论的范围，此类是 MFC 中类 CEditView 的一个无性系并以与其完全类似的方式工作，事实上，它是依照类 CEditView 仿造出来的。类 CRTFView 只是封装类 RTE 中视图的功能，当选择每个字符或段落的格式菜单项时，便调用相应于此菜单项的类方法。

当调用格式的方法时，在各种情况下的过程都是相同的。首先，获取所选择区域的当前字符格式信息，此过程是通过调用编辑控制类的方法 GetCharFormat 或 GetParaFormat 来完成的。然后，审查此格式信息，查看当前的格式信息是否包含用户请求的风格位（黑体、斜体等）。

如果所获取的格式信息的风格位不包含用户请求的风格，则通过“或（OR）”操作包含起用户所请求的风格，如果风格位已经包含了用户请求的风格，则通过“异或（XOR）”操作把用户所请求的风格去掉。然后，根据此风格位设置屏蔽码，并调用编辑控制的方法 SetCharFormat 或 SetParaFormat，便向控制发送了 Windows 消息，从而把新的风格应用到被标记的文本。

注释

本节可做为 Windows 95 环境下完整的文本编辑器的基础，或者也可用于显示从 HTML 文档到 RTF 格式的 Word 文档的任何文档。

此系统一个明显的作用存在于带标记的文本文件环境中。程序员只须编写一个预处理程序，通过分析文本标记以及在 RTE 控制中设置相应的风格标志，便可以把文本装入到编辑控制中。反之亦然，可以写出标记来代替风格位。

最后，需要注意的是：RTE 控制响应普通文本控制所响应的全部消息，因此，可以把前面章节中所讨论的有关编辑控制的使用方法利用到 RTE 控制中，并希望它们做类似的工作。虽然诸如 EM_SETSEL 的普通消息只接受整数范围的值，但 RTE 控制定义了一套 EM_SETSELEX 形式的消息，这套消息可以利用长整型范围的值。

9.7 改变编辑控制中插入光标的类型

问题

有许多膝上型计算机用户，在他们小尺寸的屏幕上，看不清编辑域中的光标，如何使光标变大从而使他们能够很容易地看到呢？是否存在一种设置光标大小或设置较醒目图案的一般方法，从而使膝上型计算机用户（以及视力差的用户）能够很容易地看清楚当前是哪个编

辑控制呢?

方法

如果存在一种解决复杂问题的简单方法,那是再好不过的。这里,通过对 WindowsAPI 的简单调用,就能够快速、容易地把光标(或插入光标,众所周知,在 Windows 程序设计的说法中,光标是鼠标的代表)改变为较大的尺寸或较醒目的图案。

函数 CreateCaret 与 ShowCaret 是本节中将介绍的特殊的 API 函数,虽然 MFC 提供了一些有关插入光标函数的接口,但这里是由 API 提供的一种较丰富、较易于理解的方法来完成此工作的例子。

步骤

打开与运行按照如下步骤生成的例子程序 CH97.MAK。选择主菜单 Dialog,并从此菜单中选择菜单项 Change Caret Type,此时,便会看到类似于图 9-7 所示的对话框,点击按钮 Change Caret,然后在编辑域点击,可以看到此时插入光标变成了非常宽的闪烁线,点击按钮 Bitmap Caret,再次在编辑域中点击,便会显示一种看起来象“I”的新型的插入光标。如果想实现上述功能,请按如下步骤进行。

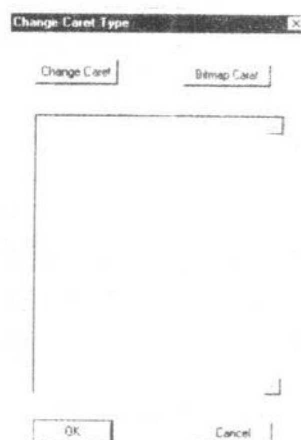


图 9-7 改变插入光标对话框

1. 在 Visual C++中,创建新的项目文件 CH97.MAK。进入 AppStudio,创建新的对话框,在此对话框窗口的顶端,添加两个标题分别为 Change Caret 和 Bitmap Caret 的按钮,在对话框的中间添加一个编辑域,把按钮 OK 与按钮 Cancel 移到对话框的底部。

2. 进入 ClassWizard,为刚定义的对话框模板创建一个新的对话框类,把此类命名为 CCaretDlg。在 ClassWizard 中,从下拉 列表中选择 CCaretDlg,从对象列表中选择对象 IDC_BUTTON1,从消息列表中选择消息 BN_CLICKED,点击按钮 Add Function 添加新的函数 OnChangeCaret。把下面的代码输入到 CCaretDlg 的方法 OnChangeCaret 中。

```
void CCaretDlg:: OnChangeCaret ()
{
    caret_type=1;
}
```

3. 在 ClassWizard 中,从对象列表中选择对象 IDC_BUTTON2. 从消息列表中选择消息 BN_CLICKED,点击按钮 Add Function 添加新的函数 OnBitmapCaret。把下面的代码输入到 CCaretDlg 的方法 OnBitmapCaret 中。

```
void CCaretDlg:: OnBitmapCaret ()
{
    caret_type=2;
}
```

4. 从 ClassWizard 的对象列表中选择对象 IDC_EDIT1,从消息列表中选择消息 EN_SETFOCUS,点击按钮 Add Function 添加新的函数 OnGetEditFocus。把下面的代码输入到 CCaretDlg 的方法 OnGetEditFocus 中。

```
void CCaretDlg:: OnGetEditFocus ()
{
    switch (caret _ type) {
        case1:
    :: CreateCaret (GetDlgItem (IDC _ EDIT1) ->m _ hWnd, NULL, 6, 4);
    break;
        case2:
          HBITMAP hBitmap=LoadBitmap (AfxGetInstanceHandle (),
                                            MAKEINTRESOURCE (IDB _ BITMAP1));
    :: CreateCaret (GetDlgItem (IDC _ EDIT1) ->m _ hWnd, hBitmap, 0, 0);
    break;
    }
    if ( (caret _ type)
        :: ShowCaret (GetDlgItem (IDC _ EDIT1) ->m _ hWnd);
}
```

5. 从 ClassWizard 的对象列表中选择对象 IDC _ EDIT1，从消息列表中选择消息 EN _ KILLFOCUS，点击按钮 Add Function 添加新的函数 OnLoseEditFocus。把下面的代码输入到 CCaretDlg 的方法 OnLoseEditFocus 中。

```
void CCaretDlg:: OnLoseEditFocus ()
{
    if (caret _ type)
    DestroyCaret ();
}
```

6. 最后，把下面一行添加到 CCaretDlg 的构造函数中。

```
caret _ type=0;
```

与此同时，把下面的行添加到 CCaretDlg 的头文件中。

```
private:
    intcaret _ type;
```

7. 接下来，重新进入 AppStudio。从菜单列表中选择主菜单对象，添加标题为 Dialog 的主菜单，在此主菜单中添加标题为 Change Caret Type、标识符为 ID _ CARET _ DLG 的菜单项。保存菜单，然后退出 AppStudio。

8. 在 ClassWizard 中，从下拉列表中选择对象 CCh97App。从对象列表中选择对象 ID _ CARET _ DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的函数 OnCaretDlg。把下面的代码输入到 CCh97App 的方法 OnCaretDlg 中。

```
void CCh97App:: OnCaretDlg ()
{
    CCaretDlgdlg;
    dlg. DoModal ();
}
```

9. 最后，把下面一行添加到文件 CH9. CPP 顶端的 include 列中。

```
#include" caretdlg. h"
```

10. 编译与运行此例子程序。

用法

当用户选择按钮 Change Caret 或按钮 Bitmap Caret 时，只需把类中的一个标志设定为给定的值。当进入编辑域时，对话框类便接收到一个 EN _GETFOCUS 消息，此消息将被方法 OnGetEditFocus 捕获，该方法利用 API 函数 CreateCaret 来创建新的插入光标，方法 ShowCaret 用来使插入光标显示。

按钮 Bitmap Caret 把插入光标的类型设置为位图。当调用函数 CreateCaret 时，或者传递位图句柄，或者传递插入光标的长度和宽度。位图句柄是自己应用程序中创建并通过函数 LoadBitmap 装入的位图的索引，LoadBitmap 也是 API 中的函数。

为了在本节中演示例子，利用 AppStudio 创建一个位图，保存此位图，设定其标识符为 IDB _BITMAP1。虽然 8×8 格的位图显示起来效果最好，但插入光标的位图可以为任意大小，如果程序员在应用程序中利用的是比较大的字体，则一定要利用较宽、较高的位图，在此过程中应该注意：当显示位图时，位图将被反转，因此，只有位图的白色部分在屏幕上的插入光标中显露出来。

注释

定制插入光标是 Windows API 系统利用最少的部分之一。用户可能比较喜欢显示专用的插入光标，但应用程序却很少利用这一特性来适应他们。

如果在应用程序中使定制的插入光标显得突出，那么膝上型计算机用户以及视力差的用户一定会感到定制的插入光标对他们来说是一种可喜的宽慰。认真地考虑一下，创建一个可以被他们利用的实用程序。

9.8 改变插入光标的闪烁速度

问题

对于大多数用户来说，插入光标或编辑光标闪烁得太快或太慢都会使他们不易辨别，因此希望能够允许用户来定制插入光标的闪烁速度，以便他们能够较容易地阅读编辑框中的文本并找到它们相应的位置。

如何利用 Windows API 的功能让用户来定制插入光标的闪烁速度呢?这样的想法是可能的呢，还是需要自己绘制光标并利用某类计时器更新光标呢?

方法

在 Windows 95 可定制的领域里，如果用户想作某种改变而操作系统却不允许改变，这确实是件糟透的事情，Windows 95 充分地考虑了这一特殊问题，并通过对 API 函数 SetCaret-BlickTime 的简单调用，解决了这一问题。

函数 SetCaretBlinkTime 顾名思义，用来改变编辑域中光标（也就是插入光标）的闪烁速度。与此相伴的函数 GetCaretBlinkTime 用来返回插入光标闪烁速度的当前设置值，当利用诸如 SetCaretBlinkTime 的系统函数时，应用程序通常应“考虑周到”，即在完成使用某一属性后，应把它恢复为原来的设置值。插入光标的闪烁速度是为 Windows 中的所有应用程序设置的，所以当从自己的编辑窗口中移去光标时，应通过把闪烁速度重新设置为先前的值或缺省的值，来恢复 Windows 的设定。

本节中，将讨论如何在应用程序中减慢或加快插入光标的闪烁速度。

步骤

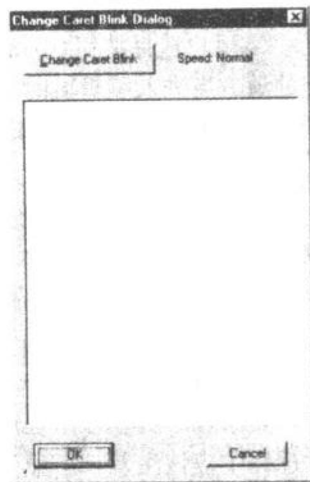


图 9-8　改变插入光标闪烁速度的对话框

打开与运行按照如下步骤生成的例子程序 CH98. MAK。选择主菜单 Dialog，并从此菜单中选择菜单项 Change Caret Blink Speed，此时，便会看到类似于图 9-8 所示的对话框，点击按钮 Change Caret Blink，然后在编辑域中点击一下，可以看到插入光标开始正常闪烁，多次点击按钮 Change Caret Blink，每次点击按钮后，在编辑域中点击一下，可以看到插入光标的闪烁速度会发生各种改变，并且在与此按钮相邻的文本域中显示出闪烁速度。如果想实现上述功能，请按如下步骤进行。

1. 在 Visusl C＋＋中，利用 AppWizard 创建项目文件 CH98. MAK。进入 AppStudio，创建新的对话框，在此新的对话框中添加标题为 Change Caret Blink 的按钮以及标题为 Speed：Normal 的文本域。在对话框中再添加一个编辑域，把按钮 OK 与按钮 Cancel 移到对话框的底部。

2. 进入 ClassWizard，为刚定义的对话框模板创建一个新的对话框类。把此类命名为 CBlinkDlg。在 ClassWizard 中，从下拉 列表中选择 CBlinkDlg，从对象列表中选择对象 IDC _ BUTTON1，从消息列表中选择消息 BN _ CLICKED，点击按钮 Add Function 添加新的函数 OnChangeCursorSpeed。把下面的代码输入到 CBlinkDlg 的方法 OnChangeCursorSpeed 中。

```
void CBlinkDlg::OnChangeCursorSpeed ()
{
   //Three different options: Normal, Slow, Fast
   switch (cur_setting) {
      case0: //Normal
          cur_setting=1;
          SetCaretBlinkTime (old_time);
          GetDlgItem (IDC_TEXT1) ->SetWindowText (" Speed: Normal");
          break;
      case1: //Slow
          cur_setting=2;
          GetDlgItem (IDC_TEXT1) ->SetWindowText (" Speed: Slow");
          SetCaretBlinkTime (old_time * 2);
          break;
      case 2: //Fast
          cur_setting=0;
          GetDlgItem (IDC_TEXT1) ->SetWindowText (" Speed: Fast");
          SetCaretBlinkTime (old_time/2);
          break;
  }
}
```

3. 把下面两行添加到类 CBlinkDlg 的构造函数中。

```
old _ time=GetCaretBlinkTime ();
cur _ setting=0;
```

4. 把下面几行添加到类 CBlinkDlg 的头文件 BLINKDLG.H 中。

```
private:
    UINT old _ time;
    short cur _ setting;
```

5. 接下来，重新进入 AppStudio。从菜单列表中选择主菜单对象，添加标题为 Dialog 的主菜单，在此主菜单中添加标题为 Change Caret Blink Speed、标识符为 ID _ CARET _ SPEED 的菜单项，保存菜单，然后退出 AppStudio。

6. 在 ClassWizard 中，从下拉列表中选择对象 CCh98App。从对象列表中选择对象 ID _ CARET _ SPEED，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的函数 OnCaretSpeed。把下面的代码输入到 CCh98App 的方法 OnCaretSpeed 中。

```
void CCh98App:: OnCaretSpeed ()
{
    CBlinkDlg dlg;
    dlg.DoModal ();
}
```

7. 最后，把下面一行添加到文件 CH9.CPP 顶端的 include 列中。

```
#include" blinkdlg.h"
```

8. 编译与运行此例子程序。

用法

当用户选择对话框并点击按钮 Change Caret Blink 时，方法 OnChangeCursorSpeed 被调用，此方法检查成员变量 cut _ setting 的当前值，并在方法 SetCaretBlinkTime 中设置相应的速度。在对话框的构造函数中，通过对函数 GetCaretBlinkTime 的调用，来获取插入光标闪烁速度的初始设置值。在例子程序中，将根据此初始设置值，使闪烁的时间或者减半（加快闪烁）或者倍增（减慢闪烁）从而来改变光标的闪烁速度，应该注意：整个一次闪烁的时间为所设置时间的两倍，这是因为：这个时间除包括打开光标（光标变亮）所用的时间外，还要包括关闭光标（光标变暗）所用的时间。

第10章　打　　印

对于用户来说，打印机是现有的最重要的外部设备之一。与调制解调器、扫描仪以及其他一些硬件设备不同，大多数用户更熟悉与习惯于利用打印机来处理一些事务。无论是主机用户，还是工作站用户都会十分了解打印机，并且认识到了打印输出的重要性。对当今所谓的“无纸”办公环境具有讽刺意味的是，当前在高质量、高清晰度输出上的花费比以往的任何时期都要高。

在打印领域里，Windows 95 在 Windows 3.1 的基础上迈入了完美的阶段，许多打印驱动程序除通过操作系统较好地控制输出与字体选择外，还支持直接打印。当程序要求把几百块不同的代码与众多打印机打交道时，过去时代的 MS－DOS 是无能为力的，而 Windows 95 则有了较大的飞跃。目前，Windows 程序员只需关心的是如何利用打印机的驱动程序接口以及如何调用 Windows API 函数来驱动打印机，至于有关硬件的信号交换和打印机的控制代码，都是程序员所不关心的。

本章中，我们将考察几个与打印机有关的 API 函数，并且介绍在应用程序中如何利用这些 API 函数。在这里，将会学习到：如何使打印输出看起来更漂亮，如何处理用户没有打印机时的情况，如何确定系统中所用打印机的性能等。由于 Windows 设法为最终用户提供单一的、一致的用户接口，因此在这里还将学习到：如何在开发程序时，如同所有的专业程序一样，利用相同的通用对话框。

1. 如何确定打印机的性能

众所周知，不同的打印机具有不同的性能，例如，当在一台打印机上输出图形时，最好要确定一下此打印机是否支持图形。打印超出当前所安装的打印机的范围时，在一定程度上会给用户带来烦恼，在本节中，将介绍如何确定当前所用打印机的性能及其所支持与不支持的功能。

2. 如何确定当前打印机的页面大小和方向

当在 Windows 中打印文档时，需要知道当前打印机所能打印的页面大小以及方向，因为对于同样大小页面来说，利用横向模式打印与利用纵向模式打印，其效果看起来是完全不同的。本节中，将介绍如何来确定当前打印机所能打印的页面大小与方向（横向或者纵向）。

3. 如何利用通用打印机对话框

用户喜欢利用相同的接口来处理同样的事情，Windows 95 允许用户以某种一致的方式来查看某些对话框。本节中，将讨论打印机的通用对话框（Windows 3.x 目前所支持的方式），并介绍如何利用通用对话框从用户那里获取打印机信息。

4. 如何打印到文件

通常不能够一定保证在任何需要打印的时候，手边就有现成的可供使用的打印机。目前移动式计算环境的最现实问题之一是：许多膝上型计算机的用户想打印文件，但手边没有一台打印机，在这种情况下，就需要把数据打印到文件中并存放在磁盘上，然后在办公室里，再把此文件输出到打印机上打印出来。在本节中，我们将讨论打印到文件，并介绍如何在某些

情况下修改打印输出，利用 print－to－file 选项把数据打印到文件。

5. 如何确定可用的打印机字体

当使用打印机时，用户会发现有些字体印出的效果比另一些字体较好，产生这种现象有很多原因，其主要原因在于有些字体在屏幕上的显示效果不如被“拉伸”后在打印机上输出的效果好。在本节中，我们将介绍如何查找某一打印机所支持的字体以及如何选择这些字体。

6. 如何确定打印队列的状态

在 Windows 95 与 Windows NT 中，提供了检查打印队列状态的能力，这是在网络环境下所必须拥有的功能。本节中，将讨论如何确定某一给定打印机队列的状态。

表 10-1 中列出了本章所用到的 Windows 95 API 函数。

表 10-1　第 10 章中用到的 Windows 95 的 API 函数

DeviceCapabilities	GetProfileString	GlobalAlloc
GlobalLock	GlobalUnlock	GlobalFree
GetPrinterDeviceDefaults	Escape	GetDeviceCaps
EnumFontFamilies	MakeProcInstance	FreeProcInstance
EnumPrinters	EnumJobs	OpenPrinter
LocalAlloc	LocalFree	

10.1　确定打印机的性能

问题

有时需要知道在 Windows 中当前打印机的某些性能。实际上，最好能知道所有安装的打印机以及用户所选择的任一打印机的性能，这些所需的信息包括用户所选打印机的页面大小（最好以象素表示）以及其尺寸（以毫米的十分之一为单位）。

如何才能利用 Windows API 函数来查找所安装的打印机以及这些打印机的页面大小与尺寸呢？

方法

为了确定在 Windows 下所安装的打印机，用户必须会阅读系统的信息。信息被保存在 Windows 目录结构下的 WIN.INI 文件中，当获得此信息后，就可列出用户所使用的打印机。关于确定打印机的其他信息（如页面的大小以及尺寸），是比较复杂的。

Windows 所支持的每种打印机都有一个实现该打印机所需功能的驱动程序，此驱动程序实际上是 Windows 装入的一个特殊形式的动态连接库，以便访问打印机的特定函数。动态装入的方法意味着程序员无需知道如何与每个打印机通信，操作系统也不必把所有支持的打印机的全部信息都保存在内存中。

与打印机的驱动程序传递信息是一件很麻烦的事情。首先，必须把打印机驱动程序装入到内存中，接着从打印机驱动程序中获取信息，在驱动程序内设置的标准函数 DeviceCapabilities 是用来查询此信息的。

在本节中，将介绍查找所安装的打印机的列表，装入所请求的打印机的驱动程序，而后从打印机驱动程序中获取所需信息的必要步骤。

步骤

打开与运行按照如下步骤生成的 Visual C＋＋的例子程序 CH101.MAK。选择菜单 Dia-

log，并从此菜单中选择菜单项 Printer Capabilities，此时将会出现一个对话框，当在对话框左边的列表框中选择所列出的某一打印机（如果有多个打印机的话）时，便显示如图 10-1 所示的对话框。

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH101.MAK。

2. 进入 AppStudio，创建新的对话框模板。

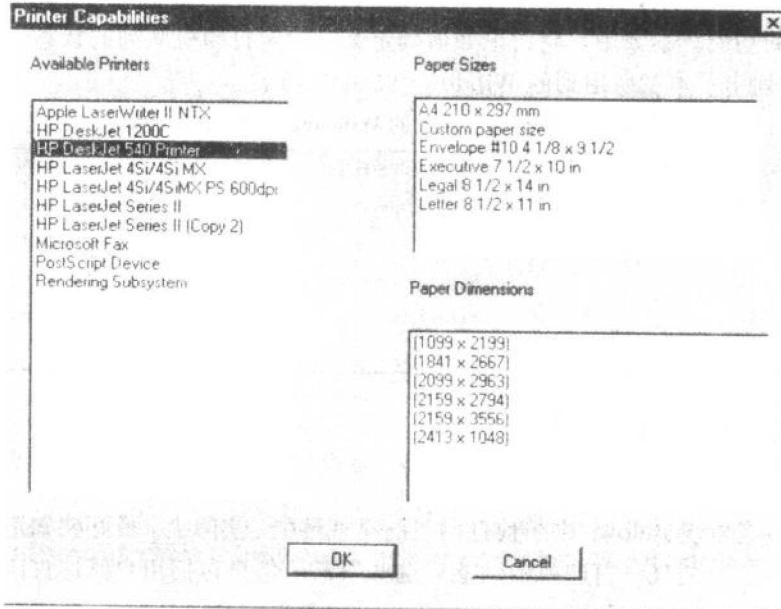


图 10-1 显示选中的打印机及其性能的对话框

3. 在新的对话框模板中，把按钮 OK 与按钮 Cancel 移到对话框的底部，在对话框的顶端添加标题为 Available Printers 的静态文本域，紧跟在此文本域的下面，添加一个列表框。

4. 在对话框的右边，添加第二个标题为 Paper size 的静态文本域，并在其下面添加一个列表框，接着，在此列表框的下面，添加第三个标题为 Paper Dimensions 的静态文本域，并在此文本域下面添加第三个列表框。

5. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，把此新类命名为 CPrinterCapDlg。在 ClassWizard 中，从下拉组合框中选择 CPrinterCapDlg，从对象列表中选择对象 CPrinterCapDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function 添加新函数 OnInitDialog，把下面的代码添加到 CPrinterCapDlg 的方法 OnInitDialog 中。

```
BOOL CPrinterCapDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    char allDevices [4096];
    char *ptr;
    GetProfileString ("devices", NULL, "", allDevices, sizeof (allDevices) );
    // Add the names of the printer devices to the list box
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
```

```
    // Get each name out of the device list
    ptr = allDevices;
    while ( * ptr)
    {
        list->AddString ( ptr );
        ptr += strlen (ptr) + 1 ;
    }
    return TRUE; // return TRUE unless you set the focus to a control
}
```

6. 在ClassWizard中,从下拉组合框中选择CPrinterCapDlg.从对象列表中选择对象IDC_LIST1,从消息列表中选择消息LBN_SELCHANGE,点击按钮Add Function添加新函数OnSelectPrinter,把下面的代码添加到CPrinterCapDlg的方法OnSelectPrinter中。

```
void CPrinterCapDlg:: OnSelectPrinter ()
{
    // Get the selected printer
    char printer [256];
    char driver [_MAX_PATH];
    char *ptr;
    DWORD num_sizes, num_dimensions;
    WORD FAR      *pawPaperList;
    HANDLE     hPaperList, hPaperDims;
    POINT FAR     *paptPaperList;
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    list->GetText ( list->GetCurSel (), printer );
    GetProfileString (" devices", printer, "", driver, sizeof (driver));
    // Get the driver name out of the device string
    ptr = strtok (driver, ",");
    ptr = strtok (NULL, ",");
    // Number of paper sizes
    num_sizes = DeviceCapabilities (printer, " LPT1", (WORD) DC_PAPERS,
                        (LPSTR) NULL, (LPDEVMODE) NULL);
    // Number of paper dimensions
    num_dimensions = DeviceCapabilities (printer, " LPT1", (WORD) DC_PAPERSIZE,
                        (LPSTR) NULL, (LPDEVMODE) NULL);
    // allocate space for paper sizes
    hPaperList = GlobalAlloc (GMEM_MOVEABLE, num_sizes * sizeof (WORD));
    pawPaperList = (WORD FAR * ) GlobalLock (hPaperList);
    // allocate space for paper dimensions
    hPaperDims = GlobalAlloc (GMEM_MOVEABLE, num_dimensions * sizeof (POINT));
    paptPaperList = (POINT FAR * ) GlobalLock (hPaperDims);
    // fill buffer with paper list
    DeviceCapabilities (printer, " LPT1", (WORD) DC_PAPERS,
```

```
                    (LPSTR) pawPaperList, (LPDEVMODE) NULL);
    // fill buffer with paper dimensions
    DeviceCapabilities (printer, " LPT1", (WORD) DC _PAPERSIZE,
                    (LPSTR) paptPaperList, (LPDEVMODE) NULL);
    // Load paper dimension list and paper size list
    CListBox *list2 = (CListBox * ) GetDlgItem (IDC _LIST2);
    CListBox *list3 = (CListBox * ) GetDlgItem (IDC _LIST3);
    // Clear the lists first
    list2->ResetContent ();
    list3->ResetContent ();
    char buffer [80];
    for ( int i=0; i< (int) num _dimensions; ++i ) {
        // First the paper size
    if ( (pawPaperList [i] < MAX _PAPERS) && (pawPaperList [i] > 0))
      list2->AddString ( szPaperList [pawPaperList [i]] );
    else
      list2->AddString ( szPaperList [0] );
    // Next the dimensions
    sprintf (buffer, " (%d x %d) ", paptPaperList [i] .x, paptPaperList [i] .y);
    list3->AddString ( buffer );
}
GlobalUnlock (hPaperList);
GlobalUnlock (hPaperDims);
GlobalFree (hPaperList);
GlobalFree (hPaperDims);
}
```

7. 把下面的代码块添加到源文件 PRINTER.CPP 的顶端。

```
char *szPaperList [] = {
                        " Custom paper size",
                        " Letter 8 1/2 x 11 in",
                        " Letter Small 8 1/2 x 11 in",
                        " Tabloid 11 x 17 in",
                        " Ledger 17 x 11 in",
                        " Legal 8 1/2 x 14 in",
                        " Statement 5 1/2 x 8 1/2 in",
                        " Executive 7 1/2 x 10 in",
                        " A3 297 x 420 mm",
                        " A4 210 x 297 mm",
                        " A4 Small 210 x 297 mm",
                        " A5 148 x 210 mm",
                        " B4 250 x 354",
                        " B5 182 x 257 mm",
                        " Folio 8 1/2 x 13 in",
```

```
                                    " Quarto 215 x 275 mm",
                                    " 10x14 in",
                                    " 11x17 in",
                                    " Note 8 1/2 x 11 in",
                                    " Envelope #9 3 7/8 x 8 7/8",
                                    " Envelope #10 4 1/8 x 9 1/2",
                                    " Envelope #11 4 1/2 x 10 3/8",
                                    " Envelope #12 4 3/4 x 11",
                                    " Envelope #14 5 x 11 1/2",
                                    " C size sheet",
                                    " D size sheet",
                                    " E size sheet"};
#define MAX_PAPERS                  27
```

8. 进入 AppStudio，选择菜单资源列表。选择菜单 IDR_CH101TYPE，并添加标题为 Dialog 的新的主菜单，在此主菜单上，添加标题为 Printer Capabilities、标识符为 ID_PRINTER_CAP 的新的菜单项，保存菜单资源与资源文件。

9. 进入 ClassWizard 并从下拉组合框中选择对象 CCh101App，从对象列表中选择对象 ID_PRINTER_CAP，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnPrinterCap。把下面的代码输入到 CCh101App 的方法 OnPrinterCap 中。

```
void CCh101App::OnPrinterCap ()
{
    CPrinterCapDlg  dlg;
    dlg.DoModal ();
}
```

10. 把下面一行代码添加到源文件 CH101.CPP 的顶端。

```
#include "printerc.h"
```

11. 编译与运行此例子程序。

用法

当用户从主菜单中选择菜单项 Printer Capbalities 后，即创建对话框，此时，响应 Windows 的消息 WM_INITDIALOG，方法 OnInitDialog 被调用。在此方法中，程序将从存放打印机名的 WIN.INI 文件中检索字符串，然后分析该字符串，并将各个打印机名存放在第一个列表框中。

当由于用户选择某个打印机而改变了列表框选项时，即调用方法 OnSelectPrinter。此方法接着为所请求的打印机调用函数 DeviceCapabilities，然后检索该打印机所支持的页面大小与页面尺寸的种类数目，接着再次调用函数 DeviceCapabilities 来分配适当大小数组，而后赋上相应的值。

应该注意，在 Windows 3.x 中，函数 DeviceCapabilities 由打印机驱动程序来装入，而在 Windows 95 中，已不必要装入打印机的驱动函数了。

10.2 确定当前打印机的页面大小和方向

问题

如果想知道当前打印机所设置的页面大小（按照英寸）与方向（横向还是纵向），是否有通过 Windows API 函数来直接获取这些信息的方法呢？

方法

获取当前打印机所设置的页面大小与方向的方法是完全可能的，不过，要获取这些信息，涉及到一些有关打印机设置的操作，并且需要为所考虑的打印机创建设备描述表。

本节中，将介绍在没有为用户提供一般所需的通用对话框的情况下，如何直接获取打印机设备描述表。如果给出了设备描述表，便可以检索有关计算页面大小（按英寸）所需的信息了。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH102.MAK。选择菜单 Dialog，并从此菜单中选择菜单项 Printer Statistics，此时便会看到类似于图 10-2 所示的对话框。

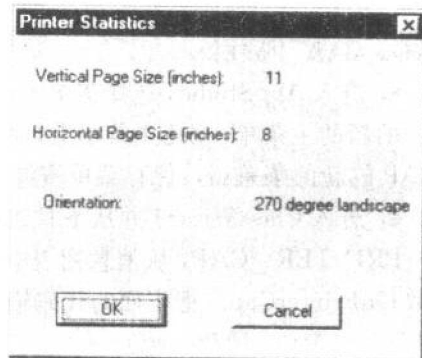


图 10-2 显示当前被选中的打印机的页面大小与方向的对话框

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH102.MAK。

2. 进入 AppStudio，创建新的对话框模板。

3. 在新的对话框模板中，把按钮 OK 与按钮 Cancel 移到对话框的底部，把标题为 Vertical Page Size (inches)：的静态文本域添加到对话框中，添加标题为空的第二个文本域，使它与第一个文本域水平方向对齐，同时把第二个文本域的标识符设定为 ID_VERT_PAGE_SIZE。

4. 在静态文本域 Vertical Page Size (inches)：的下方并与其左对齐，添加一个标题为 Horizontal Page Size (inches)：的文本域。紧挨此文本域的右边，添加一个标题为空、标识符为 ID_HORZ_PAGE_SIZE 的文本域。

5. 在对话框的左边并紧挨文本域 Horizontal Page Size (inches)：的下方，添加标题为 Orientation 的第三个文本域，紧跟在 Orientation 之后，添加另一个标题为空、标识符为 ID_ORIENTATION 的文本域。

6. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，把此新类命名为 CPrinterStatDlg。从对象列表中选择对象 CPrinterStatDlg，从消息列表中选择消息 WM_INITDIALOG，把下面的代码输入到 CPrinterStatDlg 的方法 OnInitDialog 中。

```
BOOL CPrinterStatDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ()
    // Get the device context for the printer
    CDC *printer_dc = new CDC;
    CPrintDialog dlg ( FALSE );
    if (! AfxGetApp () ->GetPrinterDeviceDefaults (&dlg.m_pd))
    {
```

```
        return (FALSE);
    }
    HDC hdcPrint = dlg.CreatePrinterDC ();
    printer_dc->Attach ( hdcPrint );
    // Next, get the physical page size for the printer
    POINT p;
    Escape (printer_dc->m_hDC, GETPHYSPAGESIZE, NULL, NULL, &p);
    // Now get the physical size for the printer
    int log_x = printer_dc->GetDeviceCaps (LOGPIXELSX);
    int log_y = printer_dc->GetDeviceCaps (LOGPIXELSY);
    // Get the real size
    int x_size = p.x / log_x;
    int y_size = p.y / log_y;
    // Set those static text fields
    char buffer [80];
    sprintf (buffer, "%d", x_size);
    GetDlgItem (ID_HORZ_PAGE_SIZE) ->SetWindowText (buffer);
    sprintf (buffer, "%d", y_size);
    GetDlgItem (ID_VERT_PAGE_SIZE) ->SetWindowText (buffer);
    // Get the default printer from Win.INI
    char temp [256];
    GetProfileString (" windows", " device", ",,,",
        temp, 256);
    char *printer_name, *driver_name, *port_name;
    printer_name = strtok (temp, (const char *) ",");
    driver_name = strtok ( (char *) NULL, (const char *) ", ");
    port_name = strtok ( (char *) NULL, (const char *) ", ");
    // Get the orientation
    int orientation = DeviceCapabilities ( printer_name, port_name, DC_ORIENTATION, NULL,
NULL );
    // Set the static text field based on the return from the function
    switch ( orientation ) {
      case 0:
      GetDlgItem (ID_ORIENTATION) ->SetWindowText ( " Portrait" );
    break;
      case 90:
      GetDlgItem (ID_ORIENTATION) ->SetWindowText ( " 90 degree Landscape" );
    break;
      case 270:
      GetDlgItem (ID_ORIENTATION) ->SetWindowText ( " 270 degree landscape" );
    break;
      default:
      GetDlgItem (ID_ORIENTATION) ->SetWindowText ( " Unknown" );
```

```
    break;
    }
    // Free up the memory
    delete printer_dc;
    return TRUE; // return TRUE unless you set the focus to a control
}
```

7. 进入 AppStudio 并在菜单 IDR_CH102TYPE 中添加新菜单 Dialog。在此菜单 Dialog 中添加标题为 Printer Statistics、标识符为 ID_PRINT_STATISTICS 的新的菜单项。

8. 进入 ClassWizard，从下拉组合框中选择对象 CCh102App，从对象列表中选择对象 ID_PRINTER_STATISTICS，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的函数 OnPrinterStatistics。把下面的代码输入到 CCh102App 的方法 OnPrinter-Statistics 中。

```
void CCh102App::OnPrinterStatistics ()
{
    CPrinterStatDlg    dlg;
    dlg.DoModal ();
}
```

9. 把下面一行代码添加到源文件 CH102.CPP 的顶端。

```
#include " printers.h"
```

10. 编译与运行此例子程序。

用法

利用打印机设备描述表，可以取得许多有关设置当前打印机的数据信息，在本例中，我们所感兴趣的是打印机逻辑的与物理的页面大小以及其方向的设置。此过程中的第一步是：获取所选打印机设备描述表的句柄，这是通过利用通用对话框类 CPrintDialog，使用打印机的当前设置来创建新的打印机设备描述表来完成的。

当创建设备描述表后，程序就利用这个 DC 来获取打印机页面的物理大小，这是通过利用 Windows API 的函数 Escape 来完成的，打印机的水平与垂直分辨率是通过调用打印机 DC 的方法 GetDeviceCaps 来获得的。通过用打印机的逻辑页面大小除物理页面分辨率就得到以英寸表示大小的打印页。

打印的方向是通过调用 Windows API 的函数 Escape 和设定参数 GETSETPRINT-ORIENT 来获得的。返回值通常为 1 或 2，1 代表纵向模式（正规的打印布局），2 代表横向模式（侧转打印）。

当获得所有的数据信息之后，这些数据便通过利用 API 函数 SetWindowText 被存放在相应的静态文本域中。

10.3 利用通用打印机对话框

问题

在使用打印机时，我们通常需要利用通用对话框来设置一定的参数以便进行正确的打印，但对于打印对话框中的某些选项，在有些情况下是不必要或者对于某些用户来说是没有意义的。那么，如何利用 Windows API 来定制通用打印对话框？又如何能够使通用打印对话框按

照用户所需的要求与形式来显示呢？

方法

如果对于每一个程序都要求通用对话框有完全相同的标准，那么通用对话框将很难应用，许多程序并不需要整个对话框中的所有功能，或者要求在某些情况下禁止对话框中的某些功能。Windows 95 通用对话框适应了所有这些要求，即只要通过向对话框创建例程传递一组标志，就能定制通用的打印机对话框。

本节中，将讨论通用打印机对话框的可配置部分以及用户在自己的应用程序中如何设置这些选项。这里将显示一个对话框，此对话框中列出了所有的结构选项，用户可以从这些选项中作任意的选择，程序将根据用户所作的选择，显示出相应结构与形态的打印机对话框。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH103. MAK。选择菜单 Dialog，并从此菜单中选择菜单项 Printer Dialog Options，此时便会看到类似于图 10-3 所示的对话框，选择核选框 Hide Print To File、Disable Page Numbers 以及 Disable Selection，点击按钮 Show Dialog，便出现如图 10-4 所示的对话框。

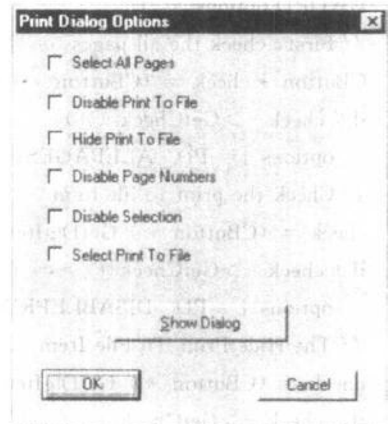


图 10-3 显示设置通用打印机对话框选项的对话框

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH103. MAK。

2. 进入 AppStudio，创建新的对话框模板。在对话框的中心位置处，添加垂直排放的六个核选框。依次设定这些核选框的标题为：Select All Pages，Disable Print To File，Hide Print To File，Disable Page Numbers，Disable Selection 以及 Select Print To File。

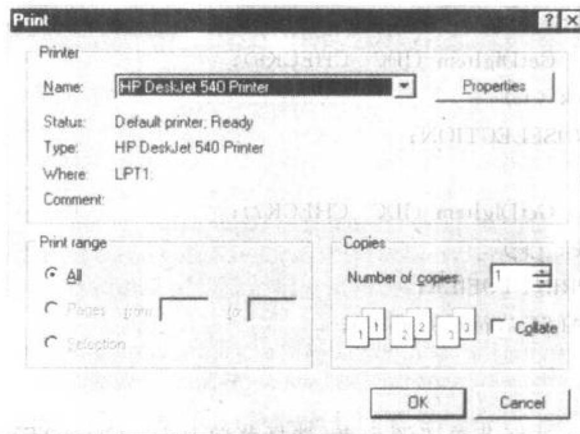


图 10-4 反映 Print Dialog Options 对话框中所选择选项的通用打印机对话框

3. 在对话框中添加新的命令按钮，把此按钮的标题设定为 &Show Dialog。

4. 把按钮 OK 与按钮 Cancel 移到对话框的底部并设定此对话框的标题为 Print Dialog Options。

5. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，把此新类命名为 CPrintDlgOptions。

6. 在 ClassWizard 中，从下拉组合框中选择类 CPrintDlgOptions，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 添加新的函数 OnShowDialog。把下面的代码输入到类 CPrintDlgOptions 的方法 OnShowDialog 中。

```
void CPrintDlgOptions:: OnShowDialog ()

    // Get the options from the checkboxes.
    DWORD options = 0;
    // First, check the all pages
    CButton *check = (CButton *) GetDlgItem (IDC_CHECK1);
    if ( check->GetCheck () )
        options |= PD_ALLPAGES;
    // Check the print to file item
    check = (CButton *) GetDlgItem (IDC_CHECK3);
    if ( check->GetCheck () == 0 )
        options |= PD_DISABLEPRINTTOFILE;
    // The Hide Print To File Item
    check = (CButton *) GetDlgItem (IDC_CHECK4);
    if ( check->GetCheck () )
        options |= PD_HIDEPRINTTOFILE;
    // Disable Page Numbers
    check = (CButton *) GetDlgItem (IDC_CHECK5);
    if ( check->GetCheck () )
        options |= PD_NOPAGENUMS;
    // Disable selection
    check = (CButton *) GetDlgItem (IDC_CHECK6);
    if ( check->GetCheck () )
        options |= PD_NOSELECTION;
    // Select Print to File
    check = (CButton *) GetDlgItem (IDC_CHECK7);
    if ( check->GetCheck () )
        options |= PD_PRINTTOFILE;
    CPrintDialog dlg ( FALSE, options, this );
    dlg.DoModal ();
```

7. 进入 AppStudio，选择菜单资源列表。选择菜单 IDR_CH103TYPE，添加标题为 Dialog 的主菜单，在菜单 Dialog 中添加标题为 Printer Dialog Options、标识符为 ID_PRINT_OPTIONS 的新的菜单项，保存此菜单资源与资源文件。

8. 进入 ClassWizard，从下拉组合框中选择对象 CCh103App，从对象列表中选择对象 ID

_PRINT_OPTIONS，从消息列表中选择消息COMMAND，点击按钮Add Function添加新的函数OnPrintOptions。把下面的代码输入到CCh103App的方法OnPrintOptions中。

```
void CCh103App:: OnPrintOptions ()
{
    CPrintDlgOptions dlg;
    dlg.DoModal ();
}
```

9. 把下面一行代码添加到源文件CH103.CPP的顶端。

```
#include " printdlg.h"
```

10. 编译与运行此例子程序。

用法

通用打印机对话框接受一个包含一组可配置选项的标志参数，此参数中的选项编码为位，用来表示哪些选项是程序员要求在最后输出的对话框中出现的。

用户通过选择对话框中的选项，确定通用打印对话框的显示形状。需要注意的是：有些选项之间是相互排斥的，例如：当选择选项Hide Print To File的同时，选择选项Print To File是没有什么意义的，这两种选项虽然是相互矛盾的，但通用打印机对话框还是接受他们，只不过在对话框中，相当于有一个隐藏的核选框。

注释

由于这是个十分通用并经常被定制的对话框，所以也可以利用Delphi与Visual Basic来创建这样的对话框，下面，就介绍利用这两种工具来创建这样对话框的方法。首先，看一下利用Delphi创建这样对话框的步骤。

1. 在Delphi中创建新的项目文件PRNTDLG.DPR。在项目文件的表单中添加新的打印机通用对话框，另外，在此对话框的表单中添加四个核选框。

2. 把这四个核选框的标题分别设置为：Disable Print To File，Hide Print To File，Disable Page Numbers，Disable Selection。

3. 在表单中再添加两个新的按钮Show Dialog与Close。双击按钮Show Dialog，把下面的代码添加到表单的方法TForm1.Button1Click中。

```
procedure TForm1.Button1Click (Sender: TObject);
begin
   if ( CheckBox2.Checked ) then
       begin
       PrintDialog1.Options := PrintDialog1.Options + [poPrintToFile];
       PrintDialog1.PrintToFile := Ture;
   end;
  if ( CheckBox3.Checked) then
   PrintDialog1.Options := PrintDialog1.Options + [poDisablePrintToFile];
  if ( CheckBox4.Checked) then
   PrintDialog1.Options := PrintDialog1.Options - [poPageNums];
  if ( CheckBox5.Checked) then
   PrintDialog1.Options := PrintDialog1.Options - [poSelection];
PrintDialog1.Execute;
```

```
end;
```

4. 双击按钮 Close，把下面的代码添加到方法 TForm1.Button2Click 中。

```
procedure TForm.Button2Click (Sender: TObject);
begin
   Close
end;
```

5. 编译与运行此例子程序。

接着，按如下步骤用 Visual Basic 来创建这样的对话框。

1. 在 Visual Basic 中创建新的项目文件 PRTCOM.MAK。创建一个名为 PRTCOM.FRM 的新的表单。

2. 在此表单中添加六个标题分别为 Enable Print All Pages，Disable Print To File，Hide Print To File，Enable Page Numbers，Disable Selection 以及 Set Print To File 的按钮。

3. 在表单中再添加标题为 Show Dialog 的命令按钮，双击此按钮，把下面的代码添加到函数 Command1 _ Click 中。

```
Private Sub Command1 _Click ( )
    If Check1.Value Then
       CMDialog1.Flags = CMDialog1.Flags Or PD _ALLPAGES
    Else
       CMDialog1.Flags = CMDialog1.Flags And Not PD _ALLPAGES
    End If
    If Check3.Value Then
       CMDialog1.Flags = CMDialog1.Flags Or PD _DISABLEPRINTTOFILE
    Else
       CMDialog1.Flags = CMDialog1.Flags And Not PD _DISABLEPRINTTOFILE
    End If
    If Check4.Value Then
       CMDialog1.Flags = CMDialog1.Flags Or PD _HIDEPRINTTOFILE
    Else
       CMDialog1.Flags = CMDialog1.Flags And Not PD _HIDEPRINTTOFILE
    End If
    If Check5.Value Then
       CMDialog1.Flags = CMDialog1.Flags Or PD _NOPAGENUMS
    Else
       CMDialog1.Flags = CMDialog1.Flags And Not PD _NOPAGENUMS
    End If
    If Check6.Value Then
       CMDialog1.Flags = CMDialog1.Flags Or PD _NOSELECTION
    Else
       CMDialog1.Flags = CMDialog1.Flags And Not PD _NOSELECTION
    End If
    CMDialog1.ShowPrinter
End Sub
```

4. 把下面的内容添加到表单的 General 部分。

```
Attribute VB_Name = " Form1"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Const PD_ALLPAGES = &H0&
Const PD_SELECTION = &H1&
Const PD_PAGENUMS = &H2&
Const PD_NOSELECTION = &H4&
Const PD_NOPAGENUMS = &H8&
Const PD_COLLATE = &H10&
Const PD_PRINTTOFILE = &H20&
Const PD_PRINTSETUP = &H40&
Const PD_NOWARNING = &H80&
Const PD_RETURNDC = &H100&
Const PD_RETURNIC = &H200&
Const PD_RETURNDEFAULT = &H400&
Const PD_SHOWHELP = &H800&
Const PD_USEDEVMODECOPIES = &H40000&
Const PD_DISABLEPRINTTIFILE = &H80000&
Const PD_HIDEPRINTTOFILE = &H100000&
```

5. 编译与运行此例子程序。

10.4 打印到文件

问题

在应用程序中，经常需要把数据库中的报告在输出设备上打印出来，在一般情况下，这种输出设备是打印机。但有时，对于使用膝上计算机的用户来说，需要把他们的文档永久地记录下来，但手边却没有可供使用的打印机来输出他们的数据。如果用户仅使用选项 Print To File，程序就会无法识别向何处输出，Windows 也就会显示信息 Print Not Ready（或者是其他错误）。那么，当用户需要时，如何把数据打印到文件呢？

方法

打印到文件是一个相当主动而又很容易实现的过程。基本的方法是，创建一个通用的打印机对话框，让用户在其中输入有关信息，并选择返回标志，当打印到文件的标志被设置后，向用户要求要打印到的文件的名字，然后作一些必要的工作使得打印输出是基于磁盘而不是基于打印机的。

本节中，将执行上述的过程，为用户显示通用的打印对话框，并允许用户在对话框中选择核选框 Print To File，如果此核选框被选择，程序中将显示通用对话框 File Save As，当用户在这个对话框中输入文件名之后，该文件就会被打开，一批记录就被写到此文件中，写完之后，文件便被关闭，整个打印到文件的过程即告完成。

本程序包括的主要内容有：用来打印与保存文件的通用对话框、检查从打印机通用对话框中返回的标志以及利用通用对话框 File Save As 的结果来创建新文件。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH104.MAK。选择菜单 File，并从此菜单中选择菜单项 Print。此时便显示打印机通用对话框，点击核选框 Print To File，选择按钮 OK，便会显示如图 10-5 所示的通用对话框 Save As，把存放报告的文件名输入到指定的区域并按下按钮 OK，报告便会被写到所指定的基于磁盘的文件里。

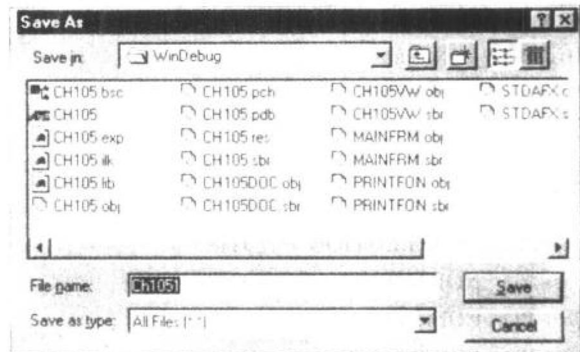


图 10-5　保存文件的通用对话框

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH104.MAK。

2. 进入 ClassWizard，从下拉组合框中选择此项目文件的视图对象 CCh104View，从对象列表中选择对象 ID_FILE_PRINT，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的函数 OnFilePrint。

3. 把下面的代码输入到 CCh104View 的方法 OnFilePrint 中。

```
void CCh104View:: OnFilePrint ()
{
    // Put up a print dialog
    CPrintDialog dlg (FALSE, PD_ALLPAGES | PD_USEDEVMODECOPIES );
    if ( dlg.DoModal () == IDOK ) {
    // See if they wanted to print to file or print to printer
    if ( dlg.m_pd.Flags & PD_PRINTTOFILE ) {
        // Get the output file here.
        CFileDialog fdlg (FALSE);
    if ( fdlg.DoModal () == FALSE )
        return;
    // Open the output file here
    CString outFile = fdlg.GetPathName ();
    FILE *fp = fopen ( outFile, " w" );
    if ( fp == (FILE *) NULL ) {
        MessageBox (" Unable to create/open output file", " Error", MB_OK );
        return;
    }
    for ( int i=0; i<10; ++i ) {
        fprintf (fp, " Record %d Line 1\n", i);
        fprintf (fp, " Record %d Line 2\n", i);
        fprintf (fp, " Record %d Line 3\n\n", i);
    }
    fclose (fp);
```

}

}

4. 编译与运行此例子程序。

用法

当创建打印机通用对话框后，检查域 Flags 来确定选项 Print To File 是否被包含在对话框中，在本例中，由于未在选项标志中指定标志 PD_HIDEPRINTTOFILE，所以通用对话框中包含选项 Print To File。有关如何创建通用对话框的详细信息，请参见 10.3 节。

当用户点击打印机通用对话框中的按钮 OK 后，对话框的域 Flags 将按照用户的要求重新设置，如果用户选择选项 Print To File，则在域 Flags 中将设置为标志 PD_PRINTTOFILE，这时，程序就知道用户要求把输出定向到基于磁盘的文件而非打印机本身了。

当确定选择核选框 Print To File 后，就需要用户给出存放输出内容的文件，这是通过利用通用文件对话框来完成的。把通用文件对话框中的第一个参数设置为 FALSE，此对话框就成为 Save As 对话框；如果把此参数设置为 TRUE，此对话框看起来就象 File Open 通用对话框了。这类似于打印通用对话框，当参数设置为 FALSE 时，以 Print 对话框的形式出现，当参数设置为 TRUE 时，就以 Print Setup 对话框的形式出现。

当用户在通用文件对话框中输入文件名后，按下按钮 OK，程序就会连同其完全路径名一起来检索此名字的文件，然后根据检索的结果创建或打开与截取此文件，如果不能创建或打开该文件，就会显示一条错误信息，相反，程序就会通过循环把所有的数据输出到此文件中。

10.5 确定可用的打印机字体

问题

通常在把任何输出内容发送到打印机之前，需要查询此打印机支持哪些字体，特别当我们要求打印出的字体正是我们想要的字体而非仅仅象我们想要的字体时，就有必要查找打印机可用的字体。只有通过获取打印机所支持的字体并当利用这些字体时，才能使屏幕上显示的字体完全和打印机打印出的字体相一致。

那么，如何利用 Windows 95 的 API 函数来获取当前缺省打印机的所有可用的打印机字体呢？

方法

当前，用户要做的一件最重要的事情是确保在程序中所用的字体在一般计算机上是可得到的，这样也就保证了用户生成的打印输出是可读的，因为在有些情况下，我们会遇到一些十分烦恼的事情，例如，满屏幕显示的是完全可读的数据，而在打印机上呈现出的却是一些难以理解的乱糟糟的线段。

本节中，将讨论对给定的打印机查找其全部可用字体的必要过程。要完成这个过程，需要利用 Windows API 函数 EnumFontFamilies、MakeProcInstance、FreeProcInstance 以及一些通用打印机对话框的函数。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH105.MAK。选择菜单 Dialog，并从此菜单中选择菜单项 List Printer Fonts，此时便会显示类似于图 10-6 所示的对话

框，在此对话框中，显示出了当前打印机的所有可用字体。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH105.MAK。

2. 进入 AppStudio，创建新的对话框模板。把按钮 OK 与按钮 Cancel 移到对话框的底部，在此对话框中，添加一个列表框，保存此对话框模板。

3. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，把此新类命名为 CPrinterFontDlg。

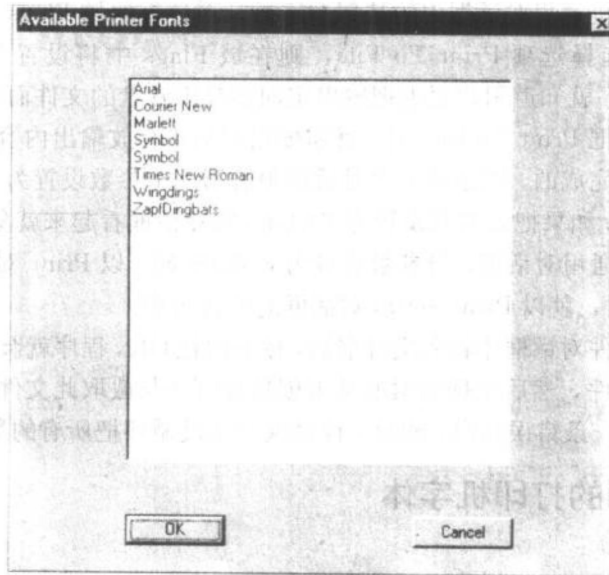


图 10-6 显示可用打印机字体的对话框

4. 在 ClassWizard 中，从下拉组合框中选择 CPrinterFontDlg。从对象列表中选择对象 CPrinterFontDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function 添加新的函数 OnInitDialog，把下面的代码输入到 CPrinterFontDlg 的方法 OnInitDialog 中。

```
BOOL CPrintFontDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    // Get the device context for the printer
    CDC *printer_dc = new CDC;
    CPrintDialog dlg ( FALSE );
    if (! AfxGetApp () ->GetPrinterDeviceDefaults (&dlg.m_pd))
    {
        return (FALSE);
    }
    HDC hdcPrint = dlg.CreatePrinterDC ();
    printer_dc->Attach ( hdcPrint );
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
```

```
    FARPROC EnumFontProc = MakeProcInstance ( ( FARPROC ) EnumFontsFunc,
AfxGetInstanceHandle () );
    ::EnumFontFamilies ( printer_dc->m_hDC, NULL , (OLDFONTENUMPROC) EnumFontProc,
(LPARAM) list);
    FreeProcInstance ( EnumFontProc );
    return TRUE; // return TRUE unless you set the focus to a control
}
```

5. 把下面的代码添加到源文件 PRINTFON.CPP 中，并注意添加到方法 OnInitDialog 的前面。

```
int CALLBACK EnumFontsFunc (LOGFONT FAR *lplf , NEWTEXTMETRIC FAR
                    *lpntm , int FontType, LPARAM lpData)
{
    CListBox *list = (CListBox *) lpData;
    list->AddString (lplf->lfFaceName);
    return 1;
    }
```

6. 进入 AppStudio，选择菜单资源列表。选择菜单 IDR_CH105TYPE，添加标题为 Dialog 的新的主菜单，在此主菜单上，添加一个标题为 List Printer Fonts、标识符为 ID_LIST_FONTS 的新的菜单项，保存此菜单资源与资源文件。

7. 进入 ClassWizard，从下拉组合框中选择对象 CCh105App。从对象列表中选择对象 ID_LIST_FONTS，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的函数 OnListFonts，把下面的代码输入到 CCh105App 的方法 OnListFonts 中。

```
void CCh105App:: OnListFonts ()
{
    CPrintFontDlg dlg;
    dlg.DoModal ();
}
```

8. 把下面一行代码添加到源文件 CH105.CPP 的顶端。

```
#include " printfon.h"
```

9. 编译与运行此例子程序。

用法

Windows 95 API 函数 EnumFontFamilies 接收一个回调函数，对输出设备描述表所支持的每种字体都将调用它一次。为了保证仅获取该打印机可用的字体，特别为此打印机创建了一个设备描述表，并且把此设备描述表传递给函数 EnumFontFamilies，针对于此设备描述表所支持的每种字体，通过对回调函数的每次调用，它们的名字就会被存放在输出列表框中。

为了创建打印机设备描述表，必须首先获取缺省打印机的信息。在 MFC 中，这个工作是通过调用应用程序对象的方法 GetPrinterDeviceDefaults 来完成的。另外还可以通过从 WIN.INI 文件中获取“设备”信息来得到所有打印机设备的信息，这时，我们就可以为所有安装的打印机挑选字体了。

当获得打印机信息之后，设备描述表就被创建并且与打印机 DC 对象相连，接着，回调函数通过利用 API 函数 MakeProcInstance 被创建并且转化为函数 EnumFontFamilies 所适合

的类型。EnumFontFamilies 函数的最后一个参数是用来让程序员把指定的应用程序数据传递给回调函数的，在本例中，我们传递了一个指针，此指针指向对话框中的列表框，传递此指针的目的是为了让回调函数知道存放字体名字的位置。

在回调函数 EnumFont 中一定要记住返回一个非零的值，如果此函数返回 0，则将终止列举，那么在列表框中就只能列出一种字体了。

注释

以上仅是“如何向用户显示输出设备上可用字体”这一过程的第一步，下一步要做的工作是：如何确保只有能够被打印的字体才能在屏幕上正常地显示出来，换句话说也就是：在屏幕上显示的字体就是能够被打印的字体。

10.6 确定打印队列的状态

问题

有时，程序员需要显示存在于系统中的各种打印机中打印作业的状态，但似乎不能找到一种方法来查询当前哪些打印机是可用的以及在这些打印机上正在运行哪些作业。

那么，如何利用 Windows API 函数来查询可用的打印机及其所处的工作状态呢？

方法

在 Windows NT 版本以及 Windows 95 中，增加了一批全新的、专门用来处理打印机及打印队列状态的 API 函数。在本节中，将讨论 API 函数 EnumPrinters 与 EnumJobs，这两个函数可提供给程序员确定打印作业状态所需的信息。

在 Windows 95 中添加的函数 EnumPrinters，用来列出所有正在有效运行的打印机的名称及其标识符，此函数可用于列出用户能够访问的无论是在本地工作的还是在网络上运行的所有打印机，在本节的例子中，我们只考虑本地打印机的情况。

新的 Windows 95 API 中增加的另一个函数 EnumJobs，用来列出所指定打印机上正在打印的作业，为了能够使用这个函数，必须有打印机的句柄，这就需要使用另外一个 API 函数 OpenPrinter。

需要注意的是，因为这几个 API 函数的功能只在 Windows 95 以及 Windows NT 中能够实现，所以我们必须使用 Visual C++ 2.1（或以上版本）来编译下面所设计的项目文件。

步骤

打开与运行按照如下步骤生成的 Visual C++ 2.1 的例子程序 CH106.MAK。选择菜单 Dialog，并从此菜单中选择菜单项 Printer Jobs，此时显示一个所有可用打印机的对话框，在第一个列表框中选择其中一个打印机后，即在第二个列表框中显示该打印机的作业列表，图 10-7 所示的例子表示：在被选中的打印机上只有一个正在运行的打印作业。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++ 2.x 中利用 AppWizard 创建新的项目文件 CH106.MAK。

2. 进入 AppStudio，创建新的对话框模板。在新的对话框模板中，把按钮 OK 与按钮 Cancel 移到对话框的底部，在此对话框中添加两个列表框，并在这两个列表框的上面添加两个标题分别为 Available Printers 以及 Jobs：的静态文本域，然后保存此对话框模板。

3. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，把此新类命名为 CPrintQDlg。

4. 在 ClassWizard 中，从下拉组合框中选择 CPrintQDlg。从对象列表中选择对象 CPrint-QDlg，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Function 添加新的函数 OnInitDialog，把下面的代码输入到 CPrintQDlg 的方法 OnInitDialog 中。

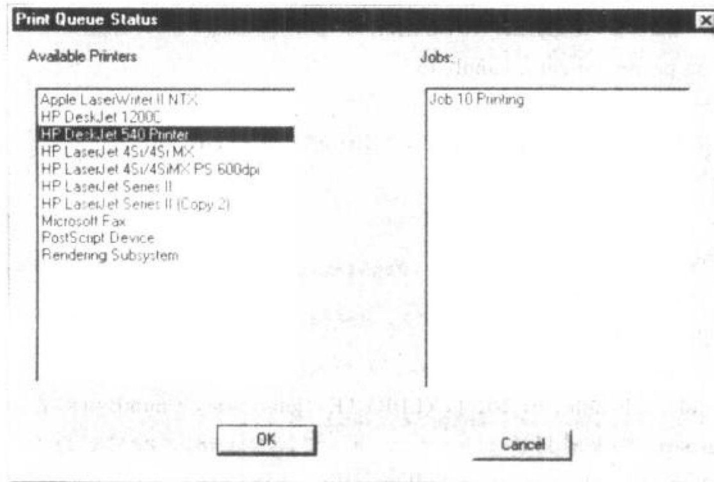


图 10-7　显示打印机作业队列及状态的对话框

```
BOOL CPrintQDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    int size = 4096;
    unsigned long sizeNeeded = 0;
    unsigned long numPrinters;
    PPRINTER_INFO_1 pPrinters;
    pPrinters = (PPRINTER_INFO_1) LocalAlloc ( (LMEM_FIXED | LMEM_ZEROINIT), size );
int ret = EnumPrinters ( PRINTER_ENUM_LOCAL, NULL, 1, (LPBYTE) pPrinters, size,
&sizeNeeded, &numPrinters);
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    for ( int i=0; i< (int) numPrinters; ++i ) {
        list->AddString ( pPrinters [i] .pName );
    }
    LocalFree ( pPrinters );
    return TRUE; // return TRUE unless you set the focus to a control
                 // EXCEPTION: OCX Property Pages should return FALSE
}
```

5. 在 ClassWizard 中，从下拉组合框中选择 CPintQDlg。从对象列表中选择对象 IDC _ LIST1，从消息列表中选择消息 LBN _ SELCHANGE，点击按钮 Add Function 添加新的函数 OnSelectPrinter，把下面的代码输入到 CPrintQDlg 的方法 OnSelectPrinter 中。

```
void CPrintQDlg::OnSelectPrinter ()
{
```

```
HANDLE handle;
JOB_INFO_1 jobs [10];
// Get the string from the list box
CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
char buffer [256];
list->GetText ( list->GetCurSel (), buffer );
// First, open the printer to get a handle to it
if ( ! OpenPrinter (buffer, &handle, NULL ) ) {
   MessageBox ( " Could not open printer", " Error", MB_OK );
return;
}
// Enumerate the jobs
DWORD size = sizeof (jobs);
DWORD numBytes = 0;
DWORD numEntries = 0;
int ret = EnumJobs ( handle, 0, 10, 1, (LPBYTE) jobs, size, &numBytes, &numEntries );
// Add the results to the jobs list
CListBox *list2 = (CListBox *) GetDlgItem (IDC_LIST2);
for ( int i=0; i< (int) numEntries; ++i ) {
   char buffer [80];
sprintf (buffer, " Job %ld", jobs [i] .JobId );
if ( jobs [i] .pStatus != NULL )
   strcat ( buffer, jobs [i] .pStatus );
else {
     switch ( jobs [i] .Status ) {
case JOB_STATUS_PAUSED:
     strcat ( buffer, " Paused");
break;
     case JOB_STATUS_PRINTED:
     strcat ( buffer, " Printed");
     break;
case JOB_STATUS_PRINTING:
     strcat ( buffer, " Printing");
       break;
case JOB_STATUS_SPOOLING:
     strcat ( buffer, " Spooling");
       break;
     }
}
list2->AddString ( buffer );
}
}
```

6. 进入 AppStudio，选择菜单资源列表。选择菜单 IDR_CH106TYPE，添加标题为 Dialog

的新的主菜单，在此主菜单上，添加标题为 Printer Jobs、标识符为 ID _ PRINTER _ JOBS 的新菜单项，保存此菜单资源与资源文件。

7. 进入 ClassWizard，从下拉组合框中选择 CCh106App。从对象列表中选择对象 ID _ PRINTER _ JOBS，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新的函数 OnPrinterJobs，把下面的代码输入到 CCh106App 的方法 OnPrinterJobs 中。

```
void CCh106App::OnPrinterJobs ( )
{
CPrintQDlg dlg;
dlg.DoModal;
}
```

8. 把下面一行代码添加到源文件 CH106.CPP 的顶端。

```
#include " printqdl.h"
```

9. 编译与运行此例子程序。

用法

函数 EnumPrinters 采用相当多的参数，但在这些参数中，最重要的是第一个与第四个参数。第一个参数用来指定所需要查看的打印机，在本例中，只请求查看与计算机物理相连的本地打印机的情况，故此参数被设定为 PRINTER _ ENUM _ LOCAL，它的另外一个可替换的参数用来指定网络打印机，也就是说此计算机与网络相连的所有打印机。

函数 EnumPrinters 的第四个参数是一个被分配了空间的结构缓冲区，它将用来返回有关打印机的信息。如果此缓冲区没有被分配足够的空间，函数 EnumPrinters 就会失败并且 API 函数 GetLastError 将返回空间不够的信息，在本例中，参数 sizeNeeded 用来存放返回所有信息所需的空间大小值。

当打印机列表被设置后，内存就会被清理，数据被存放在对话框的列表框中。用户此时就可从列表框中选择所需的打印机了，当某打印机被选择后，就会调用方法 OnSelectPrinter，此方法利用 API 函数 OpenPrinter 来检索所选择打印机的句柄，如果程序没有此打印机的访问权限，此函数就调用失败。当调用成功时，此打印机的句柄就被传递给 API 函数 EnumJobs，此函数返回由参数 size 所指定数目的一串打印机的作业。另外，此函数还能够返回以下两种结构类型中的任意一种，一种比较简单的结构类型是 JOB _ INFO _ 1，另一种比较详细而且复杂的结构类型是 JOB _ INFO _ 2，至于需要哪种结构类型，可通过向此函数中第四个参数传递相应的值来确定。

当函数 EnumJobs 成功地返回后，打印机作业的 ID 连同这些作业的状态就被显示在列表框中，如果没有作业，则列表框为空。关于函数 EnumJobs，有一点需要注意：在有些文档中，把该函数的第二个参数（此参数用来描述从第几个作业开始）认为是从 1 开始算起的，这样的说法是不对的，打印作业是从 0 开始算起的，也就是说当想要显示所有打印作业时，此参数应设置为 0，如果此参数被设置为大于 0 的值，则会从打印队列中“丢失”作业。

第 11 章　应用程序之间的通信

在最初的 Windows 操作系统中，最有用而却未引起注意的特征之一是：它不仅能够在同一时间内运行多个程序，而且能够在这些程序之间进行通信。Windows 的剪贴板是程序间进行通信的最初方式，之后，这种通信方式就被在程序间进行动态数据交换的方式（DDE 方式，Dynamic Data Exchange）所取代，在 Windows 3.1 中，Microsoft 的开发人员引入了在多程序间通信的一种较新和较好的方式，即对象的链接与嵌入（Object Linking and Embedding，OLE）方式。

本章中，将讨论如何从一个程序向另一个程序获取信息的各种方法。我们首先考虑从 File Manager 中拖一个对象放到自己的应用程序中以及从自己的一个应用程序中拖一个对象放到另一个应用程序中的情况；接着，将考察出自于客户机和服务器的早期数据交换方式，即 DDE 方式；最后，我们将讨论用在 OLE 应用程序之中的对象嵌入的概念。

在当今世界，几乎没有哪个程序是孤立存在的，程序之间、设备之间以及性能之间都存在着或多或少的交互作用。当把一个文件发送到打印假脱机系统，或利用设备驱动程序访问扫描仪，或利用 OLE 把 Microsoft Excel 电子数据表格嵌入到应用程序时，都是在进行信息通信。因此，应用程序间的通信将成为 Windows 应用程序的一个不可缺少的部分，做为一名高级应用程序开发人员，理所当然地应当掌握 Windows 95 的这些重要的技术和方法。

1. 如何支持从系统程序拖放到应用程序

本节将学习如何接受从 File Manager 拖放到应用程序中的文件名。在本节中将了解到：当应用程序正在运行时，如何接受这些文件名，以及当从 File Manager 或 Windows Explorer 中给定文件名时，如何利用此隐藏资源来运行应用程序。

2. 如何支持从应用程序拖放到另一应用程序

本节将学习如何把文件名拖放到其他应用程序并运行它们，或者把现有的文件拖放到正在运行的应用程序以便此应用程序把它打开。在这里，还将学习如何确定某程序实例是否支持拖放，以及如何把拖放的信息发送给其他正在运行的应用程序。

3. 如何查看剪贴板的内容

当用户从菜单 Edit 中选择菜单项 Paste 时，在应用程序中会发生什么呢？剪贴板支持存储多少种类型的数据呢？本节将讨论如何查看剪贴板中的内容以及查找在剪贴板中是否有想要粘贴到应用程序中的数据类型。

4. 如何利用剪贴板进行剪切、拷贝和粘贴

当用户在应用程序中选择数据后，可能需要对其进行剪切、拷贝或粘贴，本节将讨论如何通过利用 Windows 95 API 使剪贴板支持应用程序中的所有操作。

5. 如何编写动态数据交换客户程序

如果在自己应用程序中能够利用其他应用程序的功能，那是再好不过的。例如，Microsoft Word 就有一些非常好用但却难以实现的功能，那么为什么不把 Word 的这些功能引入到自己应用程序中呢？本节中，将讨论如何实现DDE客户服务以便使其他应用程序为自己执行

任务。

6. 如何编写动态数据交换服务程序

如果在 Windows 桌面上只有一个“引擎”在运行而多个应用程序被“挂”在此“引擎”上以执行各自相关的任务，那不是很好吗？数据库的事务处理、打印服务、数据查找以及硬件接口等，都能够做为 DDE 服务器设备来实现，该设备使运行的程序利用作为“引擎”的实例来执行任务。本节中，将讨论基本 DDE 服务程序的需求，以及如何把这些需求放入到应用程序中。

7. 如何在动态数据交换中支持系统主题

DDE 中的系统主题对于客户程序询问 DDE 服务器中提供何种功能是非常有用的。程序员通过系统主题可以查到服务器提供何种服务，以及有什么其他主题可以应用等，本节中，将讨论有关 DDE 中系统主题的需求以及如何在应用程序中实现这些需求。

8. 如何使文件对象的链接与嵌入（OLE）兼容

如果想使文件能够在 OLE 环境下应用，那么这些文件就需要与复合文档的 OLE 规范一致。本节中，将讨论利用复合文档所需做的必要工作以及如何在应用程序中利用此功能。

9. 如何创建 OLE 服务程序对象

本节中，将编写一个 OLE 服务程序对象。此对象能够被嵌入到其他应用程序中，并仍然执行自己应用程序的任务。在这里，将一步步讨论如何创建一个 OLE 服务程序，以及在 Windows 95 中利用此服务程序所需做的必要工作。

表 11-1 列出了本章用到的 Windows 95 API 函数。

表 11-1 第 11 章中用到的 Windows 95 API 函数

DragAcceptFiles	DragQueryFile	SetCapture
GetAsyncKeyState	ReleaseCapture	MessageBox
SendMessage	ScreenToClient	GlobalFree
PsotMessage	GlobalAlloc	GlobalLock
GlobalUnlock	GlobalReAlloc	OpenClipboard
IsClipboardFormatAvailable	GetClipboardData	MakeProcInstance
DdeInitialize	DdeCreateStringHandle	DdeConnect
DdeGetLastError	DdeDisconnect	DdeFreeStringHandle
DdeUninitialize	DdeClientTransaction	DdeGetData
DdeCmpStringHandles	DdeCreateDataHandle	DdeNameService
DdePostAdvise	StgOpenStorage	StgCreateDocfile

11.1 支持从系统程序拖放到应用程序

问题

通常，用户希望接受从其他应用程序诸如 Windows Explorer 或 File Manager（在 Windows 3.x 中）中拖放的文件名。如何来接受这些文件名并对其进行处理呢？由于 MFC 只具有接受文件的功能而没有处理这些文件的功能，因此我们就需要利用 Windows 95 的 API 函数来完成这一任务。

那么，如何利用 Windows 95 API 函数来获取被用户拖放到应用程序窗口的文件名并对其进行处理呢？

方法

拖放文件的技术在 Windows 95 以及 Windows 3.x 中都证明是十分成功的，因此，在 Windows 95 中，一些基本的文件操作实际上都已转向了利用拖放的方式来进行。

把文件拖放到应用程序使得该应用程序窗口能够接受此文件的过程实际上并不难，只要知道其工作原理，就会发现它不过是一件十分简单的事情。

MFC 只有一种功能用于此过程的 DragAcceptFiles 部分，即通知 Windows 系统此窗口将接受拖放到它上面的文件。

此过程的第二部分是处理所获得的 Windows 信息 WM_DROPFILES，此消息表明 Windows 正试图把文件拖放到应用程序的窗口中。本节中，将介绍如何在应用程序窗口中利用此消息来获取被拖放的文件名。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1101.MAK。启动程序并在启动时选择列表框中显示的第二个文档模板，此时将会看到一个有空白列表框嵌入的视图，接着，打开 Windows Explorer 并从显示的目录中选择一个文件，选择此文件后，按下鼠标左键把鼠标移到所显示的视窗的区域内，文件就被拖放到自己的窗口中，释放鼠标左键，这时就会看到在视图的列表框中显示出该文件名，如图 11-1 所示。

执行下面的过程，便会实现上述功能。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH1101.MAK。进入资源编辑器并创建一个新的对话框，把新对话框的风格位设置如下：Not Visible、Child Window、No Border，在此对话框中添加一个列表框。

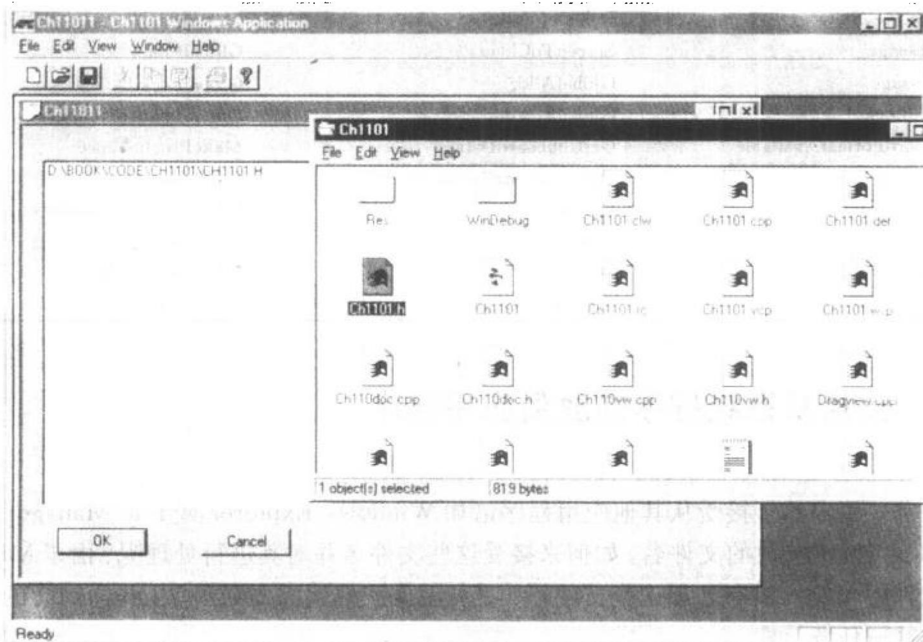


图 11-1　从 Windows Explorer 中拖放文件的视图

2. 选择 ClassWizard，为新的对话框模板创建新的对话框类 CDragView，把此新类的基类设置为 CFormView，保存此类并退出 ClassWizard。

3. 在类 CDragView 中，添加名为 OnInitialUpdata 的函数。把下面的代码输入到 CDragView 的 OnInitialUpdata 方法中。

```
void CDragView:: OnInitialUpdata (void)
{
    CView:: OnInitialUpdate ();
    DragAcceptFiles ();
}
```

4. 利用下面的语法把新的条目添加到类 CDragView 的消息映射中，只需添加下面标记为黑体的行即可。

```
BEGIN_MESSAGE_MAP (CDragView, CFormView)
    //{{AFX_MSG_MAP (CDragView)
    ON_WM_DROPFILES ()
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}} AFX_MSG_MAP
END_MESSAGE_MAP ()
```

5. 继续把下面定义的方法添加到 CDragView 的源文件中。

```
void CDragView:: OnDropFiles (HDROP hDropInfo)
{
    char fileName [_MAX_PATH];
    // Assign the list box object
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    // First, determine how many files are dropped.
    UINT numFiles = DragQueryFile (hDropInfo, -1, NULL, 0);
    // Next, loop through and add the files to the listbox
    for ( UINT i=0; i<numFiles; ++i ) {
        // Get the file name
        DragQueryFile (hDropInfo, i, fileName, _MAX_PATH);
        // Store it in the list box
        list->AddString ( fileName );
    }
}
```

6. 把下面的说明添加到 CDragView 的头文件 DRAGVIEW.H 中，同样，这里也只需添加标记为黑体的行。

```
protected:
    virtual void OnInitialUpdate (void);
    virtual ~CDragView ();
    virtual void DoDataExchange (CDataExchange * pDX);
    // DDX/DDV support
    // Generated message map functions
    //{{AFX_MSG (CDragView)
```

```
afx_msg void OnDropFiles (HDROP hDropInfo);
```

7. 把下面几行添加到例子程序对象 CCh1101App 的方法 InitInstance 中，这些代码应添加在 InitInstance 中紧挨着 pDocTemplate 的下面。

```
pDocTemplate = new CMultiDocTemplate (
    IDR_CH1101TYPE,
    RUNTIME_CLASS (CCh1101Doc),
    RUNTIME_CLASS (CMDIChildWnd),        // standard MDI child frame
    RUNTIME_CLASS (CDragView));
AddDocTemplate (pDocTemplate);
```

8. 最后，把下面一行添加到 CCh1101App 的源文件 CH1101.CPP 的顶端。

```
#include "dragview.h"
```

9. 编译与运行此例子程序。

用法

当调用方法 DragAcceptFiles 时，Windows 将利用风格位 WS_EX_ACCEPTFILES 来标记应用程序窗口，此风格位向 Windows 表明，当把文件拖放到此应用程序窗口并释放鼠标键时，一个用来表明此事件发生的消息便被发送到此窗口。

当鼠标键被释放后，消息 WM_DROPFILES 就被发送到鼠标所在的窗口，此消息有一个参数存放在消息结构的 wParam 元素中，此参数包含一个存放在 Windows 中的拖放信息结构的句柄。

当收到具有拖放结构的消息 WM_DROPFILES 后，就可以做下面的事情。首先，调用函数 DragQueryFile 来查找在此结构中存放了多少个文件，这是通过在调用此函数时，把参数 index 设置为－1 来完成的。

当知道结构中存放的文件数后，就可以通过多次调用函数 DragQueryFile 从此结构中提取文件名，并根据自己的需要来处理这些文件名。在本例中，对这些文件名所进行的处理，只是把它们存放在视图的列表框中而已。

注释

本节所介绍的内容是许多不同应用程序的基础。我们可以根据对这些被拖放的文件名的不同处理，编制出用户所需的不同应用程序。例如，一个编辑器或许就要利用被拖放的文件名把此文件装入到编辑缓冲区，并把它显示在编辑窗口中，一个筛选程序或许也需接受文件名，并利用它把一些用户定义的词从文本中筛选出来。

本节是理解 Windows 中所有其他拖放操作的核心部分，如果想在 Windows 95 中进一步做其他有关拖放文件的工作，那么，理解这一节所描述的过程是十分必要的。

11.2 支持从应用程序拖放到另一应用程序

问题

在上一节中，介绍了如何把文件从 Windows Explorer 中拖放到应用程序的窗口中。但是当不仅需要具有把文件拖放到应用程序窗口的功能，而且需要具有在应用程序的两个窗口间互相拖放的功能时，Windwos 95 是如何做的呢？

方法

让我们首先来考察一下 Windows Explorer 是如何把文件拖放到应用程序窗口中的。当用

户从 Windows Explorer 中选择文件并把它拖放到应用程序窗口中时，Windows Explorer 向应用程序发送了一个 WM _ DROPFILES 消息。

在应用程序中，我们可以利用相同的方法把文件从一个窗口拖放到另一个窗口，所需做的就是让识别消息 WM _ DROPFILES 的程序能够接收到与此相同的消息，以便让用户把文件拖放到这些应用程序。本节中将对这种方法作深入地讨论，利用这种方法，使得创建程序过滤器成为可能，过滤器筛选文件，然后让用户利用一个简单的拖放操作把筛选的结果“链接”到其他的应用程序中。

本节中，我们将讨论消息WM _DROPFILES的结构，并介绍如何创建适合此结构的实例。

步骤

打开与运行上节所生成的 Visual C++的例子程序 CH1101. MAK。启动程序并在启动时选择列表框中显示的第二个文档模板，此时将看到一个嵌有空白列表框的视图。

接着，打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1102. MAK。选择菜单 File 的菜单项 New 或在工具条中选择图标 New File，然后，选择第二个文档模板并点击按钮 OK，一个装满文件的列表框的视图便会显示出来，从列表框中选择一个文件，然后把鼠标移到视窗的其余部分（在列表框之外的部分），按下鼠标左键并拖动鼠标到第一个（列表框为空的）视图，释放鼠标左键，则会看到此文件名被显示在视图的列表框中，如图 11-2 所示。

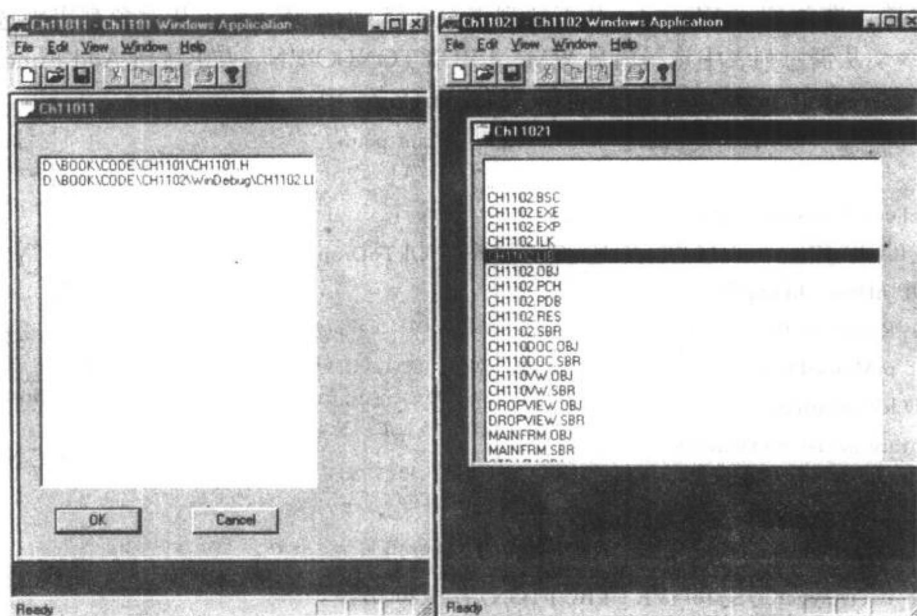


图 11-2　从一个应用程序拖放文件到另一个应用程序的拖放视图

为了在例子程序中实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件，把此项目文件命名为 CH1102. MAK。进入资源编辑器，创建新的对话框，把新对话框的风格位设置如下：Not Visible、Child Window、No Border，在此对话框中添加一个列表框。

2. 选择ClassWizard，为新的对话框模板创建新的对话框类，命名此新类为CDropView，把此新类的基类设置为CFormView，保存此类，退出ClassWizard。

3. 在类CDropView中，添加名为OnInitialUpdata的函数，把下面的代码添加到CDropView的方法OnInitialUpdata中。

```
void CDropView::OnInitialUpdate (void)
{
    WIN32_FIND_DATA fileinfo;
    // Get the list box
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    // Add the files in the current directory to it
    HANDLE hdl = FindFirstFile ( "*.*", &fileinfo );
    int ret = TRUE;
    if ( hdl == INVALID_HANDLE_VALUE )
       ret = FALSE;
    while ( ret ) {
       list->AddString ( fileinfo.cFileName );
       ret = FindNextFile ( hdl, &fileinfo );
    }
}
```

4. 接着，进入ClassWizard，从下拉列表中选择CDropView 。从对象列表中选择对象CDropView，从消息列表中选择消息WM_LBUTTONDOWN，点击按钮Add Function，把下面的代码添加到CDropView的方法OnLButtonDown中。

```
void CDropView::OnLButtonDown (UINT nFlags, CPoint point)
{
    char szDropPathName [200];
    BOOL fCallDefProc = FALSE, fInNonClientArea, fOkToDrop;
    HDROP hDrop, hDropT;
    LONG lResult = 0;
    POINT ptMousePos;
    HWND hWndSubject;
    // Capture mouse movements
    SetCapture ();
    // Wait for mouse to be released
    do {
    } while (GetAsyncKeyState (VK_LBUTTON) & 0x8000);
    ReleaseCapture ();
    // See if it is in a valid window
    ::GetCursorPos (&ptMousePos);
    hWndSubject = ::WindowFromPoint (ptMousePos);
    if (! IsWindow (hWndSubject))
 {
        fOkToDrop = FALSE;
```

```
        }
else
        {
        fOkToDrop = TRUE;
        }
if (! fOkToDrop) {
        :: MessageBox (NULL, "Not a window", "Error", MB_OK );
        return;
}
// Is the cursor in the window's non-client area?
fInNonClientArea = (HTCLIENT ! =
        :: SendMessage (hWndSubject, WM_NCHITTEST, 0,
        (LONG) MAKELONG (ptMousePos.x, ptMousePos.y)));
        // Create drop-file memory block and initialize it
:: ScreenToClient (hWndSubject, &ptMousePos);
hDrop = DragCreateFiles (&ptMousePos, fInNonClientArea);
if (hDrop == NULL)
        {
        :: MessageBox (NULL,
            "Insufficient memory to drop file (s) .",
            "Error", MB_OK);
        return;
}
// Add the file by getting the text from the list box
CListBox *list = (CListBox * ) GetDlgItem (IDC_LIST1);
int idx = list->GetCurSel ();
list->GetText ( idx, szDropPathName );
// Get the current directory name
char buffer [_MAX_DIR];
char temp [_MAX_PATH];
_getcwd ( buffer, _MAX_DIR );
sprintf (temp, "%s\%s", buffer, szDropPathName );
hDropT = DragAppendFile (hDrop, temp);
if (hDropT == NULL) {
            :: MessageBox (NULL,
            "Insufficient memory to drop file (s) .",
            "Error", MB_OK);
        GlobalFree (hDrop);
        hDrop = NULL;
        return;
}
else {
        hDrop = hDropT;
```

```
    }
    if (hDrop != NULL)
    {
        // All pathnames appended successfully,
        // post the message
        // to the drop-file client window
          :: PostMessage (hWndSubject, WM_DROPFILES, (WPARAM) hDrop, 0L);
        // Don't free the memory,
        // the Dropfile client will do it
    }
    CFormView:: OnLButtonDown (nFlags, point);
}
```

5. 接着，把下面的代码块添加到 DROPVIEW.CPP 的顶端，紧接在宏调用 IMPLEMENT_DYNCREATE 的后面。

```
typedef struct {
        WORD wSize;                  // Size of data structure
        POINT ptMousePos;            // Position of mouse cursor
        BOOL fInNonClientArea;       // Was the mouse in the
                                     // window's non-client area
} DROPFILESTRUCT, FAR *LPDROPFILESTRUCT;
HDROP FAR DragCreateFiles (LPPOINT lpptMousePos,
                                BOOL fInNonClientArea)
{
    HGLOBAL hDrop;
    LPDROPFILESTRUCT lpDropFileStruct;
    // GMEM_SHARE must be specified because the block will
    // be passed to another application
    hDrop = GlobalAlloc (GMEM_SHARE | GMEM_MOVEABLE | GMEM_ZEROINIT, sizeof
(DROPFILESTRUCT) + 1);
    // If unsuccessful, return NULL
    if (hDrop == NULL) return ( (HDROP) hDrop);
    // Lock block and initialize the data members
    lpDropFileStruct = (LPDROPFILESTRUCT) GlobalLock (hDrop);
    lpDropFileStruct->wSize = sizeof (DROPFILESTRUCT);
    lpDropFileStruct->ptMousePos = *lpptMousePos;
    lpDropFileStruct->fInNonClientArea = fInNonClientArea;
    GlobalUnlock (hDrop);
    return ( (HDROP) hDrop);
}
HDROP FAR DragAppendFile (HGLOBAL hDrop, LPCSTR szPathname)
{
    LPDROPFILESTRUCT lpDropFileStruct;
    LPCSTR lpCrnt;
```

```
    WORD wSize;
    lpDropFileStruct = (LPDROPFILESTRUCT) GlobalLock (hDrop);
    // Point to first pathname in list
    lpCrnt = (LPSTR) lpDropFileStruct + lpDropFileStruct->wSize;
    // Search for a pathname were first byte is a zero byte
    while ( * lpCrnt)        // While the 1st char of path is non-zero
      {
      while ( * lpCrnt) lpCrnt++; // Skip to zero byte
      lpCrnt++;
      }
    // Calculate current size of block
    wSize = (WORD) (lpCrnt - (LPSTR) lpDropFileStruct + 1);
    GlobalUnlock (hDrop);
    // Increase block size to accommodate new pathname being appended
    hDrop = GlobalReAlloc (hDrop, wSize + lstrlen (szPathname) + 1,
                 GMEM_MOVEABLE | GMEM_ZEROINIT | GMEM_SHARE);
    // Return NULL if insufficient memory
    if (hDrop == NULL) return ( (HDROP) hDrop);
    lpDropFileStruct = (LPDROPFILESTRUCT) GlobalLock (hDrop);
    // Append the pathname to the block
    lstrcpy ( (LPSTR) lpDropFileStruct + wSize - 1, szPathname);
    GlobalUnlock (hDrop);
    return ( (HDROP) hDrop);     // Return the new handle to the block
}
```

6. 把下面的说明添加到 CDropView 的头文件 DROPVIEW.H 中。

```
protected:
virtual void OnInitialUpdate (void);
```

7. 把下面几行添加到例子程序对象 CCh1102App 的方法 InitInstance 中，这些代码应添加在 InitInstance 中紧挨着 pDocTemplate 的下面。

```
pDocTemplate = new CMultiDocTemplate (
    IDR_CH1102TYPE,
    RUNTIME_CLASS (CCh1102Doc),
    RUNTIME_CLASS (CMDIChildWnd),        // standard MDI child frame
    RUNTIME_CLASS (CDropView));
AddDocTemplate (pDocTemplate);
```

8. 最后，把下面一行添加到 CCh1102App 的源文件 CH1102.CPP 的顶端。

```
#include "dropview.h"
```

9. 编译与运行此例子程序。

用法

首先，用户从 MFC 窗口或视图的列表框中选择一个文件，然后在视图的用户区中某一位置按下鼠标左键，这时就会启动视图类的方法 OnLButtonDown。

方法 OnLButtonDown 通过为窗口设置捕捉(设置此捕捉的目的在于：即使鼠标落在窗口

边界之外，Windows 也在不断地发送鼠标的消息，直到用户释放鼠标键）来等待用户释放鼠标左键（由函数 GetAsyncKeyState 来检查鼠标的状态），当用户释放鼠标键后，函数 GetCursorPos 便被调用，此函数用来检索鼠标光标所在位置的屏幕坐标。接着，通过调用函数 WindowFromPoint 来确定鼠标此时落在了哪个窗口上，如果它落在的位置没有窗口，则函数返回。

如果在释放鼠标的位置处找到了窗口，则调用函数 DragCreateFiles，此函数首先建立一个标志为 GMEM_SHARED 的全局内存块，标志 GMEM_SHARED 用来表明：其他的应用程序也能够象操作系统或分配内存的应用程序那样，来利用此内存块。内存块首先将被转化为 LPDROPFILESTRUCT 型的结构，然后它将利用所找到的窗口以及一个用来说明鼠标是否落来在此窗口非客户区的标志来进行初始化，接着此内存块被开锁，并把它的句柄返回给调用函数。

接下来，程序查找正确的列表项，并提取此表项的文本，然后把此文本传递给函数 DragAppendFile，函数 DragAppendFile 用来把文件添加到鼠标所落在窗口的文件列表中，最后，整个结构（利用句柄）通过调用函数 PostMessage 被发送到鼠标落在的窗口。在这里，特别需要注意的是：要利用函数 PostMessage 而不要利用函数 SendMessage 来发送消息，因为利用函数 SendMessage 将会形成一个无限循环。

还需要注意的是：当分配了全局内存块并把它发送到其他应用程序用于拖放文件之后，不要释放此全局内存块，因为释放它的工作是由其他窗口或操作系统来处理的。

11.3 查看剪贴板的内容

问题

有时需要知道在应用程序中是否有文本被粘贴在剪贴板中，或者需要根据剪贴板中是否有用于拷贝与粘贴的文本，来激活或禁止对话框中的拷贝、粘贴功能，例如，当有被选择的文本时，就激活按钮 Copy，当没有被选择的文本时，就禁止按钮 Copy。

那么，如何来查看剪贴板中的内容呢？

方法

在 Windows 中，利用剪贴板的功能进行通信是一个非常好的办法。利用剪贴板，可以在应用程序之间很容易地共享信息，并且还不会象利用 DDE 或 OLE 进行通信那样需要很多的额外开销。

在本节中，我们将讨论如何查看剪贴板中可供使用的内容以及如何从中提取信息，另外，还将说明如何利用一行简单代码就能把剪贴板的文本直接粘贴到编辑控制。

值得注意的是，虽然本节中介绍的各种功能都是十分简单的，但把这些功能放在一起并要理解清楚所有事件发生的顺序时，就会变得很麻烦，另外还必须注意有关分配与释放空间、锁定与开锁对象的方法，以确保不使系统崩溃。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1103.MAK。选择菜单 Dialog 的菜单项 Clipboard Dialog，此时便会看到一个新的对话框窗口，如图 11-3 所示，如果在剪贴板中有内容，则会看到按钮 Copy 处于激活状态，并且在对话框顶端的静态文本域中显示出剪贴板中的内容，如果剪贴板中没有任何文本格式的内容，则在静态文本域中将什么也看

不到，并且按钮 Copy 处于变灰状态。

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH1103. MAK。进入资源编辑器，从菜单列表中选择主菜单，并添加标题为 Dialog 的新主层菜单，在主菜单 Dialog 上，添加标题为 Clipboard Dialog 的菜单项，并设定此下拉菜单项的标识符为 ID _ CLIP _ DLG。

2. 在资源列表中，通过点击 Dialog 资源创建一个新的对话框模板，点击按钮 New，把按钮 OK 与 Cancel 移到对话框的底部，在对话框的顶端，添加标题为 Current Clipboard Contents 的静态文本域，在此文本域的下面添加第二个无标题、标识符为 IDC _ CLIP _ CONTENTS 的文本域。

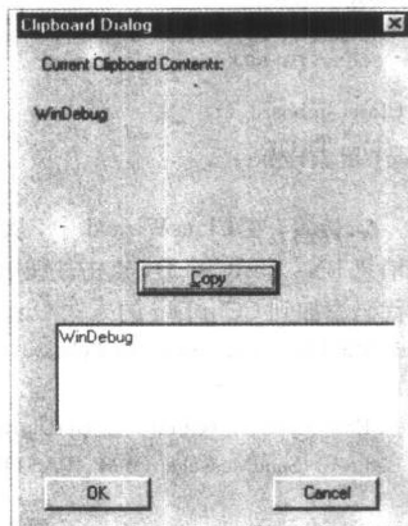


图 11-3 拷贝剪贴板的对话框

3. 在此空的静态文本域的下面添加按钮，把此按钮的标识符设定为 IDC _ BUTTON1、标题设定为 &Copy。

4. 在此对话框中，紧挨着按钮的下面添加编辑域，删除编辑域中的任何标题。

5. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，并把此新类命名为 CClipDlg。从 ClassWizard 的对象列表中选择标识符 CClipDlg，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Function，把下面的代码添加到 CClipDlg 的方法 OnInitDialog 中。

```
BOOL CClipDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    CenterWindow ();
    // First, open the clipboard
    OpenClipboard ();
    // See if the format is available
    if ( ! IsClipboardFormatAvailable (CF_TEXT) ) {
      // NO data available. Disable copy button.
      GetDlgItem (IDC_BUTTON1) ->EnableWindow (FALSE);
    }
    else {
      // Data available. Enable copy button
      GetDlgItem (IDC_BUTTON1) ->EnableWindow (TRUE);
    // Get the data
    HANDLE hData = GetClipboardData (CF_TEXT);
    // Unlock the handle
    LPSTR lpData = (LPSTR) GlobalLock (hData);
    // Allocate a block of this size
```

```
    char *str = new char [GlobalSize (hData)];
    // Copy it into the string
    lstrcpy ( str, lpData );
    // Set the static text field to be this data
    GetDlgItem (IDC_CLIP_CONTENTS) ->SetWindowText ( str );
    // Clean up after ourselves
    delete str;
    GlobalUnlock ( hData );
    }
  CloseClipboard ();
  return TRUE;
}
```

6. 接着，在 ClassWizard 中，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 添加新函数 OnCopyFromClipboard。把下面的代码添加到 CClipDlg 的方法 OnCopyFromClipboard 中。

```
void CClipDlg:: OnCopyFromClipboard ()
{
    CEdit *edit = (CEdit *) GetDlgItem (IDC_EDIT1);
    edit->SendMessage (WM_PASTE);
}
```

7. 在 ClassWizard 中，从下拉列表中选择对象 CCh1103App。从对象列表中选择对象 ID_CLIP_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnClipDlg。把下面的代码添加到 CCh1103App 的方法 OnClipDlg 中。

```
void CClipDlg:: OnClipDlg ()
{
    CClipDlg dlg;
    dlg.DoModal ();
}
```

8. 最后，把下面一行添加到源文件 CH1103.CPP 顶端的 include 文件列中。

```
#include "clipdlg.h"
```

9. 编译与运行此例子程序。

用法

当第一次显示对话框时，方法 OnInitDialog 被调用。此方法首先通过调用函数 OpenClipboard 打开剪贴板以便进行读写操作。接着，通过调用函数 IsClipboardFormatAvailable 来确定剪贴板中的内容是否是所需要的，由于我们只考虑文本内容，所以在调用此函数时，利用 CF_TEXT 做为所需格式的参数，此函数将扫描剪贴板，从而确定其中是否有所指定格式的条目（在这里为文本格式），如果有，此函数将返回 TRUE；否则，返回 FALSE。

如果在剪贴板中没有文本，方法 OnInitDialog 只是禁止按钮 Copy。如果在剪贴板中有文本，此方法便利用函数 GetClipboardData 来获取此文本的句柄，函数 GetClipboardData 返回一个全局内存块的句柄，该内存块通过函数 GlobalLock "锁定"，从而使其能够被访问。函数 GlobalLock 返回一个能够被用来访问内存块中文本的临时指针，利用此指针，把剪贴板中的

数据拷贝到局部缓冲区，然后把此缓冲区传递给静态文本控制以便显示剪贴板中的文本。最后，通过开锁句柄并释放在程序中分配的数据空间来进行清理工作。

当用户按下按钮 Copy 后，函数 OnCopyFromClipboard 便被调用，此函数只是向编辑控制发送一个消息，告诉编辑控制把剪贴板的内容拷贝到它自已的缓冲区。

值得注意的是：当数据被拷贝后，它就与剪贴板的内容相互独立，因此改变编辑域中的内容不会影响到剪贴板中的内容。为了证明这一点，我们可以修改一下编辑域中的文本，然后再次按下 Copy 按钮，这时会发现，在编辑域中重新显示的文本依旧是被修改前的内容。

注释

利用剪贴板进行通信是应用程序间通信的最简单方法，也是在应用程序间传递简单文本或位图的最好方法，它通过用户进行剪切与粘贴的操作，在两个应用程序之间传递数据．从而完成程序间的信息通信。

当然，利用剪贴板非常简单．但它却是 Windows API 函数的一个重要组成部分。用户可能比较推崇（或害怕）利用 DDE 或 OLE 的方式在应用程序间进行可视编辑以及异步通信，但他们也需要掌握与应用剪贴板提供的拷贝、剪切与粘贴的功能。

11.4 利用剪贴板进行剪切、拷贝和粘贴

问题

在对话框的编辑控制中，常需要使用剪切、拷贝以及粘贴的命令，而目前对话框的编辑控制中只提供了利用 Ctrl－X、Ctrl－Y 与 Ctrl－Z 等键盘命令来执行这些操作。如果不想让用户利用这种方式进行操作，那么，如何在对话框中创建一些按钮使得用户可以利用这些按钮而不必使用键盘来执行这些命令呢？

方法

在 Windows 2.x 中，一般使用快捷键进行剪切、拷贝以及粘贴的命令，这些命令不太直观，当今用户已不再采用。

在有些编辑窗口中，利用菜单命令来执行有关编辑的操作，在前面的章节中，曾经介绍过如何利用快捷键在模式对话框中进行工作。

实践证明，对于在编辑窗口中执行剪切、拷贝以及编辑的操作，无论是使用快捷键还是使用菜单，都不如使用按钮那样简单、快速。本节中，我们将讨论如何使用按钮，在对话框中进行同编辑框的信息通信，从而完成利用剪贴板进行剪切、拷贝与粘贴的操作。这个过程听起来似乎需作很多工作，其实从下面的步骤可以看出，我们只需用几行代码就可完成这个任务。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1104.MAK。从菜单 Dialog 中选择菜单项 Cut Copy Paste，此时便会看到一个新的对话框窗口，如图 11-4 所示，在对话框顶端的编辑框中输入几行文本，然后从中选择一些文本，接着在编辑框中某一新位置点击一下，使得插入光标移到该位置处，点击按钮 Copy，接着点击按钮 Paste，此时，便会看到刚才所选择的文本被拷贝到了当前光标所在的位置处。利用按钮 Cut、Copy 以及 Paste 进行不同组合的操作，便可看出他们各自的功能。

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH1104.MAK。进入资源编辑器，从菜单列表中选择主菜单，利用标题 Dialog 添加新的主菜单，在主菜单 Dialog 上，添加标题为 Cut Copy Paste 的菜单项，并设定此下拉菜单项的标识符为 ID _ EDIT _ FUNCS。

2. 在资源列表中，通过点击 Dialog 资源创建新的对话框模板，点击按钮 New，把按钮 OK 与 Cancel 移到对话框的底部，在对话框的顶端，添加一个编辑框，在此编辑框的下面添加三个按钮。

3. 设定第一个按钮的标题为 &Cut，设定第二个按钮的标题为 &Copy，设定第三个按钮的标题为 &Paste。

4. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，并把此类命名为 CEditFuncDlg。

5. 从 ClassWizard 的对象列表中选择对象 IDC _ BUTTON1，从消息列表中选择消息 BN _ CLICKED，点击按钮 Add Function 添加新函数 OnCut，把下面的代码添加到 CEditFuncDlg 的方法 OnCut 中。

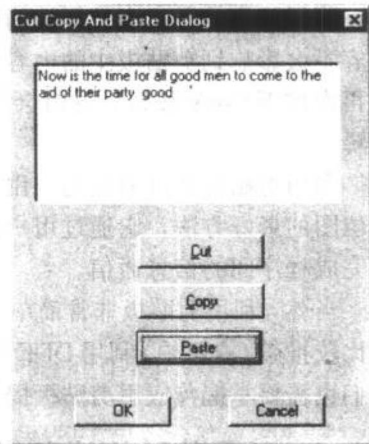


图 11-4 剪切、拷贝与粘贴对话框

```
void CClipDlg:: OnCut ()
{
    CEdit *edit = (CEdit *) GetDlgItem (IDC_EDIT1);
    edit->SendMessage (WM_CUT);
}
```

6. 从 ClassWizard 的对象列表中选择对象 IDC _ BUTTON2，从消息列表中选择消息 BN _ CLICKED，点击按钮 Add Function 添加新函数 OnCopy。把下面的代码添加到 CEditFuncDlg 的方法 OnCopy 中。

```
void CClipDlg:: OnCopy ()
{
    CEdit *edit = (CEdit *) GetDlgItem (IDC_EDIT1);
    edit->SendMessage (WM_COPY);
}
```

7. 从 ClassWizard 的对象列表中选择对象 IDC _ BUTTON3，从消息列表中选择消息 BN _ CLICKED，点击按钮 Add Function 添加新函数 OnPaste。把下面的代码添加到 CEditFuncDlg 的方法 OnPaste 中。

```
void CClipDlg:: OnPaste ()
{
    CEdit *edit = (CEdit *) GetDlgItem (IDC_EDIT1);
    edit->SendMessage (WM_PASTE);
}
```

8. 在 ClassWizard 中，从下拉列表中选择对象 CCh1104App。从对象列表中选择对象 ID _ EDIT _ FUNCS，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnCutCopyPasteDlg。把下面的代码添加到 CCh1104App 的方法 OnCutCopyPasteDlg 中。

```
void CCh1104App:: OnCutCopyPasteDlg ()
```

```
{
    CEditFuncDlg dlg;
    dlg.DoModal ();
}
```

9. 最后，把下面一行添加到的源文件CH1104.CPP顶端的include文件列中。

```
#include "editfunc.h"
```

10. 编译与运行此例子程序。

用法

本节所介绍的例子，或许是本书中最简单的例子，在这里，所做工作的只是编制了几个简单的函数。当按下按钮后，便调用相应的类方法，这些按钮只是把Windows API命令WM _ CUT、WM _ COPY以及WM _ PASTE发送给编辑控制。

然而，在"幕后"发生的是完全不同的状况。编辑控制根据传递的命令，来确定是把数据放置到剪贴板中（当接收Copy或Cut命令时）还是从剪贴板中获取信息（当接收Paste命令时），本例中，数据只是在一个应用程序中利用剪贴进行处理的。事实上，完全可以把一个应用程序中的文本拷贝或剪切到剪贴板，而后把剪贴板中的内容粘贴到另一个应用程序中，此例同上节的例子相比，是一个潜在的进程间通信的较完整例子。从这里，我们也可以看出Windows API函数所具有的强大功能。

注释

我们可以利用Visual Basic编制出同样的程序。下面的步骤就说明这一点。

1. 创建一个新的项目文件或在现有的项目文件中添加新的表单。在表单的顶端添加一个多行编辑框并在此编辑框的下面添加三个按钮。

2. 设定第一个按钮的标题为Cut，设定第二个按钮的标题为Copy，设定第三个按钮的标题为Paste。

3. 把下面的说明添加到此表单的general区域。

```
Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Long) As LongConst WM _CUT = &H300
Const WM _COPY = &H301
Const WM _PASTE = &H302
```

4. 双击按钮Cut，把下面的代码添加到表单的方法Command1 _ Click中。

```
Private Sub Command1 _Click ( )
    x = SendMessage (Text1.hwnd, WM _CUT, 0, 0)
End Sub
```

5. 双击按钮Copy，把下面的代码添加到表单的方法Command2 _ Click中。

```
Private Sub Command2 _Click ( )
    x = SendMessage (Text1.hwnd, WM _COPY, 0, 0)
End Sub
```

6. 双击按钮Paste，把下面的代码添加到表单的方法Command3 _ Click中。

```
Private Sub Command3 _Click ( )
    x = SendMessage (Text1.hwnd, WM _PASTE, 0, 0)
End Sub
```

7. 运行此例子程序，其结果将与本节中利用Visual C++编制的例子相同。

11.5 编写动态数据交换客户程序

问题

如果想编写一个用来进行动态数据交换(DDE)的客户程序，最好是具有同 Windows 的 Program Manager 或其他动态数据交换的系统服务器进行数据交换的能力。

那么，如何利用 Windows API 函数来创建这样的 DDE 客户程序呢?

方法

DDE 是应用程序之间进行通信的一种最好方法，这种方法，既没有象利用 OLE 方法那样，需要很多额外开销，同时又比利用剪贴板所提供的功能高级，总之，在实现程序间通信这一问题上，它是功能性与复杂性上的一个较好折衷。

本节中，将讨论如何利用 DDE 库函数的基本功能(实际上是 DDEML 库)来进行与 Program Manager 的通信。在本节的例子中，将说明如何把 Program Manager 做为一个服务器连接起来，如何列出“开始”菜单的“程序”部分中所显示的组项，以及执行完这些工作后，如何取消同服务器的连接等。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1105.MAK，便会看到一个新的对话框窗口，如图 11-5 所示。点击按钮 Connect，状态指示器就会改变，表明服务器已被连接，接着，点击按钮 Show Groups，此时就会弹出一个消息框并显示出安装在程序管理器中的当前组项，如图 11-6 所示，最后，点击按钮 Disconnect，此时对话框中显示的状态指示器表明：例子程序已经脱离了同服务器的连接。

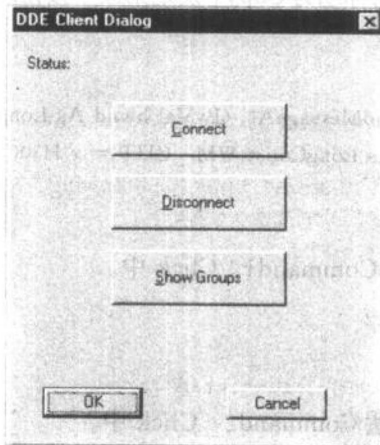


图 11-5　DDE 对话框

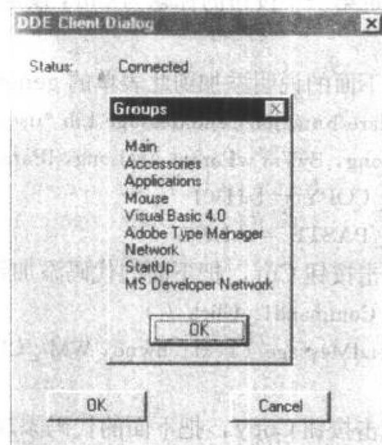


图 11-6　在 DDE 对话框中显示程序开始菜单中所有组项的对话框

为了在应用程序中实现上述功能，按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH1105.MAK。进入资源编辑器并从菜单列表中选择主菜单，利用标题 Dialog 添加新的主菜单，在主菜单 Dialog 上，添加标题为 Dde Dialog 的菜单项，并把此下拉菜单项的标识符设定为 ID_DDE_DLG。

2. 在资源列表中，通过选择 Dialog 资源创建新的对话框模板，然后点击按钮 New，把按钮 OK 与 Cancel 移到对话框的底部，在对话框的顶端，添加一个静态文本框，紧挨着此静态文本框的右边，再添加一个空的静态文本框，在文本框的下面添加三个按钮。

3. 设定第一个静态文本框的标题为 Status，标识符为 IDC_STATIC；设定第二个静态文本框的标题为空，标识符为 IDC_STATUS。

4. 设定第一个按钮的标题为 Connect，设定第二个按钮的标题为 Disconnect，设定第三个按钮的标题为 Show Groups。

5. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，并把此类命名为 CDDeDialog。

6. 从 ClassWizard 的对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 添加新函数 OnConnect。把下面的代码添加到 CDDeDialog 的方法 OnConnect 中。

```
void CDDeDialog::OnConnect ()
{
    FARPROC callBack = MakeProcInstance ( (FARPROC) DdeCallBack, AfxGetInstanceHandle () );
    switch (DdeInitialize (&InstId, (PFNCALLBACK) callBack, APPCMD_CLIENTONLY, 0)) {
      case DMLERR_NO_ERROR :
          // Create string handles for topic and service
      Service = DdeCreateStringHandle (InstId, "PROGMAN", CP_WINANSI);
      Topic = DdeCreateStringHandle (InstId, "PROGMAN", CP_WINANSI);
      if (Service && Topic) {
      HConv = ::DdeConnect (InstId, Service, Topic, 0);
      switch (DdeGetLastError (InstId)) {
      case DMLERR_NO_ERROR :
      GetDlgItem (IDC_STATUS) ->SetWindowText ("Connected");
      return;
    default :
      MessageBox ( "Unable to connect to ProgMan", "Error", MB_OK );
      return;
      }
    } else {
    MessageBox ( "Unable to create string handles!", "Error", MB_OK );
    return;
    }
default :
      MessageBox ( "Error initializing DDE system!", "Error", MB_OK );
      return;
    }
}
```

7. 从 ClassWizard 的对象列表中选择对象 IDC_BUTTON2，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 添加新函数 OnDisconnect。把下面的代码添加到 CDDeD-

ialog 的方法 OnDisconnect 中。

```
void CDDeDialog:: OnDisconnect ()
{
    if (HConv)
       DdeDisconnect (HConv);
    GetDlgItem (IDC_STATUS) ->SetWindowText ("Disconnected");
    if (InstId) {
       DdeFreeStringHandle (InstId, Service);
       DdeFreeStringHandle (InstId, Topic);
       DdeUninitialize (InstId);
    }
}
```

8. 从 ClassWizard 的对象列表中选择对象 IDC_BUTTON3,从消息列表中选择消息 BN_CLICKED,点击按钮 Add Function 添加新函数 OnShowGroups,把下面的代码添加到 CDDeDialog 的方法 OnShowGroups 中。

```
void CDDeDialog:: OnShowGroups ()
{
    DWORD len;
    char commands [1024];
    unsigned char * TempText;
    sprintf (commands, "Groups");
    if (TempStringHandle)
        DdeFreeStringHandle (InstId, TempStringHandle);
    TempStringHandle = DdeCreateStringHandle (InstId, commands, CP_WINANSI);
    hProgData = :: DdeClientTransaction (NULL, 0, HConv, TempStringHandle,
                            CF_TEXT, XTYP_REQUEST, 5000, NULL);
    len = :: DdeGetData (hProgData, NULL, 0, 0);
    TempText = new unsigned char [len];
    len = :: DdeGetData (hProgData, TempText, len, 0);
    MessageBox ( (const char * ) TempText, "Groups", MB_OK );
}
```

9. 把下面的代码添加到类 CDDeDialog 的构造函数的上面。

```
static CDDeDialog * theDialog = NULL;
HDDEDATA FAR PASCAL DdeCallBack (WORD type, WORD, HCONV, HSZ,
                            HSZ, HDDEDATA, DWORD, DWORD)
{
    switch (type) {
       case XTYP_DISCONNECT:
         if ( theDialog )
            theDialog->GetDlgItem (IDC_STATUS) ->SetWindowText ("Disconnected");
       break;
    case XTYP_ERROR:
```

```
    MessageBox (NULL, "A critical DDE error has occured.", "Error",
            MB_ICONINFORMATION);
    }
    return 0;
}
```

10. 把下面的代码添加到类 CDDeDialog 的构造函数中，注意这里只添加标记为黑体的行。

```
CDDeDialog::CDDeDialog (CWnd * pParent /* =NULL */)
: CDialog (CDDeDialog::IDD, pParent)
{
    theDialog = this;
    InstId = 0;
    //{{AFX_DATA_INIT (CDDeDialog)
    //NOTE: the ClassWizard will add member initialization here
    //}} AFX_DATA_INT
}
```

11. 把下面几行添加到类 CDDeDialog 的头文件 DDEDIALO.H 中，另外，还要包括标准的 include 文件 DDEML.H。

```
private:
    DWORD     InstId;
    HCONV     Hconv;
    HSZ       Service;
    HSZ       Topic;
    HSZ       TempStringHandle;
    HDDEDATA hProgData;
```

12. 在 ClassWizard 中，从下拉列表中选择对象 CCh1105App。从对象列表中选择对象 ID_DDE_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnCDDeDialog。把下面的代码添加到 CCh1105App 的方法 OnCDDeDialog 中。

```
void CCh1105App::OnCDDeDialog ()
{
    CDDeDialog dlg;
    dlg.DoModal ();
}
```

13. 最后，把下面一行添加到的源文件 CH1105.CPP 顶端的 include 文件列中。

```
#include "ddedialo.h"
```

14. 把 Windows 库 DDEML.LIB 添加到项目文件中，编译与运行此例子程序。

用法

当用户从菜单中选择 Dde Dialog 后，类 CDDeDialog 就被创建，此时类 CDDeDialog 的构造函数被调用，构造函数把用于 DDE 会话的实例句柄初始化为 0。必须记住，当调用函数 DdeInitialize 时，如果此实例句柄不为 0，则 DDE 系统将会崩溃。

当用户点击按钮 Connect 后，下面的事件将会发生。

首先，通过调用 API 的函数 MakeProcInstance，对话框类创建一个回调函数处理程序。接着，API 函数 DdeInitialize 被调用，它用来初始化 DDE 系统。如果没有错误，就会创建两个字符串句柄，字符串句柄就是在应用程序之间来回传递的全局内存块。由于应用程序中所利用的内存都是局部于此应用程序的，所以只有利用全局内存块才能被多个应用程序所共享。

当以上工作成功地完成之后，DDE 系统就被连接到做为服务器的 Program Manager。当用户点击按钮 Show Groups 后，系统便利用从调用函数 DdeInitialize 得来的句柄向服务器发送命令“Groups”，这是通过向服务器发送 XTYP_REQUEST 命令来完成的，此命令表明客户程序需要返回一些数据，这些数据是通过全局句柄来返回的，在后面，将利用全局句柄来返回这些数据。为了获取所要的数据，需调用函数 DdeGetData 两次。第一次调用此函数，设置 NULL 指针做为参数用来返回数据所需空间的大小，根据返回的大小，为数据分配一个存储块。接着，利用此存储块作为指针值再一次调用函数 DdeGetData，则所要的数据就被存放在此存储块中返回，继而这些数据便在例子程序中显示出来。

最后，按钮 Disconnect 用来取消对话框与做为服务器的 Program Manager 的连接，并释放前面分配字符串句柄时所占用的空间。

注释

动态数据交换是两个程序间进行通信的常用方法，这种方法在很大程度上比使用 OLE 的方法易于理解与调试，而与剪贴板相比，它能提供更强的功能。

11.6 编写动态数据交换服务程序

问题

如果想要创建一个 DDE 服务程序，在应用程序中可利用此服务程序来服务于其他应用程序的数据请求。那么，如何编写一个能在不同应用程序中都易于使用的服务程序呢？

方法

编写 DDE 服务程序是一件技巧性很强而又十分复杂的事情，在这里，我们不只是简单地编写一个服务程序，而是要创建一个在应用程序中可利用的通用构架。下面，将一步步地讨论在服务程序中使自己的信息生效的过程。

步骤

为了创建一个 DDE 服务程序，首先在例子程序中创建文件 SERVER.CPP。

1. 把下面的代码输入到文件 SERVER.CPP 中。

```
#include <stdafx.h>
#define _NODEFINEHSZ
#include "ddeml.h"
const TOPICS = 2;
const ITEMS = 2;
class CDdeServer {
private:
    FARPROC   lpfnDdeCallBack;          // Callback function
    HSZ       ghszDDEMLDemo;            // Service name
    HSZ       ghszTopics [TOPICS];      // topics the server supports
    HSZ       ghszItems [ITEMS];        // items the server supports
```

```
    DWORD       idInst;                     // instance identifier
    BOOL        bAdviseLoopActive;          // Flag for advise loop
public:
    CDdeServer (void);
    BOOL Initialize (void);
    int Uninitialize (void);
    void SetAdvise (BOOL flag) { bAdviseLoopActive = flag; };
    int Advise (int whichTopic, int whichItem);
    DWORD GetDdeInstance (void) { return idInst; };
    HSZ GetTopic (int i) { return ghszTopics [i]; };
    HSZ GetItem (int i) { return ghszItems [i]; };
    HSZ GetTopicHandle (void) { return ghszDDEMLDemo; };
};
static CDdeServer *theServer = NULL;
HDDEDATA EXPENTRY _DdeCallBack (WORD        wType,
                                WORD        wFmt,
                                HCONV       hConv,
                                HSZ         hsz1,
                                HSZ         hsz2,
                                HDDEDATA    hDDEData,
                                DWORD       dwData1,
                                DWORD       dwData2)
{
    int i, j;
    switch (wType)
    {
    case XTYP_ADVSTART:
            // Tell object to set advise flag to on
            theServer->SetAdvise (TRUE);
            return (HDDEDATA) TRUE;
      case XTYP_ADVREQ:
            // Return a global handle to your information here
            return NULL;
      case XTYP_ADVSTOP:
            theServer->SetAdvise (FALSE);
            return (HDDEDATA) NULL;
      case XTYP_CONNECT:
            // The server gets an XTYP_CONNECT when the client requests a
            // conversation (via DdeConnect) . The server should return TRUE
            // if it supports the topic, and FALSE otherwise.
            for (i = 0; i < TOPICS; i++)
            {
              if (! DdeCmpStringHandles (hsz1, theServer->GetTopic (i) ))
```

```
            return (HDDEDATA) TRUE;
        }
        return FALSE; // None found to match
case XTYP_WILDCONNECT:
// Return a pair of topic/items for each supported.
HSZPAIR ahszp [ (TOPICS + 1)]; // for the HSZPÁIRS that we need to
                                               // return to the DDEML.
if ( (hsz2 ! = theServer->GetTopicHandle ()) &&.(hsz2 ! = NULL))
{
   // we only support the DDEMLDemo service
   return (HDDEDATA) NULL;
}
// scan the topic table and create hsz pairs
j = 0;
for (i = 0; i < TOPICS; i++)
{
   if (! DdeCmpStringHandles (hsz1, theServer->GetTopic (i) ))
   {
      ahszp [j] .hszSvc = theServer->GetTopicHandle ();
ahszp [j] .hszTopic = theServer->GetTopic (i);
j++;
}
}
// The array of hszpairs should end with NULL string handles
ahszp [j] .hszSvc = ahszp [j] .hszTopic = NULL;
j++;
// send it back
return (DdeCreateDataHandle (theServer->GetDdeInstance (),
                             (LPBYTE) &ahszp [0],
                             sizeof (HSZPAIR) * j,
                             0L,
                             0,
                             wFmt,
                             0));
case XTYP_REQUEST:
      // The server gets an XTYP_REQUEST when the client requests
      // information via DdeClientTransaction (...XTYP_REQUEST) .
      // This transaction must be handled differently for each
      // topic and format.
      // Is this a request on the System topic?
      if (! DdeCmpStringHandles (hsz1, theServer->GetTopic (0)))
:
   return (HDDEDATA) NULL; // no match
```

```
}
// Is this a CF_TEXT-format request on the item?
if ( (wFmt == CF_TEXT) && (! DdeCmpStringHandles (hsz2,
                                theServer->GetTopic (1))))
{
   HDDEDATA hReturn;
int nValue = 256; // Value to return
// Create data handle for the number the user selected to
// pass on to the client.
hReturn = DdeCreateDataHandle (theServer->GetDdeInstance (),
                              // instance identifier
          (LPBYTE) &nValue, // source buffer
          sizeof (int),      // length of object
          0,                 // offset into source buffer
                             theServer->GetItem (1),
          wFmt,              // clipboard data format
          0);                // creation flags
     if (! hReturn)
     {
   WORD wError;
   char szBuffer [128];
   wError = DdeGetLastError (theServer->GetDdeInstance ());
   wsprintf (szBuffer, "DdeCreateDataHandle failed. Error #%#0x", wError);
   MessageBox (NULL, szBuffer, "Error", MB_OK);
   return (HDDEDATA) NULL;
   }
// send handle to the requested data back to client
return hReturn;
}
// Is this a CF_BITMAP-format request on the GENERAL topic?
if ( (wFmt == CF_BITMAP) && (! DdeCmpStringHandles (hsz2,
                                theServer->GetItem (1))) )
{
   return (HDDEDATA) NULL;
}
else
{
   // Request on unsupported topic and/or format.
   return (HDDEDATA) NULL;
}
return (HDDEDATA) NULL;
case XTYP_CONNECT_CONFIRM:
   return (HDDEDATA) NULL;
```

```
    case XTYP_POKE:
      return (HDDEDATA) DDE_FNOTPROCESSED;
    case XTYP_REGISTER:
      return (HDDEDATA) NULL;
    case XTYP_UNREGISTER:
      return (HDDEDATA) NULL;
    case XTYP_DISCONNECT:
      return (HDDEDATA) NULL;
    case XTYP_EXECUTE:
      return DDE_FNOTPROCESSED;
    default:
      return (HDDEDATA) NULL;
    }
}
CDdeServer:: CDdeServer (void)
{
    idInst = 0;
    ghszDDEMLDemo = NULL;
    lpfnDdeCallBack = NULL;
    bAdviseLoopActive = FALSE;
    theServer = this;
}
BOOL CDdeServer:: Initialize ()
{
    lpfnDdeCallBack = MakeProcInstance ( (FARPROC) _DdeCallBack,
                                         AfxGetInstanceHandle ());
    // Initialize the system
    if (DdeInitialize (&idInst, (PFNCALLBACK) lpfnDdeCallBack,
                                 APPCLASS_STANDARD, 0L))
      return FALSE;
    // Create string handle for the Service name
    ghszDDEMLDemo = DdeCreateStringHandle (idInst, "Demo Service",
                                           CP_WINANSI);
    if (! ghszDDEMLDemo)
      return FALSE;
    // Create string handles to the Topic names
    ghszTopics [0] = DdeCreateStringHandle (idInst, "Sample", CP_WINANSI);
    ghszTopics [1] = DdeCreateStringHandle (idInst, SZDDESYS_TOPIC,
                                           CP_WINANSI);
    // Create string handles for the Item names
    ghszItems [0] = DdeCreateStringHandle (idInst, "Item1", CP_WINANSI);
    ghszItems [1] = DdeCreateStringHandle (idInst, "Item2", CP_WINANSI);
    // Register the service
```

```
    if (! DdeNameService (idInst, ghszDDEMLDemo, NULL, DNS_REGISTER))
       return FALSE;
    return TRUE;
}
int CDdeServer:: Uninitialize ()
{
    // Unregister the service
    DdeNameService (idInst, ghszDDEMLDemo, NULL, DNS_UNREGISTER);
    // terminate all conversations
    DdeUninitialize (idInst);
    return 0;
}
int CDdeServer:: Advise (int whichTopic, int whichItem)
{
    if (bAdviseLoopActive) {
    // send XTYP_ADVREQ to own callback
    if (! DdePostAdvise (idInst, ghszTopics [whichTopic], ghszItems [whichItem]))
       return FALSE;
       return TRUE;
    }
    return FALSE;
}
```

2. 要改变的第一个选项是主题的数目和名字。在类的初始化函数中，可利用数组 ghszTopics 来指定主题，数组 ghszTopics 存放服务程序的主题名，这些项被存放在数组 ghszItem 中，通过增加与修改这些数组，就能改变服务程序的主题与项。

3. 服务程序的当前名字为 Demo Service，此名字可以通过在函数 Initialize 中修改 DdeCreateStringHandle 的调用来加以改变，这正是首次调用函数 DdeCreateStringHandle 的目的。

4. 上面所述的正是在 Initialize 与 UnInitialize 中所必须定制的内容，在回调函数中，可以修改为下面的内容。对于条件 XTYP_ADVREQ，在假定通知请求设置为 TRUE 并且数据已修改时向调用函数返回数据，利用这个条件把具有新信息的全局句柄返回给调用程序．此全局句柄通过利用下面的代码得到。

```
hData = DdeCreateDataHandle (idInst,
        (LPBYTE) &data,              //The data to pass
        size of (data),              //The size of the data to pass
        0,                           //The offset from the beginning
                                     //of the data
        ghszItems [whichData],       //The item to use
        wFmt,                        // Format of the data on the
                                     //clipboard (CF_TEXT or CF_BITMAP) .
        HDATA_APPOWNED);             //Indicates that the data is
                                     //owned by the server
```

5. 条件 XTYP _ REQUEST 显示了一个如何把数字传递回调用它的客户程序的简单实例。可以利用条件 XTYP _ REQUEST 来扩充返回的数据，只要如同上面例子那样创建一个需要发送与传递的数据实例即可。

用法

解释 DDE 的具体工作过程已超出本书的范围。在这里，我们只针对此实例与工作的每一步进行分析。首先，初始化 DDE 系统，这是任何 DDE 系统所必要的步骤，接着，服务程序被注册到 DDE 系统，这也是任何 DDE 服务程序所必须的。

初始化和注册后，开始系统的可选部分。通常，系统为其所支持的所有主题与项分配字符串句柄，主题类似于“股票价格”或“PLU 值”，项类似于特定的服务，诸如“价格”或“名称”。例如，如果在销售细目系统中利用 DDE，则主题或许是 PLU 值，而项可能是“价格”、“现有数量”以及“应征税的标志”，这些会从服务器送到客户。

服务程序利用回调函数接收所有新的 DDE 命令，必须处理的命令有 XTYP _ CONNECT、XTYP _ WILDCONNECT 与 XTYP _ REQUEST，这些是实现服务程序所必须的基本功能。

通知语句（XTYP _ ADVSTART、XTYP _ ADVREQ、XTYP _ ADVSTOP）被用于在客户与服务器之间创建一个“热”链接，当客户投寄一个“通知开始”的消息后，服务程序将把数据上的任何变化“通知”客户。

关于 DDE 命令的更多信息，请参阅 DDE 参考手册。

注释

编写 DDE 服务程序是一件不应该轻视的困难任务。服务程序中需要的许多函数已经在 Windows API 函数或其他服务应用程序中存在。在考虑写一个成熟的服务程序之前，应该认真考虑在应用程序中使用 OLE，并使应用程序成为 OLE 服务程序。

如果的确想创建一个 DDE 服务程序，建议使用 Microsoft 无偿提供的“ez—server”代码，它包含在下一节中。该代码实现了一个通用的 DDE 服务程序，此服务程序能够被嵌入到用户的应用程序，它包含了实现系统主题、反馈以及 DDE 服务程序的所有其他特征的完整代码。

11.7 在动态数据交换中支持系统主题

问题

如果希望编写一个支持系统主题的完整的 DDE 服务程序，就像 Microsoft Word 与 Excel 中所创建的服务程序那样，系统主题包括所有需要的信息以便开发者查找 DDE 服务程序所提供的服务，并给予自己应用程序的用户以同样的功能。那么，如何利用 DDE API 函数的强大功能来实现这样一个服务程序呢？

方法

在编写 DDE 服务程序时，各种细节性的问题层出不穷，这些问题的频繁出现使得 Microsoft 决定直接来帮助开发者克服在编写 DDE 服务程序时遇到的内在问题。

Microsoft 免费提供了一个支持 DDE 中所有基本系统函数的一般 DDE 服务程序文件，此文件可以被包含在开发者的应用程序中，并可执行有关实现一个支持系统主题的 DDE 服务程序的所有基本工作。

本节中，将考察此 DDE 服务程序，并讨论它如何利用 Windows API 函数来实现这一支

持系统主题的服务程序。这里，还将说明开发人员如何以少量的精力或毫不费力地在应用程序中利用这些 API 来增加功能。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1107.MAK。选择主菜单 DDE，并从此菜单中选择选项 DDE Initialize，此时便会看到一个消息框，如图 11-7 所示，此消息框用来表明支持主题的 DDE 系统已经被建立。

按照如下的步骤，就能实现上述功能。

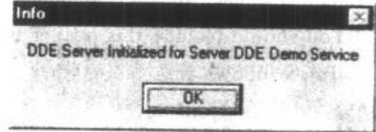


图 11-7 初始化 DDE 的消息框

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH1107.MAK。

2. 进入资源编辑器，在项目文件的主菜单中添加新的主菜单 DDE。在菜单 DDE 中，添加标识符为 ID_DDE_INIT 的菜单项，并把此下拉菜单项的标题设定为 DDE Initialize。

3. 关闭资源编辑器，保存资源文件。进入 ClassWizard，从下拉列表中选择对象 CCh1107App，从对象列表中选择对象 ID_DDE_INIT，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnDdeInit。把下面的代码添加到 CCh1107App 的 OnDdeInit 方法中。

```
void CCh1107App:: OnDdeInit ()
{
    if ( InitializeDDE (AfxGetInstanceHandle (),
                "DDE Demo Service",
                NULL,
                NULL,
                APPCLASS_STANDARD) == TRUE ) {
      MessageBox ( NULL, "DDE Server Initialized for Server DDE Demo Service","Info", MB_OK );
    }
    else
      MessageBox ( NULL, "Unable to initialize server!", "Error", MB_OK );
}
```

4. 接下来，重新进入资源编辑器。在主菜单 DDE 中添加第二个菜单项。把此菜单项的标识符设定为 ID_DDE_UNINT、标题设定为 DDE Uninitialize。

5. 进入 ClassWizard，从下拉列表中选择对象 CCh1107App。从对象列表中选择对象 ID_DDE_INIT，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnDdeInit。把下面的代码添加到 CCh1107App 的方法 OnDdeInit 中。

```
void CCh1107App:: OnDdeUninit ( )
{
UninitializeDDE ( );
}
```

6. 把下面几行代码添加到源文件 CH1107.CPP 的顶端。

```
extern "C" {
```

```
#define _NODEFINEHSZ
#include "stddde.h"
};
```

7. 接下来，把下面两个文件存放到此例子程序的子目录中。

```
/*
    stddde.h
    This header file defines all the constants, structures
    and APIs used to interface with stddde.c.
    You should include this header file in your own application
    after windows.h
*/
#ifndef _STDDDE_
#define _STDDDE_
#include <ddeml.h>
//
// String names for standard Windows Clipboard formats
//
#define SZCF_TEXT               "TEXT"
#define SZCF_BITMAP             "BITMAP"
#define SZCF_METAFILEPICT       "METAFILEPICT"
#define SZCF_SYLK               "SYLK"
#define SZCF_DIF                "DIF"
#define SZCF_TIFF               "TIFF"
#define SZCF_OEMTEXT            "OEMTEXT"
#define SZCF_DIB                "DIB"
#define SZCF_PALETTE            "PALETTE"
#define SZCF_PENDATA            "PENDATA"
#define SZCF_RIFF               "RIFF"
#define SZCF_WAVE               "WAVE"
#define SZCF_UNICODETEXT        "UNICODETEXT"
#define SZCF_ENHMETAFILE        "ENHMETAFILE"
//
// String names for some standard DDE strings not
// defined in DDEML.H
//
#define SZ_READY                "Ready"
#define SZ_BUSY                 "Busy"
#define SZ_TAB                  "\t"
#define SZ_RESULT               "Result"
#define SZ_PROTOCOLS            "Protocols"
#define SZ_EXECUTECONTROL1      "Execute Control 1
//
// Definition for a DDE Request processing function
```

```
//
typedef HDDEDATA (DDEREQUESTFN) (UINT wFmt, HSZ hszTopic, HSZ hszItem);
typedef DDEREQUESTFN * PDDEREQUESTFN;
//
// Definition for a DDE Poke processing function
//
typedef BOOL (DDEPOKEFN) (UINT wFmt, HSZ hszTopic, HSZ hszItem, HDDEDATA hData);
typedef DDEPOKEFN * PDDEPOKEFN;
//
// Definition for a DDE Execute processing function
//
typedef BOOL (DDEEXECFN) (HSZ hszTopic, HDDEDATA hData);
typedef DDEEXECFN * PDDEEXECFN;
//
// Definition for a DDE execute command procession function
//
typedef BOOL (DDEEXECCMDFN) (struct _DDETOPICINFO FAR * pTopic,
                        LPSTR pszResultString,
                         UINT uiResultSize,
                         UINT uiNargs,
                         LPSTR FAR * ppArgs);
typedef DDEEXECCMDFN FAR * PDDEEXECCMDFN;
//
// Structure used to hold a clipboard id and its text name
//
typedef struct _CFTAGNAME {
    WORD wFmt;
    LPSTR pszName;
} CFTAGNAME, FAR * PCFTAGNAME;
//
// Structure used to store information on a DDE execute
// command processor function
//
typedef struct _DDEEXECCMDFNINFO {
    struct _DDEEXECCMDFNINFO FAR * pNext; // pointer to the next item
    struct _DDETOPICINFO FAR * pTopic;    // pointer to the topic it belongs to
    LPSTR pszCmdName;                     // The name of the command
    PDDEEXECCMDFN pFn;                    // A pointer to the function
    UINT uiMinArgs;                       // min number of args
    UINT uiMaxArgs;                       // max number of args
} DDEEXECCMDFNINFO, FAR * PDDEEXECCMDFNINFO;
//
// Structure used to store information on a DDE item
```

```
//
typedef struct _DDEITEMINFO {
    struct _DDEITEMINFO FAR *pNext; // pointer to the next item
    LPSTR pszItemName;        // pointer to its string name
    HSZ hszItemName;        // DDE string handle for the name
    struct _DDETOPICINFO FAR *pTopic; // pointer to the topic it belongs to
    LPWORD pFormatList;        // ptr to null term list of CF format words.
    PDDEREQUESTFN pfnRequest;        // pointer to the item specific request processor
    PDDEPOKEFN pfnPoke;        // pointer to the item specific poke processor
    HDDEDATA hData;        // data for this item (not used by stddde.c)
} DDEITEMINFO, FAR *PDDEITEMINFO;
//
// Structure used to store information on a DDE topic
//
typedef struct _DDETOPICINFO {
    struct _DDETOPICINFO FAR *pNext; // pointer to the next topic
    LPSTR pszTopicName;        // pointer to its string name
    HSZ hszTopicName;        // DDE string handle for the name
    PDDEITEMINFO pItemList;        // pointer to its item list
    PDDEEXECFN pfnExec;        // pointer to the generic execute processor
    PDDEREQUESTFN pfnRequest;        // pointer to the generic request processor
    PDDEPOKEFN pfnPoke;        // pointer to the generic poke processor
    PDDEEXECCMDFNINFO pCmdList; // pointer to the execute command list
} DDETOPICINFO, FAR *PDDETOPICINFO;
//
// Structure used to store information about a DDE conversation
//
typedef struct _DDECONVINFO {
    struct _DDECONVINFO FAR *pNext; // pointer to the next one
    HCONV hConv;        // handle to the conversation
    HSZ hszTopicName;        // HSZ for the topic of the conversation
    PDDEITEMINFO pResultItem;        // pointer to a temp result item
} DDECONVINFO, FAR *PDDECONVINFO;
//
// Structure used to store information on a DDE server
// which has only one service
//
typedef struct _DDESERVERINFO {
    LPSTR lpszServiceName;        // pointer to the service string name
HSZ hszServiceName;        // DDE string handle for the name
PDDETOPICINFO pTopicList;        // pointer to the topic list
DWORD dwDDEInstance;        // DDE Instance value
PFNCALLBACK pfnStdCallback;        // pointer to standard DDE callback fn
```

```
PFNCALLBACK pfnCustomCallback;// pointer to custom DDE callback fn
PDDECONVINFO pConvList;          // pointer to the active conversation list
} DDESERVERINFO, FAR * PDDESERVERINFO;
//
// Functions provided by stddde.c
//
extern BOOL InitializeDDE (HANDLE hInstance,
                    LPSTR lpszServiceName,
                    LPDWORD lpdwDDEInst,
                    PFNCALLBACK lpfnCustomCallback,
                    DWORD dwFilterFlags);
extern void UninitializeDDE (void);
extern PDDETOPICINFO AddDDETopic (LPSTR lpszTopic,
                PDDEEXECFN pfnExec,
                PDDEREQUESTFN pfnRequest,
                PDDEPOKEFN pfnPoke);
extern BOOL RemoveDDETopic (LPSTR lpszTopic);
extern PDDEITEMINFO AddDDEItem (LPSTR lpszTopic,
                LPSTR lpszItem,
                LPWORD pFormatList,
                PDDEREQUESTFN lpReqFn,
                PDDEPOKEFN lpPokeFn);
extern BOOL RemoveDDEItem (LPSTR lpszTopic, LPSTR lpszItem);
extern PDDEEXECCMDFNINFO AddDDEExecCmd (LPSTR pszTopic,
                LPSTR pszCmdName,
                PDDEEXECCMDFN pExecCmdFn,
                UINT uiMinArgs,
                UINT uiMaxArgs);
extern BOOL RemoveDDEExecCmd (LPSTR pszTopic, LPSTR pszCmdName);
extern void PostDDEAdvise (PDDEITEMINFO pItemInfo);
extern PDDETOPICINFO FindTopicFromName (LPSTR lpszName);
extern PDDEITEMINFO FindItemFromName (PDDETOPICINFO pTopic, LPSTR lpszItem);
extern PDDETOPICINFO FindTopicFromHsz (HSZ hszName);
extern PDDEITEMINFO FindItemFromHsz (PDDETOPICINFO pTopic, HSZ hszItem);
extern LPSTR GetCFNameFromId (WORD wFmt, LPSTR lpBuf, int iSize);
extern WORD GetCFIdFromName (LPSTR pszName);
#endif // _STDDDE_
// COPYRIGHT:
//
// (C) Copyright Microsoft Corp. 1993. All rights reserved.
//
// You have a royalty-free right to use, modify, reproduce and
// distribute the Sample Files (and/or any modified version) in
```

```
// any way you find useful, provided that you agree that
// Microsoft has no warranty obligations or liability for any
// Sample Application Files which are modified.
//
/*
    stddde.c
    This module implements basic DDE support for an application.
    It supports the system topic and allows topics and items to
    be dynamically added and removed while the application is running.
    It only supports CF_TEXT for system items.
    It assumes that there is only one service provided and that
    the server is never too busy to process a command.
    All objects created by this module are created using _calloc
    which allows efficient use of global memory and zero initializes
    each one.
*/
#include <windows.h>
#include "stddde.h"
#include <malloc.h>
#include <string.h>
//
// Local constants
//
#define MAXFORMATS 128 // max no of CF formats we will list
//
// Local data
//
static DDESERVERINFO ServerInfo;
//
// Format lists
//
static WORD SysFormatList [] = {
    CF_TEXT,
    NULL};
//
// Standard format name lookup table
//
CFTAGNAME CFNames [] = {
    CF_TEXT,            SZCF_TEXT,
    CF_BITMAP,          SZCF_BITMAP,
    CF_METAFILEPICT,    SZCF_METAFILEPICT,
    CF_SYLK,            SZCF_SYLK,
```

```
    CF_DIF,             SZCF_DIF,
    CF_TIFF,            SZCF_TIFF,
    CF_OEMTEXT,         SZCF_OEMTEXT,
    CF_DIB,             SZCF_DIB,
    CF_PALETTE,         SZCF_PALETTE,
    CF_PENDATA,         SZCF_PENDATA,
    CF_RIFF,            SZCF_RIFF,
    CF_WAVE,            SZCF_WAVE,
    NULL,               NULL
    };
//
// Local functions
//
HDDEDATA SysReqTopics (UINT wFmt, HSZ hszTopic, HSZ hszItem);
HDDEDATA SysReqItems (UINT wFmt, HSZ hszTopic, HSZ hszItem);
HDDEDATA SysReqFormats (UINT wFmt, HSZ hszTopic, HSZ hszItem);
HDDEDATA TopicReqFormats (UINT wFmt, HSZ hszTopic, HSZ hszItem);
HDDEDATA FAR PASCAL _StdCallback (WORD wType,
                WORD wFmt,
                HCONV hConv,
                HSZ hsz1,
                HSZ hsz2,
                HDDEDATA hData,
                DWORD dwData1,
                DWORD dwData2);
HDDEDATA DoWildConnect (HSZ hszTopic);
BOOL DoCallback (WORD wType,
            WORD wFmt,
            HCONV hConv,
            HSZ hsz1,
            HSZ hsz2,
            HDDEDATA hData,
            HDDEDATA *phReturnData);
HDDEDATA MakeDataFromFormatList (LPWORD pFmt, WORD wFmt, HSZ hszItem);
void AddFormatsToList (LPWORD pMain, int iMax, LPWORD pList);
static BOOL AddConversation (HCONV hConv, HSZ hszTopic);
static BOOL RemoveConversation (HCONV hConv, HSZ hszTopic;
static PDDECONVINFO FindConversation (HSZ hszTopic);
//
// Initialize all the DDE lists for this server and
// initialize DDEML.
//
BOOL InitializeDDE (HANDLE hInstance,
```

```
        LPSTR lpszServiceName,
         LPDWORD lpdwDDEInst,
         PFNCALLBACK lpfnCustomCallback,
         DWORD dwFilterFlags)
{
    UINT uiResult;
    //
    // Make a proc instance for the standard callback
    //
    ServerInfo.pfnStdCallback =
        (PFNCALLBACK) MakeProcInstance ( (FARPROC) _StdCallback, hInstance);
    //
    // Make sure the application hasn't requested any filter options
    // which will prevent us from working correctly.
    //
    dwFilterFlags &= ! ( CBF_FAIL_CONNECTIONS
            | CBF_SKIP_CONNECT_CONFIRMS
            | CBF_SKIP_DISCONNECTS);
    //
    // Initialize DDEML. Note that DDEML doesn't make any callbacks
    // during initialization so we don't need to worry about the
    // custom callback yet.
    //
    uiResult = DdeInitialize (&ServerInfo.dwDDEInstance,
             ServerInfo.pfnStdCallback,
             dwFilterFlags,
             0);
    if (uiResult ! = DMLERR_NO_ERROR) return FALSE;
    //
    // make a proc instance for the custom callback if there is one
    //
    if (lpfnCustomCallback) {
       ServerInfo.pfnCustomCallback =
         (PFNCALLBACK) MakeProcInstance ( (FARPROC) lpfnCustomCallback,
                hInstance);
    }
    //
    // Return the DDE instance id if it was requested
    //
    if (lpdwDDEInst) {
        * lpdwDDEInst = ServerInfo.dwDDEInstance;
    }
    //
```

```
    // Copy the service name and create a DDE name handle for it
    //
    ServerInfo.lpszServiceName = lpszServiceName;
    ServerInfo.hszServiceName = DdeCreateStringHandle (ServerInfo.dwDDEInstance,
                                    lpszServiceName,
                                    CP_WINANSI);
    //
    // Add all the system topic items to the service tree
    //
    AddDDEItem (SZDDESYS_TOPIC,
            SZDDESYS_ITEM_TOPICS,
            SysFormatList,
            SysReqTopics,
            NULL);
    AddDDEItem (SZDDESYS_TOPIC,
            SZDDESYS_ITEM_SYSITEMS,
            SysFormatList,
            SysReqItems,
            NULL);
    AddDDEItem (SZDDESYS_TOPIC,
            SZDDE_ITEM_ITEMLIST,
            SysFormatList,
            SysReqItems,
            NULL);
    AddDDEItem (SZDDESYS_TOPIC,
            SZDDESYS_ITEM_FORMATS,
            SysFormatList,
            SysReqFormats,
            NULL);
    //
    // Register the name of our service
    //
    DdeNameService (ServerInfo.dwDDEInstance,
            ServerInfo.hszServiceName,
            NULL,
            DNS_REGISTER);
    return TRUE;
}
//
// Tidy up and close down DDEML
//
void UninitializeDDE (void)
{
```

```
PDDETOPICINFO pTopic;
PDDEITEMINFO pItem;
//
// Unregister the service name
//
DdeNameService (ServerInfo.dwDDEInstance,
          ServerInfo.hszServiceName,
          NULL,
          DNS_UNREGISTER);
//
// free the name handle
//
DdeFreeStringHandle (ServerInfo.dwDDEInstance, ServerInfo.hszServiceName);
//
// Walk the server topic tree, freeing all the other string handles
//
pTopic = ServerInfo.pTopicList;
while (pTopic) {
     DdeFreeStringHandle (ServerInfo.dwDDEInstance, pTopic->hszTopicName);
     pTopic->hszTopicName = NULL;
     //
     // Free any item handles it owns
     //
     pItem = pTopic->pItemList;
     while (pItem) {
     DdeFreeStringHandle (ServerInfo.dwDDEInstance, pItem->hszItemName);
pItem->hszItemName = NULL;
pItem = pItem->pNext;
}
pTopic = pTopic->pNext;
}
//
// Release DDEML
//
DdeUninitialize (ServerInfo.dwDDEInstance);
ServerInfo.dwDDEInstance = NULL;
//
// Free any proc instances we created
//
if (ServerInfo.pfnCustomCallback) {
     FreeProcInstance ( (FARPROC) ServerInfo.pfnCustomCallback);
     ServerInfo.pfnCustomCallback = NULL;
     }
```

```
    FreeProcInstance ( (FARPROC) ServerInfo.pfnStdCallback);
    ServerInfo.pfnStdCallback = NULL;
}
//
// Find a topic by its name
//
PDDETOPICINFO FindTopicFromName (LPSTR lpszName)
{
    PDDETOPICINFO pTopic;
pTopic = ServerInfo.pTopicList;
while (pTopic) {
    if (lstrcmpi (pTopic->pszTopicName, lpszName) == 0) {
        break;
    }
    pTopic = pTopic->pNext;
    }
    return pTopic;
}
//
// Find a topic by its HSZ
//
PDDETOPICINFO FindTopicFromHsz (HSZ hszName)
{
    PDDETOPICINFO pTopic;
    pTopic = ServerInfo.pTopicList;
    while (pTopic) {
    if (DdeCmpStringHandles (pTopic->hszTopicName, hszName) == 0) {
        break;
    }
        pTopic = pTopic->pNext;
    }
    return pTopic;
}
//
// Find an item by its name in a topic
//
PDDEITEMINFO FindItemFromName (PDDETOPICINFO pTopic, LPSTR lpszItem)
{
    PDDEITEMINFO pItem;
    pItem = pTopic->pItemList;
    while (pItem) {
        if (lstrcmpi (pItem->pszItemName, lpszItem) == 0) {
            break;
```

```
        }
    pItem = pItem->pNext;
    }
    return pItem;
}
//
// Find an item by its HSZ in a topic
//
PDDEITEMINFO FindItemFromHsz (PDDETOPICINFO pTopic, HSZ hszItem)
{
    PDDEITEMINFO pItem;
    pItem = pTopic->pItemList;
    while (pItem) {
        if (DdeCmpStringHandles (pItem->hszItemName, hszItem) == 0) {
            break;
        }
        pItem = pItem->pNext;
    }
    return pItem;
}
//
// Add a new topic and default processing for its item list and formats
//
PDDETOPICINFO AddDDETopic (LPSTR lpszTopic,
            PDDEEXECFN pfnExec,
            PDDEREQUESTFN pfnRequest,
            PDDEPOKEFN pfnPoke)
{
    PDDETOPICINFO pTopic;
    //
    // See if we already have this topic
    //
    pTopic = FindTopicFromName (lpszTopic);
    if (pTopic) {
        //
        // We already have this one so just update its info
        //
        pTopic->pfnExec = pfnExec;
        pTopic->pfnRequest = pfnRequest;
        pTopic->pfnPoke = pfnPoke;
    } else {
        //
        // Create a new topic
```

```
        //
        pTopic = calloc (1, sizeof (DDETOPICINFO));
        if (! pTopic) return NULL;
        //
        // Fill out the info
        //
        pTopic->pszTopicName = lpszTopic;
            pTopic->hszTopicName = DdeCreateStringHandle (ServerInfo.dwDDEInstance,
                lpszTopic,
                CP_WINANSI);
        pTopic->pItemList = NULL;
        pTopic->pfnExec = pfnExec;
        pTopic->pfnRequest = pfnRequest;
        pTopic->pfnPoke = pfnPoke;
        //
        // Add it to the list
        //
        pTopic->pNext = ServerInfo.pTopicList;
        ServerInfo.pTopicList = pTopic;
        //
        // Add handlers for its item list and formats.
        //
        AddDDEItem (lpszTopic,
            SZDDE_ITEM_ITEMLIST,
            SysFormatList,
            SysReqItems,
            NULL);
    AddDDEItem (lpszTopic,
              SZDDESYS_ITEM_FORMATS,
              SysFormatList,
              TopicReqFormats,
              NULL);
     }
     return pTopic;
}
//
// Remove a topic and all its items. If there is an active
// conversation on the topic then disconnect it.
//
BOOL RemoveDDETopic (LPSTR lpszTopic)
{
    PDDETOPICINFO pTopic, pPrevTopic;
    PDDECONVINFO pCI;
```

```
//
// See if we have this topic by walking the list
//
pPrevTopic = NULL;
pTopic = ServerInfo.pTopicList;
while (pTopic) {
if (lstrcmpi (pTopic->pszTopicName, lpszTopic) == 0) {
   //
   // Found it. Disconnect any active conversations on this topic
   //
   while (pCI = FindConversation (pTopic->hszTopicName)) {
   //
// Tell DDEML to disconnect it
   //
   DdeDisconnect (pCI->hConv);
   //
   // We don't get notified until later that it's gone
   // so remove it from the list now so we won't keep
   // finding it in this loop.
   // When DDEML sends the disconnect notification later
   // it won't be there to remove but that doesn't matter.
   //
   RemoveConversation (pCI->hConv, pCI->hszTopicName);
}
//
// Free all the items in the topic
//
while (pTopic->pItemList) {
   if (! RemoveDDEItem (lpszTopic,
                pTopic->pItemList->pszItemName)) {
              return FALSE; // some error
          }
        }
//
// Unlink it from the list.
//
if (pPrevTopic) {
   pPrevTopic->pNext = pTopic->pNext;
} else {
ServerInfo.pTopicList = pTopic->pNext;
}
//
// Release its string handle
```

```
        //
        DdeFreeStringHandle (ServerInfo.dwDDEInstance,
                             pTopic->hszTopicName);
        //
        // Free the memory associated with it
        //
        free (pTopic);
        return TRUE;
        }
        pTopic = pTopic->pNext;
        }
        //
        // We don't have this topic
        //
        return FALSE;
}
//
// Add a new item.
//
PDDEITEMINFO AddDDEItem (LPSTR lpszTopic,
                 LPSTR lpszItem,
                 LPWORD lpFormatList,
                 PDDEREQUESTFN lpReqFn,
                 PDDEPOKEFN lpPokeFn)
{
     PDDEITEMINFO pItem = NULL;
     PDDETOPICINFO pTopic;
     //
     // See if we have this topic already
     //
     pTopic = FindTopicFromName (lpszTopic);
     if (! pTopic) {
          //
          // We need to add this as a new topic
          //
          pTopic = AddDDETopic (lpszTopic,
               NULL,
               NULL,
               NULL);
     }
     if (! pTopic) return NULL; // failed
     //
     // See if we already have this item
```

```
    //
    pItem = FindItemFromName (pTopic, lpszItem);
    if (pItem) {
        //
        // Just update the info in it
        //
        pItem->pfnRequest = lpReqFn;
        pItem->pfnPoke = lpPokeFn;
        pItem->pFormatList = lpFormatList;
    } else {
    //
    // Create a new item
    //
    pItem = calloc (1, sizeof (DDEITEMINFO));
    if (! pItem) return NULL;
    //
    // Fill out the info
    //
    pItem->pszItemName = lpszItem;
    pItem->hszItemName = DdeCreateStringHandle (ServerInfo.dwDDEInstance,
                        lpszItem,
                        CP_WINANSI);
        pItem->pTopic = pTopic;
        pItem->pfnRequest = lpReqFn;
        pItem->pfnPoke = lpPokeFn;
        pItem->pFormatList = lpFormatList;
          //
          // Add it to the existing item list for this topic
          //
          pItem->pNext = pTopic->pItemList;
          pTopic->pItemList = pItem;
      }
     return pItem;
}
//
// Remove an item from a topic.
//
BOOL RemoveDDEItem (LPSTR lpszTopic, LPSTR lpszItem)
{
    PDDETOPICINFO pTopic;
    PDDEITEMINFO pItem, pPrevItem;
    //
    // See if we have this topic
```

```
    //
    pTopic = FindTopicFromName (lpszTopic);
    if (! pTopic) {
        return FALSE;
    }
    //
    // Walk the topic item list looking for this item.
    //
    pPrevItem = NULL;
    pItem = pTopic->pItemList;
    while (pItem) {
        if (lstrcmpi (pItem->pszItemName, lpszItem) == 0) {
            //
            // Found it. Unlink it from the list.
            //
            if (pPrevItem) {
                pPrevItem->pNext = pItem->pNext;
            } else {
                pTopic->pItemList = pItem->pNext;
            }
            //
            // Release its string handle
            //
            DdeFreeStringHandle (ServerInfo.dwDDEInstance,
                        pItem->hszItemName);
            //
            // Free the memory associated with it
            //
            free (pItem);
            return TRUE;
        }
        pPrevItem = pItem;
        pItem = pItem->pNext;
        }
    //
    // We don't have that one
    //
    return FALSE;
}
//
// Get the text name of a Clipboard format from its id
//
LPSTR GetCFNameFromId (WORD wFmt, LPSTR lpBuf, int iSize)
```

```
{
    PCFTAGNAME pCTN;
    //
    // Try for a standard one first
    //
    pCTN = CFNames;
    while (pCTN->wFmt) {
        if (pCTN->wFmt == wFmt) {
            strncpy (lpBuf, pCTN->pszName, iSize);
            return lpBuf;
        }
        pCTN++;
    }
    //
    // See if it's a registered one
    //
    if (GetClipboardFormatName (wFmt, lpBuf, iSize) == 0) {
        //      // Nope. It's unknown
        //
        *lpBuf = '\0';
    }
    return lpBuf;
}
//
// Post an advise request about an item
//
void PostDDEAdvise (PDDEITEMINFO pItemInfo)
{
    if (pItemInfo && pItemInfo->pTopic) {
        DdePostAdvise (ServerInfo.dwDDEInstance,
                pItemInfo->pTopic->hszTopicName,
                pItemInfo->hszItemName);
    }
}
//
// DDE callback function called from DDEML
//
HDDEDATA FAR PASCAL _StdCallback (WORD wType,
                        WORD wFmt,
                          HCONV hConv,
                        HSZ hsz1,
                            HSZ hsz2,
                        HDDEDATA hData,
```

```
                    DWORD dwData1,
                    DWORD dwData2)
{
    HDDEDATA hDdeData = NULL;
    switch (wType) {
    case XTYP_CONNECT_CONFIRM:
        //
        // Add a new conversation to the list
        //
        AddConversation (hConv, hsz1);
        break;
    case XTYP_DISCONNECT:
        //
        // Remove a conversation from the list
        //
        RemoveConversation (hConv, hsz1);
        break;
    case XTYP_WILDCONNECT:
        //
        // We only support wild connects to either a NULL service
        // name or to the name of our own service.
        //
        if ( (hsz2 == NULL)
        || ! DdeCmpStringHandles (hsz2, ServerInfo.hszServiceName)) {
            return DoWildConnect (hsz1);
            }
            break;
            //
            // For all other messages we see if we want them here
            // and if not, they get passed on to the user callback
            // if one is defined.
            //
    case XTYP_ADVSTART:
    case XTYP_CONNECT:
    case XTYP_EXECUTE:
    case XTYP_REQUEST:
    case XTYP_ADVREQ:
    case XTYP_ADVDATA:
    case XTYP_POKE:
            //
            // Try and process them here first.
            //
            if (DoCallback (wType,
```

```
                    wFmt,
                    hConv,
                    hsz1,
                    hsz2,
                    hData,
                    &hDdeData)) {
                return hDdeData;
            }
            //
            // Fall Through to allow the custom callback a chance
            //
        default:
            if (ServerInfo.pfnCustomCallback != NULL) {
              return (ServerInfo.pfnCustomCallback (wType,
                                wFmt,
                                hConv,
                                hsz1,
                                hsz2,
                                hData,
                                dwData1,
                                dwData2));
                }
            }
        return (HDDEDATA) NULL;
}
//
// Process a generic callback
//
BOOL DoCallback (WORD wType,
                    WORD wFmt,
                      HCONV hConv,
                      HSZ hszTopic,
                      HSZ hszItem,
                      HDDEDATA hData,
                      HDDEDATA *phReturnData)
{
    PDDETOPICINFO pTopic;
    PDDEITEMINFO pItem;
    LPWORD pFormat;
    PDDEPOKEFN pfnPoke;
    CONVINFO ci;
    PDDEREQUESTFN pfnRequest;
    //
```

```
// See if we know the topic
//
pTopic = FindTopicFromHsz (hszTopic);
if (! pTopic) {
    return FALSE;
}
//
// See if this is an execute request for the topic
//
if (wType == XTYP_EXECUTE) {
    //
    // See if the user supplied a function to handle this
    //
    if (pTopic->pfnExec) {
        //
        // Call the exec function to process it
        //
        if ( (*pTopic->pfnExec) (wFmt, hszTopic, hData)) {
            *phReturnData = (HDDEDATA) DDE_FACK;
        return TRUE;
    }
}
//
// Either no function or it didn't get handled by the function
//
*phReturnData = (HDDEDATA) DDE_FNOTPROCESSED;
return TRUE;
}
//
// See if this is a connect request. Accept it if it is.
//
if (wType == XTYP_CONNECT) {
    *phReturnData = (HDDEDATA) TRUE;
    return TRUE;
}
//
// For any other transaction we need to be sure this is an
// item we support and in some cases, that the format requested
// is supported for that item.
//
pItem = FindItemFromHsz (pTopic, hszItem);
if (! pItem) {
    //
```

```
        // Not an item we support
        //
        return FALSE;
    }
    //
    // See f this is a supported format
    //
    pFormat = pItem->pFormatList;
    while ( * pFormat) {
        if ( * pFormat == wFmt) break;
        pFormat++;
    }
    if (! * pFormat) return FALSE; // not one we support
    //
    // Now just do whatever is required for each specific transaction
    //
    switch (wType) {
    case XTYP_ADVSTART:
        //
        // Start an advise request. Topic/item and format are ok.
        //
        * phReturnData = (HDDEDATA) TRUE;
        break;
    case XTYP_ADVDATA:
        //
        // Some data for us. See if we have a poke function for
        // this item or for this topic in general.
        //
        * phReturnData = (HDDEDATA) DDE_FNOTPROCESSED;
        pfnPoke = pItem->pfnPoke;
        if (! pfnPoke) pfnPoke = pTopic->pfnPoke;
        if (pfnPoke) {
            if ( ( * pfnPoke) (wFmt, hszTopic, hszItem, hData)) {
                //
                // Data at the server has changed. See if we
                // did this ourself (from a poke) or if it's from
                // someone else. If it came from elsewhere then post
                // an advise notice of the change.
                //
                ci.cb = sizeof (CONVINFO);
                if (DdeQueryConvInfo (hConv, (DWORD) QID_SYNC, &ci)) {
                  if (! (ci.wStatus & ST_ISSELF)) {
                    //
```

```
                // It didn't come from us
                //
                DdePostAdvise (ServerInfo.dwDDEInstance,
                               hszTopic,
                               hszItem);
            }
            }
        *phReturnData = (HDDEDATA) DDE_FACK; // say we took it
    }
    }
    break;
case XTYP_POKE:
    //
    // Make sure we're not trying to poke ourself and just
    // post an advise so we'll get it as and ADVDATA transaction
    // a bit later.
    //
    *phReturnData = (HDDEDATA) DDE_FNOTPROCESSED;
    ci.cb = sizeof (CONVINFO);
    if (DdeQueryConvInfo (hConv, (DWORD) QID_SYNC, &ci)) {
      if (! (ci.wStatus & ST_ISSELF)) {
        //
        // Not trying to poke ourself in the foot so
        // send the advise notice
        //
        DdePostAdvise (ServerInfo.dwDDEInstance,
                                    hszTopic,
                                    hszItem);
        *phReturnData = (HDDEDATA) DDE_FACK; // say we took it
      }
    }
    break;
case XTYP_ADVREQ:
case XTYP_REQUEST:
    //
    // Attempt to start an advise or get the data on a topic/item
    // See if we have a request function for this item or
    // a generic one for the topic
    //
    pfnRequest = pItem->pfnRequest;
if (! pfnRequest) pfnRequest = pTopic->pfnRequest;
if (pfnRequest) {
    *phReturnData = (*pfnRequest) (wFmt, hszTopic, hszItem);
```

```
    } else {
        *phReturnData = (HDDEDATA) NULL;
    }
    break;
    default:
        break;
    }
    //
    // Say we processed the transaction in some way
    //
    return TRUE;
}
//
// Return the current topics list
//
HDDEDATA SysReqTopics (UINT wFmt, HSZ hszTopic, HSZ hszItem)
{
    HDDEDATA hData;
    PDDETOPICINFO pTopic;
    int cb, cbOffset;
    //
    // Create an empty data object to fill
    //
    hData = DdeCreateDataHandle (ServerInfo.dwDDEInstance,
                                 NULL,
                                 0,
                                 0,
                                 hszItem,
                                 wFmt,
                                   0);
pTopic = ServerInfo.pTopicList; cbOffset = 0;
while (pTopic) {
    //
    // put in a tab delimiter unless this is the first item
    //
    if (cbOffset ! = 0) {
        DdeAddData (hData, SZ _ TAB, lstrlen (SZ _ TAB), cbOffset);
        cbOffset += lstrlen (SZ _ TAB);
    }
    //
    // Copy the string name of the topic
    //
    cb = lstrlen (pTopic->pszTopicName);
```

```
    DdeAddData (hData, pTopic->pszTopicName, cb, cbOffset);
    cbOffset += cb;
    pTopic = pTopic->pNext;
}
//
// Put a NULL on the end
//
DdeAddData (hData, "", 1, cbOffset);
return hData;
}
//
// Return the system item list
//
HDDEDATA SysReqItems (UINT wFmt, HSZ hszTopic, HSZ hszItem)
{
    HDDEDATA hData;
    PDDETOPICINFO pTopic;
    PDDEITEMINFO pItem;
    int cb, cbOffset;
    //
    // Find the system topic
    //
    pTopic = FindTopicFromHsz (hszTopic);
    if (! pTopic) return NULL;
    //
    // Create an empty data object to fill
    //
    hData = DdeCreateDataHandle (ServerInfo.dwDDEInstance,
                    NULL,
                    0,
                    0,
                    hszItem,
                  wFmt,
                    0);
    //
    // Walk the item list
    //
    cbOffset = 0;
    pItem = pTopic->pItemList;
    while (pItem) {
        //
        // put in a tab delimiter unless this is the first item
        //
```

```
        if (cbOffset ! = 0) {
            DdeAddData (hData, SZ_TAB, lstrlen (SZ_TAB), cbOffset);
            cbOffset += lstrlen (SZ_TAB);
        }
        //
        // Copy the string name of the item
        //
        cb = lstrlen (pItem->pszItemName);
        DdeAddData (hData, pItem->pszItemName, cb, cbOffset);
        cbOffset += cb;
        pItem = pItem->pNext;
    }
    //
    // Put a NULL on the end
    //
    DdeAddData (hData, "", 1, cbOffset);
    return hData;
}
//
// Return the system formats list which is the union of
// all formats supported.
//
HDDEDATA SysReqFormats (UINT wFmt, HSZ hszTopic, HSZ hszItem)
{
    HDDEDATA hData;
    PDDETOPICINFO pTopic;
    PDDEITEMINFO pItem;
    WORD wFormats [MAXFORMATS];
    wFormats [0] = NULL; // start with an empty list
    //
    // Create an empty data object to fill
    //
    hData = DdeCreateDataHandle (ServerInfo.dwDDEInstance,
                    NULL,
                    0,
                    0,
                    hszItem,
                    wFmt,
                    0);
    //
    // Walk the topic list
    //
    pTopic = ServerInfo.pTopicList;
```

```
    while (pTopic) {
        //
        // Walk the item list for this topic
        //
        pItem = pTopic->pItemList;
        while (pItem) {
            //
            // Walk the formats list for this item
            // adding each one to the main list if unique
            //
            AddFormatsToList (wFormats, MAXFORMATS, pItem->pFormatList);
            pItem = pItem->pNext;
        }
        pTopic = pTopic->pNext;
    }
    //
    // Walk the table and build the text form
    //
    return MakeDataFromFormatList (wFormats, wFmt, hszItem);
}
//
// Return a topic formats list. This is the union of all
// the formats supported by this topic.
//
HDDEDATA TopicReqFormats (UINT wFmt, HSZ hszTopic, HSZ hszItem)
{
    HDDEDATA hData;
    PDDETOPICINFO pTopic;
    PDDEITEMINFO pItem;
    WORD wFormats [MAXFORMATS];
    //
    // Find the topic info
    //
    pTopic = FindTopicFromHsz (hszTopic);
    if (! pTopic) return NULL;
    wFormats [0] = NULL; // start with an empty list
    //
    // Create an empty data object to fill
    //
    hData = DdeCreateDataHandle (ServerInfo.dwDDEInstance,
                    NULL,
                    0,
                    0,
```

```
                        hszItem,
                        wFmt,
                        0);
    //
    // Walk the item list for this topic
    //
    pItem = pTopic->pItemList;
    while (pItem) {
        //
        // Walk the formats list for this item
        // adding each one to the main list if unique
        //
        AddFormatsToList (wFormats, MAXFORMATS, pItem->pFormatList);
        pItem = pItem->pNext;
    }
    //
    // Walk the table and build the text form
    //
    return MakeDataFromFormatList (wFormats, wFmt, hszItem);
}
//
// Process a wild connect request.
// Since we only support one service, this is much simpler.
// If hszTopic is NULL we supply a list of all the topics we
// currently support. If it's not NULL, we supply a list
// of topics (zero or one items) which match the requested topic.
// The list is terminated by a NULL entry.
//
HDDEDATA DoWildConnect (HSZ hszTopic)
{
    PDDETOPICINFO pTopic;
    int iTopics = 0;
    HDDEDATA hData;
    PHSZPAIR pHszPair;
    //
    // See how many topics we will be returning
    //
    if (hszTopic == NULL) {
        //
        // Count all the topics we have
        //
        pTopic = ServerInfo.pTopicList;
        while (pTopic) {
```

```
            iTopics++;
            pTopic = pTopic->pNext;
        }
    } else {
        //
        // See if we have this topic in our list
        //
        pTopic = ServerInfo.pTopicList;
        while (pTopic) {
            if (DdeCmpStringHandles (pTopic->hszTopicName, hszTopic) == 0) {
                iTopics++;
                break;
            }
            pTopic = pTopic->pNext;
        }
    }
    //
    // If we have no match or no topics at all, just return
    // NULL now to refuse the connect
    //
    if (! iTopics) return (HDDEDATA) NULL;
    //
    // Allocate a chunk of DDE data big enough for all the HSZPAIRS
    // we'll be sending back plus space for a NULL entry on the end
    //
    hData = DdeCreateDataHandle (ServerInfo.dwDDEInstance,
                    NULL,
                    (iTopics + 1) * sizeof (HSZPAIR),
                    0,
                    NULL,
                    0,
                    0);
    //
    // Check we actually got it.
    //
    if (! hData) return (HDDEDATA) NULL;
    pHszPair = (PHSZPAIR) DdeAccessData (hData, NULL);
    //
    // Copy the topic data
    //
    if (hszTopic == NULL) {
        //
        // Copy all the topics we have (includes the system topic)
```

```
        //
        pTopic = ServerInfo.pTopicList;
        while (pTopic) {
            pHszPair->hszSvc = ServerInfo.hszServiceName;
            pHszPair->hszTopic = pTopic->hszTopicName;
            pHszPair++;
            pTopic = pTopic->pNext;
        }
    } else {
        //
        // Just copy the one topic asked for
        //
        pHszPair->hszSvc = ServerInfo.hszServiceName;
        pHszPair->hszTopic = hszTopic;
        pHszPair++;
    }
    //
    // Put the terminator on the end
    //
    pHszPair->hszSvc = NULL;
    pHszPair->hszTopic = NULL;
    //
    // Finished with the data block
    //
    DdeUnaccessData (hData);
    //
    // Return the block handle
    //
    return hData;
}
HDDEDATA MakeDataFromFormatList (LPWORD pFmt, WORD wFmt, HSZ hszItem)
{
    HDDEDATA hData;
    int cbOffset, cb;
    char buf [256];
    //
    // Create an empty data object to fill
    //
    hData = DdeCreateDataHandle (ServerInfo.dwDDEInstance,
                            NULL,
                            0,
                            0,
                            hszItem,
```

```
                        wFmt,
                        0);
    //
    // Walk the format list
    //
    cbOffset = 0;
    while (*pFmt) {
        //
        // put in a tab delimiter unless this is the first item
        //
        if (cbOffset != 0) {
            DdeAddData (hData, SZ_TAB, lstrlen (SZ_TAB), cbOffset);
            cbOffset += lstrlen (SZ_TAB);
        }
        //
        // Copy the string name of the format
        //
        GetCFNameFromId (*pFmt, buf, sizeof (buf));
        cb = lstrlen (buf);
        DdeAddData (hData, buf, cb, cbOffset);
        cbOffset += cb;
        pFmt++;
    }
    //
    // Put a NULL on the end
    //
    DdeAddData (hData, "", 1, cbOffset);
    return hData;
}
//
// Add a list of formats to main list ensuring that each item
// only exists in the list once.
//
void AddFormatsToList (LPWORD pMain, int iMax, LPWORD pList)
{
    LPWORD pFmt, pLast;
    int iCount;
    if (!pMain || !pList) return;
    //
    // Count what we have to start with
    //
    iCount = 0;
    pLast = pMain;
```

```
    while ( * pLast) {
        pLast++;
        iCount++;
    }
    //
    // Walk the new list ensuring we don't add the item if there
    // isn't room or we have it already.
    //
    while ( (iCount < iMax) && * pList) {
        //
        // See if we have this one
        //
        pFmt = pMain;
        while ( * pFmt) {
            if ( * pFmt == * pList) {
                //
                // Already got this one
                //
                goto next_fmt; // I know. I hate this too.
            }
            pFmt++;
        }
        //
        // Put it on the end of the list
        //
        * pLast++ = * pList;
        iCount++;
next_fmt:
        pList++;
    }
    //
    // Stick a null on the end to terminate the list
    //
    * pLast = NULL;
}
//
// Find the first occurence of a topic conversation in our list
//
static PDDECONVINFO FindConversation (HSZ hszTopic)
{
    PDDECONVINFO pCI;
    //
    // Try to find the info in the list
```

```
    //
    pCI = ServerInfo.pConvList;
    while (pCI) {
        if (DdeCmpStringHandles (pCI->hszTopicName, hszTopic) == 0) {
            return pCI;
        }
        pCI = pCI->pNext;
    }
    return NULL;
}
//
// Add a conversation to our list
//
static BOOL AddConversation (HCONV hConv, HSZ hszTopic)
{
    PDDECONVINFO pCI;
    //
    // Allocate some memory for the info and fill it in
    //
    pCI = calloc (1, sizeof (DDECONVINFO));
    if (! pCI) {
        return FALSE;
    }
    pCI->hConv = hConv;
    pCI->hszTopicName = hszTopic;
    //
    // Add it into the list
    //
    pCI->pNext = ServerInfo.pConvList;
    ServerInfo.pConvList = pCI;
    return TRUE;
}
//
// Remove a conversation from our list
//
static BOOL RemoveConversation (HCONV hConv, HSZ hszTopic)
{
    PDDECONVINFO pCI, pPrevCI;
    //
    // Try to find the info in the list
    //
    pCI = ServerInfo.pConvList;
    pPrevCI = NULL;
```

```
    while (pCI) {
        if ( (pCI->hConv == hConv)
        && (DdeCmpStringHandles (pCI->hszTopicName, hszTopic) == 0)) {
            //
            // Found it. Unlink it from the list
            //
            if (pPrevCI) {
                pPrevCI->pNext = pCI->pNext;
            } else {
                ServerInfo.pConvList = pCI->pNext;
            }
            //
            // Free the memory
            //
            free (pCI);
            return TRUE;
        }
        pPrevCI = pCI;
        pCI = pCI->pNext;
    }
    //
    // Not in the list
    //
    return FALSE;
}
```

8. 把文件 STDDD.CPP 添加到项目文件。

9. 修改来自于 Visual C++的 include 集的文件 DDEML.H。找出此文件中读取 DECLARE_HANDLE32 (HSZ) 的行，把下面一行代码添加到此行的上面。

```
#ifndef _NODEFINEHSZ
```

把下面一行代码添加到该行的下面。

```
#endif
```

10. 把 DDEML Windows 库添加到项目文件的连接选项中，这时就可编译与运行此例子程序了。

用法

关于 DDE 内部工作的详细情况，参看 11.5 与 11.6 节。这里，我们将稍加讨论有关支持 DDE 系统主题的意义以及例子程序如何帮助程序员实现这一目的。

系统主题支持下面供客户浏览的项：Topics（主题）、SysItems（系统项）、Formats（格式）、TopicItemList（主题项表）、Help（帮助）、Status（状态）以及 ReturnMessage（返回消息），如果 DDE 服务程序支持以上所有这些项，就认为与系统主题的要求一致。

“Topics 项”向客户返回一个由制表符所分隔开的、DDE 服务程序所支持的所有主题的列表，此列表根据服务程序与应用程序的状态可随时被修改。

“SysItems”返回由此列表所支持的一系列系统主题项，一个真正的服务程序将支持所有列出的主题，文件 STDDDE.CPP 中的代码就是用来实现这些的。

“Formats”向调用的客户表明此应用程序所支持的剪贴板的格式，在此程序中 CF _ TEXT 是所要求的格式，也是唯一支持的格式。

“TopicItemList”返回一系列本服务程序有关的主题，在本节的例子程序中，不存在与服务程序有关的主题。文件 STDDDE.CPP 包含函数 AddDDEItem，此函数的语法如下。

```
PDDEITEMINFO AddDDEItem (LPSTR lpszTopic,
                         LPSTR lpszItem,
                         LPWORD lpFormatList,
                         PDDEREQUESTFN lpReqFn,
                         PDDEPOKEFN lpPokeFn)
```

此函数向服务程序列表中添加一个新的 DDE 主题和项，并当客户发送主题与项的请求时为此项注册一个回调函数，参数 lpszTopic 指定主题的名字，lpszItem 指定主题中的项，参数 lpFormatList 指定主题所支持的所有格式的列表。函数 lpReqFn 处理请求调用，函数 lpPokeFn 处理 DDE _ POKE 的调用。

“Help”返回服务程序被指明部分的帮助信息。

“Status”返回在给定的通道上最后操作服务程序的状态，此消息对只返回成功与失败标志的 DDE 函数是有用的。

“ReturnMessage”返回最后的消息 DDE _ ACK 的状态与消息，它用来表明发送到服务程序的最后消息是否被确认。

注释

虽然这个特定的 DDE 服务程序只是支持系统主题，但它却是编写一个完整的 DDE 服务程序的极好开端。所有的基本功能就是使服务程序支持任何数目的主题与项，并把信息以独立于应用程序的方式返回给调用程序。

用户可以自由地把 STDDDEC.PP 与 STDDDE.H 文件包含在自己的应用程序中。

11.8 使文件对象的链接和嵌入（OLE）兼容

问题

许多人想使自己的应用程序能够适合于应用 OLE，但又不知如何下手。我们知道 MFC 包含了一些与 OLE 有关的类，不过，当要涉足这些类时，也需要慎重地考虑。

是否存在一种能简单实现应用程序的存储文件与 OLE 相容，而且同时不降低系统其余部分效率的方法呢？OLE 复合文档看起来是处理结构文件的一个非常好的方法。

方法

OLE2 的复合文档部分是系统的一种分立组件，它也是对应用程序增添 OLE 支持的一个非常好的起点。复合文档是简单的数据集合体，这些数据以层次的格式存放在磁盘上，实际上，复合文档就是小型文件系统。本节中，将讨论如何创建 OLE 复合文档，以及如何在这些复合文档间“流动”，以便把数据保存到磁盘使得在以后能够容易地再调用。

OLE 复合文档差不多是一个完整的磁盘系统。它包括自己的文件分配表（FAT）以及目录项、数据等，它将与基于 DOS 的 FAT 系统或者 Windows 95/NT 的 NTFS 文件系统协同

工作。为此，讨论利用复合文档的 OLE 就十分有意义。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH1108.MAK。选择主菜单 Dialog 中的菜单项 OLE File Dialog，便会看到一个新的对话框窗口，如图 11-8 所示，在编辑框中输入字符串，然后点击按钮 Add，不断重复这一过程，将在列表框中显示多条字符串，当对列表框中显示的字符串感到满意时，点击对话框中的按钮 Save。

接着，点击对话框中的按钮 Load，列表就在列表框中重新出现，此时，列表框中的内容以复合文档的形式被保存到了磁盘上，并在任何时刻都可被重新调用到应用程序。

图 11-8 OLE 文件对话框

为了在例子程序中实现上述功能，按如下步骤进行。

1. 在 Visul C++中创建项目文件 CH1108.MAK。为此项目文件选择选项 OLE，点击单选按钮 Container，这将创建利用 OLE 容器进行工作的基本功能，进入资源编辑器，从菜单列表中选择主菜单，在主菜单上添加标题为 Dialog 的菜单，在此菜单上添加标题为 OLE File Dialog 的下拉菜单项，设定此下拉菜单项的标识符为 ID_OLE_FILE_DLG。

2. 把按钮 OK 移到对话框的底部，并把此按钮的标题改为 Close，从对话框中删除按钮 Cancel，添加标题为“Enter a string:”的静态文本域，在紧接着此文本域的右边添加一个编辑框，在编辑框的右边添加标题为 &Add 的按钮。

3. 在以上添加的控制的下面，添加一个列表框。

4. 在列表框的下面添加标题分别为 Load、Save 的两个按钮。

5. 进入 ClassWizard，为刚定义的模板创建名为 COleFileDlgd 的新对话框类，并保存此类的定义。

6. 在 ClassWizard 中，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function 添加新函数 OnAdd。在 COleFileDlg 的方法 OnAdd 中添加下面的代码。

```
void COleFileDlg::OnAdd ()
{
    // Get the edit text
    char s [80];
    CEdit *edit = (CEdit *) GetDlgItem (IDC_EDIT1);
    memset ( s, 0, 80 );
    edit->GetWindowText ( s, 80 );
    // If there was anything there, put it into the list box
    if ( strlen (s) ) {
        CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
        list->AddString ( s );
    }
    // Clear the edit field text
```

```
    edit->SetWindowText ( "" );
    // Reset the focus to the edit field for more typing
    edit->SetFocus ();
}
```

7. 在ClassWizard中，从对象列表中选择对象IDC_BUTTON2，从消息列表中选择消息BN_CLICKED，点击按钮Add Function添加新函数OnLoad。在COleFileDlg的方法OnLoad中添加下面的代码。

```
void COleFileDlg:: OnLoad ()
{
    LPSTORAGE pStorage;
    LPSTREAM pStream;
    USHORT szwFile [512], szwInfo [512];
    // Try to open the storage device.
    // Make a Unicode copy of the filename.
    mbstowcs (szwFile, "test.doc", sizeof (szwFile));
    mbstowcs (szwInfo, "INFO", sizeof (szwFile));
    HRESULT hResult = StgOpenStorage (szwFile, NULL, STGM_READ |
                      STGM_SHARE_EXCLUSIVE |
                      STGM_DIRECT, NULL, NULL, &pStorage);
    if ( hResult ! = S_OK ) {
       // Couldn't open it.
       MessageBox ( "Couldn't open storage device!", "Error", MB_OK );
       return;
    }
    // Try to open the stream for input
    hResult = pStorage->OpenStream ( szwInfo, NULL, STGM_READ |
                      STGM_SHARE_EXCLUSIVE |
                      STGM_DIRECT, NULL, &pStream );
    if ( hResult ! = S_OK ) {
       // Couldn't open it,
       MessageBox ( "Couldn't create stream!", "Error", MB_OK );
       return;
    }
    // All went well. Read in elements and place in list box
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    list->ResetContent ();
    int numElements = 0;
    char buffer [80];
    unsigned long cb;
    pStream->Read ( &numElements, sizeof (numElements), &cb );
    for ( int i=0; i<numElements; ++i ) {
        memset ( buffer, 0, 80 );
```

```
        pStream->Read ( buffer, sizeof (buffer), &cb );
        list->AddString ( buffer );
    }
    // Close the stream
    pStream->Release ();
    pStorage->Release ();
}
```

8. 在ClassWizard中，从对象列表中选择对象IDC_BUTTON3，从消息列表中选择消息BN_CLICKED，点击按钮Add Function添加新函数OnSave。在COleFileDlg的方法OnSave中添加下面的代码。

```
void COleFileDlg:: OnSave ()
{
    LPSTORAGE pStorage;
    LPSTREAM pStream;
    USHORT szwFile [512], szwInfo [512];
    // Try to open the storage device.
    // Make a Unicode copy of the filename.
    mbstowcs (szwFile, "test.doc", sizeof (szwFile));
    mbstowcs (szwInfo, "INFO", sizeof (szwFile));
    // Try to open the storage device.
    HRESULT hResult = StgOpenStorage (szwFile, NULL, STGM_READWRITE |
                STGM_SHARE_EXCLUSIVE |
                STGM_DIRECT, NULL, NULL, &pStorage);
    if ( hResult ! = S_OK ) {
        // Couldn't open it. Create it.
        hResult = StgCreateDocfile ( szwFile, STGM_READWRITE |
                STGM_SHARE_EXCLUSIVE |
                STGM_DIRECT | STGM_CREATE, NULL, &pStorage);
        if ( hResult ! = S_OK ) {
            MessageBox ( "Couldn't create storage device!", "Error", MB_OK );
            return;
        }
    }
    // Try to open the stream for output
    hResult = pStorage->OpenStream ( szwInfo, NULL, STGM_READ |
                    STGM_SHARE_EXCLUSIVE |
                    STGM_DIRECT, NULL, &pStream );
    if ( hResult ! = S_OK ) {
        // Couldn't open it, create it.
        hResult = pStorage->CreateStream ( szwInfo, STGM_READWRITE |
                    STGM_SHARE_EXCLUSIVE |
                    STGM_DIRECT |
```

```
                STGM_CREATE, NULL, NULL, &pStream );
        if ( hResult != S_OK ) {
            MessageBox ( "Couldn't create stream!", "Error", MB_OK );
            return;
        }
    }
    // All went well. Get elements from list box and write out
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);
    int numElements = list->GetCount ();
    char buffer [80];
    unsigned long cb;
    pStream->Write ( &numElements, sizeof (numElements), &cb );
    for ( int i=0; i<numElements; ++i ) {
        memset ( buffer, 0, 80 );
        list->GetText ( i, buffer );
        pStream->Write ( buffer, sizeof (buffer), &cb );
    }
    // Close the stream
    pStream->Release ();
    pStorage->Release ();
}
```

9. 把下面的 include 文件行添加到源文件 OLEFILED.CPP 的顶端。

```
#include "ole2.h"
```

10. 在 ClassWizard 中，从下拉列表中选择对象 CCh1108App。从对象列表中选择对象 ID _OLE_FILE_DLG，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnOleFileDlg。在 CCh1108App 的方法 OnOleFileDlg 中添加下面的代码。

```
void CCh11App:: OnOleFileDlg ()
{
    COLFileDlg dlg;
    dlg.DoModal ();
}
```

11. 把下面一行添加到 CH11.CPP 顶端的 include 列中。

```
#include "olefiled.h"
```

12. 编译与运行此例子程序。

用法

OLE 复合文档例程由两个主类元组成，类元 IStorage 与类元 IStream。IStorage 代表为 OLE 存储信息的“存储设备”，此设备可驻留在磁盘或内存中，无论驻留在哪里，其工作方式都是一样的，不管 OLE 系统确定将存放在何处（内存或磁盘），存储设备本质上是带有特征的文件系统。

类 IStream 代表存储设备中的单一文件。所谓流，顾名思义就是以程序理解的方式来存储数据的一连串字节。类 IStorage 完全理解在磁盘或内存里的文件系统的表示，类 IStream 只理

解在存储设备内给定“文件”中一定数量可用的原始数据。

为了利用存储设备，必须首先打开或创建一个存储文件。这是通过调用函数 StgOpen-Storage 来完成的，或通过调用函数 StgCreateDocFile 来创建（或截取）存储文件。无论是打开或创建，都需要向函数传递存储设备的名字，名字可以是 NULL，当名字是 NULL 时，OLE 复合文档函数将创建一个临时的存储设备，此临时存储设备在程序退出或存储设备关闭时将被删除。这些函数返回一个指向 IStorage 类变量的指针，该指针被用来引用此存储设备。

当存储设备被打开后，必须在存储设备内打开或创建一个流，这就如同 DOS 在目录结构中打开或创建文件一样，完成此任务的两个函数为 CreateStream 与 OpenStream。在这两个函数中，都为流定义了便于引用的名字以及确定流的访问权限的一组标志。打开与创建函数都返回一个指向 IStream 类变量的指针，此指针将在后面用来对流进行访问。

当流被打开或创建后，它就如同文件一样可以被读出与写入，本例中，存储了列表框中元素的数目，该数目将被写到磁盘上，在其后是这些元素本身。当读回这些数据时，只要打开流就可读入元素的数目，然后通过循环把每一项读入，最后以这些项全部被存放到列表框中而结束循环。

注释

下一节将进一步讨论 OLE2 的功能，本节是利用 OLE 的开始。随着越来越多的应用程序与 OLE 兼容，自然要求开发人员创建一些能够利用 OLE 的应用程序。

利用 OLE 复合文档的文件结构其中较大好处之一是：保证与任何正在运行的系统兼容。例如，如果要在 NTFS 系统中直接访问磁盘，就要认真考虑用复合文档来保存数据，这样，不仅可以很迅速的找到数据，而且文件系统将不影响程序的运行。

11.9 创建 OLE 服务程序对象

问题

如果为应用程序编写一个完整的 OLE 服务程序，这就意味着不仅需要处理应用程序自己运行时的情况，而且还要处理把应用程序的对象嵌入到另一个应用程序的 OLE 容器时的情况。

在服务程序与应用程序之间以极少量的重复代码来完成这一任务的最简单方法是什么呢？

方法

有一种方法能使得在应用程序部分与 OLE 服务程序部分之间根本没有代码重复吗？对于这种方法，人们可能会十分惊讶并感到难以置信。

值得高兴的是，MFC 提供了在几乎不影响应用程序设计的情况下实现 OLE 服务程序的简单机制。本节中，创建了一个实现简单编辑器的简单 OLE 服务程序，此编辑器既能以独立的方式被利用，也可以在无需修改代码的情况下做为一个被嵌入的 OLE 对象来运用，最值得一提是，整个程序只需要几行代码。

步骤

打开与运行按照如下步骤生成的例子程序 CH109.MAK。点击窗口的视图区，然后在此编辑器中输入一些字符，可以看到，例子程序的此编辑器与一般的编辑器的表现基本相同。现在，关闭例子程序，启动一个 OLE 兼容的容器应用程序，诸如 Windows 的 Word，从此应用

程序的菜单 Edit 中选择菜单项 Insert Object，然后从显示的列表中选择 Ch11 文档，接着在对话框中点击按钮 OK，编辑器便以一个小的窗口在 Word 中显示出来，如图 11-9 所示。

按照如下的步骤，就能实现上述功能。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH1109.MAK。在选项中，为此项目文件选择 OLE 选项并点击单选按钮 Full Server，这将创建 OLE 服务程序的基本功能。

2. 进入 ClassWizard，从下拉列表中选择 CCh1109View。从对象列表中选择对象 CCh1109View，从消息列表中选择消息 WM_CREATE，点击按钮 Add Function 添加新方法 OnCreate，在 CCh1109View 的方法 OnCreate 中添加下面的代码。

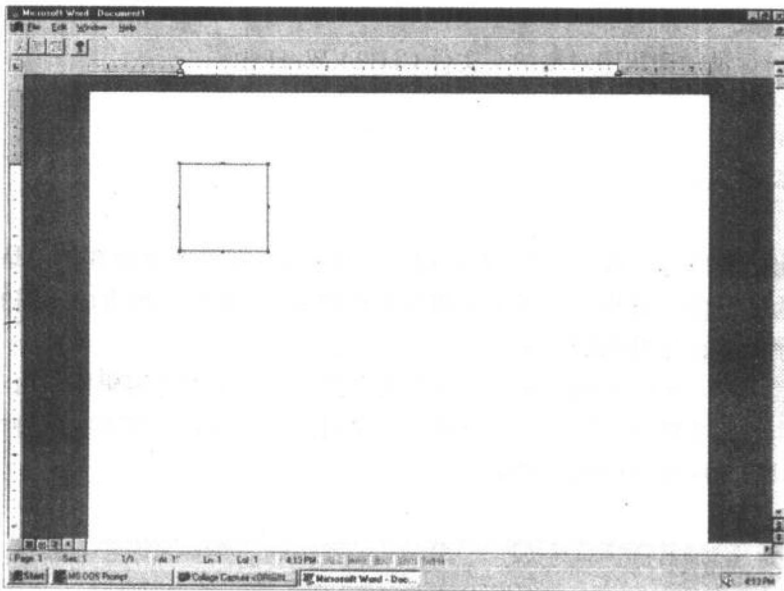


图 11-9 利用 Microsoft Word for Windows 的 OLE 服务程序例子

```
int CCh1109View:: OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CView:: OnCreate (lpCreateStruct) == -1)
        return -1;
        // Get the client rectangle for the entire view
        CRect r;
        GetClientRect (&r);
        // Create an edit control of that size
        if ( m_Edit.Create ( WS_CHILD | WS_VISIBLE | ES_MULTILINE |
            ES_UPPERCASE, r, this, 1001 ) == FALSE )
            MessageBox ("Cannot create edit box!", "Error", MB_OK );
    return 0;
}
```

3. 接着，在 ClassWizard 中，从对象列表中选择对象 CCh1109View，从消息列表中选择消息 WM_SIZE，点击按钮 Add Function 添加新方法，在 CCh1109View 的方法 OnSize 中添

加下面的代码。

```
void CCh1109View:: OnSize (UINT nType, int cx, int cy)
{
    CView:: OnSize (nType, cx, cy);
    // Get the client rectangle for the entire view
    CRect r;
    GetClientRect (&r);
    // Resize the edit control to that size
    m_Edit.MoveWindow (&r);
    m_Edit.SetFocus ();
}
```

4. 把下面一行添加到视图对象的头文件 CH11VW.H 中。

```
private:
    CEdit m_Edit;
```

5. 编译与运行此例子程序。

用法

从前面列出的代码可以看出，我们在创建应用程序时，并没有考虑其作用是做为 OLE 服务程序还是做为独立的应用程序，用于视图与文档的 MFC 类自动地生成所需的后台代码从而使得应用程序能够在任意的设置下运行。

在此例子程序中，我们在视图窗口的客户区内创建了一个多行编辑框。由于此客户区不依赖于程序是作为服务程序运行还是作为独立的应用程序运行的，所以在两种情况下都无需改动，编辑框只是为应用程序收集数据。

第 12 章　音效和音乐

从请求换盘的简单振铃到计算机游戏的音效，声音都被作为一种非视觉的交互方式，可以使得 Windows 应用程序易于使用并且充满乐趣。在本章中，首先将介绍如何在应用程序中添加重要的声音和音乐的多媒体效果，使得应用程序成为真正的 Windows 95 的应用程序，同时也将介绍如何管理音频 CD 中的曲目和如何播放音频 CD 中的音乐，如果有的程序员正准备开发某个出色的游戏，本章也将介绍如何在后台播放 MIDI 文件。

1. 如何播放音效

从商业应用到游戏，每个应用程序都需要简单的方法来播放音效，本节将示范如何以最少的编程工作在应用程序中添加音效，同时还讨论播放音效的几种方式。

2. 如何读取 CD 中的曲目信息

为收集的 CD 建立数据库！或者，替换 Windows 的 CD 播放器。本节将介绍如何实现音频 CD 的管理，介绍如何获取音频 CD 的曲目信息以及如何使用这些信息。

3. 如何播放 CD 音乐

本节将介绍如何从 CD 中播放曲目，从而使得应用程序成为真正的多媒体应用程序。

4. 如何播放 MIDI 音乐

如果有的程序员准备为应用程序添加后台音乐，例如开发新热门的计算机游戏，或者编写自己的 MIDI 音序器，本节将会有很大的帮助。本节将介绍如何以最小的处理器负载来播放音乐，使得程序员编写很小的应用程序便可以实现后台播放 MIDI 文件。

表 12-1 中列出了本章中使用的 Windows API。

表 12-1　在第 12 章中使用的 Windows API

PlaySound	mciSendString
mciSendCommand	mciGetErrorString
MM_MCINOTIFY	GetOpenFileName

12.1　播放音效

问题

有的程序员希望在应用程序中添加简单的音效，但是又不想为此而开发一个完整的音乐系统，是否有一种以少量代码来实现音效的方法呢？

方法

用于 Windows 应用程序的音效通常都是以采样的声波数据存放在 .WAV 文件中，使用 API 函数 PlaySound 来回放这些 wave 文件便可以非常简单地实现把音效添加到应用程序中。函数 PlaySound 是高层的音频函数，它以设备无关的方式驱动 PC 喇叭或声卡，使得程序员只需编写少量的代码便可以实现声音的播放，可以推断此方法必然缺乏对声音播放方式的直接控制。不过函数 PlaySound 总是以最好的方式来播放声音的，播放的频率取决于声音的采样频率和声卡所能回放的频率，不能通过此 API 函数调用来进行改变（尽管用户可以使用

控制面板来改变某些设置），同样，也不能通过 API 函数 PlaySound 的调用来实现混声或暂停声音的播放。

尽管有这些局限性，但是除了少数需要很强功能的应用程序外，函数 PlaySound 提供足够的灵活性用于大多数需求的应用程序中。在下面的步骤中将介绍如何使用此 API 函数来实现应用程序中音效的添加。

函数 PlaySound 有三个参数，此函数的语法为

```
BOOL PlaySound (LPCTSTR lpszName, // sound string
                HANDLE hModule,   // sound resource
                DWORD fdwSound);  // sound type
```

第一个参数是由应用程序提供的指针，通常指向要播放的 wave 文件名，也可用来传送资源名或别名（例如：system exclamation sound）。第二个参数是 HINSTANCE 句柄，指向某个可执行程序的实例，只有在设置了标志 SND _ RESOURCE（表示声音从资源中装入）时才使用此参数。

第三个参数是标志字，用来通知函数 PlaySound 有关声音数据的来源以及声音播放的方式。表 12-2 列出了所有的标志。

表 12-2 有关函数 PlaySound 的标志

标志	含义
SND _ ALIAS	参数 lpszName 指定登记表 [sounds] 部分中的一项，不能与 SND _ FILENAME 或 SND _ RESOURCE 同时使用
SND _ ASYNC	声音一开始播放后 PlaySound 就返回，不能与 SND _ SYNC 同时使用
SND _ FILENAME	命名参数指定 wave 文件名。不能与 SND _ ALIAS 或 SND _ RESOURCE 同时使用
SND _ NODEFAULT	即使找不到所请求的声音也不要播放系统缺省声音
SND _ NOWAIT	不要等待可用的 wave 设备，如果 wave 设备忙或不可用则调用返回
SND _ RESOURCE	命名参数指定资源名，不能与 SND _ ALIAS 或 SND _ FILENAME 同时使用
SND _ SYNC	函数 PlaySound 直到声音播放完才返回，不能与 SND _ ASYNC 同时使用

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将弹出一个小的对话框，如图 12-1 所示，此对话框有两个按钮，点击按钮 Close，则关闭此例子程序，点击按钮 Play Sound，则开始播放声音。

注：需要有声卡或类似的硬件，并且要将需要的驱动程序安装好。如果不清楚如何安装声卡驱动程序，可以查阅 Windows 手册。

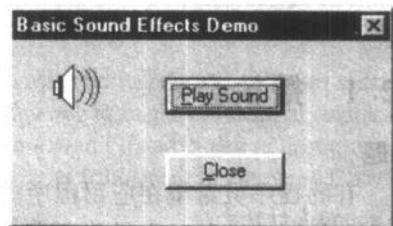


图 12-1 例子程序 SOUNDFX

实现例子程序的具体步骤如下：

1. 首先创建工作目录 SOUNDFX，用来存放所有的源文件。

2. 创建文本文件 SOUNDFX. RC，在文件中添加下列行，这就是用来定义对话框的资源脚本。

```
/*-----------------------------------------------------------*/
/*                                                           */
```

```
/* MODULE: SOUNDFX.RC                                                    */
/* PURPOSE: This is the resource script which defines the dialog         */
/*          and icon for the application.                                */
/*                                                                       */
/* --------------------------------------------------------------------- */
#include <windows.h>
#include "soundfx.rh"
IDD_PLAYSOUND DIALOG 78, 62, 157, 71
STYLE DS_MODALFRAME | WS_POPUP  WS_VISIBLE | WS_CAPTION |
    WS_SYSMENU | DS_3DLOOK
CAPTION "Basic Sound Effects Demo"
FONT 8, "MS Sans Serif"
{
    DEFPUSHBUTTON "&Play Sound", ID_PLAYSOUND, 63, 14, 50, 14,
                        BS_DEFPUSHBUTTON | WS_TABSTOP
    PUSHBUTTON "&Close", IDCANCEL, 63, 43, 50, 14, WS_TABSTOP
    ICON IDI_SOUNDS, IDI_SOUNDS, 15, 8, 18, 20
}
IDI_SOUNDS ICON "sounds.ico"
```

3. 创建另一文件 SOUNDFX. RH，此文件将被包含在刚创建的资源脚本中以及例子程序的源文件中，此文件定义的标识符用来标识对话框中的资源，在文件中添加下列代码。

```
#ifndef __SOUNDFX_RH
/* --------------------------------------------------------------------- */
/*                                                                       */
/* MODULE: SOUNDFX.RH                                                    */
/* PURPOSE: This include file defines the resouce IDs used in the        */
/*          application.                                                 */
/*                                                                       */
/* --------------------------------------------------------------------- */
#define __SOUNDFX_RH
#define IDI_SOUNDS       1001
#define IDD_PLAYSOUND    200
#define ID_PLAYSOUND     101
#endif /* not __SOUNDFX_RH */
```

4. 为例子程序准备图标，此图标显示在对话框中，使用资源编辑器创建一个 32×32 象素的图标，并保存在文件 SOUNDS. ICO 中。

5. 现在为例子程序创建源文件 SOUNDFX. C。在文件的顶部添加下列注释和说明，特别应该注意的是：在本例子程序中，除了包含通常的 Windows 的头文件外，还包含文件 MMSYSTEM. H，此文件是 Windows 多媒体扩充的主要头文件，此文件说明函数 PlaySound 的原型，并定义调用此函数时所使用的常量。

```
/* --------------------------------------------------------------------- */
/*                                                                       */
/* MODULE: SOUNDFX.C                                                     */
/* PURPOSE: This module demonstrates how you can use the                 */
```

```
/*          PlaySound API function to play sound effects from a file        */
/*          or from your application's resources.                          */
/*                                                                          */
/*--------------------------------------------------------------------------*/
#define STRICT
#include <windows.h>
#include <winnt.h>
#include <mmsystem.h>
#include "soundfx.rh"
```

6. 在已输入的代码下面添加下列代码。函数 PlaySoundEffect 调用函数 PlaySound 来播放声音，在本例子程序中，使用了标志 SND _ ASYNC，表示一旦开始声音播放，函数 PlaySound 便返回，如果希望例子程序暂停以等待声音播放结束，可以使用标志 SND _ SYNC 来替代标志 SND _ ASYNC，标志 SND _ SYNC 通知函数 PlaySound 直到声音播放结束后才交回控制给应用程序。

```
/*--------------------------------------------------------------------------*/
/* This function plays a sound file using the PlaySound function.           */
/* This API function also allows waves to be played from a resource,        */
/* or directly from a buffer in memory.                                     */
/*--------------------------------------------------------------------------*/
void PlaySoundEffect (HWND hWnd)
{
   BOOL ok;
   ok = PlaySound ("DEMO.WAV", NULL, SND_FILENAME | SND_ASYNC | SND_
NODEFAULT);
   if (! ok)
   MessageBox (hWnd,"Unable to play DEMO.WAV","PlaySound", MB_ICONEXCLAMATION | MB
                    _OK);
}
```

7. 在源文件中添加下面所示的函数 HandleDialog。此函数用来处理传送给对话框的 Windows 消息。此函数响应消息 WM _COMMAND，用来处理按钮点击。如果点击按钮 Play Sound，则此函数调用函数 PlaySoundEffect；如果点击按钮 Close，则此函数关闭对话框。

```
/*--------------------------------------------------------------------------*/
/* This function handles the messages sent to the dialog. When the          */
/* dialog receives the IDCANCEL command, the dialog is closed.              */
/* When the function receives the ID_PLAYSOUND command, the                 */
/* PlaySoundEffect function is called.                                      */
/*--------------------------------------------------------------------------*/
BOOL CALLBACK HandleDialog (HWND hWnd, UINT message,
                           WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
      case WM_INITDIALOG:
         return TRUE;
```

```
    case WM _COMMAND:
            if (wParam == IDCANCEL)
        {
            DestroyWindow (hWnd);
        return TRUE;
        }
            else
        if (wParam == ID _PLAYSOUND)
        {
            PlaySoundEffect (hWnd);
            return TRUE;
        }
        break;
    case WM _DESTROY:
        PostQuitMessage (0);
        return TRUE;
    }
    return FALSE;
}
```

8. 在源文件中添加下列代码。函数 WinMain 是 C 语言 Windows 应用程序的入口点或起始点，此函数创建对话框，接着在一个循环中调用函数 GetMessage、IsDialogMessage、TranslateMessage 以及 DispatchMessage，接收来自 Windows 的消息，并将消息传送给对话框的处理程序。

```
/*------------------------------------------------------------------------*/
/* Our main function — set up and run the application.                    */
/*------------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG       msg;
    HWND      hWnd;
    if ( (hWnd = CreateDialog (hInstance, MAKEINTRESOURCE (IDD _PLAYSOUND),
                               NULL, HandleDialog)) == NULL)
        return FALSE;
    while (GetMessage (&msg, NULL, NULL, NULL))
        if (! IsDialogMessage (hWnd, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    return msg. wParam;
```

9. 在成功地运行此例子程序前还需要一个文件，此文件要在当前的目录下，文件名应为 DEMO. WAV，此文件便是要播放的声音文件，可以拷贝其他应用程序中的某个 . WAV 文件.

也可以从 Windows\MEDIA 目录中拷贝标准的 Windows 声音。

注释

函数 PlaySound 使得程序员能够以少量代码便可以实现相当好的功能。在开发具有音效的应用程序时还需考虑的另外一个问题是：如何安排 wave 文件。在本节的例子程序中，要播放的声音文件是在当前目录下查找的，而在某些商品化的应用程序中，有可能是在应用程序所驻留的目录下查找声音文件，这两个方法都需要声音文件存在或者同应用程序一起提供。一个更好的方法是将声音数据作为资源封装在可执行文件中，调用函数 PlaySound 时，设置标志为 SND_RESOURCE，并在命名参数中指定资源名。

12.2 读取 CD 中的曲目信息

问题

有的程序员希望编写数据库来保存自己 CD 的曲目信息，是否有方法来读取 CD 中的信息呢？

方法

尽管音频 CD 上并没有存储有关曲目的详细信息，也不是以便于 Windows 应用程序读取的方式存储的，但是至少可以获取到曲目数以及每个曲目的长度，长度是以分秒来表示的。

这些信息都是通过媒体控制接口（MCI）来获取的。MCI 是 Windows 95 多媒体功能的应用程序编程接口，提供了对 CD、视盘、MIDI、动画以及 wave 设备的高层存取。有两种方法来访问 MCI 函数：第一种方法是使用函数 mciSendString 来发送字符串消息，例如"play cdaudio form track 5"，此方法对于面向字符串的高级语言如 Visual Basic 是非常有用的。第二种方法是使用函数 mciSendCommand，此函数使用的是 C 语言风格的结构和常量。由于使用函数 mciSendString 可以使得代码易读性强，并且可以移植到大多数的编程语言中，所以本章的例子程序中都使用此函数。

要查询 CD 首先要打开 CD 音频设备，然后便可以获取 CD 上的曲目数和每个曲目的长度。MCI 函数还能以多种格式指定曲目的当前位置信息和曲目的长度信息，大概最有用的格式是 TMSF，信息是以曲目、分、秒、帧的格式来返回的，在音频技术中，例如 CD 曲目中，术语帧似乎是没有什么意义的，此术语是用在视频领域中的。本节的例子程序是用来获取 CD 曲目信息的，所以把帧看作七十五分之一秒。

函数 mciSendCommand 的使用

在本章的所有例子程序中，都使用了函数 mciSendString 来完成同媒体控制接口的交互。此函数使用英语似的命令字符串来同系统进行通信，尽管此方法可以使得代码清晰，但是有些 C 语言程序员可能会厌烦太多的字符串创建和转换。作为另外一种选择，可以在应用程序中使用函数 mciSendCommand，此函数接受整数类型的命令和标志，需要时也接受结构指针。此函数的语法如下：

```
MCIERROR mciSendCommand (
    MCIDEVICEID  IDDevice, // device ID
    UNIT         uMsg, // command to send
    DWORD        fdwCommand, //specific flags
    DWORD        dwParam); // structure pointer
```

参数 IDDevice 用来传送一个句柄给多媒体设备，此句柄在打开设备时由 MCI_OPEN_PARAMS 结构返回。参数 uMsg 包含了要执行的命令，命令 MCI_OPEN 用来打开设备，命令

MCI _ CLOSE 用来关闭设备，命令 MCI _ PLAY 用来播放某个曲目或连续播放，命令 MCI _ STATUS 检索曲目和状态信息。第三个参数用来传送附加的操作标志，在此参数中使用 MCI _ NOTIFY 使得命令异步执行，并且在命令完成后 MCI 发送通知给应用程序。

最后一个参数 dwParam 用来传送结构的地址，某些 MCI 命令需要结构来定义任何附加参数（例如曲目号或设备别名）。关于使用函数 mciSendCommand 的详细信息可以参见 API 参考手册。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将弹出一个小的对话框，如图 12-2 所示，如果已安装 CD 播放器并且驱动器中有音频 CD，则例子程序将列出所有的曲目以及曲目的长度，否则例子程序将显示错误信息，并且列表框仍为空。要改变列出的曲目信息，可以插入另外一张 CD 盘并点击按钮 Refresh，使得例子程序从当前 CD 盘上读取数据。点击按钮 Close 则关闭例子程序。

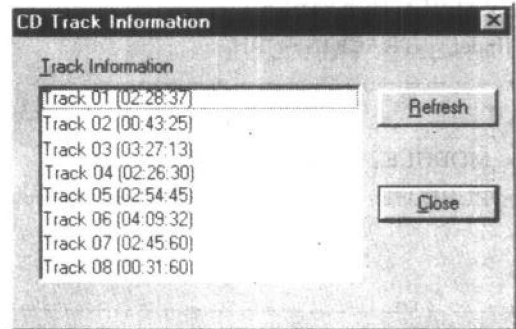


图 12-2 例子程序 TRACKINF

注：运行例子程序前需要安装 CD 播放器以及需要的驱动程序，如果不清楚如何安装声卡以及（或者）CD 播放器的驱动程序，可以查阅 Windows 说明手册。

实现例子程序的具体步骤如下：

1. 首先创建目录 TRACKINF，并以此目录为工作目录，来存放例子程序的所有源文件。

2. 创建资源脚本，用来定义对话框。尽管使用资源编辑器可以非常容易地完成资源脚本的创建，但在这里以文本文件的形式来创建会更加有用。创建文件 TRACKINF. RC，并在文件中输入下列行，此脚本包含了文件 TRACKINF. RH，接着定义要显示的对话框。

```
/*-----------------------------------------------------------------*/
/*                                                                 */
/* MODULE: TRACKINF.RC                                             */
/* PURPOSE: This is the resource script which defines the dialog   */
/*          and icon for the application.                          */
/*                                                                 */
/*-----------------------------------------------------------------*/
#include <windows.h>
#include "trackinf.rh"
IDD_TRACKINFO DIALOG 78, 62, 207, 112
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
        WS_CAPTION | WS_SYSMENU
CAPTION "CD Track Information"
FONT 8, "MS Sans Serif"
{
    LTEXT "&Track Information", -1, 11, 7, 59, 8
    LISTBOX IDL_TRACKLIST, 11, 19, 133, 81, LBS_NOTIFY | WS_BORDER |
```

```
            WS_BORDER | WS_VSCROLL
    PUSHBUTTON "&Refresh", ID_REFRESH, 152, 20, 50, 14, WS_TABSTOP
    DEFPUSHBUTTON "&Close", IDCANCEL, 152, 57, 50, 14,
                        BS_DEFPUSHBUTTON | WS_TABSTOP
}
```

3. 定义资源标识符，这些标识符曾在资源脚本中使用。创建文本文件 TRACKINF.RH，在文件中添加下列定义：

```
#ifndef __TRACKINF_RH
/*--------------------------------------------------------------*/
/*                                                              */
/* MODULE: TRACKINF.RH                                          */
/* PURPOSE: This include file defines the resource IDs used in the */
/*          application.                                        */
/*                                                              */
/*--------------------------------------------------------------*/
#define __TRACKINF_RH
#define IDD_TRACKINFO    200
#define IDL_TRACKLIST    102
#define ID_REFRESH       103
#endif /* not __TRACKINF_RH */
```

4. 现在创建例子程序的源文件。创建新文件 TRACKINF.C，并在文件中添加下列注释和 #include 语句。注意在本例子程序中，除了通常的 Windows 的 include 文件外，还包含了文件 MMSYSTEM.H，此文件定义了有关多媒体和 MCI 应用程序编程接口的函数原型和常量。

```
/*--------------------------------------------------------------*/
/*                                                              */
/* MODULE: TRACKINF.C                                           */
/* PURPOSE: This application demonstrates the use of the        */
/*          mciSendCommand API function to retrieve information */
/*          about the tracks on a CD.                           */
/*                                                              */
/*--------------------------------------------------------------*/
#define STRICT
#include <windows.h>
#include <winnt.h>
#include <mmsystem.h>
#include <stdlib.h>
#include "trackinf.rh"
```

5. 在同一源文件中添加下列函数。函数 GetTrackInfo 打开 CD 音频设备，如果成功，则使用命令“status cdaudio number of tracks”来获取 CD 上的曲目数，接着循环所有的曲目，对于每个曲目，使用命令“status cdaudio length track”来查询。接着使用返回的字符串连接曲目号形成一行文本，接着将文本插入到列表框中。注意：使用命令“set cdaudio time format

TMSF”调用函数 mciSendString，此命令确保长度查询的返回值为所要求的格式。

```
/*-------------------------------------------------------------------*/
/* This function attempts to read the track information for the CD  */
/* Audio device, and inserts the information as strings into a list  */
/* box. The function returns TRUE if successful, or FALSE if there    */
/* is no CD in the drive.                                             */
/*-------------------------------------------------------------------*/
BOOL GetTrackInfo (HWND hWnd)
{
    HWND            hWndList;
    char            retBuf [60], trackDesc [60];
    char            buf [256];
    DWORD           numTracks, track;
    MCIERROR        mciError;
    /* Get the handle of the listbox to add the items to. */
    hWndList = GetDlgItem (hWnd, IDL_TRACKLIST);
    mciError = mciSendString ("open cdaudio", retBuf,
                        sizeof (retBuf), NULL);
    if (mciError == 0)
    {
        mciError = mciSendString ("status cdaudio number of tracks",
                                retBuf, sizeof (retBuf), NULL);
        if (mciError == 0)
        {
            // Set the type of time information to be returned.
            mciError = mciSendString ("set cdaudio time format TMSF", NULL, 0, NULL);
            // Loop through and retrieve info for each track.
            numTracks = atoi (retBuf);
            for (track = 1; ((mciError == 0) &&
                        (track <= numTracks)); track++)
            {
              // Get the length of each track, and add the info
                // to the listbox.
                wsprintf (trackDesc,"status cdaudio length track %d", track);
                mciError = mciSendString (trackDesc, retBuf,
                                    sizeof (retBuf), NULL);
                if (mciError == 0)
                {
                wsprintf (buf,"Track %02ld (%s)", track, retBuf);
                SendMessage (hWndList, LB_ADDSTRING, 0, (LPARAM) buf);
                }
            }
        }
```

```
    // Close the device for the moment.
    mciSendString ("close cdaudio", NULL, 0, NULL);
    }
    // Return TRUE if successful, otherwise display the error
    // and return FALSE.
    if (mciError == 0)
        return TRUE;
    else
    {
        mciGetErrorString (mciError, buf, sizeof (buf));
        MessageBox (hWnd, buf,"Track Info", MB_ICONSTOP | MB_OK);
        return FALSE;
    }
}
```

6. 添加下面的函数 HandleDialog，此函数响应传送给对话窗口的 Windows 消息。特别地，为了响应消息 WM_INITDIALOG，此函数填充列表框，为了响应消息 ID_REFRESH，此函数也完成同样的操作，在后一种情况中，此函数在添加新的曲目信息前首先清除列表框中的内容。此函数还通过关闭对话框来响应消息 IDCANCEL，通过结束例子程序来响应消息 WM_DESTROY。

```
/*--------------------------------------------------------------------*/
/* This function handles the messages sent to the dialog.             */
/*--------------------------------------------------------------------*/
BOOL CALLBACK HandleDialog (HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            GetTrackInfo (hWnd);
            return TRUE;
        case WM_COMMAND:
            if (wParam == IDCANCEL)
            {
                DestroyWindow (hWnd);
                return TRUE;
            }
            else
                if (wParam == ID_REFRESH)
                {
                    // The user clicked the Refresh button. Clear
                    // the contents of the list box and refresh
                    // the list.
```

```
                SendDlgItemMessage (hWnd, IDL_TRACKLIST,
                                    LB_RESETCONTENT, 0, 0);
                GetTrackInfo (hWnd);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage (0);
            return TRUE;
    }
    return FALSE;
}
```

7. 最后，在源文件中添加下列代码。此为函数 WinMain，当例子程序启动时 Windows 调用此函数，此函数创建对话框，接着循环处理消息直到例子程序结束。

```
/*------------------------------------------------------------------------*/
/* Our main function - set up and run the application.                   */
/*------------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG         msg;
    HWND        hWnd;
    if ((hWnd = CreateDialog (hInstance, MAKEINTRESOURCE(IDD_TRACKINFO),NULL,Handle-
                    Dialog)) == NULL)
        return FALSE;
    while (GetMessage (&msg, NULL, NULL, NULL))
        if (! IsDialogMessage (hWnd, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    return msg.wParam;
}
```

注释

MCI 库提供的曲目信息和状态信息在单独使用时是没有什么用的，可能的用处是在播放 CD 时提供动态的标示，以及用来创建用户 CD 库的信息数据库，用户可以插入光盘，只需简单的填充艺术家和标题信息，而曲目信息可以从 CD 中直接获取。

12.3 播放 CD 音乐

问题

如何播放音频 CD 上的音乐呢？当用户正在工作时能否在后台播放呢？

方法

利用媒体控制接口（MCI）函数，可以编写非常简单的高层代码来实现整个 CD 的播放，

或者在CD上选择曲目播放。MCI是一个API函数集，使得程序员可以编写高层的、设备独立的代码，MCI解释来自应用程序的命令并传送给设备驱动程序，设备驱动程序完成同硬件设备的交互。使用MCI有两个途径：使用函数mciSendCommand来发送命令给系统并接收结果，通信是通过使用一些C语言类型的结构和常量来完成的。完成同样的功能还可以使用一个更简单的方法，即使用函数mciSendString，此函数接受类似英语的字符串命令，返回的结果也是一个字符串，此方法可读性强，并且易于移植到其他的编程语言和编程环境。

播放音频CD需要三个步骤：

1. 打开设备。
2. 播放CD。
3. 关闭CD。

按照上面的步骤，使用函数mciSendString可以实现一个非常简单的程序，即一个简单的CD播放器。如果要求更加实用，则还需要添加另外的功能，如可以选择曲目，可以在后台播放音乐，这样便可以边听音乐边工作。但是这样便需要再添加两个步骤，因此步骤如下：

1. 打开设备。
2. 获取曲目信息，以便用户选取。
3. 播放整个CD或CD的某些曲目。
4. 等待CD播放结束。
5. 关闭CD。

在12.2节中，详细介绍了如何从音频CD中获取曲目信息并将其显示出来。打开、播放和关闭CD都是一些简单的命令，所以剩下的问题是如何等待CD播放结束而不需要将计算机挂起。在Windows 95中，可以启动另外一个单独的线程来播放CD，但是还有一个更加简单的方法：在mciSendString发送的命令末添加关键字“notify”，此方法使得命令变为异步执行，一旦进程启动则返回控制给应用程序，当进程结束或失败时，应用程序可以接收到消息MM_MCINOTIFY。

在下面的步骤中，将介绍如何实现一个简单的CD播放器。

步骤

按照下列步骤实现一个例子程序。此例子程序开始显示的对话框如图12-3所示，左边的列表显示的是当前插入CD的曲目，下面是按钮Play和Stop。开始的时候，按钮Play是激活的，按钮Stop是禁止的，一旦用户选取了某个曲目并点击按钮Play，例子程序便开始播放选中的曲目，在CD播放时，按钮Play变灰，但可以点击按钮Stop来中止播放。一旦中止了播放，则按钮Play又变为激活，按钮Stop又变为禁止。

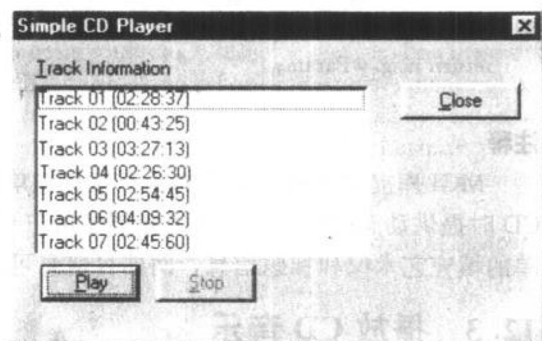


图12-3 例子程序PLAYCD

注：需要有CD播放器和安装必要的驱动程序，如果不清楚如何安装声卡和（或）CD播放器的驱动程序，可以查阅Windows说明手册。

实现例子程序的具体步骤如下：

1. 创建新目录 PLAYCD，作为本例子程序的工作目录，用来存放例子程序的所有源文件。

2. 创建资源脚本，定义本例子程序的对话框。使用文本编辑器创建新文件 PLAYCD.RC，在文件中键入下列资源编译器命令，这些命令定义了用来播放 CD 的对话框，对话框包括一个用于曲目选取的列表框、按钮 Play 和 Stop，还有按钮 Close 用来终止例子程序的运行，开始时按钮 Play 和 Stop 是禁止的。

```
/*-----------------------------------------------------------*/
/*                                                           */
/* MODULE: PLAYCD.RC                                         */
/* PURPOSE: This is the resource script which defines the dialog */
/*          and icon for the application.                    */
/*                                                           */
/*-----------------------------------------------------------*/
#include <windows.h>
#include "playcd.rh"
IDD_PLAYCD DIALOG 34, 35, 220, 116
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
CAPTION "Simple CD Player"
FONT 8, "MS Sans Serif"
{
    LTEXT "&Track Information", -1, 9, 6, 58, 8
    LISTBOX IDL_TRACKLIST, 9, 18, 139, 68, LBS_STANDARD
    DEFPUSHBUTTON "&Play", ID_PLAYTRACK, 11, 87, 41, 14,
                        BS_DEFPUSHBUTTON | WS_TABSTOP
    PUSHBUTTON "&Close", IDCANCEL, 161, 16, 50, 14, WS_TABSTOP
    PUSHBUTTON "&Stop", ID_STOP, 59, 87, 41, 14, WS_DISABLED | WS_TABSTOP
}
```

3. 现在创建一个 include 文件，定义一些资源标识符，这些资源标识符用在资源脚本中，在按钮的状态改变时或者填充查询列表框时，例子程序也引用这些标识符。创建新文件 PLAYCD.RH，在文件中添加下列定义：

```
#ifndef __PLAYCD_RH
/*-----------------------------------------------------------*/
/*                                                           */
/* MODULE: PLAYCD.RH                                         */
/* PURPOSE: This include file defines the resouRce Id's used in the */
/*          application.                                     */
/*                                                           */
/*-----------------------------------------------------------*/
#define __PLAYCD_RH
#define IDD_PLAYCD         200
```

```
#define IDL_TRACKLIST       101
#define ID_PLAYTRACK        102
#define ID_STOP             103
#endif /* not _PLAYCD_RH */
```

4. 在创建 include 文件后，现在为例子程序编写源代码。创建另一新文件 PLAYCD.C，在文件的顶部添加下列注释和语句，特别应该注意 #include 语句，除了通常的 Windows 的 include 文件，还包含了文件 MMSYSTEM.H，所有的多媒体应用程序都需要此文件，此文件定义了常量和函数原型，它们构成了 MCI 和其他多媒体系统间的接口。

```
/*-----------------------------------------------------------*/
/*                                                           */
/* MODULE: PLAYCD.C                                          */
/* PURPOSE: This application demonstrates the use of the     */
/*          mciSendCommand API function to retrieve information */
/*          about the tracks on a CD.                        */
/*                                                           */
/*-----------------------------------------------------------*/
#define STRICT
#include <windows.h>
#include <winnt.h>
#include <mmsystem.h>
#include <stdlib.h>
#include "playcd.rh"
```

5. 在同一源文件中添加下列函数。函数 GetTrackInfo 首先尝试打开 cdaudio 设备，如果成功，则获取当前 CD 上的曲目数，遍历这些曲目，以 TMSF (tracks, minutes, seconds, 以及 frames) 格式获取每个曲目的长度，并将查询结果以字符串的形式放入列表框中。

```
/*-----------------------------------------------------------*/
/* This function attempts to read the track information for the CD */
/* Audio device, and inserts the information as strings into a list */
/* box. The function returns TRUE if successful, or FALSE if there  */
/* is no CD in the drive.                                    */
/*-----------------------------------------------------------*/
void GetTrackInfo (HWND hWnd)
{
    HWND        hWndList;
    char            retBuf [60], trackDesc [60];
    char            buf [256];
    DWORD       numTracks, track;
    MCIERROR    mciError;
    // Get the handle of the listbox to add the items to.
    hWndList = GetDlgItem (hWnd, IDL_TRACKLIST);
    mciError = mciSendString ("open cdaudio", retBuf,
                        sizeof (retBuf), NULL);
```

```
if (mciError == 0)
{
    mciError = mciSendString ("status cdaudio number of tracks",
                              retBuf, sizeof (retBuf), NULL);
    if (mciError == 0)
    {
        // Set the type of time information to be returned.
        mciError = mciSendString ("set cdaudio time format TMSF",
                                  NULL, 0, NULL);
        // Loop through and retrieve info for each track.
        numTracks = atoi (retBuf);
        for (track = 1; ((mciError == 0) &&
                 (track <= numTracks)); track++)
        {
            // Get the length of each track, and add the info to
            // the listbox.
            wsprintf (trackDesc,"status cdaudio length track %d",
                   track);
            mciError = mciSendString (trackDesc, retBuf,
                                      sizeof (retBuf), NULL);
            if (mciError == 0)
            {
                wsprintf (buf,"Track %02ld (%s)", track, retBuf);
                SendMessage (hWndList, LB_ADDSTRING, 0, (LPARAM) buf);
            }
        }
    }
    // Close the device for the moment.
    mciSendString ("close cdaudio", NULL, 0, NULL);
    }
// Display the error if one occurred.
if (mciError != 0)
{
    mciGetErrorString (mciError, buf, sizeof (buf));
    MessageBox (hWnd, buf,"Play CD", MB_ICONSTOP | MB_OK);
}
}
```

6. 下面的函数用来启动CD播放，此函数需要获取选中的曲目号以及列表框中列出的曲目总数。MCI的播放命令有三种可能的形式：命令“play cdaudio”表示从CD的当前位置开始播放，直到CD的末尾；在命令中指明开始播放的曲目，例如“play cdaudio from 3”，表示从指定的曲目开始播放，直到光盘的末尾；在命令中再指明要播放到的曲目，例如“play cdaudio from 3 to 4”，表示从曲目3开始播放，直到曲目4，如果曲目4不存在，则MCI返回错

误。

下面的代码从选中的曲目开始播放，直到下一曲目的开始，如果选中的曲目是CD的最后一个曲目，则播放到CD的末尾。无论哪一种情况，都只播放一个曲目。

```
/*---------------------------------------------------------------*/
/* This function opens the CD Audio device and plays the selected  */
/* track. The function specifies the NOTIFY option, so a message   */
/* will be sent when the track is completed.                        */
/*---------------------------------------------------------------*/
void PlayTrack (HWND hWnd)
{
    char              buf [60];
    MCIERROR          mciError;
    LRESULT           trackID;
    HWND              hWndList;
    DWORD             numTracks;
    // See if there is a track selected in the list box. If there
    // is, play that track, otherwise display an error and exit.
    hWndList = GetDlgItem (hWnd, IDL_TRACKLIST);
    if (hWndList)
        trackID = SendMessage (hWndList, LB_GETCURSEL, 0, 0);
    else
        trackID = LB_ERR;
    if (trackID == LB_ERR)
    {
        MessageBox (hWnd,"Please select a track to play","Play CD",
                    MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    // Open the device.
    mciError = mciSendString ("open cdaudio", buf, sizeof (buf), NULL);
    if (mciError == 0)
    {
        // Set the type of time information used.
        mciSendString ("set cdaudio time format TMSF",
                    NULL, 0, NULL);
        mciSendString ("status cdaudio number of tracks",
                    buf, sizeof (buf), NULL);
        numTracks = atoi (buf);
        // Play the specified track. Using the word notify
        // tells the MCI interface to send a notification message
        // to the specified window when done.
        trackID--;
```

```
        if (trackID < numTracks)
          wsprintf (buf,"play cdaudio from %ld to %ld notify",
                    trackID, trackID + 1);
        else
          wsprintf (buf,"play cdaudio from %ld notify", trackID);
        mciError = mciSendString (buf, NULL, 0, hWnd);
        if (mciError == 0)
        {
          // Enable the Stop button and gray the Play button.
          EnableWindow (GetDlgItem (hWnd, ID_PLAYTRACK), FALSE);
          EnableWindow (GetDlgItem (hWnd, ID_STOP), TRUE);
        }
        else // Close the device if playing failed.
            mciSendString ("close cdaudio", NULL, 0, NULL);
    }
    // Display the error if one occurred.
    if (mciError != 0)
    {
        mciGetErrorString (mciError, buf, sizeof (buf));
        MessageBox (hWnd, buf,"Play CD", MB_ICONSTOP | MB_OK);
    }
}
```

注：在上面的代码中，如果调用成功则不关闭CD音频设备，成功的调用将启动CD的播放，一旦播放结束则例子程序将接收到一个消息。

7. 函数TrackDone用来响应消息MM_MCINOTIFY。无论异步的MCI命令是成功还是失败，此消息在命令结束时被发送，将此函数添加在同一源文件中。

```
/*-------------------------------------------------------------------*/
/* This function handles the message sent to the dialog when the    */
/* cd is finished playing.                                           */
/*-------------------------------------------------------------------*/
void TrackDone (HWND hWnd)
{
    // Enable the Play button and gray the Stop button.
    EnableWindow (GetDlgItem (hWnd, ID_PLAYTRACK), TRUE);
    EnableWindow (GetDlgItem (hWnd, ID_STOP), FALSE);
    // Close the CD device.
    mciSendString ("close cdaudio", NULL, 0, NULL);
}
```

8. 在源文件PLAYCD.C中添加下列代码。当用户点击按钮Stop时将调用此函数，此函数还在例子程序结束时被调用。因此，在使用函数mciSendString发送消息“stop cdaudio”给例子程序前，此函数需要确定按钮Stop是否激活的，从而确定CD是否正在播放。

```
/*-------------------------------------------------------------------*
```

```
/* This function responds when the user clicks the stop button. It      */
/* checks to ensure that the stop button is enabled (which tells us     */
/* that the CD is playing), and stops the CD device.                    */
/*--------------------------------------------------------------------- */
void StopButtonClicked (HWND hWnd)
{
    HWND        hWndStop;
    MCIERROR        mciError;
    char        buf [100];
    hWndStop = GetDlgItem (hWnd, ID_STOP);
    if ( (hWndStop) && (IsWindowEnabled (hWndStop)))
    {
        // Tell the CD to stop playing.
        mciError = mciSendString ("stop cdaudio", NULL, 0, NULL);
        // Display the error if one occurred.
        if (mciError ! = 0)
        {
            mciGetErrorString (mciError, buf, sizeof (buf));
            MessageBox (hWnd, buf,"Play CD", MB_ICONSTOP | MB_OK);
        }
    }
}
```

9. 在源文件中添加如下所示的函数 HandleDialog。此函数是一个回调函数，用来处理发送给对话框的消息，此函数响应许多 Windows 消息。处理的第一个消息是 WM_INITDIALOG。为了响应此消息，函数将 CD 的曲目信息显示在列表框中。当此函数接收到命令消息 ID_PLAYTRACK 时，调用前面编写的函数 PlayTrack 来开始播放选中的曲目。此函数还响应命令消息 ID_STOP，调用函数 StopButtonClicked 来中止音乐的播放。另外，此函数在对话框关闭时调用函数 StopButtonClicked。

此函数处理的最特别的消息是 MM_MCINOTIFY，此消息的发送是因为在函数 PlayTrack 中调用函数 mciSendString 时指定了“notify”，此消息用来通知命令（在这里是指播放曲目）已经结束、失败或者被中断。参数 LPARAM 中包含发送消息的设备 ID，参数 WPARAM 中包含发送消息的原因解释。在本例子程序中，只有一个设备发送消息，并且忽略原因的解释，只是调用函数 TrackDone 来改变按钮的状态（激活或禁止）。

```
/*--------------------------------------------------------------------- */
/* This function handles the messages sent to the dialog. When the      */
/* dialog receives the IDCANCEL command, the dialog is closed.          */
/*--------------------------------------------------------------------- */
BOOL CALLBACK HandleDialog (HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
```

```
        case WM_INITDIALOG:
            GetTrackInfo (hWnd);
            return TRUE;
        case WM_COMMAND:
            switch (wParam)
            {
                case IDCANCEL:
                    DestroyWindow (hWnd);
                    return TRUE;
                case ID_PLAYTRACK:
                    PlayTrack (hWnd);
                    return TRUE;
                case ID_STOP:
                    StopButtonClicked (hWnd);
                    return TRUE;
                default:
                    break;
    }
            break;
        case MM_MCINOTIFY:
            TrackDone (hWnd);
        break;
        case WM_DESTROY:
            StopButtonClicked (hWnd);
            PostQuitMessage (0);
            return TRUE;
    }
    return FALSE;
}
```

10. 最后，添加函数 WinMain。当例子程序第一次被装入时调用此函数，用来初始化对话框并循环处理消息，使得例子程序可以运行。在源文件 PLAYCD.C 的末尾添加下列代码：

```
/*---------------------------------------------------------------------------*/
/* Our main function — set up and run the application.                       */
/*---------------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG       msg;
    HWND      hWnd;
    if ( (hWnd = CreateDialog (hInstance, MAKEINTRESOURCE (IDD_PLAYCD),
                        NULL, HandleDialog)) == NULL)
    return FALSE;
    while (GetMessage (&msg, NULL, NULL, NULL))
```

```
    if (! IsDialogMessage (hWnd, &msg))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
  return msg.wParam;
}
```

注释

例子程序 PLAYCD 示范了如何使用 Windows 的媒体控制接口来播放 CD 上特定的曲目。可以扩充本例子程序的功能来播放整个 CD，也可以使本节的例子程序与 12.2 节中讨论的例子程序相结合，以获取 CD 的曲目信息，实现一个具有数据库功能的 CD 播放器，这样用户便可以选择喜欢和不喜欢的曲目，CD 播放器则只播放选取的曲目。另外还可以编写一个非常小的应用程序，当用户在前台工作时可以在后台简单的随机播放曲目。

12.4 播放 MIDI 音乐

问题

有的程序员希望自己的应用程序运行时在后台播放音乐。但是要实现此功能，需要的 wave 数据可能会使得应用程序相当庞大，是否有方法利用大多数声卡支持的 MIDI 功能，并以很少的硬盘和内存空间实现后台音乐的播放呢?

方法

通常的 Windows 声音在大多数的声卡上都能产生正常品质的输出效果，产生令人激动并有用的声音，当然，同样的技术也可以用来为游戏或其他的多媒体应用程序播放后台音乐。但是，即使是几秒钟音乐的采样数据也是相当庞大的，一个明显的问题是需要为大量的数据寻找硬盘空间，另外还需要占用大量的处理器时间来为声卡装入和准备这些数据。

解决此问题的方法是使用合成音乐，而不是使用采样的声音，存储和播放合成音乐使用乐器数字接口，或简称 MIDI。大多数的声卡都支持某预设范围内的标准合成器声音，即声卡存储产生声音所需要的全部波形，或者至少能够使用算法来近似波形。应用程序所需要的仅仅是一个 .MID 文件，此文件用来存放用于播放每个音符的音调、持续时间和乐器，这些数据非常少，并且需要较少的处理器时间来操作。

那么，如何来使用声卡的 MIDI 功能呢? 通常，此问题的答案随厂商的不同而不同，甚至随声卡的不同而不同。但在 Microsoft Windows 3.1 中，Microsoft 添加了多媒体系统的 API，提供了一个通用的方法来使用 MIDI 功能，而不需要知道声卡的确切接口。在本节的例子程序中，将使用 Windows 多媒体 API 后台播放 MIDI 文件。

播放 MIDI 文件类似于播放其他的多媒体设备。首先需要打开适当的设备，然后发送播放命令以及播放文件所需要的信息。有两个 API 函数可以用来发送命令给多媒体设备。C 语言程序员通常使用函数 mciSendCommand，此函数使用 C 语言风格的结构和常量来完成任务。另一个函数是 mciSendString，此函数使用字符串语法，用其他语言编程时非常易于使用，也使得程序的可读性强，因此本节使用此函数来同 MIDI 设备通信。

第二个需要掌握的技巧是：在应用程序执行其他函数时，如何使得 MIDI 设备在后台进行

播放。实现此功能的关键是使用关键字 "notify"，添加此关键字给多媒体系统函数，使得一旦开始播放音乐，Windows 就返回控制给应用程序，使得 MIDI 设备在后台操作，当音乐播放完后再通知应用程序。在下面的步骤中将介绍如何使用此方法来播放 MIDI 文件

步骤

按照下列步骤实现一个例子程序。运行此例子程序，点击按钮 Browse，寻找一个 MIDI 文件来播放，可以双击来打开文件，也可在标准的打开文件对话框中点击按钮 Open。

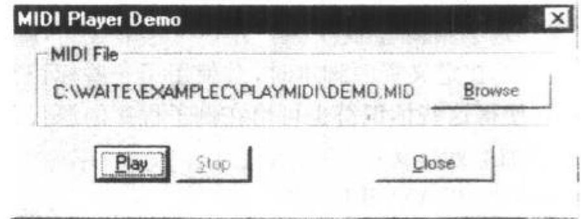


图 12-4　例子程序 PLAYMIDI

一旦选取一个 MIDI 文件，则文件名将显示在对话框中，并且按钮 Play 将被激活，图 12-4 所示的就是例子程序的当前状态。通过点击按钮 Play 来播放选取的文件，当文件正在播放时，按钮 Play 将变灰，按钮 Stop 将被激活。可以等待音乐播放结束，也可点击按钮 Stop 来中止音乐的播放，无论哪一种情况，按钮 Play 都将再被激活，而按钮 Stop 将变灰。如果准备关闭例子程序，则可以点击按钮 Close。

下面就是实现此例子程序的步骤，将说明如何使用多媒体系统 API 来播放 MIDI 文件，以及如何在后台播放音乐。

注：必须有支持 MIDI 播放功能的声卡，并且安装必要的驱动程序，如果不清楚如何来安装声卡驱动程序和 MIDI 映象程序，可以查阅 Windows 说明手册。

实现例子程序的具体步骤如下：

1. 首先创建目录 PLAYMIDI，用来存放例子程序的文件。

2. 接着为例子程序创建资源脚本 PLAYMIDI.RC，在文件中添加下列行，用来定义对话框，此对话框在例子程序运行时将显示在屏幕上。

```
/*-------------------------------------------------------------*/
/*                                                             */
/* MODULE: PLAYMIDI.RC                                         */
/* PURPOSE: This is the resource script which defines the dialog */
/* and icon for the application.                               */
/*                                                             */
/*-------------------------------------------------------------*/
#include <windows.h>
#include "playmidi.rh"
IDD_PLAYMIDI DIALOG 30, 62, 233, 70
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
CAPTION "MIDI Player Demo"
FONT 8, "MS Sans Serif"
{
    GROUPBOX "MIDI File", ID_GROUPBOX1, 9, 3, 219, 33, BS_GROUPBOX
    LTEXT "", ID_FILENAME, 16, 18, 152, 8
```

```
    DEFPUSHBUTTON "&Browse", ID_BROWSE, 174, 15, 50, 14,
                        BS_DEFPUSHBUTTON | WS_TABSTOP
    PUSHBUTTON "&Play", ID_PLAY, 33, 43, 31, 14, WS_DISABLED | WS_TABSTOP
    PUSHBUTTON "&Stop", ID_STOP, 67, 43, 31, 14, WS_DISABLED | WS_TABSTOP
    PUSHBUTTON "&Close", IDCANCEL, 150, 43, 50, 14, WS_TABSTOP
}
```

3. 在定义资源脚本时，曾使用了一些标识符，现在创建一个include文件来定义这些标识符，使得这些标识符也可用在例子程序的源代码中。创建文本文件PLAYMIDI.RH，在文件中添加下列定义：

```
#ifndef __PLAYMIDI_RH
/*------------------------------------------------------------------------*/
/*                                                                        */
/* MODULE: PLAYMIDI.RH                                                    */
/* PURPOSE: This include file defines the resource IDs used in the        */
/*          application.                                                  */
/*                                                                        */
/*------------------------------------------------------------------------*/
#define __PLAYMIDI_RH
#define IDD_PLAYMIDI        200
#define ID_GROUPBOX1        101
#define ID_FILENAME         102
#define ID_BROWSE           103
#define ID_PLAY             104
#define ID_STOP             105
#endif /* not __PLAYMIDI_RH */
```

4. 准备工作就绪后，开始着手主任务，即为例子程序编写C语言源代码。创建文件PLAYMIDI.C，并在文件中添加下列行，这些行包含了例子程序所需要的头文件，特别应注意包含了文件MMSYSTEM.H，此文件提供了Windows多媒体函数所需用的常量、结构和函数原型。

```
/*------------------------------------------------------------------------*/
/*                                                                        */
/* MODULE: PLAYMIDI.C                                                     */
/* PURPOSE: This application demonstrates how you can use the             */
/*          multimedia system API to play MIDI files using the MIDI       */
/*          mapper.                                                       */
/*                                                                        */
/*------------------------------------------------------------------------*/
#define STRICT
#include <windows.h>
#include <winnt.h>
#include <commdlg.h>
#include <mmsystem.h>
```

```
#include "playmidi.rh"
```

5. 在同一源文件中添加下列代码。函数 BrowseClicked 用来显示标准的打开文件对话框，向用户询问 MIDI 文件名。此函数首先初始化一个 OPENFILENAME 结构，接着调用 Windows API 函数 GetOpenFileName。如果编写程序使用的是某个应用程序框架，或者某些语言如 Delphi、Visual Basic，则可以利用开发环境的功能来显示对话框，获取文件名。

```
static char *FileNameFilter = "Midi Sequences (*.mid) \0*.MID\0All Files
                                              (*.*) \0*.*\0";
/*---------------------------------------------------------------------------*/
/* This function is called when the Browse button is clicked. It            */
/* uses a file open common dialog box to get the name of a MIDI             */
/* sequence file. If a valid file name is returned, the function            */
/* sets the ID_FILENAME static text to the name of the file. If the         */
/* user chooses Cancel, the previous string is retained (which may          */
/* be an emtpy string).                                                     */
/*---------------------------------------------------------------------------*/
void BrowseClicked (HWND hWnd)
{
    char                fileName [255];
    OPENFILENAME        ofStruct;
    // Get the previous file name, if any.
    fileName [0] = 0;
    GetDlgItemText (hWnd, ID_FILENAME, fileName, sizeof (fileName));
    // Initialise the open file structure.
    ofStruct.lStructSize = sizeof (OPENFILENAME);
    ofStruct.hwndOwner = hWnd;
    ofStruct.hInstance = 0;
    ofStruct.lpstrFilter = (LPSTR) FileNameFilter;
    ofStruct.lpstrCustomFilter = NULL;
    ofStruct.nMaxCustFilter = 0;
    ofStruct.nFilterIndex = 1;
    // Stores the result in this variable.
    ofStruct.lpstrFile = fileName;
    ofStruct.nMaxFile = sizeof (fileName);
    ofStruct.lpstrFileTitle = NULL;
    ofStruct.nMaxFileTitle = 0;
    ofStruct.lpstrInitialDir = NULL;
    ofStruct.lpstrTitle = "Choose a MIDI sequence file";
    ofStruct.Flags = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
            OFN_PATHMUSTEXIST;
    ofStruct.nFileOffset = 0;
    ofStruct.nFileExtension = 0;
    ofStruct.lpstrDefExt = "*";
```

```
ofStruct.lCustData = NULL;
ofStruct.lpfnHook = NULL;
ofStruct.lpTemplateName = NULL;
/* Call the common dialog routine, then store the new
     file name if the user clicks OK, and enable or disable
     the Play button depending upon whether there is a file
     currently specified. */
if (GetOpenFileName (&ofStruct))
{
     SetDlgItemText (hWnd, ID_FILENAME, fileName);
EnableWindow (GetDlgItem (hWnd, ID_PLAY), TRUE);
}
else
     if (fileName [0] == 0)
          EnableWindow (GetDlgItem (hWnd, ID_PLAY), FALSE);
}
```

6. 在源文件中添加函数 PlayClicked，此函数是播放 MIDI 文件的核心代码。首先调用函数 mciSendString 来打开指定的文件，选项 “type sequencer” 用来告诉 MCI 此文件是 MIDI 文件，应该使用音序器来播放，在文件没有 .MID 后缀时这是非常需要的。当设备打开后，此函数再一次调用函数 mciSendString，命令为 “play myseq notify”，从而开始了音乐播放，并且立即返回例子程序。当音乐播放结束时，多媒体系统将发送消息 MM_MCINOTIFY 给例子程序的窗口。

```
/*-----------------------------------------------------------------*/
/* This function uses the MIDI sequencer device to open the        */
/* specified file and play it. By specifying the notify keyword,   */
/* control is returned to the application as soon as the sequence */
/* begins playing, and the application will be notified when it is */
/* complete.                                                       */
/*-----------------------------------------------------------------*/
void PlayClicked (HWND hWnd)
{
     char              inBuf [300], outBuf [60], fileName [255];
     MCIERROR          mciError;
     /* Retrieve the file name from the static text control.
          The fileName must be valid otherwise the Play button
          would not be enabled. */
     GetDlgItemText (hWnd, ID_FILENAME, fileName, sizeof (fileName));
     // Open the device.
     wsprintf (inBuf,"open %s type sequencer alias myseq", fileName);
     mciError = mciSendString (inBuf, outBuf, sizeof (outBuf), NULL);
     if (mciError == 0)
     {
```

```
        /* Play the MIDI file. Using the word notify
           tells the MCI interface to send a notification
           message to the specified window when done. */
        mciError = mciSendString ("play myseq notify", NULL, 0, hWnd);
        if (mciError == 0)
        {
            // Enable the Stop button and gray the Play button.
                EnableWindow (GetDlgItem (hWnd, ID_PLAY), FALSE);
                EnableWindow (GetDlgItem (hWnd, ID_STOP), TRUE);
            }
    else // Close the device if playing failed.
            mciSendString ("close myseq", NULL, 0, NULL);
    }
    // Display any error messages.
    if (mciError != 0)
    {
        mciGetErrorString (mciError, inBuf, sizeof (inBuf));
    MessageBox (hWnd, inBuf,"Play MIDI", MB_ICONSTOP | MB_OK);
    }
}
```

7. 添加下面的函数 MIDINotify。当对话框接收到来自 Windows 的消息 MM_MCINOTIFY 时，在对话框的消息处理函数中将调用此函数。此函数激活按钮 Play，按钮 Stop 变灰，并关闭 MIDI 设备。

```
/*----------------------------------------------------------------------*/
/* This function handles the message sent to the dialog when the        */
/* sequence is finished playing.                                         */
/*----------------------------------------------------------------------*/
void MIDINotify (HWND hWnd)
{
    // Enable the Play button and gray the Stop button.
    EnableWindow (GetDlgItem (hWnd, ID_PLAY), TRUE);
    EnableWindow (GetDlgItem (hWnd, ID_STOP), FALSE);
    // Close the device.
    mciSendString ("close myseq", NULL, 0, NULL);
}
```

8. 在源文件中添加函数 StopClicked，当用户点击按钮 Stop 时将调用此函数。由于只有在 MIDI 文件播放时按钮 Stop 才被激活，所以也只有在此时例子程序才能接收到中止消息，当用户点击按钮 Close 关闭例子程序时，此函数也将被调用，这样，就能确保例子程序结束时释放 MIDI 设备。为了响应此消息，此函数使用 API 函数 mciSendString 发送一个中止消息给音序器，为此，多媒体系统将中止 MIDI 音乐文件的播放，并发送一个通知消息给对话框窗口。

```
/*----------------------------------------------------------------------*/
/* This function is called when the user clicks the Stop button, and    */
```

```
/* also when the application is about to shut down. It checks to see       */
/* if the MIDI device is still playing, and if so, stops it.                 */
/*---------------------------------------------------------------------------*/
void StopClicked (HWND hWnd)
{
    HWND          hWndStop;
    MCIERROR        mciError;
    char          buf [100];
    hWndStop = GetDlgItem (hWnd, ID_STOP);
    if ( (hWndStop) && (IsWindowEnabled (hWndStop)))
    {
        // Tell MCI to stop playing the sequence.
        mciError = mciSendString ("stop myseq", NULL, 0, NULL);
        // Display any error message if it failed.
        if (mciError ! = 0)
        {
        mciGetErrorString (mciError, buf, sizeof (buf));
        MessageBox (hWnd, buf,"Play MIDI", MB_ICONSTOP | MB_OK);
        }
    }
}
```

9. 现在为对话框,同样也是为例子程序,添加主消息处理函数,在源文件 PLAYMIDI.C 的末尾添加下列代码。为了响应消息 WM_INITDIALOG,此函数初始化对话框,设置文件名为空,并禁止按钮 Play 和 Stop。为了响应命令消息 ID_BROWSER,此函数调用前面刚添加的函数 BrowseClicked,让用户选取要播放的文件,并激活按钮 Play。为了响应命令消息 ID_PLAY,此函数调用函数 PlayClicked 来开始播放选定的 MIDI 文件,并激活按钮 Stop。为了响应命令消息 ID_STOP,此函数调用函数 StopClicked 来中止 MIDI 设备的播放。这些都是 WM_COMMAND 命令消息,并不异乎寻常。

在此函数处理的消息中最有意思的消息是 MM_MCINOTIFY。由于在函数 PlayClicked 中开始播放音乐时使用了关键字"notify",所以当 MIDI 设备结束音乐的播放时将发送此消息给对话框窗口。此消息的参数 WPARAM 用来说明消息的原因(为什么停止播放的原因),以及参数 LPARAM 给出发送此消息的设备的设备 ID,尽管本例子程序没有用到设备 ID,但是对于支持几个模拟多媒体设备的应用程序则是有用的(例如:同时使用 .WAV 声音和 MIDI)。

表 12-3 列出了由参数 WPARAM 返回的可能原因。

表 12-3 发送消息 MM_MCINOTIFY 的原因

标志	含义
MCI_NOTIFY_ABORTED	设备接收到某个命令,使得当前的回调情况不再发生。如果新命令中断了当前命令,并且也要求通知,则可能发生此种情况
MCI_NOTIFY_SUCCESSFUL	成功地结束播放

（续）

标志	含义
MCI _ NOTIFY _ SUPERSEDED	设备接收到另外的命令，此命令设置了标志 “notify”
MCI _ NOTIFY _ FAILTURE	设备执行命令时发生错误

```
/* ------------------------------------------------------------------------ */
/* This function handles the messages sent to the dialog. When the          */
/* dialog receives the IDCANCEL command, the dialog is closed.              */
/* ------------------------------------------------------------------------ */
BOOL CALLBACK HandleDialog (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            // Set the initial state of the controls.
            SetDlgItemText (hWnd, ID_FILENAME,"");
            EnableWindow (GetDlgItem (hWnd, ID_PLAY), FALSE);
            EnableWindow (GetDlgItem (hWnd, ID_STOP), FALSE);
            return TRUE;
        case WM_COMMAND:
            switch (wParam)
            {
                case IDCANCEL:
                    DestroyWindow (hWnd);
                    return TRUE;
                case ID_BROWSE:
                    BrowseClicked (hWnd);
                    return TRUE;
                case ID_PLAY:
                    PlayClicked (hWnd);
                    return TRUE;
                case ID_STOP:
                    StopClicked (hWnd);
                    return TRUE;
                default:
                    break;
            }
            break;
        case MM_MCINOTIFY:
            MIDINotify (hWnd);
            break;
```

```
        case WM_DESTROY:
            StopClicked (hWnd);
            PostQuitMessage (0);
            return TRUE;
    }
    return FALSE;
}
```

10. 最后，在源文件中添加下面的函数 WinMain。当 Windows 运行此例子程序时调用此函数，此函数创建对话框，并循环处理消息直到例子程序结束。

```
/*-----------------------------------------------------------------------*/
/* Our main function -- set up and run the application.                  */
/*-----------------------------------------------------------------------*/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    MSG        msg;
    HWND       hWnd;
    if ( (hWnd = CreateDialog (hInstance, MAKEINTRESOURCE (IDD_PLAYMIDI),
                    NULL, HandleDialog)) == NULL)
        return FALSE;
    while (GetMessage (&msg, NULL, NULL, NULL))
        if (! IsDialogMessage (hWnd, &msg))
        {
            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    return msg.wParam;
}
```

注释

通过本节的例子程序可以看到：利用 API 函数 mciSendCommand 执行适当的命令可以完成 MIDI 文件的播放，另外，在应用程序中利用消息 MM_MCINOTIFY 和标志 “notify” 可以实现音乐的后台播放。显然，利用此技巧可以实现许多有用的应用程序，一个最好的例子是实现游戏的后台音乐。在游戏开始时初始化 MIDI 设备，发送命令 “play” 来开始音乐的播放，当用户退出游戏时或结束某一关时则发送命令 “stop” 和 “close”。通过响应消息 MM_MCINOTIFY，可以在每次 MIDI 音乐播放结束时再重新开始。

第 13 章　有关窗口的应用

在 Windows 95 API 的所有组件中，没有哪一个比窗口更重要。窗口是所有控制、图形以及与用户间进行交互的基本构造块。在 Windows 95 中，窗口被用于应用程序的视图、对话框、消息框以及控制等。

在这一章里，将介绍 Windows 95 中有关窗口与窗口系统的一些较有意义的用法。通过利用表 13-1 中所列出的函数，将讨论如何创建屏幕保护、如何创建无标题的窗口以及在窗口显示或处于极小化时的情况下如何改变窗口的表现方式。

1. 如何编写屏幕保护程序

本节中，将讨论利用 Visual C++编写屏幕保护程序所需做的必要工作以及如何利用 Windows API 来实现屏幕保护，还可以看到作为一种屏幕保护方式的文本占据屏幕时的情形。

2. 如何创建无标题条的窗口

具有简单边框的窗口在一些情况下是十分有用的。本节中，将讨论如何在对话框中使用只具有简单边框而无标题条的窗口来显示文本，此窗口作为对话框的子窗口，并以一个具有多行、多种字体、多种颜色的“超”静态文本域的方式进行工作。

3. 如何保持一个窗口在所有其他窗口的上面

对于许多实用程序来说，使一个窗口“上浮”在桌面上的所有其他窗口之上的能力，对于为用户提供一个稳定的信息来源是至关重要的。本节中，将介绍如何使一个窗口总保持在所有其他窗口的上面（即便用户把另一个窗口放在它的“上面”）。

4. 如何把一个窗口移到所有其他窗口的下面

同前一节相似，使一个窗口移到其他窗口的“下面”也是经常需要用到的一种做法。对话框中的面板就应具有这一功能，另外，当一个不应该被用户看到的“供应商”的应用程序运行时，也应该把此窗口放到其他窗口的下面。这一任务可以通过把窗口设置在显示顺序的底部来完成。

5. 如何改变鼠标光标的形状

当在 Windows 中工作时，向用户表明应用程序的工作状态经常是必不可少的。在 Windows 3.x 中众所周知的“漏斗”形的鼠标光标就是具有此功能的例子，当然，也可以把显示的光标利用中间有一斜杠的圆圈来代替，以用来表明在某处不能执行某种特定的操作。本节中，将看到预定义的鼠标光标的形状以及如何创建自己喜欢的光标形状。

6. 如何改变应用程序窗口极小化时的图标

在 Windows 3.x 中，当应用程序的主窗口被极小化后，显示出的图标用来向用户表明应用程序的类型以及它所处于的状态。在 Windows 95 中，应用程序的标题在任务条中表明了这一点。这里，我们将讨论如何修改窗口被极小化时所显示的图标，该图标也将做为应用程序的图标在 Windows 95 的 Explorer 窗口中显示。

7. 如何装入另一应用程序的图标

当需要在应用程序中利用图标时，能够从其他应用程序中“挪用”图标以便在自己应用程序中运用是一件比较好的事情，这样，不仅节省了重建它们所需的开销而且还为用户提供了一致的界面。本节中，将讨论如何从系统的其他应用程序中提取图标以便在自己的程序中运用。

8. 如何移动和缩放应用程序中的窗口

当在应用程序中需要利用可变大小的表单或对话框时，通常需要缩放和移动窗口中的控制，以便不管用户如何扩展窗口，总能使窗口的显示保持一致。本节中，将利用 Windows API 的强大功能，使控制适应于其父窗口的变化而变化从而被改变大小或移动到新的位置。

表 13-1 列出了本章中用到的 Windows 95 API 函数。

表 13-1　第 13 章中用到的 Windows 95 API 函数

GetClientRect	SelectObject	PatBlt
GetSystemMetrics	GetDC	SetTextColor
SetBkColor	TextOut	ReleaseDC
InvalidateRect	CreateEx	SetCursor
SetTimer	GetCursorPos	PostMessage
SetCursorPos	KillTimer	GetWindowRect
Create	Create WindowEx	CreateFont
SetWindowPos	ShowWindow	DestroyCursor
SetClassWord	LoadCursor	LoadIcon
DestroyIcon	SetClassLong	SetWindowText
SendDlgItemMessage	MoveWindow	

13.1 编写屏幕保护程序

问题

在想要编写屏幕保护程序时，虽然知道有关的基本过程（封锁屏幕，直到按下任一键后重新显示屏幕），但并不知道创建保护程序的细节性问题，也不知道如何通过 Windows 来调用所创建的屏幕保护程序。那么，如何利用 API 以及 MFC 来创建屏幕保护程序呢?

方法

利用 MFC 编写屏幕保护程序只需要掌握几个关键步骤。首先，需要创建一个占据整个屏幕并且无边框或标题的主窗口，关于这一点可以通过调用 API 函数 CreateWindow 来完成。其次，需要知道如何使用特定的颜色来封锁屏幕，这一步，可以通过处理窗口的 API 消息 WM_ERASEBKGND 并把窗口“涂”成黑色来完成。最后，需要做一些有关捕捉用户输入的事情以便当用户按下任意键或移动鼠标时来捕捉用户的输入。

在此屏幕保护程序例子中（此例子显然还不太完善），程序将在屏幕的随机位置不断地绘制字符串，直到用户按下任一键或移动鼠标为止，当绘到第一百个字符串时，屏幕将被刷新以免屏幕显得太难看。

步骤

打开与运行按照如下步骤生成的例子程序。利用 Debug|Options 屏幕把应用程序的命令行参数设置为“—s”，这表明此应用程序将作为一个屏幕保护程序运行。运行此应用程序，便

可以看到在屏幕的随机位置上绘出的文本，即类似于图 13-1 中所显示的屏幕，当对所显示的结果感到满意时，按下任意键或移动鼠标将结束此保护程序的运行。

为了编写基于此例子的屏幕保护程序，请按照如下步骤进行。

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH131.MAK。在 AppWizard 中选择按钮 Options，并关闭 Multiple Document Interface、Initial Toolbar、Print Preview、Custom VBX 控制以及 Context Senstive Help。

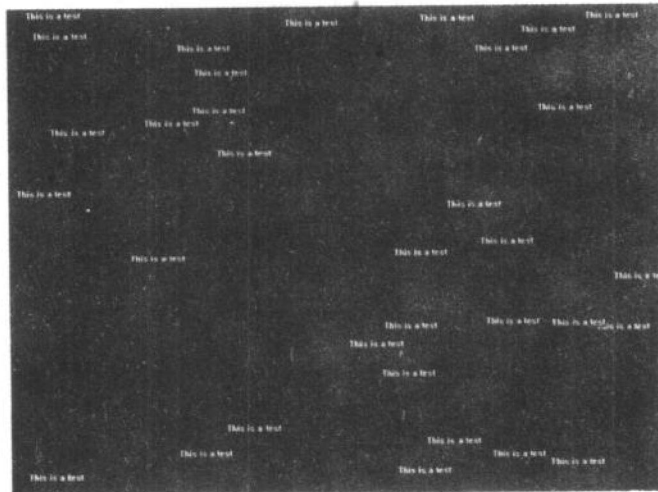


图 13-1　运行中的屏幕保护程序

2. 进入 ClassWizard，选择按钮 Add Class 添加新类 CScreenSaverWnd，并把类 CWnd 作为此类的基类。

3. 在 ClassWizard 中，从对象列表中选择对象 CScreenSaverWnd，从消息列表中选择消息 WM_ERASEBKGND，点击按钮 Add Function 添加新函数 OnEraseBkgnd。在 CScreenSaverWnd 的函数 OnEraseBkgnd 中输入下面的代码。

```
BOOL CScreenSaverWnd:: OnEraseBkgnd (CDC * pDC)
{
    COLORREF rgbBkColor = RGB (0, 0, 0);
    //
    // Erase only the area needed
    //
    CRect rect;
    GetClientRect (&rect) ;
    //
    // Make a brush to erase the background.
    //
    CBrush NewBrush (rgbBkColor) ;
    pDC->SetBrushOrg (0, 0) ;
    CBrush * pOldBrush = (CBrush * ) pDC->SelectObject (&NewBrush);
```

```
    //
    // Paint the Background....
    //
    pDC->PatBlt (rect.left, rect.top, rect.Width (), rect.Height (), PATCOPY);
    pDC->SelectObject (pOldBrush);
    return TRUE;
}
```

4. 在 ClassWizard 中，从对象列表中选择对象 CScreenSaverWnd，从消息列表中选择消息 WM_TIMER，点击按钮 Add Function 添加新函数 OnTimer。在 CScreenSaverWnd 的方法 OnTimer 中输入下面的代码。

```
void CScreenSaverWnd:: OnTimer (UINT nIDEvent)
{
    static int counter = 0;
    int x = rand () % :: GetSystemMetrics ( SM_CXSCREEN );
    int y = rand () % :: GetSystemMetrics ( SM_CYSCREEN );
    // Draw a new text string at the random coordinates.
    CDC *dc = GetDC ();
    dc->SetTextColor ( RGB (255, 255, 255) );
    dc->SetBkColor ( RGB (0, 0, 0) );
    dc->TextOut ( x, y, "This is a test", strlen ("This is a test"));
    ReleaseDC (dc);
    // Allow 100 text messages to be displayed. After that clear the screen
    counter ++;
    if ( counter == 100 ) {
        counter = 0;
    InvalidateRect (NULL);
    }
    CWnd:: OnTimer (nIDEvent);
}
```

5. 把下面的方法添加到源文件底部的类 CScreenSaver 中。

```
BOOL CScreenSaverWnd:: Create ()
{
    const char * pszClassName
        = AfxRegisterWndClass (CS_HREDRAW|CS_VREDRAW|
                               CS_SAVEBITS|CS_DBLCLKS);;
    return CWnd:: CreateEx (WS_EX_TOPMOST,
                            pszClassName,
                            "", WS_POPUP | WS_VISIBLE,
                            0, 0,
                            :: GetSystemMetrics (SM_CXSCREEN),
                            :: GetSystemMetrics (SM_CYSCREEN), NULL,
                            NULL);
```

```
}
void CScreenSaverWnd:: PostNcDestroy ()
{
    delete this;
}
LRESULT CScreenSaverWnd:: WindowProc (UINT nMsg, WPARAM wParam, LPARAM lParam)
{
    static BOOL        fHere = FALSE;
    static POINT       ptLast;
    POINT              ptCursor, ptCheck;
    switch (nMsg) {
    case WM_SYSCOMMAND:
        if ( (wParam == SC_SCREENSAVE) || (wParam == SC_CLOSE)) {
            return FALSE;
        }
        break;
    case WM_DESTROY:
        PostQuitMessage (0);
        break;
    case WM_SETCURSOR:
        SetCursor (NULL);
        break;
    case WM_NCACTIVATE:
        idTimer = SetTimer (1, 100, NULL);
        if (wParam == FALSE) {
            return FALSE;
        }
        break;
    case WM_ACTIVATE:
    case WM_ACTIVATEAPP:
        if (wParam != FALSE) break;
        // Only fall through if we are losing the focus...
    case WM_MOUSEMOVE:
        if (! fHere) {
            GetCursorPos (&ptLast);
            fHere = TRUE;
            } else {
            GetCursorPos (&ptCheck);
            if (ptCursor.x = ptCheck.x - ptLast.x) {
              if (ptCursor.x < 0) ptCursor.x *= -1;
            }
            if (ptCursor.y = ptCheck.y - ptLast.y) {
              if (ptCursor.y < 0) ptCursor.y *= -1;
```

```
            }
            if ( (ptCursor.x - ptCursor.y) > MAXMOVE) {
              PostMessage (WM_CLOSE, 0, 0l);
            }
                }
           break;
    case WM_LBUTTONDOWN:
    case WM_MBUTTONDOWN:
    case WM_RBUTTONDOWN:
        GetCursorPos (&ptCursor);
        ptCursor.x ++;
        ptCursor.y ++;
        SetCursorPos (ptCursor.x, ptCursor.y);
        GetCursorPos (&ptCheck);
        if (ptCheck.x != ptCursor.x && ptCheck.y != ptCursor.y)
        ptCursor.x -= 2;
        ptCursor.y -= 2;
        SetCursorPos (ptCursor.x, ptCursor.y);
    case WM_KEYDOWN:
    case WM_SYSKEYDOWN:
//          PostMessage (WM_CLOSE, 0, 0l);
        break;
    }
    return CWnd::WindowProc (nMsg, wParam, lParam);
}
```

6. 接着，把下面几行添加到类 CScreenSaverWnd 的头文件 SCREENSA.H 中。

```
const MAXMOVE = 100;
private:
    UNIT idTimer;
public:
BOOL Create ();
protected:
    virtual LRESULT WindowProc (UNIT nMsg, WPARAM wParam, LPARAM lParam);
    virtual void PostNcDestroy ();
```

7. 利用下面的代码修改应用程序对象 CCh131App 的方法 InitInstance。

```
BOOL CCh131App:: InitInstance ()
{
    if ( ( (! lstrcmpi (m_lpCmdLine,"/s") || ! lstrcmpi (m_lpCmdLine,"-s")) || ! lstrcmpi (m_lpCmd-
Line,"s"))) {
        // Run as screen saver.
        CScreenSaverWnd * pWnd = new CScreenSaverWnd;
        pWnd->Create ();
       m_pMainWnd = pWnd;
```

```
        return TRUE;
    } else {
        // Configuration option.
        return FALSE;
    }
}
```

8. 把下面一行代码添加到应用程序对象源文件 CH131.CPP 的顶端。

```
#include "screensa.h"
```

9. 如果愿意的话，此时可以把 MAINFRM.CPP、MAINFRM.H 以及文档/视图文件从项目文件中删除，因为此时这些文件已经不再有用。

10. 利用下面一行代码修改此例子程序的定义文件 CH131.DEF。

```
DESCRIPTION 'SCRNSAVE : My Screen Saver'
```

11. 创建可执行文件。为了运行此可执行文件并允许其作为屏幕保护程序运行，必须把它重新命名为.SCR 文件（CH131.SCR）并把它移入到系统的 Windows 目录中。控制板将查找它（通过列在定义文件的 DESCRIPTION 行中的名字）并允许选择它作为当前的屏幕保护程序。

用法

Windows 屏幕保护程序只是其扩展名重新被命名为.SCR 的可执行文件，它接受两种类型的命令行参数：运行命令参数（-s）与配置命令参数（-c）。这里的屏幕保护程序省略了配置命令参数，不过，在封锁屏幕前可以很容易通过增加一个对话框来配置诸如所要显示的字符串以及所要显示字符串的数目等设置。

当 Windows 启动屏幕保护程序时，它通过发送命令行参数-s 来启动，然后就按照标准的窗口规则运行。例子程序的方法 InitInstance 创建了类 CScreenSaverWnd 的一个实例并把主窗口变量指向此实例，例子程序的整个逻辑部分单独地驻留在类 CScreenSaverWnd 中。

当创建类 CScreenSaverWnd 时，便调用方法 Create，此方法利用 Windows API 函数 CreateEx 创建一个新窗口。函数 CreateEx 允许在桌面上创建一个没有父窗口控制的主层窗口，这里创建的窗口无边框或标题，窗口的大小通过利用 API 函数 GetSystemMetrics 被设置成整个屏幕的大小。

新建的窗口利用窗口函数 WindowProc 来处理所有发送给此窗口的消息，它只处理少数几个消息，主要有移动鼠标、按下鼠标按钮的消息以及键盘通知消息，当稍一移动鼠标或按下任一键时，窗口就会关闭。

调用类的方法 OnEraseBkgnd 来响应 Windows 的消息 WM_ERASEBKGND，此消息表明窗口应该刷新自己的背景了，这里，只是利用 GDI 函数 SelectObject（来获取与设置一个新的颜色刷）与函数 PalBlt 把背景刷新为黑色。

调用函数 OnTimer 来响应在 WindowProc 的 NcActivate 情形下创建计时器，此函数每隔 50ms 被调用一次，它只是利用随机数字发生器在屏幕上确定一点，然后把字符串输出到屏幕的此位置处，字符串是通过利用函数 TextOut 显示到屏幕上的。需注意的是：在函数 TextOut 调用之前，需利用 GDI 函数 SetTextColor 与 SetBkColor 来设置前景与背景颜色。

最后，方法 OnTimer 检查是否已显示了 100 个字符串，如果是的话，便通过调用 API 函数 InvalidateRect 来刷新窗口并重新开始显示。

注释

本节介绍的如何创建屏幕保护程序的例子是一个非常简单但却十分完整的例子，它虽然没有做出诸如分形几何图形或漂亮对象等一些比较新奇的东西，但无论从哪方面看，它都是一个完整的屏幕保护程序。

为了使做为屏幕保护的图形显得生动活泼，可以在程序中增加处理多种字体、文本的旋转以及变化被选择的字符串等功能；也可以相应于配置命令（—c），增加对屏幕保护程序的配置功能，从而使屏幕保护程序更完美。

13.2 创建无标题条的窗口

问题

有时需要创建无标题条或无标题的窗口置于对话框中，它主要用来提供能够显示多行文本以及多种字体的静态文本窗口。

那么，如何创建这样的窗口以及如何把它置放于对话框中所想要的位置呢？

方法

本节中，将讨论如何改变 Windows API 函数 CreateWindowEX 中的参数，从而来创建没有标题或标题条的窗口。这里，将利用对话框类来管理此子窗口，然后利用此子窗口来显示不同字体的文本。

在设计时，为了在对话框中利用子窗口，在资源编辑器（无论选择的是何种对话框编辑器）中设计对话框类模板时，需要为它留出空间。本例子程序中，创建了一个静态文本域“placeholder”，当建立新窗口时，此文本域将用来确定此窗口的位置。

步骤

打开与运行按照如下步骤生成的例子程序 CH132.MAK。从主菜单 Dialog 中选择菜单项 Dialog with captionless window，这时将会显示如图 13-2 所示的对话框，可以看到，在对话框中央的静态文本域中显示着三行不同风格与大小的字体。

为了在例子程序中实现上述功能，请按如下步骤进行。

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件 CH132.MAK。利用资源编辑器，为此例子程序创建一个标题为 Test Captionless Window Dialog 的新的对话框。

2. 在对话框的顶端，添加标题为 Multi _ Line Multi _ Font Text Box 的静态文本域。在紧接此静态文本域的下面，添加在水平和垂直方向上较大尺寸的第二个文本域，设定此静态文本域的标识符为 ID _ PLACEHOLDER，并且无标题。

3. 进入 ClassWizard，为刚创建的对话框模板生成一个新的对话框类，把此对话框类命名为 CNoCaptionDlg。从 ClassWizard 中的对象列表中选择对象 CNoCaptionDlg，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Class，把下面的代码添加到 CNoCaptionDlg 的方法 OnInitDialog 中。

```
BOOL CNoCaptionDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    // Create a new window for this dialog in place of the "placeholder"
    // static text field
```

```
    CWnd * w = GetDlgItem (IDC_PLACEHOLDER);
    CRect r, dr;
    w->GetWindowRect ( &r );
    GetWindowRect ( &dr );
    r.top -= dr.top;
    r.left -= dr.left;
    r.bottom -= dr.top;
    r.right -= dr.left;
    wnd = new CFancyTextWnd;
    wnd->Create( NULL, "", WS_VISIBLE | WS_BORDER | WS_CHILD, r, this, 1010, NULL );
    wnd->ShowWindow (SW_SHOW);
    return TRUE; // return TRUE unless you set the focus to a control
}
```

4. 把下面的代码添加到类 CNoCaptionDlg 的析构函数中。

```
CNoCaptionDlg:: ~CNoCaptionDlg ( void )
{
delete wnd;
}
```

5. 最后，把下面几行代码添加到 CNoCaptionDlg 的头文件 NOCAPTION. H 中。这里，只添加标记为黑体的代码。

```
#include "fancytex.h"
class CNoCaptionDlg : public CDialog
{
// Construction
public:
    CNoCaptionDlg (CWnd * pParent = NULL);    // standard constructor
private:
    CFancyTextWnd * wnd;
public:
~CNoCaptionDlg ();
```

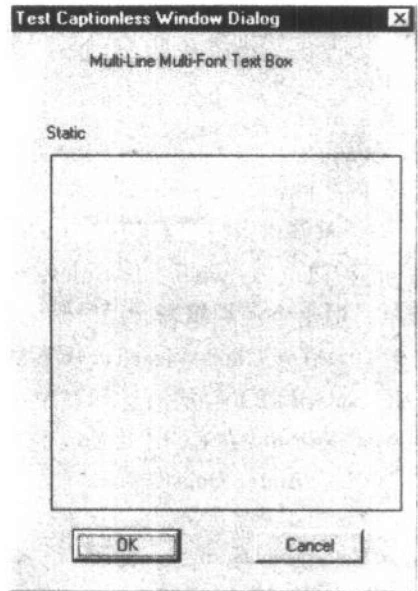


图 13-2 嵌入无标题窗口的对话框

6. 重新进入 ClassWizard，选择按钮 Add Class 添加新类。把此新类命名为 CFancyTextWnd，并把其基类设定为 CWnd，从对象列表中选择类 CFancyTextWnd，从消息列表中选择消息 WM_PAINT，在 CFancyTextWnd 的方法 OnPaint 中添加下面的代码。

```
void CFancyTextWnd:: OnPaint ()
{
    CPaintDC dc (this); // device context for painting
    dc.SetTextColor ( RGB (255, 255, 255) );
    dc.SetBkColor ( GetSysColor (COLOR_BTNFACE) );
    dc.SelectObject ( &f1 );
```

```
    dc.TextOut ( 0, 0, "This is a test", strlen ("This is a test"));
    dc.SelectObject ( &f2 );
    dc.TextOut ( 0, 24, "This is a test", strlen ("This is a test"));
    dc.SelectObject ( &f3 );
    dc.TextOut ( 0, 40, "This is a test", strlen ("This is a test"));
}
```

7. 接着，把下面代码添加到类 CFancyTextWnd 的构造函数 CFancyTextWnd::CFancy-TextWnd 中。

```
CFancyTextWnd:: CFancyTextWnd ()
{
    f1.CreateFont ( -14, 0, 0, 0, FW_NORMAL, 0, 0, 0, 0, 0, 0, 0, 0, NULL );
    f2.CreateFont ( -16, 0, 0, 0, FW_NORMAL, TRUE, 0, 0, 0, 0, 0, 0, 0, NULL );
    f3.CreateFont ( -18, 0, 0, 0, FW_BOLD, 0, 0, 0, 0, 0, 0, 0, 0, NULL );
    }
```

8. 在此类的头文件 FANCYTEX.H 中添加下面的说明。

```
private:
    CFont f1;
    CFont f2;
    CFont F3;
```

9. 重新进入资源编辑器，添加标题为 Dialogs 的新的主层菜单。在主菜单 Dialogs 中，添加标题为 Dialog with captionless window、标识符为 ID_NO_CAPTION 的菜单项，保存此菜单，退出资源编辑器。

10. 进入 ClassWizard，从下拉列表中选择对象 CCh132App。从对象列表中选择对象 ID_NO_CAPTION，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnNoCaption。在 CCh132App 的方法 OnNoCaption 中输入下面的代码。

```
void CCh132App:: OnNoCaption ()
{
    CNoCaptionDlg dlg;
    dlg.DoModal ();
    }
```

11. 在源文件 CH132.CPP 的顶端添加下面的 include 文件行。

```
#include "nocaptio.h"
```

12. 编译与运行此例子程序。

用法

当用户通过从菜单中选择菜单项 Dialog with captionless window 把窗口类实例化时，对话框对象就通过调用类 CNoCaptionDlg 的方法 OnInitDialog 被建立。

OnInitDialog 方法通过调用类 CWnd 的方法 Create 来创建新窗口。Create 方法将直接调用 Windows API 函数 CreateWindowEx，此函数创建一个新窗口的句柄并把新创建的窗口置于屏幕上，Create 方法通知 Windows 系统将把什么样的风格参数赋值给窗口。一般情况下，窗口的风格包括 WS_CAPTION、WS_CHILD、WS_VISIBLE 以及 WS_BORDER，但在本例子程序中，由于省略了 WS_CAPTION 风格位，所以生成的是无标题或无标题条的窗口。

13.3 保持一个窗口在所有其他窗口的上面

问题

有时，应用程序的主窗口需要一直保持在桌面的所有其他窗口之上，比如当程序是一个显示用户信息的简单实用程序时，如果它不能总保持在其他窗口的上面，用户就看不到所需的信息，这样其意义也就不大了。

那么，如何使主窗口总保持在窗口显示栈的最顶端呢？

方法

Windows 95 同其他版本的 Windows 一样，有一个“Z 次序”的概念，它用来表明窗口在屏幕上显示的次序。Z 次序最低的窗口显示在栈的底部，Z 次序最高的窗口显示在栈的顶端。

API 函数 SetWindowPos 能够用来改变当前桌面上显示的所有窗口的 Z 次序。本节中，将讨论利用 API 函数 SetWindowPos 使某窗口总是保持在所有其他窗口之上的简单方法。

步骤

打开与运行按照如下步骤生成的例子程序 CH133.MAK，则显示如图 13-3 所示的应用程序窗口，在此窗口中，选择主菜单 Dialog 中的菜单项 Always on top window，此时，如果把桌面上的其他窗口移动到此窗口之上，将会发现所移动的窗口总保持在此窗口的后面，而此例子程序的窗口总是显示在最上面。

为了在例子程序中实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH133.MAK。进入资源编辑器。

2. 添加标题为 Dialog 的新的主菜单。在主菜单 Dialog 上添加标题为 Always on top window、标识符为 ID_TOP_WINDOW 的菜单项。

3. 进入 ClassWizard，为例子程序选择应用程序对象 CCh133App。从对象列表中选择对象 ID_TOP_WINDOW，从消息列表中选择消息 COMMAND，点击 ClassWizard 中的按钮 Add Function 添加新函数。

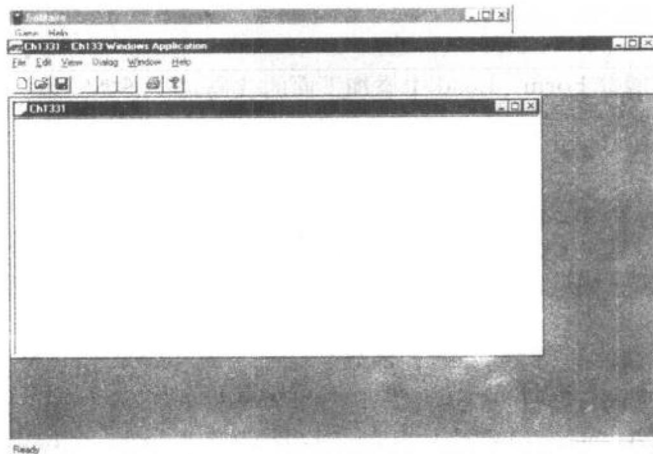


图 13-3 总保持在上面的窗口

4. 把新添加的函数命名为 OnTopWindow。在 CCh133App 的方法 OnTopWindow 中添加下面的代码。

```
void CCh133App:: OnTopWindow ()
{
    :: SetWindowPos(AfxGetMainWnd()->m_hWnd, HWND_TOPMOST,-1,-1,-1,-1, SWP_
NOMOVE | SWP_NOSIZE );
}
```

5. 编译与运行此例子程序。

用法

函数 SetWindowPos 需要七个参数，但在本例中只用了三个。第一个参数用来指定所要改变位置的窗口的句柄，本例中，利用 MFC 的内部函数得到一个指向主窗口的指针，然后利用该指针获取存放此窗口句柄的公有成员变量。第二个参数是窗口位置变量，这里它被设置成了 HWND_TOPMOST，该参数指定一个窗口句柄，此窗口将在该参数所指定的窗口之后显示，这里的特殊参数值 HWND_TOPMOST，用来向 Windows 95 操作系统表明此窗口将总是做为最上层的窗口显示。

接下来的四个参数用来为窗口指定新的位置，由于这里只对窗口的 Z 次序感兴趣，所以不需要这些参数。最后一个参数是一套标志，它用来表明哪一个参数需要用到，哪一个参数应该省略，本例中，设置了标志 SWP_NOMOVE，此标志表明将不利用前两个位置的参数，另外还设置了标志 SWP_NOSIZE，它表明也不利用后两个位置的参数。

注释

下面的步骤将介绍如何在 Visual Basic 中完成同样的事情。

1. 创建新的项目文件或在现有的项目文件中添加一个新的表单，在此表单中添加一个按钮以及其他一些所需的控制(这里添加了一个单选按钮)。

2. 把此按钮的标题设置为 Close，通过双击此按钮在函数 Command1_Click 中添加下面的代码。

```
Private Sub Command1_Click( )
    End
End Sub
```

3. 双击表单，在函数 Form_Load 中添加下面的代码。

```
Private Sub Form_Load( )
    Dim Flags As Integer
    Flags = &H1
    Flags = Flags Or &H2
    Call SetWindowPos(hWnd, -1, 0, 0, 0, 0, Flags)
End Sun
```

4. 把下面几行添加到表单的 general 部分。

```
Private Declare Sun SetWindowPos Lib "User32" (ByVal hWnd As Integer,
ByVal hWndInsertAfter As Integer, ByVal X As Integer, ByVal Y As Integer, ByVal cx As Integer,
ByVal cy As Integer, ByVal wFlags As Integer)
```

5. 运行此例子程序。可以注意到：无论调用什么，表单总是保持在最上面。

13.4 把一个窗口移到所有其他窗口的下面

问题

有时需要把主窗口移到窗口栈的底部，从而使得当其他应用程序正在运行时，此主窗口能够从屏幕上暂时消失。比如，当应用程序创建了另一任务时，希望此任务能够运行在应用程序窗口的前面以便用户只看到该任务的运行情况。

那么，如何把应用程序的窗口移到所有其他窗口的下面呢?

方法

同前一节相同，我们可以利用 API 函数 SetWindowPos 把一个窗口移到窗口“Z 次序”的后面。在上一节中，我们利用了特殊设置 HWND_TOPMOST 确保了窗口总是在所有其他窗口的上面。

但是目前还没有一个真正的相应设置来使得一个窗口总保持在其他窗口的下面。设置 HWND_BOTTOM 只是使窗口初始时显示在下面，但当利用鼠标点击此窗口后，它就会移回到“Z 次序”的上面，不过，没有设置 HWND_TOPMOST 的窗口是决不会移动到有设置 HWND_TOPMOST 的窗口的上面的。

本节中，将讨论函数 SetWindowPos 另一方面的用法，并介绍如何把窗口暂时地移到“Z 次序”栈的底部。

步骤

打开与运行按照如下步骤生成的例子程序 CH134.MAK。选择主菜单 Dialog 中的菜单项 Move Window To Bottom，将会注意到，此窗口立刻落到了所有其他窗口之后，如图 13-4 所示。

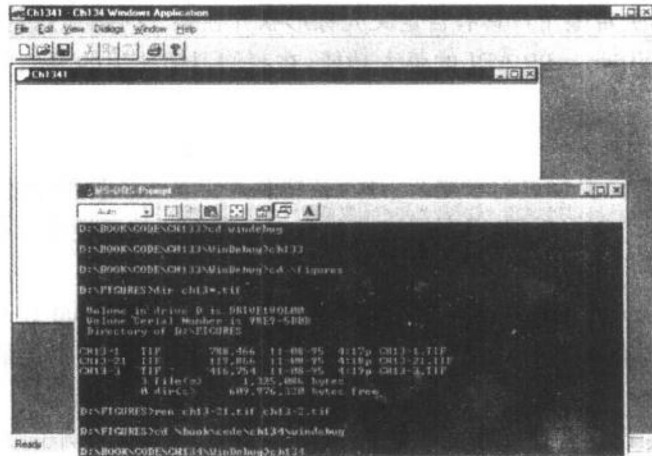


图 13-4 移到后面的窗口

为了在例子程序中实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH134.MAK。进入资源编辑器。

2. 添加标题为 Dialog 的新的主菜单。在主菜单 Dialog 上添加标题为 Move Window To

Bottom、标识符为 ID _ BOTTOM _ WINDOW 的菜单项。

3. 进入 ClassWizard，为例子程序选择应用程序对象 CCh134App。从对象列表中选择对象 ID _ BOTTOM _ WINDOW，从消息列表中选择消息 COMMAND，点击 ClassWizard 中的按钮 Add Function 添加新函数。

4. 把新添加的函数命名为 OnBottomWindow。在 CCh134App 的方法 OnBottomWindow 中添加下面的代码。

```
void CCh134App::OnBottomWindow ()
{
  ::SetWindowPos(AfxGetMainWnd()->m_hWnd, HWND_BOTTOM,-1,-1,-1,-1, SWP_NOMOVE | SWP_NOSIZE );
}
```

5. 编译与运行此例子程序。

用法

当用户选择了菜单命令 Move Window To Bottom 后，例子程序便为此窗口句柄调用 API 函数 SetWindowPos，并在函数中利用标志 HWND _ BOTTOM，此标志表明：该窗口要移动到当前窗口栈的底部。虽然此窗口并不总是在栈的底部，但可以相信，在创建其他窗口或将此窗口引至窗口栈的顶端之前，此窗口将保持在桌面的底部。

13.5 改变鼠标光标的形状

问题

我们通常希望能够根据应用程序的状态来改变鼠标光标的形状，例如，当在执行比较繁重的后台打印作业时，希望鼠标光标以"漏斗"形状显示；当在编辑模式下工作时，希望鼠标光标以"I"形状显示，而且，更希望用户具有自定义光标形状并在某一时刻显示它的能力。

那么，如何利用 Window 95 API 的强大功能，在我们习惯使用的环境中改变光标的形状呢？

方法

这个问题实际做起来要比说起来复杂一些，虽然装入与显示各种类型的光标（包括用户自己定义的光标）是一种平常的任务，但要把这些改变与正在运行的环境结合起来却是一件不太容易的事情。

例如，当利用 Visual C++把光标改变为"一条线"时，由于 Visual C++系统的内部功能，使得当鼠标一被移动，鼠标光标就要变回到它的缺省形状，因此要保持最初打算使用的光标形状就变得十分困难。

不过，Windows 的设计者已经很好地解决了这一问题，在本节的例子中将会看到这一点，他们不仅提供了允许程序员修改屏幕上鼠标光标形状的功能，而且通过给出 Windows 消息，使得无论鼠标何时移动，都允许程序员重新设置光标的形状。

本节中，将讨论 API 函数 LoadCursor 与 SetCursor 以及 Windows 消息 WM _ SETCURSOR。

步骤

打开与运行按照如下步骤生成的例子程序 CH135.MAK。这时，将会看到一个普通的多

文档界面窗口(即 MFC 中的视图),选择主菜单 Cursor 中的任一菜单项,这些选项包括 Hourglass、I—Beam、Cross 以及 User Defined。在选择了这些选项的其中之一后,把鼠标移回到 MDI 窗口中,将会看到鼠标光标变成了从菜单中所选择的形状。MDI 窗口与光标选项菜单如图 13-5 所示。

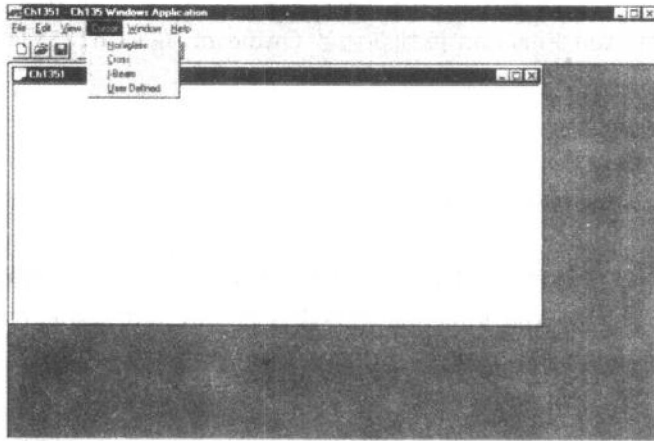


图 13-5　显示鼠标光标选项菜单的 MDI 窗口

为了在例子程序中实现上述功能,请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH135.MAK。进入资源编辑器。

2. 添加标题为 Cursor 的新的主菜单。在此主菜单 Cursor 上添加标题分别为 Hourglass、I—Beam、Cross 以及 User Defined 的菜单项。

3. 设定 Hourglass 菜单项的标识符为 ID _ HOURGLASS、I—Beam 菜单项的标识符为 ID _ IBEAM、Cross 菜单项的标识符为 ID _ CROSS、User Defined 菜单项的标识符为 ID _ USER _ CURSOR。

4. 在资源编辑器中创建新的菜单项。从请求创建资源类型的对话框中选择 Cursor,当显示光标框后,绘制一个自己喜欢的光标,在 Properties 屏中,把此光标的标识符设定为 IDC _ MOUSE _ CRUSOR。

5. 退出资源编辑器,保存新的资源,进入 ClassWizard,从下拉列表中选择 CCh135View。

6. 从对象列表中选择对象 ID _ HOURGLASS,从消息列表中选择消息 COMMAND,点击 ClassWizard 中的按钮 Add Function 添加新函数 OnHourglass,在 CCh135View 的方法 OnHourglass 中添加下面的代码。

```
void CCh135View::OnHourglass( )
{
    curCursor = ::LoadCursor(NULL, IDC_WAIT);
}
```

7. 从对象列表中选择对象 ID _ CROSS,从消息列表中选择消息 COMMAND,点击 ClassWizard 中的按钮 Add Function 添加新函数 OnCross,在 CCh135View 的方法 OnCross

中添加下面的代码。

```
void CCh135View::OnCross( )
{
    curCursor = ::LoadCursor(NULL, IDC_CROSS);
}
```

8. 从对象列表中选择对象 ID_IBEAM,从消息列表中选择消息 COMMAND,点击 ClassWizard 中的按钮 Add Function 添加新函数 OnIbeam,在 CCh135View 的方法 OnIbeam 中添加下面的代码。

```
void CCh135View::OnIbeam( )
{
    curCursor = ::LoadCursor(NULL, IDC_IBEAM);
}
```

9. 从对象列表中选择对象 ID_USER_CURSOR,从消息列表中选择消息 COMMAND,点击 ClassWizard 中的按钮 Add Function 添加新函数 OnUserCursor,在 CCh135View 的方法 OnUserCursor 中添加下面的代码。

```
void CCh135View::OnUserCursor( )
{
    curCursor = hMouseCursor;
}
```

10. 接下来,从对象列表中选择对象 CCh135View,从消息列表中选择消息 WM SETCURSOR,点击按钮 Add Function 添加新函数 OnSetCursor,在 CCh135View 的方法 OnSetCursor 中添加下面的代码。

```
BOOL CCh135View::OnSetCursor(CWnd * pWnd, UINT nHitTest, UINT message )
{
    ::SetCursor(curCursor);
    return TURE;
}
```

11. 在类 CCh135View 的构造函数中添加下面代码。

```
CCh135View::CCh135View( )
{
    //Load the special cursor for our application
    hMouseCursor = ::LoadCursor(AfxGetInstanceHandle( ),
MAKEINTRESOURCE(ID_MOUSE_CURSOR));
    curCursor = ::LoadCursor(NULL, IDC_IBEAM);
}
```

12. 最后,为类 CCh135View 添加析构函数,并在此析构函数中添加下面的代码。

```
CCh135View::~CCh135View( )
{
    ::DestroyCursor(hMouseCursor);
}
```

13. 在类 CCh135View 的头文件 CC135VIEW.H 中添加下面几行代码。

```
private:
    HCURSOR hMouseCursor;
    HCURSOR curCursor;
```

14. 编译与运行此例子程序。

用法

Window 95 API 的函数 LoadCursor 根据给定的实例句柄装入光标资源。如果传递给此函数的句柄是 NULL 或 0,则函数 LoadCursor 将搜寻一个标准的窗口光标,诸如"hourglass"、"cross"或"I-beam"。当查找到光标资源后,光标句柄就被取回从而被应用程序利用。

API 函数 SetCursor 利用函数 LoadCursor 返回的光标句柄在窗口中实际地显示被调用的光标,被设置的光标对任何窗口都有效,即便窗口本身并没有特地设置此光标(这就意味着对整个桌面都起了作用)。当利用函数 LoadCursor 检索到光标的句柄后,函数 SetCursor 便显示此光标。

最后,函数 DestroyCursor 对于应用程序装入的任一光标都是需要的。当在资源编辑器中创建新的光标资源并利用函数 LoadCursor 装入后,在关闭窗口时,必须把此光标资源删除从而释放其占用的内存空间。

当 Windows 95 需要设置当前光标值时,它就向窗口发送一个消息 WM_SETCURSOR,这包括改变光标或移动鼠标。窗口检索此消息并要求重置所要显示的光标形状,在本例中,各个菜单项只是把视图类的成员变量设置成所要显示的光标,然后在方法 OnSetCursor(它是响应消息 WM_SETCURSOR 的方法)中才向用户显示光标。

如果在窗口中不重置 WM_SETCURSOR 消息的处理函数,则当鼠标一移动时,操作系统就改变光标,这就使得光标看起来好象从未改变过似的。

13.6 改变应用程序窗口极小化时的图标

问题

在使用 Windows 3.x 的 SDK 以及 C 程序设计的以往日子里,要改变应用程序极小化时所显示的图标,所需做的只是指定出现在 Windows 注册函数 RegisterWindowClass 中的图标。如果需要创建一个窗口并改变此窗口极小化时的图标,则可以在注册过程中指定 NULL 图标,然后利用消息 WM_PAINT 的处理程序来绘制自己的图标。

然而,随着 MFC、Delphi 以及 Visual Basic 的出现,一些功能似乎从应用程序设计中消失了。利用 MFC,似乎还找不出什么办法能够完成过去几行代码就能完成的任务。如何通过 API 函数改变应用程序的图标呢?

方法

在对新工具失去的功能感到遗憾之前,试想一下过去创建一个窗口并把它显示在屏幕上也需要一百多行的代码。

这些暂且不论,利用 Windows API 依然可能实现所要做的事情。实际上,在注册窗口的过程中,就需要指定极小化窗口时所要显示的图标,而且,当注册窗口类后,在运行时修改图标也是可能的。事实上,这才是在 Windows 95 API 库中的函数 SetClassLong 的整个意义所在。

利用函数 SetClassLong,便可把窗口类的图标改变为极小化时所要显示的图标。

步骤

打开与运行按如下步骤生成的例子程序 CH136. MAK，将会看到一个普通的多文档界面窗口(即 MFC 中的视图)。点击此 MDI 窗口中的极小化按钮把它缩小为图标形式，此时将显示如图 13-6 所示的图标。

为了在例子程序中实现上述功能，请按如下步骤进行。

图 13-6 显示定制图标的极小化 MDI 窗口

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH136. MAK，进入 ClassWizard。

2. 点击按钮 Add Class 添加新类，把新添加的类命名为 CMyFrame 并选择 CMDIChildWnd 做为此新类的基类，接受其他的缺省值，生成新类的定义。

3. 在 ClassWizard 中，从下拉列表中选择类 CMyFrame。从对象列表中选择对象 CMyFrame，从消息列表中选择消息 WM_CREATE，点击按钮 Add Function 添加新函数 OnCreate。在 CMyFrame 的方法 OnCreate 中输入下面的代码。

```
int CMyFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    SetClassLong( m_hWnd, GCL_HICONSM, (LONG)theIcon );
    if (CMDIChildWnd::OnCreate(lpCreateStruct) == -1)
    return -1;
    // TODO: Add your specialized creation code here
    return 0;
}
```

4. 把下面的代码添加到类 CMyFram 的构造函数中。

```
CMyFrame::CMyFrame( )
{
    theIcon = ::LoadIcon(AfxGetInstanceHandle( ), MAKEINTRESOURCE(IDI_ICON1) );
}
```

5. 把下面的代码添加到类 CMyFrame 的析构函数中。

```
CMyFrame::~CMyFrame( )
{
    DestroyIcon(theIcon);
}
```

6. 把下面几行代码添加到类 CMyFrame 的头文件 MYFRAME. H 中。

```
private:
    HICON theIcon;
```

7. 从项目文件列表中选择 CH136. CPP 文件，按照下面代码修改模板实例 CCh136View 的文档模板。

```
    CMultiDocTemplate * pDocTemplate;
    pDocTemplate = new CMultiDocTemplate(
    IDR_CH136TYPE,
```

```
RUNTIME_CLASS(CCh136Doc),
RUNTIME_CLASS(CMyFrame), // standard MDI child frame
RUNTIME_CLASS(CCh136View));
AddDocTemplate(pDocTemplate);
```

8. 把下面一行添加到 CCh136App 的源文件 CH136.CPP 的顶端。

```
#include "myframe.h"
```

9. 在资源编辑器中，创建新的图标资源。绘制窗口极小化时所要显示的图标，把此图标的标识符设定为 IDI_ICON1 并把它保存起来。

10. 编译与运行此例子程序。

用法

当极小化窗口时，Windows 便自动处理显示图标化窗口的工作，它是通过检查窗口的分类词并从窗口结构的附加字节的位置提取图标来完成这一任务的。在此例子中，所要做的只是把分类词设置成自己的图标，此图标即是在窗口极小化时将被显示的图象。

注意，创建自己的框架窗口来实现上述的功能是必要的，在 ClassWizard 中建立框架窗口并通过文档模板与文档/视图系统结构相连接。

注释

需要说明的是：Win32 API 也允许程序员利用消息 WM_SETICON 来完成相同的任务。

13.7 装入另一应用程序的图标

问题

有时在应用程序中需要让用户浏览其他应用程序的图标以便从中选择一种图标用来在自己应用程序中显示，这些都是可能的，因为 Windows 3.x 中的文件管理器以及 Windows 95 的 Windows Explorer 都能够这样做。

利用 MFC 还找不出解决这一问题的方法，那么，利用 Windows 95 API 的功能有完成这一任务的办法吗？如果有的话，又该怎么做呢？

方法

MFC 包含了 Windows API 的大部分函数并为这些函数赋予了大量功能。不过，让所有 API 函数都适合于这些类是绝对不可能的，因此，有些 API 函数就被忽略掉了，另外，还有些函数认为不适合于在类库中使用也被忽略掉了，这里，所需利用的函数就属于后一类。函数 ExtractIcon 便提供了在例子程序中所需要的功能。

利用函数 ExtractIcon，只需要一个驻留在可访问磁盘上的可执行文件、动态连接库或库文件的名字，许多这样的程序、库以及 DLL 等包含了大量可在应用程序中利用的诸如图标之类的资源。

本节中，将探究 Windows API 最大程度能及的范围。因为我们不仅需要知道如何从可执行文件或动态连接库中查找存在的图标，而且还需知道如何把这些图标装入到自己的应用程序并在对话框中显示他们。

步骤

打开与运行按照如下步骤生成的例子程序 CH137.MAK。选择主菜单 Dialogs 中的菜单项 Browse Icons，则显示如图 13-7 所示的对话框，点击按钮 Select File，一个标准 Windows

的 File Open 对话框就会显示，进入到 WINDOWS 子目录，选择一个 DLL 来查看，最好的选择是 MORICONS. DLL 文件。当在 File Open 对话框中选择了此文件后，文件的名字便在对话框顶端的文本域中显示出来，图标的数目也将在对话框中按钮 Select File 下面的文本域中显示出来。

在提示“View Icon Number：”相邻的编辑域中输入 1 到文本域中显示的图标数量之间的任一数值，点击按钮 View，所选择的图标便在下面的编辑域中显示出来，图 13-8 显示的就是从 MORICONS. DLL 文件中提取出的一个图标的例子。

为了实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH137. MAK，进入资源编辑器。

2. 创建新的对话框类。在对话框的顶端添加标题为 Browsing File：的静态文本域，在与此静态文本域相邻的右边，添加第二个无标题、标识符为 ID _ FILE _ NAME 的文本域。

3. 在刚添加的两个文本域的下面添加一个标题为 Select File 的命令按钮，在此命令按钮的下面，添加一个标题为 Icons contained in this file：的静态文本域，在此静态文本域的右边添加一个无标题、标识符为 ID _ NUM _ ICONS 的文本域。

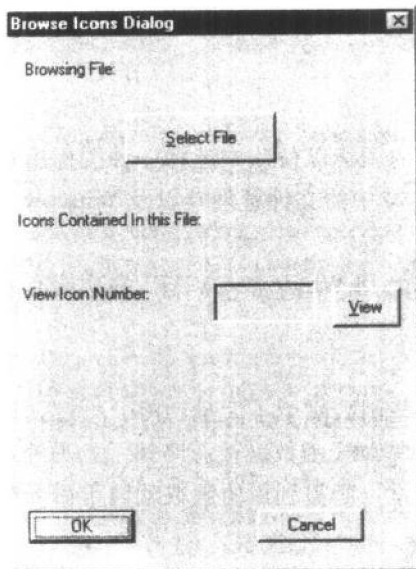


图 13-7 浏览图标对话框

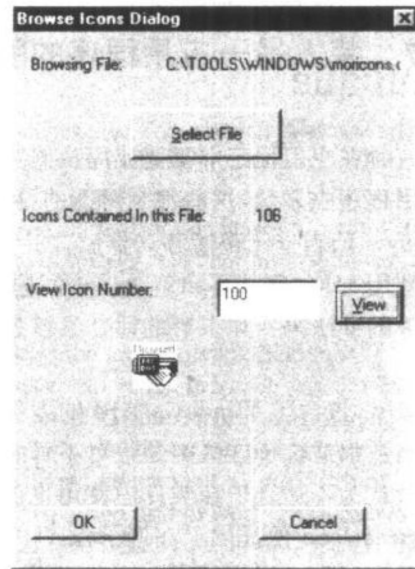


图 13-8 显示 MORICONS. DLL 中图标的浏览图标对话框

4. 在两个有关图标数目的文本域下面，添加标题为 View Icon Number：的文本域并在此文本域的右边添加一个编辑域，在编辑域的旁边，添加标题为 View 的按钮。

5. 最后，在对话框中编辑域的下面，通过在此位置添加一个图象域并把其类型设置为 I-CON 添加一个图标。

6. 选择 ClassWizard，为刚创建的模板生成一个新的对话框类，把此类命名为 CBrowse-IconsDlg。

7. 从对象列表中选择对象 IDC _BUTTON1,从消息列表中选择消息 BN _CLICKED,点击按钮 Add Function 添加新函数 OnSelectFile。在 CBrowseIconsDlg 的方法 OnSelectFile 中添加下面的代码。

```
void CBrowseIconsDlg:: OnSelectFile ()
{
    // Get the file name from the user
    CFileDialog fd (TRUE);
    if ( fd.DoModal () == IDOK ) {
        CString theFile = fd.GetPathName ();
        UpdateIcons (theFile);
        GetDlgItem (IDC _EDIT1) ->EnableWindow (TRUE);
        GetDlgItem (ID _FILE _NAME) ->SetWindowText ( theFile );
        fileName = theFile;
    }
    }
```

8. 从对象列表中选择对象 IDC _BUTTON2,从消息列表中选择消息 BN _CLICKED,点击按钮 Add Function 添加新函数 OnViewIcon。在 CBrowseIconsDlg 的方法 OnViewIcon 中添加下面的代码。

```
void CBrowseIconsDlg:: OnViewIcon ()
{
    // Get the string from the edit box
    char buffer [80];
    GetDlgItem (IDC _EDIT1) ->GetWindowText ( buffer, 80);
    // Change to a number
    int idx = atoi (buffer);
    // See how many are in the file
    GetDlgItem (ID _NUM _ICONS) ->GetWindowText ( buffer, 80 );
    int numIcons = atoi (buffer);
    if ( numIcons < idx ) {
        MessageBox ("Not that many icons in file", "Error", MB _OK );
        return;
    }
    // Extract that icon from the file
    HICON hIcon = ExtractIcon ( AfxGetInstanceHandle (), fileName, idx );
    // Update the text box with this icon
    HICON hOldIcon = (HICON):: SendDlgItemMessage ( m _hWnd,
                        IDC _ICON _PLACEHOLDER, STM _SETICON, (WPARAM) hIcon, 0 );
}
```

9. 从对象列表中选择对象 CBrowseIconsDlg，从消息列表中选择消息 WM _INITDIALOG，点击按钮 Add Function 添加新函数 OnInitDialog。在 CBrowseIconsDlg 的方法 OnInitDialog 中添加下面的代码。

```
BOOL CBrowseIconsDlg:: OnInitDialog ()
```

```
{
    CDialog:: OnInitDialog ();
    // Set window to be disabled.
    GetDlgItem (IDC_EDIT1) ->EnableWindow (FALSE);
    return TRUE; // return TRUE unless you set the focus to a control
}
```

10. 把下面的方法定义添加到类中。

```
void CBrowseIconsDlg:: UpdateIcons (CString& s)
{
    // Step 1: Try to get number of icons
    int numIcons = (int) ExtractIcon ( AfxGetInstanceHandle (), s, -1 );
    char buffer [80];
    sprintf (buffer, "%d", numIcons);
    GetDlgItem (ID_NUM_ICONS) ->SetWindowText (buffer);
}
```

11. 在 CBrowseIconsDlg 的头文件 BROWSEIC.H 中添加下面的说明。

```
private:
CString fileName;
// Construction
public:
void UpdateIcons (CString& s);
```

12. 重新进入资源编辑器，添加标题为 Dialogs 的主层菜单。在此 Dialogs 菜单中，添加标题为 Browse Icons、标识符为 ID_BROWSE_ICONS 的菜单项，保存菜单，退出资源编辑器。

13. 从 ClassWizard 的下拉列表中选择对象 CCh137App。从对象列表中选择对象 ID_BROWSE_ICONS，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnBrowseIcons。在 CCh137App 的方法 OnBrowseIcons 中添加下面的代码。

```
void CCh137App:: OnBrowseIcons ( )
{
CBrowseIconsDlg dlg;
dlg.DoModal;
}
```

14. 把下面一行添加到 CCh137App 的源文件 CH137.CPP 中。

```
#include "browseic.h"
```

15. 编译与运行此例子程序。

用法

当最初显示对话框时，因为没有数据装入，所以对话框是空的。当用户点击了按钮 Select File 后，一个通用的 File Open 对话框就会显示，当用户从中选择文件后，便为此文件调用函数 ExtractIcon，并以当前实例句柄以及作为被选择图标索引的－1 为参数，参数－1 向函数表明：它应返回在此文件中找到的图标数目，当这些完成之后，图标的数目就通过 API 函数 SetWindowText 显示在静态文本域中。

当用户在编辑框中输入一个值并选择按钮 View 后，程序通过检查文件中可用的图标数目的当前值来确定用户输入的索引值是否有效。如果索引值无效，程序将显示一条错误信息然后继续运行；如果索引值有效，函数 ExtractIcon 便重新被调用，并以所需图标的索引值、当前实例句柄以及从中提取图标的文件名为参数。如果调用成功，函数将返回此图标的句柄。

当图标句柄被返回到对话框时，函数 SendDlgItemMessage 便以窗口消息 STM_SETICON 为参数被调用。此消息向静态文本域表明：它应显示在此消息的 wparam 域中所传递的图标。当执行完这项工作之后，图标便在对话框中向用户显示出来。

13.8 移动和缩放应用程序中的窗口

问题

当允许用户对所显示的对话框的大小与位置进行调整时，由于对话框中往往包含一些控制，因此当对话框的大小与位置发生改变时，也需要对这些控制的大小与位置进行改变，以便使它们能够适应于调整后对话框的一些新安排。

那么，如何移动与缩放对话框或应用程序中的窗口以便把它们放到所需的位置呢？

方法

本节中，将利用几个不同的 Windows API 函数来实现这一目标。首先，将利用函数 GetWindowRect，它用来以屏幕坐标的方式返回一个窗口的边界框；接着，将利用函数 GetSystemMetrics 来获取对话框的某些数据；最后，利用 API 函数 MoveWindow 来实际地移动与调整对话框中控制的位置与大小。

此方法可以类似的方式运用于屏幕上的窗口，诸如视图窗口、弹出式窗口等，此时，实际的参数可能需要变化一下，因为控制对于他们的父对话框来说就是子窗口，只不过是需要一些特殊的过程来把他们摆放在正确的位置而已。

步骤

打开与运行按照如下步骤生成的例子程序 CH138.MAK。选择主菜单 Dialogs 中的菜单项 Move and Resize Dialog，此时便显示如图 13-9 所示的对话框，可以看到，此窗口的边框是属于可缩放类型的，即允许用户改变此窗口的大小，移动四周的边框直到对话框的位置与大小达到满意为止，释放鼠标，便会看到对话框中的控制将重新调整其大小并移动到对话框中相应的位置。

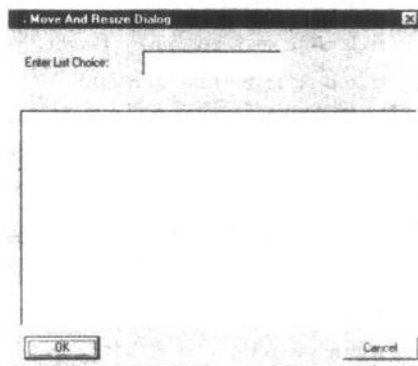


图 13-9 移动与调整对话框

为了在例子程序中实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH138.MAK，进入资源编辑器。

2. 创建一个新的对话框类。在对话框的顶端添加标题为 Enter List Choice 的静态文本域，在紧接着此静态文本域的右边创建一个编辑域。

3. 在紧跟静态文本域与编辑域的下面添加一个列表框。

4. 把按钮 OK 与 Cancel 移到对话框的底部。

5. 进入 ClassWizard，为刚创建的对话框模板生成一个新的对话框类，把此新类命名为 CMoveResizeDlg。从 ClassWizard 中的对象列表中选对象 CMoveResizeDlg，从消息列表中选择消息 WM_SIZE，点击按钮 Add Function 添加新函数 OnSize。把下面的代码添加到 CMoveResizeDlg 的方法 OnSize 中。

```
void CMoveResizeDlg::OnSize (UINT nType, int cx, int cy)
{
    CDialog::OnSize (nType, cx, cy);
    CRect er, r, rt;
    // First, get the current size of the dialog box
    GetWindowRect ( &r );
    CWnd *edit1 = GetDlgItem (IDC_EDIT1);
    if ( edit1 == NULL )
        return;
    CWnd *btn1 = GetDlgItem (IDOK);
    if ( btn1 == NULL )
        return;
    CWnd *btn2 = GetDlgItem (IDCANCEL);
    CWnd *list1 = GetDlgItem (IDC_LIST1);
    // The "pivot" point is the first edit field. This determines the high point of
    // the dialog. Use it to resize and move all other fields
    edit1->GetWindowRect ( &er );
    // Move the Ok button to be just above the bottom of the dialog box
    btn1->GetWindowRect ( &rt );
    int height = rt.bottom - rt.top;
    int width = rt.right - rt.left;
    // Reset rectangle
    rt.bottom = r.bottom - height - 10 - r.top;
    rt.top = rt.bottom - height;
    rt.left = GetSystemMetrics (SM_CXFRAME) + 10;
    rt.right = rt.left + width;
    btn1->MoveWindow ( &rt );
    // Do the same for the CANCEL button
    btn2->GetWindowRect ( &rt );
    height = rt.bottom - rt.top;
    width = rt.right - rt.left;
    // Reset rectangle
    rt.bottom = r.bottom - height - 10 - r.top;
    rt.top = rt.bottom - height;
    rt.right = r.right - r.left - 10;
    rt.left = rt.right - width;
    int bottom_of_list = rt.top - 10;
    btn2->MoveWindow ( &rt );
```

```
    // Hold onto the top and bottom parts
    rt.top = er.bottom - r.top + 10;
    rt.bottom = bottom_of_list;
    rt.left = 10;
    rt.right = r.right - r.left - 10;
    list1->MoveWindow (&rt);
}
```

6. 重新进入资源编辑器，添加标题为&Dialogs的主层菜单。在Dialogs主菜单上，添加标题为Move and Resize Dialog、标识符为ID_MOVE_DLG的菜单项，保存此菜单并退出资源编辑器。

7. 进入ClassWizard，从下拉列表中选择对象CCh138App。从对象列表中选择对象ID_MOVE_DLG，从消息列表中选择消息COMMAND，点击按钮Add Function添加新函数OnMoveDlg。把下面的代码添加到CCh138App的方法OnMoveDlg中。

```
void CCh138App::OnMoveDlg ( )
{
CMoveResizeDlg dlg;
dlg.DoModal ( );
}
```

8. 把下面一行添加到CCh138App的源文件CH138.CPP的顶端。

```
#include "moveresi.h"
```

9. 编译与运行此例子程序。

用法

当缩放对话框时，即发送消息WM_SIZE给对话框类，此消息由方法OnSize捕捉。接下来，方法OnSize通过调用API函数GetWindowRect来获取对话框本身的窗口边界框，然后利用此边框来改变对话框中其他控制的位置。

当改变对话框中控制的位置时，要利用函数MoveWindow。可惜的是：对于子控制来说，函数MoveWindow利用的是相对于其父窗口框架（在本例中是对话框）的位置，因此，为了把窗口坐标转换为此窗口的用户坐标，需从所有坐标中减去了对话框左上角的坐标值。另外，还需要根据对话框边框的宽度（用于缩放的边框）修改X坐标。当所有这些完成之后，便可调用函数MoveWindow来改变与调整控制的位置与大小。

在本例子程序中，按钮OK与Cancel总是保持其初始时的大小，而只是改变它们在对话框中的位置。对话框中央的列表框不仅改变位置而且改变大小，从而保证其落在按钮OK与顶端的编辑框之间。

注释

同样的方法可用在Delphi中，下面就是在Delphi中利用这种方法的例子。

1. 在Delphi中创建项目文件或在现存的项目文件中添加一个新表单。在此表单上，添加标题为Enter Selection的静态文本域，紧挨着此静态文本域的右边添加一个编辑域。

2. 在静态文本域与编辑框的下面添加一个列表框和两个命令按钮。

3. 点击Property Inspector的Event页中的OnResize事件，把新的事件处理程序命名为FormResize。在此表单的方法FormResize中，输入下面的代码。

```
procedure TForm1.FormResize (Sender: TObject);
var
    fr, r, r1: TRect;
    height, width : Integer;
    { Get the positon of the edit box }
    GetWindowRect ( Edit1.handle, r );
    { Get current position of OK box }
    GetWindowRect ( Button1.handle, r1 );
    { Get size of form }
    GetWindowRect ( Form1.Handle, r1 );
    { Move OK button }
    height : = r1.bottom - r1.top;
    width : = r1.right - r1.left;
    Button1.Top : =fr.bottom - 10 - height - fr.top -
                    GetSystemMetrics (SM _CYCAPTION);
    Button1.Left : = 10;
    { Move Cancel button }
    GetWindowRect ( Button2.handle, r1 );
    height : = r1.bottom - r1.top;
    width : = r1.right - r1.left;
    Button2.Top : = fr.bottom - 10 - height - fr.top -
                    GetSystemMetrics (SM _CYCAPTION);
    Button2.Left : = fr.right - width - 20 - fr.left;
    { Now, move the list box }
    ListBox1.Top : = Edit1.Top + Edit1.Height + 10;
    ListBox1.Height : = Button1.Top - 10 - ListBox1.Top;
    ListBox1.Width : = Button2.Left + Button2.Width - ListBox1.Left;
end;
```

第 14 章　程序设计的技巧

在任何程序设计环境中，都存在一些小件，如何很好地运用它们，在有经验的程序员和刚入门的程序员之间是有区别的。在 Windows 领域里，存在着组成 API 函数库的数以百计的这种“小件”，在这些函数中，许多函数由于对大多数 Windows 应用程序过于专用而很少使用，还有一些函数由于程序员未能很好理解也极少使用。

本章中，将讨论 Windows API 程序设计的一些较深奥的内容，并介绍一些小的编程技巧，这些小的技巧会使你成为一个较好的 Windows 程序设计员，更重要的是它能够使你对 Windows 程序设计环境有更深一步的了解。

在本章的几节中，将学习到如何利用表 14-1 所列出的函数进行防错性程序设计、可扩充的程序设计以及界面友好的程序设计。所有这些方面，对程序员来说都是至关重要的，而且有助于他们在程序设计领域获得进一步成功。

1. 如何确定指针是否有效

极大多数 Windows 用户的抱怨是运行应用程序时出现的一般性保护故障(GPF)，假如谨慎地从事应用程序中基本功能性的设计和实现，那么大多数 GPF 是可以避免的。对于程序员来说，大多数 GPF 是由于指针指向了无效的内存地址所引起的，使用这种错误的指针来访问或写入该指针指向的内存，势必会使得 Windows 产生 GPF 或中断，从而也使得用户感到烦恼。本节中，将考察在 Windows 中的内部指针检查例程，这种方法与生成 GPF 消息的方法是相同的，在应用程序中，可利用这种方法在引起任何故障之前确认指针的有效性。

2. 如何确定字符串是否有效

第二个导致程序崩溃的主要原因是无效字符串指针的利用。因此，当在应用程序中使用可能引起程序崩溃的（用于显示、格式化或打印字符串的）指针之前，应对其进行有效性的检查。本节中，将介绍用于检查字符串指针是否有效的 Window API 函数，并讨论如何在应用程序中利用这些函数来防止崩溃。

3. 如何在应用程序中放置版本信息

在 Windows 环境里，很少有比利用过期的资源更令人烦恼的事情了。程序崩溃、不能再现结果以及一些奇怪现象的发生，经常是由于 VBXs、动态连接库以及其他一些可执行文件与应用程序中所需的版本不匹配造成的。在前面我们曾讨论过如何检查他人的 DDL 与可执行文件的版本信息。本节中，将介绍如何把版本信息放置在自己的应用程序与 DDLs 中来供他人利用。

4. 如何编写动态连接库

动态连接库是一个可重用的 Windows 组件，它能够独立于可执行文件被装入，而且不必在编译与连接时进行联编。如人们所要求的那样，在无需改变整个可执行文件的情况下，DLLs 是包含可更新组件的非常简单的方法。另外，DLLs 能够封装相应于不同代码类型的所有不同处理函数，这也是非常有意义的。例如，如果应用程序能够读取十种不同类型的数据，而读取每种数据的方法各不相同，数据的解释也各不相同，但这十种类型的大部分代码却是相同

的。在这种情况下，可以把这些不同的代码类型放在DLLs中而只装入所需要读取与解释的版本，这样，由于在任何时刻都只有一个版本的代码而非十个不同版本的全部代码联编到可执行文件，因此会大大地节省内存。

5. 如何利用动态连接库

本节中，将讨论如何利用上节的动态连接库来子类化运行程序并将捕捉到的窗口消息记录于日志文件。

6. 如何创建各种分辨率下都能显示的应用程序

Windows是一个号称与设备无关的图形用户界面，可惜的是，许多Windows应用程序是以假定的图形显示分辨率为基础的。本节中，将利用Windows API函数确定屏幕的大小并根据实际显视器分辨率的大小（以象素为单位）把窗口显示在适当的位置。

表14-1列出了本章中用到的Windows 95 API函数。

表14-1 本章中用到的Windows 95 API函数

IsBadWriteptr	IsBadStringPfr	LocalAlloc
LocalLock	LocalUnlock	LocalFree
GetFileVersionInfoSize	GetFileVersionInfo	VerQueryValue
MoveWindow	LoadLibrary	GetProcAddress
FreeLibrary	EnumWindows	GetSystemMetrics

14.1 确定指针是否有效

问题

有时，在应用程序运行时需要确定程序中的某个指针是否有效。在不访问指针并且不引起一般性保护故障的情况下，有检查指针是否确实指向了某一变量的方法吗？

方法

认真地考虑一下，应该想到：既然Windows能够确定某指针是无效的（这就是为什么显示一般性保护故障消息的原因），那么在应用程序中，也应该能够做到这一点。

幸运的是，编写Windows的人们在API中包含了一组用于确定指针是否合法地指向存储器中某些内容的函数。实际上是用于检查指向可读存储器、可写存储器以及存放可执行代码存储器的指针的一组完整函数，不过，本节的论题，将集中在最重要的方面——可写的数据指针。

在大多数C与C++代码中，是通过让指针与NULL进行比较来检查指针的有效性；在Delphi中，是通过让指针与Nil进行比较来检查指针的有效性；当然，在Visual Basic中，没有指针。问题在于大多数C与C++的编译器没有把指针初始化为NULL值，因此忘记初始化指针是许多与指针有关的GPFs之后的基本问题。本节中，将利用API函数IsBadWritePtr来确定某指针是否有效。

步骤

打开与运行按照如下步骤生成的例子程序CH141.MAK。选择主菜单Validate中的菜单项Valid Pointer，一个显示有字符串"Pointer is Good"的消息框便会出现。接着，从菜单Validate中选择菜单项Invalid Pointer，则会看到如图14-1所示的消息框，它表明了一个无效

指针的引用，在这里，选择无效指针并不引起程序崩溃。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH141.MAK。在 AppWizard 中，为此项目文件选择选项，关闭核选框 Multiple Document Interface、Toolbar 以及 Print 与 Print PreView。

2. 进入资源编辑器，从资源列表中选择菜单资源。选择菜单 IDR_MAINFRAME（它应是现有的唯一菜单），利用标题 Validate 添加一个新的主层菜单。

3. 在菜单 Validate 上，添加标题为 Valid Pointer、标识符为 ID_VALID_PTR 的菜单项。

4. 在菜单 Validate 上，添加另一个标题为 Invalid Pointer、标识符为 ID_INVALID_PTR 的菜单项，保存资源文件，退出资源编辑器。

5. 进入 ClassWizard，从下拉组合框中选择应用程序对象 CCh141App。从对象列表中选择对象 ID_VALID_PTR，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnValidPtr。把下面的代码添加到类 CCh141App 的方法 OnValidPtr 中。

图 14-1 表明无效指针引用的对话框

```
void CCh141App:: OnValidPtr ()
{
    int *p = new int [20];
    if ( IsBadWritePtr (p, 20 * sizeof (int) ) )
        MessageBox (NULL, " Pointer is Bad!", " Error", MB_OK );
    else
        MessageBox (NULL, " Pointer is Good!", " Info", MB_OK );
    delete [] p;
}
```

6. 再次进入 ClassWizard，从下拉组合框中选择应用程序对象 CCh141App。从对象列表中选择对象 ID_INVALID_PTR，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnInvalidPtr。把下面的代码添加到类 CCh141App 的方法 OnInvalidPtr 中。

```
void CCh141App:: OnInvalidPtr ()
{
    int far *p = (int far * ) 0x20; // This pointer doesn't point at anything!
    if ( IsBadWritePtr (p, 20 * sizeof (int)) )
        MessageBox (NULL, " Pointer is Bad!", " Error", MB_OK );
    else
        MessageBox (NULL, " Pointer is Good!", " Info", MB_OK );
}
```

7. 编译与运行此例子程序。

用法

函数 IsBadWritePtr 用来验证指针的两方面事情。首先，确定指针是否真正地指向了应用

程序中的有效数据空间，由于大多数悬挂指针没有指向内存中的有效段，所以函数 IsBad-WritePtr 将捕获大多数没有被初始化的指针。

函数 IsBadWritePtr 验证的第二方面的事情是：指针指向的对象块所包含的存储空间是否至少为该函数中所指定的大小并且是可写的，另外也可以通过这种方法来检查所分配的指针是否指向有足够的空间来保存在存储块内所要存放的存储量，从而能捕捉内存是否写满。

当在第一个例子函数（OnValidPtr）中调用函数 IsBadWritePtr 时，指针指向了一个完全合法的存储块，而且此存储块包含在一个可写的存储区中，因此，此函数调用失败，从而表明这是一个正确的指针。

当在第二个例子函数 OnInvalidPtr 中调用函数 IsBadWritePtr 时，由于指针没有指向任何地方，因此函数调用成功，从而表明这是一个坏的、不能使用的指针。

注释

此程序将只捕捉未初始化或指向数据空间之外的无效指针，而并不捕捉指向被删除的对象或指向 NULL 内存地址的指针。在应用程序中使用任何指针之前，一定要注意把它核对为 NULL。

14.2 确定字符串是否有效

问题

有时，在应用程序运行时需要确定在应用程序中分配的字符串是否有效。那么，有查明字符串指针是否被分配空间以及是否指向内存中某有效字符串的方法吗？

方法

确定字符串指针(即 Windows 中的 LPSTR)是否已由程序分配空间以及它所指向的字符串是否仍然有效是完全可能的。通过本节讨论的方法并与核查 NULL 指针相结合，将会使编制出的应用程序不会出现无效的字符串指针。

本节中，将利用 API 函数 IsBadStringPtr 确定字符串指针是否正指向应用程序的数据空间以及所指向的是否为有效字符串。

步骤

打开与运行按照如下步骤生成的例子程序 CH142.MAK。选择主菜单 Validate 中的菜单项 Valid String，一个显示有字符串 " String is Good" 的消息框便会出现，接着，从菜单 Validate 中选择菜单项 Invalid String，则会看到如图 14-2 所示的消息框，它实际上用来表明一个无效字符串指针的引用，在这里，选择无效的字符串指针并不会引起程序崩溃。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH142.MAK。在 AppWizard 中，为此项目文件选择选项，关闭核选框 Multiple Document Interface、Toolbar 以及 Print 与 Print Pre-View。

2. 进入资源编辑器，从资源列表中选择菜单资源。选择菜单 IDR_MAINFRAME（它应是现存的唯一菜单)，利用标题 Validate 添加一个新的主层菜单。

3. 在菜单 Validate 上，添加标题为 Valid String、标识符为 ID

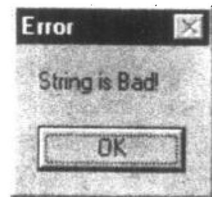


图 14-2 表明无效字符串（LPSTR）引用的消息框

_VALID_STRING 的菜单项。

4. 在菜单 Validate 上，添加另一个标题为 Invalid String、标识符为 ID_INVALID_STRING 的菜单项，保存资源文件，退出资源编辑器。

5. 进入 ClassWizard，从下拉组合框中选择应用程序对象 CCh142App。从对象列表中选择对象 ID_VALID_STRING，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnValidString。把下面的代码添加到类 CCh142App 的方法 OnValidString 中。

```
void CCh142App:: OnValidString ()
{
    HANDLE hStr = LocalAlloc (LPTR, 256);
    LPSTR string = (LPSTR) LocalLock (hStr);
    if ( IsBadStringPtr (string, 256) )
        MessageBox (NULL, " String is Bad!", " Error", MB_OK );
    else
        MessageBox (NULL, " String is Good!", " Info", MB_OK );
    LocalUnlock ( hStr );
    LocalFree ( hStr );
}
```

6. 再次进入 ClassWizard，从下拉组合框中选择应用程序对象 CCh142App。从对象列表中选择对象 ID_INVALID_STRING，从消息列表中选择消息 COMMAND，点击按钮 Add Function 添加新函数 OnInvalidString。把下面的代码添加到类 CCh142App 的方法 OnInvalidString 中。

```
void CCh142App:: OnInvalidString ()
{
    LPSTR string = (LPSTR) 0×0;
    if ( IsBadStringPtr (string, 256) )
        MessageBox (NULL, " String is Bad!", " Error", MB_OK );
    else
        MessageBox (NULL, " String is Good!", " Info", MB_OK );
}
```

7. 编译与运行此例子程序。

用法

当 Windows 利用函数 LocalAlloc 创建字符串变量时，在存储块的起点处将留出一部分用来指示存储块的状态与大小，然后，创建一个引用此存储块的句柄，Windows 将在程序中利用此句柄间接操作存储区，从而使实际的存储块能够到处移动来创建新的空间。

函数 IsBadStringPtr 只是检查字符串指针是否仍然有效以及是否指向被认为是合法的字符串。本例中，菜单项 Valid String 的句柄首先分配了一个具有 256 个字节的字符串，然后通过函数 IsBadStringPtr 来确认。除非分配失败，返回的指针将是有效的，因此函数 IsBadStringPtr 将调用失败（即返回值为假）。

在例子函数 OnInvalidString 中，新的字符串指针从没有被分配空间，因此它所指向的内存是无用的。Windows 利用函数 IsBadStringPtr 对此进行检查，然后向调用程序指明：该字符串指针是无效的。

注释

此程序将只捕获没有被初始化或没有指向有效字符串的字符串指针，而并不捕获指向被删除的对象或指向 NULL 存储器地址的指针。在应用程序中使用任何一个指针之前，一定要注意把它核对为 NULL。

14.3 在应用程序中放置版本信息

问题

应用程序的版本变换会带来一些严重的问题。如果用户想使用“技术支持问答”却不知道他们正在运行的版本，这将是一件令人头疼的事情。因为在一种版本中存在的问题不一定在另一版本中也会存在，所以，技术支持的人们自然强烈要求解决这一问题。那么，是否有一种便捷的方法来添加版本信息以便程序员在应用程序中用来确定正在运行的是何种版本?

方法

许多年来，在计算机软件中，版本问题一直是个令人头痛的问题，正如上面所说，在应用程序的一个版本中存在的问题，在另一个版本中可能就不存在。对于技术支持人员来说，会有不同的两种回答问题的说法，一种说法是：“我们将为你安装一个更新版”；而另一种说法是：“我们弄不清为什么你的应用程序不能正常执行，我们这里运行得很正常”，可以想象得到，用户会更喜欢第一种说法。

本节中，将讨论 Windows 允许用户在应用程序中嵌入的版本资源，即允许用户把可执行文件的名字、版本号以及说明性信息等置于用户自己的执行文件中，并且还将讨论如何在 About 框中为用户显示正在运行的软件的版本信息。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH143.MAK。选择菜单 Help 中的菜单项 About，则会看到如图 14-3 所示的 About 框，在此对话框中显示着该可执行文件的当前版本号。

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH143.MAK。在 AppWizard 中，为此项目文件选择选项，关闭核选框 Multiple Document Interface、Toolbar 以及 Print 与 Print PreView。

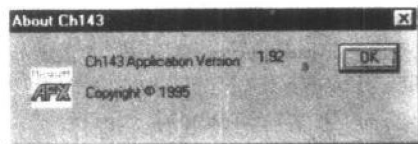


图 14-3 显示例子 Ch143 当前版本号的 About 框

2. 从项目文件列表中选择文件 CH143.RC2 并对其进行编辑，把下面的文本块输入到此文件中。

```
////////////////////////////////////////////////////////////////////////////
// Version stamp for this .EXE
#include " winver.h"
VS_VERSION_INFO VERSIONINFO
FILEVERSION          1, 0
PRODUCTVERSION       1, 0
FILEFLAGSMASK        VS_FFI_FILEFLAGSMASK
FILEFLAGS            (VS_FF_PRERELEASE | VS_FF_DEBUG)
FILEOS               VOS_DOS_WINDOWS16
```

```
FILETYPE                VFT_APP
FILESUBTYPE             VFT2_UNKNOWN
BEGIN
  BLOCK "VarFileInfo"
  BEGIN
    VALUE "Translation", 0x0409, 1252
  END
  BLOCK "StringFileInfo"
  BEGIN
    BLOCK "040904E4"
    BEGIN
        VALUE "CompanyName", "The Waite Group\0"
        VALUE "FileDescription", "Chapter 14 How To Number 3.\0"
        VALUE "FileVersion", "1.92\0"
        VALUE "InternalName", "Version Resource Example\0"
        VALUE "LegalCopyright", "Copyright (c) 1995 The Waite Group.
                                  All rights reserved.\0"
        VALUE "LegalTrademarks", "None.\0"
        VALUE "OriginalFilename", "ch143.exe\0"
        VALUE "ProductName", "Version Resource Example Ch143.exe\0"
        VALUE "ProductVersion", "1.00\0"
    END
  END
END
```

3. 进入资源编辑器，选择资源列表 Dialog，点击 ID_ABOUTBOX 项。在显示的对话框模板中，编辑所显示的字符串，把字符串 Version 后面的 1.0 删除，并利用新的静态文本域代替，把此静态文本域的标识符设定为 ID_FILE_VERSION、标题设定为空。保存此资源文件，退出资源编辑器。

4. 进入 ClassWizard，从下拉列表中选择类 CAboutDlg。从对象列表中选择类 CAboutDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function 添加新的函数 OnInitDialog。把下面的代码添加到 CAboutDlg 的方法 OnInitDialog 中。

```
BOOL CAboutDlg::OnInitDialog ()
{
    CDialog::OnInitDialog ();
    UpdateVersionInformation ();
    return TRUE; // return TRUE unless you set the focus to a control
}
```

5. 把下面的代码添加到文件 CH143.CPP 的底部。

```
void CAboutDlg::UpdateVersionInformation ()
{
    BYTE          block [1024];
    DWORD FAR * translation;
```

```
    DWORD FAR  * buffer;
    DWORD         handle;
    UINT          bytes;
    static char  * fileName = " ch143.exe";
    char          name [512]; // StringFileInfo data block.
    char          data [256];
    // Get the actual size of the information block.
    bytes = (UINT) GetFileVersionInfoSize (fileName, &handle);
    if (bytes)
    {
        // Get the actual block for the version information
        if (GetFileVersionInfo (fileName, handle, bytes, block))
        {
        if (VerQueryValue (block, " \\VarFileInfo\\Translation", (VOID FAR *
            FAR * ) &translation, (UINT FAR * ) &bytes)) {
            // The File Version for this file
            wsprintf (name, " \\StringFileInfo\\%04x%04x\\FileVersion",
                    LOWORD ( * translation), HIWORD ( * translation));
          if (VerQueryValue (block, name, (VOID FAR * FAR * ) &buffer, (UINT-
            FAR * ) &bytes)) {
          lstrcpy (data, (char far * ) buffer);
          GetDlgItem (ID_FILE_VERSION) -->SetWindowText (data);
          }
          }
          else {
          MessageBox ( " Unable to get translation type", " Error", MB_OK );
          }
        }
    }
}
```

6. 把下面的说明添加到文件 CH143.CPP 中的类 CAboutDlg 中。

```
protected:
void UpdateVersionInformation ( );
```

7. 把下面一行添加到文件 CH143.CPP 顶端的 include 文件列中。

```
#include " winver.h"
```

8. 在 Visual C++中,编辑项目文件库使它包括 Windows 库 VERSION.LIB 。编辑 Project|Files 使它包括 Visual C++安装的 LIB 目录(例如,如果安装时选择 MSVC20 做为安装目录，则应为\MSVC20\LIB\VERSION.LIB)

9. 编译与运行此例子程序。

用法

当创建 Windows 可执行文件时,可执行文件所指定的资源文件与实际的可执行文件相结合。此时，通过调用 API 函数 FindResource 与 LoadResource,资源数据便能被 Windows 系

统的其余部分所利用。此外，也可以通过调用 API 函数 GetFileVersionInfoSize 与 GetFileVersionInfo 来获取版本信息。

本例中，首先通过利用资源语句 VERSIONINFO 把数据嵌入到资源文件中。语句 VERSIONINFO 所用到的数据包括一些含有文本“FileVersion”、“ProductName”等的行，跟在这些语句后面的字符串是能够从可执行文件中检索到的数据。利用 Windows Explorer，可以从桌面上直接看到这些信息。

当从菜单 Help 中选择菜单项 About 时，对话框被创建，此时便调用对话框类的方法 OnInitDialog。此方法装入存储在文件中的资源数据，并把它放置在对话框中所添加的文本域中，从这时起，对资源文件中的资源语句所做的任何改变都会自动地反映在对话框中。这便允许程序员在不需要系统的其他部分知道当前所设置的版本号的情况下来更新版本信息或者其他字符串。

14.4 编写动态连接库

问题

当需要编写一个动态连接库来供应用程序使用时，程序员不仅要知道如何编写动态连接库中包含的所有 Windows 函数，而且还必需知道如何编写 DDL。

那么，编写 DLL 应从何种组件入手并且需要哪些组件呢？如何利用一个集成的工具环境来开发 DDL 呢？

方法

编写动态连接库曾经是一件十分繁琐的事情，程序员不仅需要记忆各种各样的项目、更新定义文件、关心存储要求，而且还有许多其他的麻烦事情。目前，在诸如 Visual C++、Delphi 等一些较新的环境中，程序员可以很容易地创建动态连接库而不必担心过去的许多困扰程序员的问题。

本节中，将创建一个简单的 DLL，让用户用来保存与恢复运行中的应用程序窗口的位置。由于这是在许多不同的应用程序中都能分享的任务（例如，保存桌面的外观），因此是一个很适合于 DLL 的工作。

步骤

打开与运行按照如下步骤生成的 Visual C++的例子程序 CH144.MAK。选择菜单 DLL Test 中的菜单项 Save Window Position，则会看到如图 14-4 所示的对话框，其中列出了可以保存的窗口。选择一个窗口然后点击按钮 OK，把此应用程序的窗口移动到桌面上新的位置，完成这些之后，再次选择例子程序 CH144，并选择菜单 DLL Test 中的菜单项 Restore Window Position。应该指出，这里的例子只是介绍 DLL 中包含的基本函数，如果想创建利用此 DLL 的应用程序，请参看 14.5 节。

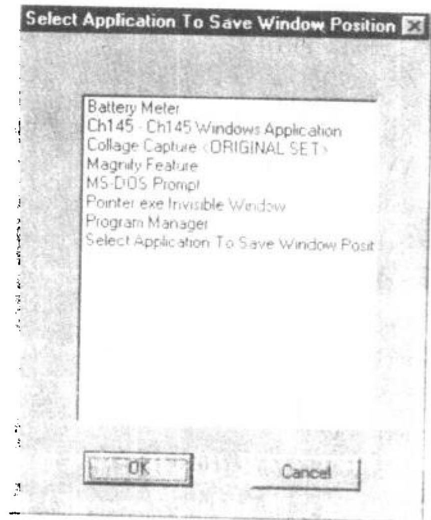


图 14-4 保存与恢复窗口位置的对话框

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C－＋中利用 Project|New 创建新的项目文件 INTFCE. MAK。从类型列表中选择 Windows 的 Dynamic Link Library (DLL)，点击按钮 OK，在接下来的对话框（选择文件）中，只需点击按钮 OK。

2. 在 Visual C＋＋中创建新的文件，并把下面的代码添加到此文件中，把此文件保存为 LIBMAIN. CPP。

```
#define STRICT
#include <windows.h>
// Turn off warning: Parameter '' is never used
#ifdef _BORLANDC_
#pragma argsused
#endif
BOOL DllEntryPoint (
    HANDLE hDLL,
    DWORD dwReason,
    LPVOID lpReserved)
{
    switch (dwReason)
        {
    case DLL_PROCESS_ATTACH:
        break;
    case DLL_PROCESS_DETACH:
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
        default:
        break;
        }
    return TRUE;
}
```

3. 在 Visual C＋＋中创建新的文件，并把下面的代码添加到此文件中，把此文件保存为 SAVEPOS. CPP。

```
#define STRICT
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern "C" {
void SaveWindow (HWND hWnd)
{
    RECT rect;
    // Get the current window rectangle
```

```
    :: GetWindowRect ( hWnd, &rect );
    // Write it to a file
    FILE *fp = fopen ( " win.pos", " w" );
    fprintf (fp, "%d, %d, %d, %d\n", rect.left, rect.top, rect.right,
          rect.bottom );
}
void RestoreWindow (HWND hWnd)
{
    RECT rect;
    // Try to open the file
    FILE *fp = fopen ( " win.pos", " r" );
    // If successful, read position
    if ( fp ! = (FILE *) NULL ) {
    fscanf (fp, "%d, %d, %d, %d", &rect.left, &rect.top, &rect.right,
          &rect.bottom );
    // And move the window there
    MoveWindow ( hWnd, rect.left, rect.top, (rect.right--rect.left),
                (rect.bottom-rect.top), TRUE);
    }
    fclose (fp);
}
}
```

4. 在 Visual C++中创建新的文件，并把下面的代码添加到此文件中，把此文件保存为CH144.DEF。

```
LIBRARY        CH144
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD MOVEABLE SINGLE
HEAPSIZE       8192
EXPORTS
               SaveWindow
               RestoreWindow
```

5. 从菜单 Project 中选择菜单项 Edit。把文件 LIBMAIN.CPP 与 SAVEPOS.CPP 添加到例子程序中。另外，把模块定义文件 CH144.DEF 也添加到例子程序中。保存此例子程序。

6. 编译此例子程序。它将创建一个新的动态连接库以及在其他应用程序中利用 DLL 所需的输入库，通过在头文件中包含 SaveWindow 与 RestoreWindow 的函数原型，并从连接库 CH144.DLL 中装入这些函数，其他应用程序就能调用 DLL。

用法

动态连接库由三个分立的部分组成。首先，初始化动态连接库，这一工作发生在文件 LIBMAIN.CPP 的函数 DllEntryPoint 中，这是所有动态连接库的标准入口与出口点。这里，它只是 DLL 的简单框架而并没有做任何工作。

初始化之后，DLL 的下一步要做的是当 DLL 从内存中被卸载时的工作。在 Windows 3.x 中，这一工作是在 WEP（Windows Exit Procedure）中完成的，而在 Windows 95 中，它也是在函数 DllEntryPoint 中完成的。这里，主要是为 DLL 做一些清理与释放空间的工作，在本例中，由于没有需要清理与释放空间的工作，因此，此函数没有做任何事情。

最后，是 DLL 的实质性部分，也就是库中所包含的实际功能。本节中，这一部分包含在文件 SAVEPOS.CPP 中，在此文件中包含有两个可输出的函数 SaveWindow 与 RestoreWindow。在模块定义文件 CH144.DEF 中，把这些函数定义为可输出的函数，这便向 Windows 95 表明这些函数可以被外部程序装入。

当 Windows 装入动态连接库时，被输出的函数表被应用程序装入，函数表用来根据名字查找函数实际的入口点。当应用程序需要装入某函数时，只需连接到 DLL 的输入库并利用该函数的原型从 DLL 调用此函数即可。

在本例中，DLL 中的函数 SaveWindow 只是以一个窗口句柄做为参数然后把此窗口的位置保存在一个文件中。当调用函数 RestoreWindow 时，此文件将被打开并读入旧的窗口位置，然后根据此位置把窗口移动到原来的位置。

注释

此例子程序的动态连接库并没有做很多的事情，不过，利用 Windows API 的强大功能做较多一些的工作是完全可能的。本例的主要意义就在于说明实现 DLL 所需做的必要工作以及在 DLL 中所需要的专用代码。

14.5 利用动态连接库

问题

在没有输入库使用的情况下，想要从动态连接库（DLL）中装入函数，就不知道函数代码驻留在存储器中的什么位置，因而也就不知道如何来获取指向库中函数的指针。

那么，如何利用 Windows API 函数获取这些信息并从 DLL 中动态地装入函数呢？

方法

从外部库中装入编译时无需知道的函数是一件十分有意义的事情。当然，函数的调用规则总是需要知道的。不过，仅仅知道所需传递的参数以及函数的名字从动态连接库中装入函数也是可能的。

Windows API 提供了一些用来处理动态连接库及动态连接库中函数的函数。本节中，将讨论两个特定的函数 LoadLibrary 与 GetProcAddress，另外，也将对函数 LoadLibrary 的伴随函数 FreeLibrary 进行讨论与使用。

本节中所使用的方法看起来似乎会令人感到迷惑，不过它主要分为如下几个步骤。首先，通过调用函数 LoadLibrary 把动态连接库从磁盘装入到内存。接着，利用函数 GetProcAddress 从库中装入函数。最后，利用指定的参数通过指针调用此函数。

步骤

在运行此例子之前，把本章第 4 节中的文件 CH144.DLL 拷贝到 WINDOWS\SYSTEM 目录之下。打开与运行按照如下步骤生成的例子程序 CH145.MAK，选择菜单 DLL Test 中的菜单项 Save Window Position，一个对话框便会显示出来，并在其中显示有可供选择的应用程序窗口。从这些窗口中选择一个并点击按钮 OK，然后把此应用程序的窗口移动到桌面上新的

位置。完成这些之后，再次选择例子程序CH145，并从菜单DLL Test中选择菜单项Restore Window Position。

虽然此程序在14.4节中已经讨论过，但是我们这里讨论的是调用DLL的程序而非DLL本身。

为了在应用程序中实现上述功能，请按如下步骤进行。

1. 在Visual C++中创建新的项目文件CH145.MAK。在AppWizard中，为此项目文件选择选项，关闭核选框Multiple Document Interface、Toolbar以及Print与Print Preview。

2. 进入资源编辑器，从资源列表中选择菜单资源。选择菜单IDR_MAINFRAME（它应该是存在的唯一菜单）并添加标题为DLL Test的主层菜单。

3. 在菜单DLL Test上，添加标题为Save Window Position、标识符为ID_SAVE_WINDOW_POS的菜单项。

4. 在菜单DLL Test上，添加另一个标题为Restore Window Position、标识符为ID_RESTORE_WINDOW_POS的菜单项。

5. 保存资源文件，退出资源编辑器。

6. 进入ClassWizard，从下拉组合框中选择应用程序对象CCh145App。从对象列表中选择对象ID_SAVE_WINDOW，从消息列表中选择消息COMMAND，点击按钮Add Function添加新函数OnSaveWindowPos。把下面的代码添加到类CCh145App的方法OnSaveWindowPos中。

```
void CCh145App::OnSaveWindowPos ()
{
    CSelectWindow dlg;
    if ( dlg.DoModal () == IDOK ) {
        SaveWindowCallback ptr;
        FARPROC f;
        hLib = LoadLibrary ( "ch144.dll" );
        f = GetProcAddress ( hLib, "SaveWindow");
        ptr = (SaveWindowCallback) f;
        theHandle = dlg.WindowHandle ();
        (*ptr) (theHandle);
        FreeLibrary ( hLib );
    }
}
```

7. 再次进入ClassWizard，从下拉组合框中选择应用程序对象CCh145App。从对象列表中选择对象ID_RESTORE_WINDOW_POS，从消息列表中选择消息COMMAND，点击按钮Add Function添加新函数OnRestoreWindowPos。把下面的代码添加到类CCh145App的方法OnRestoreWindowPos中。

```
void CCh145App::OnRestoreWindowPos ()
{
    if ( hLib ) {
        RestoreWindowCallback ptr;
        FARPROC f;
```

```
        hLib = LoadLibrary ( " ch144.dll" );
        f = GetProcAddress ( hLib, " RestoreWindow");
        ptr = (RestoreWindowCallback) f;
        ( * ptr) (theHandle);
    }
}
```

8. 进入资源编辑器，创建新的对话框模板。在此模板中添加一个列表框，把按钮 OK 与按钮 Cancel 移到模板的底部，把此对话框的标识符设定为 IDD _ DIALOG2。

9. 选择 ClassWizard，为刚创建的模板生成一个新的对话框类并把此类命名为 CSelectWindow。在 ClassWizard 中，从下拉组合框中选择对象 CSelectWindow，从对象列表中选择对象 CSelectWindow，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Function 添加新函数 OnInitDialog。把下面的代码添加到 CSelectWindow 的方法 OnInitDialog 中。

```
BOOL CSelectWindow:: OnInitDialog ()

    CDialog:: OnInitDialog ();
    CListBox *list = (CListBox * ) GetDlgItem (IDC _ LIST1);
    EnumWindows ( EnumWindowsProc, (LPARAM) list );
    UpdateData ();
    return TRUE; // return TRUE unless you set the focus to a control
}
```

10. 在 ClassWizard 中，从下拉组合框中选择对象 CSelectWindow，从对象列表中选择对象 IDOK，从消息列表中选择消息 BN _ CLICKED，点击按钮 Add Function 添加新函数 OnOk。把下面的代码添加到 CSelectWindow 的方法 OnOk 中。

```
void CSelectWindow:: OnOK ()
{
    // Get the text and window selected
    CListBox *list = (CListBox * ) GetDlgItem (IDC _ LIST1);
    int idx = list->GetCurSel ();
    if ( idx ! = LB _ ERR ) {
        list->GetText (idx, window _ text );
        window _ handle = (HWND) list->GetItemData ( idx );
        CDialog:: OnOK ();
    }
    else
        MessageBox (" You must make a selection to continue", " Error", MB _ OK );
}
```

11. 把下面几行代码添加到类 CSelectWindow 的头文件 SELECTWI.H 中。

```
private:
    CString window _ text;
    HWND window _ handle;
public:
    CString& WindowText () { return window _ text; };
```

```
HWND WindowHandle () { return window_handle; };
```

12. 把下面几行代码添加到例子程序对象 CCh145App 的头文件 Ch145.H 中。

```
class CCh145App : public CWinApp
{
private:
    HINSTANCE hLib;
    HWND      theHandle;
public:
    virtual int ExitInstance ();
```

13. 接下来，把下面的方法定义添加到例子程序的源文件 CH145.CPP 中。

```
int CCh145App:: ExitInstance ()
{
    if ( hLib )
       FreeLibrary (hLib);
    return 0;
}
```

14. 利用下面的代码修改文件 CH145.CPP，把黑体部分的行添加到此文件中。

```
typedef void ( * SaveWindowCallback) (HWND hWnd);
typedef void ( * RestoreWindowCallback) (HWND hWnd);
CCh145App:: CCh145App ()
{
    hLib = NULL;
}
/////////////////////////////////////////////////////////////////////////////
// The one and only CCh145App object
CCh145App NEAR theApp;
/////////////////////////////////////////////////////////////////////////////
// CCh145App initialization
BOOL CCh145App:: InitInstance ()
{
    hLib = NULL;
```

15. 编译与运行此例子程序。

用法

当选择菜单项 DLL Test 时，程序便试图装入 DLL 包含的获取窗口位置的函数。如果 DLL 不存在或不能被装入，函数 LoadLibrary 便返回 NULL，在这种情况下，不做任何事情。如果成功地装入了 DLL 库，一个函数指针便通过调用函数 GetProcAddress 从库中被“提取”出来。需要注意的是：DLL 的函数是做为 extern " C" 被编译的。

当函数从库中被装入后，它便通过刚获取的指针被直接调用。对于函数 SaveWindow，它只需要一个参数即要保存位置的应用程序窗口的句柄，此函数用来获取应用程序窗口的位置并把该位置保存在一个文件中。

对于函数 RestoreWindow，也需要从 DLL 中装入，此函数也是只需要一个参数，并通过指针被直接调用，从而恢复窗口的位置。

注释

在 Windows 95 中，从 DLL 中调用函数是一个强有力的编程方法。这种方法允许程序员修改自己应用程序中的功能，而无需在自己的可执行文件中驻留必要的代码。

在实际的应用程序中，DLL 的一些用法包括：硬件的专门驱动程序、国际化支持（利用资源 DLLs）以及由于不同原因而需改变代码的任何其他场合。

14.6 创建各种分辨率下都能显示的应用程序

问题

有时希望让应用程序能够总是在屏幕中央以相同的比例显示。不幸的是，由于不同监视器与视频卡的屏幕分辨率各不相同，所以对于不同的系统，程序也就会在不同的位置上显示。

那么，如何统一化程序使得它在各种不同的系统上都能在大致相同的位置显示呢?

方法

设备无关的图形用户界面似乎是完美的。Windows 试图提供一些无需知道屏幕存储定位在哪里以及无需知道安装了何种类型的视频驱动程序的图形。

不幸的是，大多数视频驱动程序利用视频卡中专门的处理来产生效果，有些还允许用户指定超出 Windows 一般范围的分辨率（例如，许多 SVGA 板的 1024×768 的分辨率）。

为了使程序真正地与设备无关，首先必须了解一些能够获取设备消息的 API 函数。本节中，将介绍函数 GetSystemMetrics 与 MoveWindow。把这两个函数结合起来使用，就能正确地指定应用程序窗口所显示的位置及其在任何监视器上所占用的屏幕空间。

本节中，最出人意料的是：完成这样的事情只需要三行代码!

步骤

打开与运行按照如下步骤生成的例子程序 CH146.MAK，将看到如图 14-5 所示的窗口。无论此程序在何种监视器或视频卡上运行，对用户来说它看起来都是相同的。

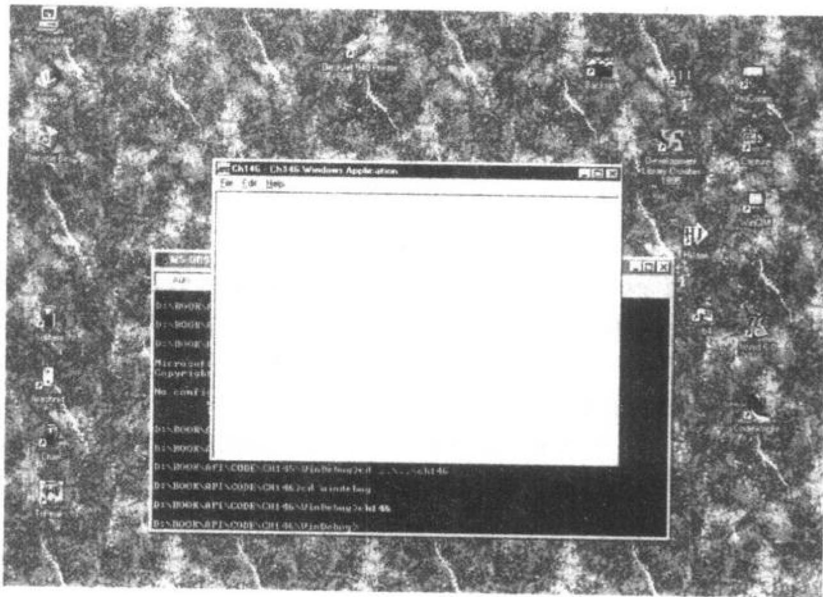


图 14-5　在指定位置显示的设备无关应用程序窗口

如果想实现上述功能，请按如下步骤进行。

1. 在 Visual C++中利用 AppWizard 创建新的项目文件 CH146.MAK。在 AppWizard 中，为此项目文件选择选项，关闭核选框 Multiple Document Interface、Toolbar 以及 Print 与 Print Preview。

2. 把下面的代码添加到源文件 CCh146App 的函数 InitInstance 中。这里只添加被标记为黑体的行。

```
// create a new (empty) document
    OnFileNew ();
    if (m_lpCmdLine [0] ! = '\0')
    {
    // TODO: add command line processing here
    }
    // Set the window size to be half the size of the screen.
    int width = GetSystemMetrics (SM_CXSCREEN);
    int height = GetSystemMetrics (SM_CYSCREEN);
    m_pMainWnd->MoveWindow (width/4, height/4, width/2, height/2 );
    return TRUE;
}
```

3. 编译与运行此例子程序。

用法

当启动此例子程序时，便调用例子程序对象的方法 InitInstance，此方法首先调用函数 GetSystemMetrics 获取屏幕的宽度与高度（以象素为单位），然后利用这两个值来计算窗口所要显示的大小与位置。本例中，所显示窗口的高度与宽度为屏幕的一半，并且放置在屏幕桌面的中央位置。

当编写 Microsoft Windows 95 的应用程序时，在屏幕中移动窗口或对话框之前，考虑用户屏幕的分辨率是十分重要的，可以利用函数 GetSystemMetrics 来测定当前屏幕设置的分辨率。

注释

本例虽然简单，但它却为 Windows 95 的应用程序提供了非常重要的基础。用户非常习惯于使用在他们的桌面上大小合适的窗口，而一般不习惯于使用那些为特定的屏幕分辨率编写的应用程序，因为这种应用程序在其他分辨率的屏幕上运行时不能显示出如人所愿的窗口。因此当编写应用程序、设计对话框以及在屏幕上显示信息时，要仔细地考虑一下所能使用的各种不同的分辨率。

第15章　完善应用程序

编写Windows应用程序并不简单，既需要清楚Windows系统的内部实现，还需要清楚应用程序所需要实现的功能。即使应用程序编写完了，还会有大量的工作，在许多时候，应用程序尽管发布了，但并没有进行彻底的完善，许多对用户很有吸引力的功能并没有实现。

在本章中，将考虑如何来完善一个应用程序。完善就是在应用程序中添加一些用户习惯使用的功能，以及一些使应用程序更加完美更加易于使用的功能。工具条是Windows应用程序的标准组件，用户已经习惯于使用此控制，应用程序还“必须”具有的其他组件有：状态条、扉屏（splash screen）、以及背景位图，本章将介绍如何来实现这些功能。

1. 如何实现上下文相关帮助

如果要选择Windows系统的某一特色添加到自己的应用程序中，当然的选择应该是上下文相关帮助。用户非常喜欢此功能，只要点击按钮Help或按下键F1，则有关用户当前任务的有用信息就会显示在屏幕上。在本节中，将介绍如何利用Windows API的强大功能来启动Windows帮助系统，使得用户可以快速查看当前任务的帮助信息。

2. 如何创建状态条

状态条是应用程序另一个不可缺少的重要的Windows功能。MFC自动地在应用程序的主窗口中添加状态条，但你是否知道如何在其他窗口中添加状态条呢？

3. 如何创建工具条

类似于状态条，工具条对用户来说也是很重要的。如果在应用程序中实现工具条，那么用户操作就只需点击按钮就可以了，而不必浏览大量繁琐的菜单和子菜单。同样，在本节中将介绍如何在非主窗口的窗口中添加工具条，同时，还将介绍浮动工具条（工具框）。

4. 如何实现在运行时修改工具条

在应用程序中，工具条是必须的，但实现用户可以定制的工具条则是大多数程序员梦寐以求的。在本节中，将介绍如何实现可定制工具条，使得用户在应用程序运行时也可修改工具条。

5. 如何从在线帮助中显示范例

Windows帮助系统允许程序员实现许多功能，其中包括从帮助系统中调用外部函数的功能。在本节中，将介绍如何使用外部函数来创建外部的功能示范。

6. 如何在应用程序启动时显示About框

许多应用程序在启动时显示About框，如Visual C++、Microsoft Word、Borland Delphi以及其他的应用程序。About框通常包含应用程序的信息，显示在屏幕上直到用户取消它。在本节中，将介绍如何显示About框，并使其保留在屏幕上直到用户取消它。此对话框在应用程序开始前显示，使得用户在应用程序的其余部分装入时可以查看一些信息。

7. 如何显示扉屏（splash screen）

扉屏是一种特殊类型的窗口，它保留在屏幕上直到一定的时间过去或用户按下鼠标键（或键盘键）。在本节中，将一步步地介绍如何创建扉屏，以及如何使其在用户点击时消失。

8. 如何确定应用程序的图标

确定应用程序极小化时显示哪个图标是非常有用的，在 Windows 95 中，应用程序显示在屏幕上时与极小化在任务条时显示不同的图标。在本节中，将介绍如何利用 Windows API 函数来确定应用程序在不同状态时显示的图标。

9. 如何显示作为窗口或对话框背景的位图

如果能够使用窗口或对话框的背景来显示一些图形，而不再是 Windows 使用的令人讨厌的灰色，是不是更好呢？许多程序员对在背景上显示位图特别感兴趣，这样可以使得应用程序充满生机。在本节中，将介绍实现的步骤以及所需要的 API 函数。

表 15-1 列出了本章使用的 Windows 95 API 函数。

表 15-1 第 15 章使用的 Windows 95 API 函数

GetFocus	GetDlgCtrlID	MessageBox
GetClientRect	MoveWindow	ShowWindow
LoadBitmap	SetButtons	FindWindow
PostMessage	UpdateWindow	GetCurrentTime
BringWindowToTop	DestroyWindow	CreateCompatibleDC
SelectObject	StretchBlt	GetObject
GetSystemMetrics	SHGetFileInfo	ImageList _ GetIcon
SendDlgItemMessage	SetBkMode	DeleteObject

15.1 实现上下文相关帮助

问题

有的程序员希望能够在基于表单的 Windows 应用程序中实现上下文帮助，尤其希望能够准确地确定用户当前正在编辑表单中的哪个域，以及在用户按下 F1 键时捕捉它。

接着应用程序就可以根据用户正在操作的域来显示相应的帮助信息或帮助文件。如何利用 Windows API 函数来实现此功能呢？

方法

此功能对用户来说是非常有用的，当用户在表单（即 MFC 中的 CFormView）中操作时，如果不清楚某个域如何操作，最后总是希望能够在帮助文件中找到相应的说明项。

在本节中，将讨论如何使用 Windows API 函数 GetFocus 和 GetDlgCtrlID 来准确确定用户正在操作的是哪个域，从而可以将相应的帮助信息装入并显示给用户。但在本节的例子程序中，将只显示包含域级帮助信息的消息框。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，可以看到一个表单显示在屏幕上，并包含几个可输入的域。在编辑域 Name 上点击鼠标并按下 F1 键，则显示如图 15-1 所示的表单和消息框。

实现例子程序的具体步骤如下：

1. 在 Visual C++中利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH151.MAK。

2. 进入资源编辑器并创建新的对话框模板。

3. 在对话框中添加组域（group field），在组域中添加三个单选按钮，单选按钮的标题分别为 Active Customer、InActive Customer、Prospect。

4. 在对话框中添加六对静态文本域和编辑域。静态文本域的标题分别为：Name，Address，City，State，Zip Code 和 Phone Number，编辑域的标题为空，标识符分别为 IDC _ EDIT1、IDC _ EDIT2、IDC _ EDIT3、IDC _ EDIT4、IDC _ EDIT5 和 IDC _ EDIT6。

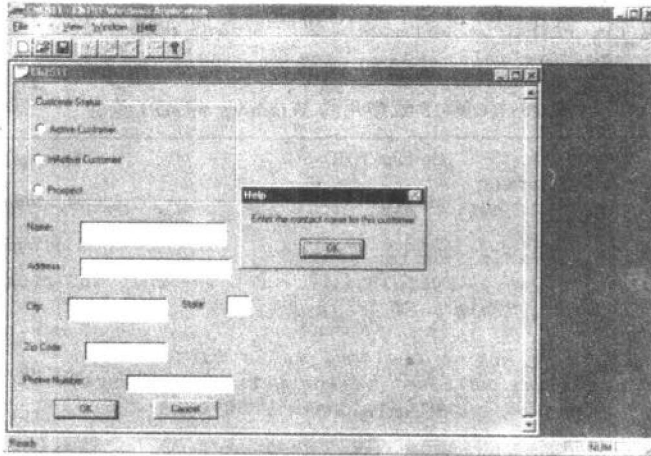


图 15-1　表单的编辑域 Name 的上下文相关帮助

5. 进入 ClassWizard，并为刚创建的对话框模板生成新的对话框类。新类命名为 CContactView，基类为 CFormView，选择对话框模板标识符 IDD _ DIALOG1 为生成此类的对话框。

6. 重新进入资源编辑器，并选择资源项 Accelerator。通过在编辑域 Key 中输入 F1 和在编辑域 ID 中输入 ID _ HELP 来添加新的快捷键，保存快捷键表并退出资源编辑器。

7. 选择类 CContactView 的源文件 CONTACTV. CPP，在类的消息映射中添加下列行（用黑体表示）：

```
BEGIN_MESSAGE_MAP (CContactView, CFormView)
    //{{AFX_MSG_MAP (CContactView)
    // NOTE — the ClassWizard will add and remove mapping macros here.
    ON_COMMAND (ID_HELP, OnHelp)
    //}} AFX_MSG_MAP
END_MESSAGE_MAP ()
```

8. 在源文件 CONTACTV. CPP 中添加下列代码：

```
void CContactView:: OnHelp ()
{
    // Get active control

    CWnd *wnd = GetFocus ();
```

```
// Get id of control
int id = wnd->GetDlgCtrlID ();

// Determine what to do based on Id

switch ( id ) {
    case IDOK: // Ok Button
        MessageBox (" The Ok button will close the form", " Help", MB_OK );
        break;
    case IDCANCEL: // Cancel Button
        MessageBox (" The Cancel Button will cancel the edit and close the
                    form", " Help", MB_OK );
        break;
    case IDC_RADIO1: // Active Customer Radio Button
        MessageBox (" Select this option for active customers\nthat have
                    ordered recently", " Help", MB_OK );
        break;
    case IDC_RADIO2: // Inactive Customer Radio Button
        MessageBox (" Select this option for inactive customers\nthat have NOT
        ordered recently", " Help", MB_OK );
        break;
    case IDC_RADIO3: // Prospect Radio Button
        MessageBox (" Select this option for prospects\nthat have never
                                ordered", " Help", MB_OK );
        break;
    case IDC_EDIT1: // Name
        MessageBox (" Enter the contact name for this customer", " Help",
                    MB_OK );
        break;
    case IDC_EDIT2: // Address
        MessageBox (" Enter the contact address for this customer", " Help",
                    MB_OK );
        break;
    case IDC_EDIT3: // City
        MessageBox (" Enter the city for this customer", " Help", MB_OK );
        break;
    case IDC_EDIT4: // State
        MessageBox (" Enter the state for this customer", " Help", MB_OK );
        break;
    case IDC_EDIT5: // Zip Code
        MessageBox (" Enter the 9 digit zip code for this customer",
                    " Help", MB_OK );
        break;
    case IDC_EDIT6: // Phone
```

```
        MessageBox (" Enter the phone number (with area code) \n for this
                customer", " Help", MB_OK );
        break;
    }
}
```

9. 在类 CContactView 的头文件 CONTACTV.H 中添加下列行，添加的行用黑体表示。

```
// Generated message map functions
    //{{AFX_MSG (CContactView)
    // NOTE - the ClassWizard will add and remove member functions here.
    afx_msg void OnHelp (void);
    //}} AFX_MSG
    DECLARE_MESSAGE_MAP ()
```

10. 最后，为例子程序修改源文件 CH151.CPP。首先，将新视图的头文件添加到 include 文件列表中，如下所示。

```
#include " contactv.h"
```

11. 接着，通过改变下列行来修改例子程序的文档模板定义，改变的行用黑体表示。

```
// Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.

    CMultiDocTemplate * pDocTemplate;
    pDocTemplate = new CMultiDocTemplate (
        IDR_CH151TYPE,
        RUNTIME_CLASS (CCh151Doc),
        RUNTIME_CLASS (CMDIChildWnd), // standard MDI child frame
        RUNTIME_CLASS (CContactView));
    AddDocTemplate (pDocTemplate);
```

12. 编译并运行此例子程序。

用法

如果程序员在应用程序的快捷键表中添加了新的项，则快捷键表将自动的将击键转化为应用程序的命令，就好像程序员在应用程序中添加了相应标识符的菜单项并选择了此菜单项一样。在本节的例子程序中，我们添加了新的命令，将 F1 键映射为命令 ID_HELP。

在本节创建的视图类中，为命令 ID_HELP 添加了消息映射项，以来捕捉此命令，这是利用 MFC 消息映射系统来完成的。如果利用 WINDOWSX.H 中定义的消息跟踪系统也可以非常容易地完成此任务，如果用户按下 F1 键，则调用函数 OnHelp。

真正的处理是在函数 OnHelp 中。首先，利用函数 GetFocus 来获取当前的控制，GetFocus 返回当前具有输入焦点的窗口。当用户按下 F1 键时，由于一个键被按下，所以具有当前输入焦点的控制也就是用户当前正操作的控制。

在确定哪个控制具有输入焦点后，便需要获取此控制的标识符，以便应用程序在需要时利用此标识符，这是通过 API 函数 GetDlgCtrlID 来完成的，此函数返回窗口的控制标识符。

一旦知道当前控制的标识符，就可以非常容易地显示出消息框，从而显示此控制的上下文相关帮助（在应用程序中定义的）。

注释

尽管在本节的例子程序中只是弹出消息框来显示控制的帮助信息，但是如果要为每个控制显示一页页的帮助信息也是非常容易的，只是为每个控制的帮助文件都添加完整的文本，本书将会变得相当杂乱，但实际上实现这样的系统是很容易的。

按照下面的步骤可以在帮助文件中实现上下文相关帮助主题。首先，创建一个新的帮助文件，包含对话框中每个控制的帮助主题，注意每页的帮助主题标识符，并创建这些常数的头文件。接着，修改类CContactView的方法OnHelp中的switch语句，用下列表单项替换每个MessageBox。

::WinHelp (m_hWnd,"helpfile.hlp", HELP_CONTEXT, identifier);

其中：参数"helpfile.hlp"为实际创建的帮助文件名，参数identifier为相应的控制标识符。

15.2 创建状态条

问题

有时，不仅需要在应用程序的主窗口上添加状态条，还需要在应用程序的普通视图上添加状态条。在应用程序主窗口的底部添加状态条（MFC实现了此功能）是相当容易的，但如何在应用程序的其他窗口上添加状态条呢？

方法

在Windows 95中，状态条窗口为标准的控制，但状态条早在Windows 3.1中就已经存在了。无论是在16位的Windows 3.1中还是在32位的Windows 95中，Microsoft基类库都包含一个封装状态条功能的"封装（wrapper）"类，在本节的例子程序中就使用了此类。如果有的程序员对直接使用API调用来创建状态条感兴趣，可以在应用程序中使用"status"类名来创建窗口。

要在视图窗口中创建状态条，只需实例化一个类型为CStatusBar的对象，并操纵此对象使其显示在窗口的顶部。如何操纵状态条呢?，这是一个难点。在本节中，将介绍创建状态条并将其显示在任何窗口中的具体步骤（尽管在本节的例子程序中选用的是视图窗口）。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示出如图15-2所示顶部有状态条的视图。

实现例子程序的具体步骤如下：

1. 在Visual C++中，利用AppWizard创建新的项目文件，并命名此项目文件为CH152.MAK。

2. 进入ClassWizard，从下拉列表中选择类CCh152View，从对象列表中选择对象CCh152View，从消息列表中选择消息WM_CREATE，点击按钮Add Function，在CCh152View的方法OnCreate中添加下列代码：

```
int CCh152View:: OnCreate (LPCREATESTRUCT lpCreateStruct)
{
  if (CView:: OnCreate (lpCreateStruct) == -1)
  return -1;
```

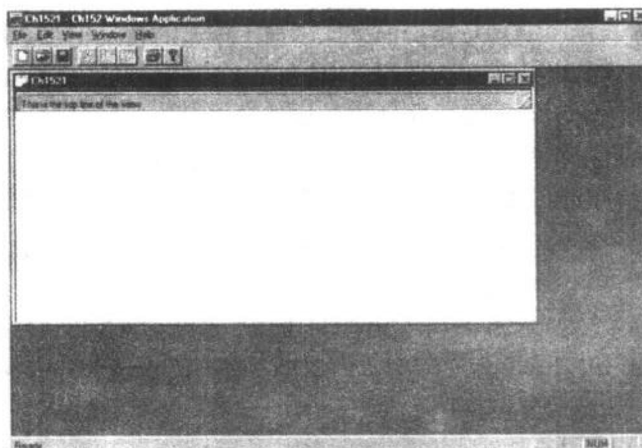


图 15-2 显示顶部状态条的 MFC 视图

```
    if ( ! topStatusBar.Create ( this, WS_CHILD | WS_VISIBLE | CBRS_TOP | WS_BORDER ) )
    {
        MessageBox (" Unable to create status bar!", " Error", MB_OK );
        return -1; // fail to create
    }

    if ( ! topStatusBar.SetIndicators (indicators, 1) )
    {
        MessageBox (" Unable to set status bar indicators!", " Error", MB_OK );
        return -1;
    }
    topStatusBar.SetPaneText (0," This is the top line of the view", TRUE);
    return 0;
}
```

3. 接着，进入 ClassWizard。从下拉列表中选择类 CCh152View，从对象列表中选择对象 CCh152View，从消息列表中选择消息 WM_SIZE，点击按钮 Add Function，在 CCh152View 的方法 OnSize 中添加下列代码：

```
void CCh152View:: OnSize (UINT nType, int cx, int cy)
{
    CView:: OnSize (nType, cx, cy);

    CRect r;
    GetClientRect (&r);
    r.bottom = 25;

    topStatusBar.MoveWindow (&r);
    UINT nID;
```

```
    UINT nStyle;
    int cxWidth;
    topStatusBar.GetPaneInfo ( 0, nID, nStyle, cxWidth );
    topStatusBar.SetPaneInfo ( 0, nID, SBPS_STRETCH, cxWidth);
}
```

4. 从项目列表中选择文件CH152VW.H，在CCh152View的类定义中添加下列说明：

```
private:
    CStatusBar topStatusBar; // Top line status bar
```

5. 从项目列表中选择文件CH152VW.CPP，在文件的顶部添加下列行：

```
static UINT BASED_CODE indicators [] =
{
    ID_SEPARATOR, // status bar indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};
```

6. 编译并运行此例子程序。

用法

MFC的状态条类（CStatusBar）在Windows 95中只是简单的封装状态条控制类型。在本节的例子程序中，只是简单地创建一个新的状态条作为其他窗口（在本节的例子程序中为视图窗口，但可以是其他任何类型的窗口）的子窗口。

创建窗口作为子窗口的技巧在于窗口的方法OnSize。在本节的例子程序中，重新定位状态条窗口，以使其放置在视图窗口的特定位置上，这是因为在应用程序中，状态条窗口通常是由父窗口来自动重新定位的，但在本节的例子程序中，状态条不可能被自动重新定位，因此就需要程序员为窗口完成此功能。

关于本节的例子程序需要注意几点。其中，使用的状态窗口没有用户常见的“窗格”，通常，状态条显示在窗口的底部，有许多窗格用来表示Control、Caps Lock、Num Lock以及其他键的状态，另外，还有窗格用来表示光标在窗口中的位置（行列值）。但在本节的例子程序中，只是使用状态条来显示静态文本（某些文档的标题或是其他的信息）。

除了缺少辅助窗格外，本节例子程序中的状态条还有另外一个特点，唯一的窗格延伸状态条的整个宽度。通常的状态条有多个文本“窗格”组成，每个文本“窗格”有固定的宽度，而整个状态条的宽度则为整个窗口的宽度。但是如果可能的话，也可以为状态条的某个窗格使用SBPS_STRETCH设置，从而使得此窗格延伸状态条的剩余宽度。在本节的例子程序中，这是在视图的方法OnSize中实现的。

注释：

在Delphi中也可以创建状态条，按照下列步骤可以实现此功能：

1. 在Delphi中创建新的项目文件，或在现有的应用程序中添加新的表单。

2. 在表单的顶部添加面板组件，设置align属性为alTop，设置alignment属性为taLeftJustify。

3. 按用户意愿设置面板组件的标题。要在运行时设置组件的标题，可以在表单的方法

TForm1.FormCreate 中使用下列代码：

```
Panel1.Caption := 'This is a top view status bar';
```

4. 编译并运行此例子程序。

15.3 创建工具条

问题

有的程序员希望能够在非主窗口的窗口中创建工具条。尽管使用 MFC 在主窗口中创建工具条是非常简单的，但很多程序员却找不到在子窗口中创建工具条的方法。

是否有方法在应用程序的子窗口中自动创建工具条呢?

方法

除了 MFC 视图的主窗口外，在其他窗口中都没有直接的方法来自动创建工具条控制。这是工具的局限，而不是基类库的局限，因为 MFC 视图窗口是可以有工具条的，但可能非常不实用，用户也不喜欢这样的实现。

由于 Windows 实际处理的 MDI 窗口是框架窗口（CMDIChildWindow）而不是实际的视图窗口（CView），因此要在 MFC 视图窗口中添加工具条，就必须首先重置（override）视图的框架窗口，添加工具条的创建和初始化。

在本节中，将介绍如何在现有的视图中添加工具条而不需要改变视图类的本身。首先，为视图创建一个新的框架窗口，接着介绍如何为框架窗口定义工具条，以便创建视图时工具条能显示出来。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示出一个顶部有工具条的视图，如图 15-3 所示。

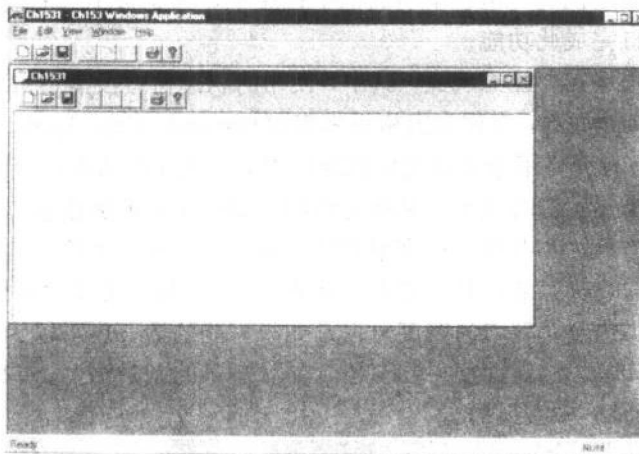


图 15-3　显示工具条的 MFC 视图

实现例子程序的具体步骤如下：

1. 在 Visual C++ 中利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH153.MAK。

2. 进入 ClassWizard，点击按钮 Add Class，新类命名为 CTBFrame，基类为 CMDIChild-Window，生成此新类的定义。

3. 在 ClassWizard 中，从对象列表中选择对象 CTBFrame，从消息列表中选择消息 WM_CREATE。在 CTBFrame 的方法 OnCreate 中添加下列代码：

```
int CTBFrame:: OnCreate (LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIChildWnd:: OnCreate (lpCreateStruct) == -1)
    return -1;
    if (! m_wndToolBar.Create (this) ||
        ! m_wndToolBar.LoadBitmap (IDR_MAINFRAME) ||
        ! m_wndToolBar.SetButtons (buttons, (sizeof (buttons) / sizeof (buttons [0]))) )
    {
        TRACE (" Failed to create toolbar\n");
        return -1; // fail to create
    }

    m_wndToolBar.ShowWindow (SW_SHOW);

    return 0;
}
```

4. 在源文件 TBFRAME.CPP 的顶部添加下列行：

```
static UINT BASED_CODE buttons [] =
{
    // same order as in the bitmap 'toolbar.bmp'
    ID_FILE_NEW,
    ID_FILE_OPEN,
    ID_FILE_SAVE,
    ID_SEPARATOR,
    ID_EDIT_CUT,
    ID_EDIT_COPY,
    ID_EDIT_PASTE,
    ID_SEPARATOR,
    ID_FILE_PRINT,
    ID_APP_ABOUT,
};
```

5. 接着，在类 CTBFrame 的头文件 TBFRAME.CPP 中添加下列说明：

```
private:
    CToolBar m_wndToolBar;
```

6. 选择应用程序对象的源文件 CH153.CPP，改变下列代码块以修改文档模板定义，改变的行用黑体表示。

```
CMultiDocTemplate * pDocTemplate;
    pDocTemplate = new CMultiDocTemplate (
```

```
    IDR_CH153TYPE,
    RUNTIME_CLASS (CCh153Doc),
    RUNTIME_CLASS (CTBFrame),
    RUNTIME_CLASS (CCh153View));
  AddDocTemplate (pDocTemplate);
```

7. 在源文件 CH153.CPP 顶部的 include 文件列表中添加下列行：

```
#include " tbframe.h"
```

8. 编译并运行此例子程序。

用法

当 MFC 创建新的 MDI 子窗口时，它首先创建 MDI 框架窗口。在应用程序中此 MDI 框架窗口是文档和视图对象的“容器”对象，包含视图窗口，也包含视图的边框和非用户区，工具条窗口就是在非用户区中创建的。在本节的例子程序中，工具条使用与例子程序主框架窗口一样的位图（和命令）。

在 MFC 中，工具条窗口依赖于运行的 Windows 版本，工具条类实际上是对内置在操作系统中的工具条控制的封装。

在本节的例子程序中，在创建框架窗口时创建工具条窗口，可以使得工具条被定位在框架窗口顶部的正确位置上，有趣的是：在父窗口被移动或改变大小时，工具条将自动随父窗口一起移动。MFC 的技术手册告诉程序员这是利用 MFC 内部的私有消息来完成的。

注释

本节的例子程序介绍了如何在视图窗口中添加工具条。但是，此例子程序只是使用了与主窗口同样的工具条位图，许多用户可能更希望在应用程序的视图中使用专门的工具条按钮，因此程序员最好在资源编辑器中创建新的工具条位图，并在视图工具条中使用新的位图及新的命令。

在 Visual Basic 中也能创建和修改工具条，下面便是例子程序的实现步骤。

1. 在 Visual Basic 中创建新的项目文件，新项目文件命名为 toolbar。在此项目文件的表单中添加工具条控制，另外在表单中添加图象列表控制。

2. 在表单中双击并在表单的方法 Form_Load 中添加下列代码：

```
Private Sub Form_Load ()
' Create object variable for the ImageList.
    Dim imgX As ListImage
' Load pictures into the ImageList Control.
    Set imgX = ImageList1.ListImages. _
    Add (, " open", LoadPicture (" bitmaps\tlbr_w95\open.bmp")) '1
    Set imgX = ImageList1.ListImages. _
    Add (, " save", LoadPicture (" bitmaps\tlbr_w95\save.bmp")) '2
    Toolbar1.ImageList = ImageList1

' Create object variable for the Toolbar.
    Dim btnX As Button
    ' Add button objects to Buttons collection using the
    ' Add method. After creating each button, set both
```

```
    ' Description and ToolTipText properties.
    Set btnX = Toolbar1.Buttons.Add (, , , tbrSeparator)
    Set btnX = Toolbar1.Buttons.Add (, " open", , tbrDefault, " open")
    btnX.ToolTipText = " Open File"
    btnX.Description = btnX.ToolTipText
    Set btnX = Toolbar1.Buttons.Add (, " save", , tbrDefault, " save")
    btnX.ToolTipText = " Save File"
    btnX.Description = btnX.ToolTipText
End Sub
```

3. 双击工具条控制，在表单的方法 Toolbar1 _ ButtonClick 中添加下列代码：

```
Private Sub Toolbar1_ButtonClick (ByVal Button As Button)
If Button.Key = " open" Then
        MsgBox " Open Button Selected"
    Else
        If Button.Key = " save" Then
            MsgBox " Save Button Selected"
        End If
    End If
End Sub
```

4. 编译并运行此例子程序。例子程序将显示出有两个按钮的工具条，工具条按钮的功能暗示也将显示出来，点击任一按钮，将弹出一个消息框，指示点击的是哪个按钮。

15.4 实现在运行时修改工具条

问题

有的程序员希望能够让用户按自己的喜好来配置显示在屏幕上的工具条，但是，又不清楚如何才能实现工具条按钮的移进移出，从而使得用户可以配置它们。

那么如何才能实现工具条按用户需求显示呢?

方法

工具条是 Windows 95 应用程序的重要部分，假如没有工具条，用户只好进入一层层的菜单和子菜单来查找所需要的命令。

也正是工具条的可定制功能将真正专业的 Windows 95 应用程序与其他的应用程序区别开来。在本节中，将介绍 Windows 95 工具条的功能，以及如何利用结合到此控制中的 MFC 的强大功能来实现动态工具条（反映用户的喜好)。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示出一个顶部有工具条的视图，如图 15-4 所示。从菜单中选择菜单 Configure 和菜单项 Toolbar，将弹出如图 15-5 所示的用于工具条定制的对话框。选择左边列表框中的第一个选项 (File New) 并点击右向箭头 (→)，选项将移动到右边的列表框中，对左边列表框中的前三个选项重复同样的过程，点击按钮 OK。视图中的工具条将改变成如图 15-6 所示，反映用户的新选择。

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为

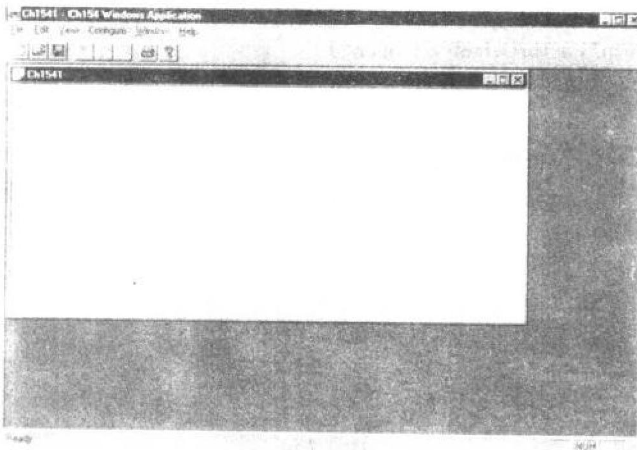


图 15-4 显示标准工具条的视图

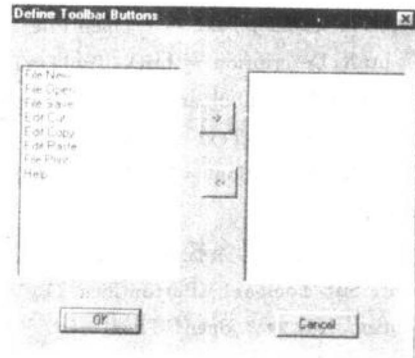


图 15-5 工具条定制对话框

CH154.MAK。

2. 进入资源编辑器并创建新的对话框模板。移动按钮 OK 和 Cancel 到对话框的底部，在对话框顶部的左右两边添加两个新的列表框。

3. 在对话框中的两个列表框之间添加两个按钮，第一个按钮的标题为→，第二个按钮的标题为←，保存对话框模板。

4. 进入 ClassWizard，为刚创建的对话框模板生成新的对话框类，新对话框类命名为 CToolbarDlg。在 ClassWizard 中，从下拉列表中选择类 CToolbarDlg，从对象列表中选择对象 CToolbarDlg，从消息列表中选择消息 WM_INITDIALOG，点击按钮 Add Function，在 CToolbarDlg 的方法 OnInitDialog 中输入下列代码：

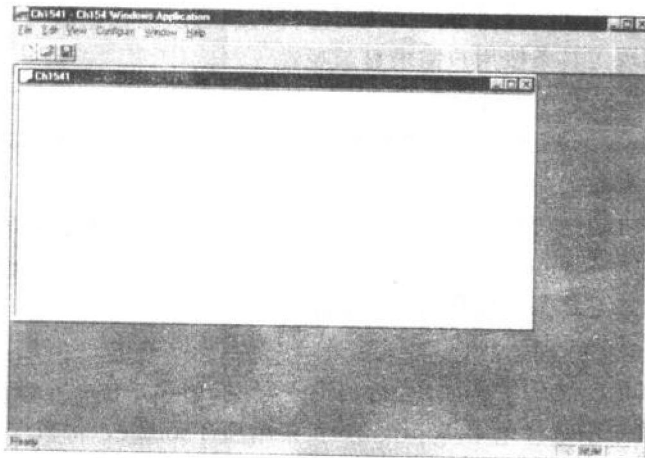


图 15-6 显示新定制工具条（反映用户的选择）的 MFC 视图

```
BOOL CToolbarDlg:: OnInitDialog ()

CDialog:: OnInitDialog ();
```

```
    CListBox *list = (CListBox *) GetDlgItem (IDC_LIST1);

    for ( int i=0; buttons [i] .string [0]; ++i ) {
        list->AddString ( buttons [i] .string );
    }

    CenterWindow ();

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

5. 在ClassWizard中，从对象列表中选择对象IDC_BUTTON1，从消息列表中选择消息BN_CLICKED，点击按钮Add Function，并命名新函数为OnMoveToList。在CToolbarDlg的新方法OnMoveToList中输入下列代码：

```
void CToolbarDlg:: OnMoveToList ()
{
    CListBox *list1 = (CListBox *) GetDlgItem (IDC_LIST1);
    CListBox *list2 = (CListBox *) GetDlgItem (IDC_LIST2);

    // See if there is anything selected

    if ( list1->GetCurSel () == LB_ERR ) {
        MessageBeep (0);
        return;
    }

    // Yes. Get the selection string

    CString sel_string;

    list1->GetText ( list1->GetCurSel (), sel_string );

    // Append it to the second list box

    list2->AddString ( sel_string );

    // Remove it from the first list box

    list1->DeleteString ( list1->GetCurSel () );
}
```

6. 在ClassWizard中，从对象列表中选择对象IDC_BUTTON2，从消息列表中选择消息BN_CLICKED，点击按钮Add Function，并命名新函数为OnMoveFromList。在CToolbarDlg的新方法OnMoveFromList中输入下列代码：

```
void CToolbarDlg:: OnMoveFromList ()
{
    CListBox *list1 = (CListBox *) GetDlgItem (IDC_LIST1);
    CListBox *list2 = (CListBox *) GetDlgItem (IDC_LIST2);

    // See if there is anything selected

    if ( list2->GetCurSel () == LB_ERR ) {
        MessageBeep (0);
        return;
    }

    // Yes. Get the selection string

    CString sel_string;

    list2->GetText ( list2->GetCurSel (), sel_string );

    // Append it to the second list box

    list1->AddString ( sel_string );

    // Remove it from the first list box

    list2->DeleteString ( list2->GetCurSel () );
}
```

7. 在 ClassWizard 中，从对象列表选择对象 IDOK，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function，并命名新函数为 OnOk。在 CToolbarDlg 的新方法 OnOk 中输入下列代码：

```
void CToolbarDlg:: OnOK ()
{
    // Initialize flags array
    for ( int k=0; k<MAX_TOOLBAR_ENTRIES; ++k )
        flags [k] = FALSE;

    // Get all of the selected items and check them in our list

    CListBox *list2 = (CListBox *) GetDlgItem (IDC_LIST2);
    CString string;

    for ( int i=0; i<list2->GetCount (); ++i ) {
```

```
    // Get the string

    list2->GetText ( i, string );
    // See which it is in our list
    int idx = -1;
    for ( int j=0; buttons [j] .string [0]; ++j )
        if ( ! strcmp (string, buttons [j] .string) )
            idx = j;
    // Set that flag to TRUE
    flags [idx] = TRUE;
    }

    CDialog:: OnOK ();
}
```

8. 在类 CToolbarDlg 的源文件 TOOLBARD.CPP 中添加下列新方法：

```
BOOL CToolbarDlg:: GetToolFlag (int cmd)
{
    for ( int i=0; buttons [i] .string [0]; ++i )
        if ( buttons [i] .id == cmd )
            return flags [i];
    return FALSE;
}
```

9. 在类 CToolbarDlg 的头文件 TOOLBARD.H 中添加下列行，添加的行用黑体表示。

```
const MAX_TOOLBAR_ENTRIES = 9;

class CToolbarDlg : public CDialog
{
private:
    BOOL flags [MAX_TOOLBAR_ENTRIES];
public:
    BOOL GetToolFlag (int cmd);
```

10. 进入资源编辑器，为 MDI 子窗口选择菜单 IDR_CH154TYPE。添加标题为 Configure 的新菜单，在菜单 Configure 中添加新的菜单项，标题为 Toolbar，保存资源文件并退出资源编辑器。

11. 在 ClassWizard 中，选择类 CMainFrame。从对象列表中选择对象 ID_CONFIGURE_TOOLBAR，从消息列表中选择消息 COMMAND，点击按钮 Add Function，并命名新方法为 OnConfigureToolbar。在 CMainFrame 的新方法 OnConfigureToolbar 中输入下列代码：

```
void CMainFrame:: OnConfigureToolbar ()
{
    CToolbarDlg dlg;
    int         id = 0;
    if ( dlg.DoModal () == IDOK ) {
```

```
    int which = 0;

    for ( int i=0; i< (sizeof (buttons) /sizeof (buttons [0])); ++i ) {
      if ( dlg.GetToolFlag (buttons [i]) == TRUE )
      m_wndToolBar.SetButtonInfo ( id++, buttons [i], TBBS_BUTTON, which );
       if ( buttons [i] ! = ID_SEPARATOR )
       which ++;
      }
      for ( i=id; i< (sizeof (buttons) /sizeof (buttons [0])); ++i )
      m_wndToolBar.SetButtonInfo ( i, 0, TBBS_SEPARATOR, 12 );
      m_wndToolBar.InvalidateRect (NULL);
  }
}
```

12. 编译并运行此例子程序。

用法

MFC 工具条控制是 Windows 95 响应某些命令的简单控制，其中一个命令可以用来设置和重新设置显示的按钮。工具条有两种基本类型的按钮。一种是命令按钮，包含图象和命令 ID，当用户点击命令按钮时，当前窗口通过消息 WM_COMMAND 发送命令 ID。另一种工具条按钮是分隔符，只占用空间但不响应用户的点击操作。

在本节的例子程序中，用户通过选取自己希望在工具条中显示的工具条按钮来配置工具条。当用户在配置对话框中点击按钮 OK 时，一个标志数组就被创建。如果选项是用户希望显示在工具条中的，则相应的标志为 TRUE；如果是不希望显示在工具条中的选项，则相应的标志为 FALSE。然后主窗口从对话框中查询标志数组，设置每个工具条按钮为工具条按钮（希望显示的）或分隔符（不希望显示的）。

注释

在本节的例子程序中，提供了一个定制工具条的简单方法，但此方法决不是完善的。首先，工具条按钮仍保持了与原设计同样的顺序，而最初级的定制功能也应该允许用户不仅可以选择自己希望显示的选项，而且也应该可以选择自己希望显示选项的顺序。

另一个可能的改进是在工具条选择对话框中实现自绘制列表框，不仅包含命令名，还应包含在工具条按钮中实际使用的图象。

最后，在定制对话框的两个列表框之间如果能够实现拖放界面，则是非常理想的。应该指出，可定制的美观的用户界面是一个优秀的 Windows 95 应用程序的重要标志。

15.5 从在线帮助中显示范例

问题

有的程序员希望能够使用户在应用程序的帮助文件中也能与应用程序进行交互，从而实现一个向导式（Wizardlike）的应用程序，利用帮助文件来驱动应用程序的运行。

是否有方法利用 Windows API 函数来实现帮助文件与应用程序之间的通信呢？

方法

Windows 帮助系统是一个非凡的创造，在用户和帮助信息之间提供了标准的界面，使得

应用程序开发者可以集中精力于帮助信息的内容，而不是其表现形式。但是，在有些情况下还需要一些特殊的功能，这些功能超出了 Windows 系统所提供的标准帮助命令。本节所讨论的问题就属于这么一种情况。

在这里我们准备实现从 Windows 帮助系统中调用外部的程序，通过在 DLL 中定义外部函数这是可以实现的。在本节中，将一步步地介绍如何实现此功能。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Help 和菜单项 Index，将显示出如图 15-7 所示的帮助窗口。点击加亮文本 Click Here To Run Dialog（需要将编译生成的文件 MYHELP.DLL 移动到 WINDOWS\SYSTEM 目录下），在例子程序的主窗口中将显示出一个对话框，如图 15-8 所示。当然，从例子程序的菜单中选择 Dialog|Show Dialog 也能将对话框显示出来。

图 15-7 例子程序的帮助窗口

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH155.MAK。

2. 进入资源编辑器并创建新的对话框。在对话框中，添加一个静态文本域，添加一个编辑域，以及三个单选按钮。静态文本域的标题为 Input String #1，单选按钮的标题分别为 Customer Type1、Customer Type2 和 Customer Type3。

3. 选择 ClassWizard，为刚创建的对话框模板生成新的对话框类，新类命名为 CMyDialog。

4. 在资源编辑器中，选择菜单资源 IDR_MAINFRAME。添加标题为 Dialog 的新菜单，

在此菜单中添加新的菜单项，标题为 Show Dialog，标识符为 ID _ SHOW _ DIALOG。

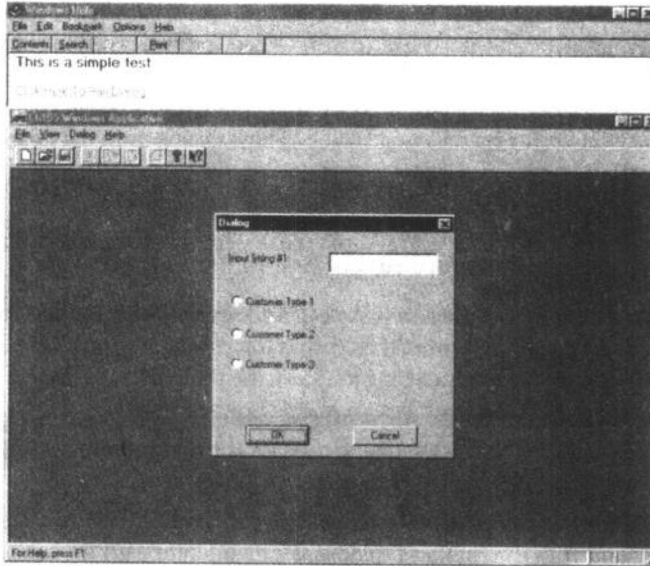


图 15-8　在帮助窗口中显示的对话框

5. 在 ClassWizard 中，选择应用程序对象 CCh155App，从对象列表中选择对象 ID _ SHOW _DIALOG，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新函数命名为 OnShowDialog。在 CCh155App 的方法 OnShowDialog 中输入下列代码：

```
void CCh155App:: OnShowDialog ()
{
    CMyDialog dlg;

    dlg.DoModal ();
}
```

6. 在 CH155.CPP 顶部的 include 文件列表中添加下列行：

```
#include " mydialog.h"
```

7. 从 CCh155App 的方法 InitInstance 中删除函数 OnFileNew 的调用，以避免例子程序打开新的视图。

8. 保存并编译例子程序 CH155.MAK。

9. 接着，创建新的项目文件 MYHELP.MAK。选择菜单 Project|New 来创建项目文件，并使其为 Windows 动态连接库。

10. 在 Visual C++中添加新文件，并在此文件中输入下列代码，保存此文件为 LIBMAIN.CPP。

```
#define STRICT
#include <windows.h>

// Turn off warning: Parameter " is never used
```

```
#ifdef _BORLANDC_
#pragma argsused
#endif

BOOL DllEntryPoint (
    HANDLE      hDLL,
    DWORD       dwReason,
    LPVOID      lpReserved)
{
    switch (dwReason)
        {
        case DLL_PROCESS_ATTACH:
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        default:
            break;
        }
    return TRUE;
}
```

11. 最后，添加第三个新文件并在此文件中输入下列代码，保存此文件为 SHOWDIAL. CPP。

```
#define STRICT
#include " windows.h"
extern " C" {
BOOL ShowDialog (char FAR *str)
{
    HWND hWnd = FindWindow (NULL, " Ch155 Windows Application");
    if ( hWnd )
        :: PostMessage (hWnd, WM_COMMAND, 32771, 0L );
    return TRUE;
}
}
```

12. 将 LIBMAIN. CPP 和 SHOWDIAL. CPP 添加到工程中，编译此 DLL。

13. 创建新文件 CH155. RTF，在此文件中输入下列文本：

```
{\rtf1\ansi \deff0
{\fonttbl {\f0\froman Tms Rmn;} {\f1\fdecor Symbol;} {\f2\fswiss Helv;}
{\f3\fmodern Courier; \f4\fswiss MS Sans Serif; \f5\fswiss Helvitica;}
{\f6\fswiss Arial; \f7\fswiss Arial Super; \f8\fswiss MS Serif;}
{\f9\froman Times; \f10\froman Times New Roman;}
```

```
}
{\colortbl;

\red0\green0\blue127;
\red0\green127\blue0;
\red0\green127\blue127;
\red127\green0\blue0;
\red127\green0\blue127;
\red127\green127\blue0;
\red127\green127\blue127;

\red192\green192\blue192;
\red0\green0\blue255;
\red0\green255\blue0;
\red0\green255\blue255;
\red255\green0\blue0;
\red255\green0\blue255;
\red255\green255\blue0;
\red255\green255\blue255;}
\f2\fs20
{# {\footnote # Test}
K {\footnote K test; contents}
$ {\footnote $ Test}
+ {\footnote + General}
{\b \fs24 This is a simple test} \par\pard
\par { {\uldb Click Here To Run Dialog} {\v ! ShowDialog (" MyDialog")}
} \par\pard
} \page
}
```

14．接着，创建新文件CH155.HPJ（Windows帮助项目文件），并在此文件中添加下列文本：

```
[OPTIONS]
  ERRORLOG = error.log
  COMPRESS = No
[BUILDTAGS]

[FILES]
  CH156.RTF

[BITMAPS]

[ALIAS]
```

```
[CONFIG]
  BrowseButtons ()
      RegisterRoutine (" myhelp.dll", " ShowDialog", " S" )
[WINDOWS]
[BAGGAGE]
```

15. 利用文件 CH155.RTF 和 CH155.HPJ 创建新的 Windows 帮助文件。

16. 运行此例子程序。

用法

在应用程序启动帮助文件时，WinHelp（Windows 帮助系统）查找注册函数。这些函数在帮助项目文件（.HPJ）的 CONFIG 部分可以找到，同时列出的还有函数所在的 DLL 以及调用函数的参数。

在帮助文件中，函数是作为 WinHelp 的宏来调用的。Windows 帮助系统接着遍历定义函数的列表来找到与此函数相匹配的函数，并试着从 DLL 中装入函数，一旦装入成功，则函数就被调用。

在本节的例子程序中，函数 ShowDialog 首先查找例子程序的主窗口，此函数知道例子程序主窗口的标题为 CH155 Windows Application，一旦找到此窗口，此函数传递（不是发送）消息给窗口，表示命令已被请求，从而可以显示出对话框。

注释

显然，这是一个通过 Windows 帮助文件调用应用程序功能的简单例子，但同时也是一个功能强大的方法。使用此方法，可以通过外部的帮助文件来执行应用程序的任何外置的功能，例如，可以用来运行示例对话框、创建显示、显示图形，或其他任何可以通过消息 COMMAND 来调用的功能。

除了调用外部的程序，还可以考虑为 WinHelp 创建基于 DLL 的函数。内部参数使得程序员可以指定帮助窗口的窗口句柄，从而可以显示动画、改变背景、或需要在帮助系统中实现的任何功能。

15.6 在应用程序启动时显示 About 框

问题

有的程序员希望能够在应用程序装入前以及显示主窗口后的几秒钟内显示 About 框，这样的实现在专业 Windows 应用程序中比比皆是。但在一般的应用程序中也需要实现此功能，而且还需要 About 框在显示一定时间后自动消失，或者在用户点击应用程序的主窗口后消失。

那么，如何利用 Windows API 函数来创建满足这些要求的 About 框呢？

方法

在程序开始前的几秒钟里显示 About 框的思想并非是新想法，此思想同样用于显示扉屏，即在程序装入时显示彩色图画给用户观看。但在本节中，只关心如何显示文本信息，如版本信息或所有权信息。

要创建自动消失的 About 框，实际上需要创建无模式对话框。所谓无模式对话框就是没

有属主和非“模式”的对话框，即在对话框显示的同时其他应用程序照常可以运行。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示如图 15-9 所示的 About 框。在主窗口出现时点击主窗口用户区中的任何地方，对话框将消失，如果用户等待 5s，对话框也会消失的。

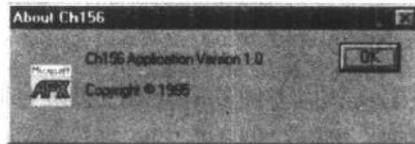


图 15-9 启动例子程序时显示的 About 框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名新的项目文件为 CH156. MAK。

2. 进入 ClassWizard，点击按钮 Add Class。新类命名为 CSplash，并选择类 CDialog 作为此类的基类，忽略关于没有为此类定义对话框模板标识符的警告信息，生成新类。

3. 在 ClassWizard 中，从下拉列表中选择类 CSplash，从对象列表中选择对象 CSplash，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Function，在 CSplash 的方法 OnInitDialog 中添加下列代码：

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

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

4. 在文件 SPLASH. CPP 中添加下列代码：

```
BOOL CSplash:: Create (CWnd * pParent)
{
    // { {AFX_DATA_INIT (CSplashWnd)
      // NOTE: the ClassWizard will add member initialization here
    //}} AFX_DATA_INIT

    if (! CDialog:: Create (CSplash:: IDD, pParent))
    {
      TRACE0 (" Warning: creation of CSplashWnd dialog failed\n");
      return FALSE;
    }
    return TRUE;
}
```

5. 在头文件 SPLASH. H 中添加或修改下列用黑体表示的行：

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

```
    ~CSplash ();                    // standard destructor
    BOOL Create (CWnd * pParent);

// Dialog Data
    //{{AFX_DATA (CSplash)
    enum { IDD = IDD_ABOUTBOX };
        // NOTE: the ClassWizard will add data members here
    //}} AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange (CDataExchange * pDX); // DDX/DDV support

    // Generated message map functions
    //{{AFX_MSG (CSplash)
    virtual BOOL OnInitDialog ();
        // NOTE: the ClassWizard will add member functions here
    //}} AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
```

6. 从项目列表中选择文件CH156.H，在CCh156App的类定义中添加下列行。添加的行同样用黑体表示。

```
#include "splash.h"

class CCh156App : public CWinApp
{
private:
    CSplash m_splash;
    DWORD m_dwSplashTime;
public:
    CCh156App ();

// Overrides
    virtual BOOL InitInstance ();
    virtual BOOL OnIdle (LONG lCount);
    virtual BOOL PreTranslateMessage (MSG * pMsg);
```

7. 从项目列表中选择源文件CH156.CPP，在类CCh156App的方法InitInstance中做下列修改（黑体表示）：

```
BOOL CCh156App:: InitInstance ()
{
    if (m_splash.Create (m_pMainWnd)) {
        m_splash.ShowWindow (SW_SHOW);
```

```
    m_splash.UpdateWindow ();
}
m_dwSplashTime = ::GetCurrentTime ();

// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need.

SetDialogBkColor (); // Set dialog background color to gray
LoadStdProfileSettings (); // Load standard INI file options (including MRU)
// Register the application's document templates. Document templates
// serve as the connection between documents, frame windows and views.

CMultiDocTemplate * pDocTemplate;
pDocTemplate = new CMultiDocTemplate (
    IDR_CH156TYPE,
    RUNTIME_CLASS (CCh156Doc),
    RUNTIME_CLASS (CMDIChildWnd), // standard MDI child frame
    RUNTIME_CLASS (CCh156View));
AddDocTemplate (pDocTemplate);

// create main MDI Frame window
CMainFrame * pMainFrame = new CMainFrame;
if (! pMainFrame->LoadFrame (IDR_MAINFRAME))
   return FALSE;
m_pMainWnd = pMainFrame;

// create a new (empty) document
OnFileNew ();

if (m_lpCmdLine [0] != '\0')
{
   // TODO: add command line processing here
}
// The main window has been initialized, so show and update it.
pMainFrame->ShowWindow (m_nCmdShow);
pMainFrame->UpdateWindow ();
m_splash.BringWindowToTop ();

return TRUE;
}
```

8. 在源文件 CH156.CPP 中添加下列函数：

```
BOOL CCh156App:: OnIdle (LONG lCount)
{
    // call base class idle first
    BOOL bResult = CWinApp:: OnIdle (lCount);

    // then do our work
    if (m_splash.m_hWnd != NULL) {
        if (:: GetCurrentTime () - m_dwSplashTime > 5000) {
            // timeout expired, destroy the splash window
            m_splash.DestroyWindow ();
            m_pMainWnd->UpdateWindow ();

            // NOTE: don't set bResult to FALSE,
            // CWinApp:: OnIdle may have returned TRUE
        }
        else {
            // check again later...
            bResult = TRUE;
        }
    }
    return bResult;
}
```

9. 在源文件CH156.CPP中添加下列函数：

```
BOOL CCh156App:: PreTranslateMessage (MSG * pMsg)
{
    BOOL bResult = CWinApp:: PreTranslateMessage (pMsg);

    if (m_splash.m_hWnd != NULL &&
        (pMsg->message == WM_KEYDOWN ||
        pMsg->message == WM_SYSKEYDOWN ||
        pMsg->message == WM_LBUTTONDOWN ||
        pMsg->message == WM_RBUTTONDOWN ||
        pMsg->message == WM_MBUTTONDOWN ||
        pMsg->message == WM_NCLBUTTONDOWN ||
        pMsg->message == WM_NCRBUTTONDOWN ||
        pMsg->message == WM_NCMBUTTONDOWN)) {
    m_splash.DestroyWindow ();
    m_pMainWnd->UpdateWindow ();
}
    return bResult;
}
```

10. 编译并运行此例子程序。

用法

当例子程序启动时，在例子程序对象类的方法 InitInstance 中创建无模式对话框（用作 About 框）。一旦此对话框被创建，则在创建主窗口时将被自动置于窗口的前面，接着显示在那里直到用户等待 5s（5 * 1000ms）或者点击鼠标，或者按下某键后才消失。

如果用户点击鼠标或按下某键，则调用例子程序对象的方法 PreTranslateMessage，此方法检查 About 框窗口是否存在，如果存在，则撤消此窗口，更新主窗口并使程序继续执行。

如果用户等待 5s，则调用方法 OnIdle，每当系统有空闲时间来执行例子程序的空闲处理时就调用此方法。在本节的例子程序中，方法 OnIdle 用来检查撤消窗口的时间是否到了，如果到了，则撤消 About 框窗口，而程序继续执行。

注释

使用 Delphi 也能完成同样的任务，下面便是实现的步骤。

1. 在 Delphi 中创建新的项目文件。从工具条中选择按钮 New Form，从表单库中选择表单 About Box。

2. 双击项目中的第一个表单（form1），在方法 TForm1. FormCreate 中添加下列代码：

```
procedure TForm1. FormCreate (Sender: TObject);
begin
  SetWindowPos (AboutBox. handle, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOSIZE Or SWP_
NOMOVE);
end;
```

3. 使用 Object Inspector，为 Form1 的方法 OnClick 添加一个方法，新方法命名为 DoClick。在 TForm1 的方法 DoClick 中添加下列代码：

```
procedure TForm1. DoClick (Sender: TObject);
begin
  if AboutBox <> Nil then
    begin
      AboutBox. Hide;
      AboutBox. Free;
      AboutBox := Nil;
    end;
end;
```

4. 使用 Object Inspector，为消息 OnKeyDown 添加新的方法，新方法命名为 DoKeyDown。在 TForm1 的方法 DoKeyDown 中添加下列代码：

```
procedure TForm1. DoKeyDown (Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if AboutBox <> Nil then
    begin
      AboutBox. Hide;
      AboutBox. Free;
      AboutBox := Nil;
    end;
end;
```

5. 在表单 TForm1 中添加一个计时器对象，在 Object Inspector 中为 Form1 选择方法

OnTimer，并创建新的方法 DoTimer 来处理此消息。在 TForm1 的方法 DoTimer 中添加下列代码：

```
procedure TForm1.DoTimer (Sender; TObject);
begin
    if AboutBox <> Nil then
      begin
          AboutBox.Hide;
          AboutBox.Free;
          AboutBox := Nil;
          Timer1.Enabled := False;
      end;
end;
```

6. 从主菜单中选择 View|Project Source 来编辑工程源文件。修改工程源文件如下，修改或添加的行用黑体表示。

```
program Splash
uses
    Forms,
    Splsh in 'SPLSH.PAS' {Form1},
    Splsh2 in 'SPLSH2.PAS' {AboutBox},
    Winprocs;
{$R *.RES}
begin
    AboutBox := TAboutBox.Create (Application);
    AboutBox.Show;
    AboutBox.Update;
    Application.CreateForm (TForm1, Form1);
    Application.Run;
end
```

7. 编译并运行此例子程序。

15.7 显示扉屏 (splash screen)

问题

有的程序员希望在自己的应用程序启动时显示扉屏。用户希望的是一幅漂亮的位图，此位图显示几秒钟并在程序开始时消失，但有的程序员还希望能够让用户使用鼠标点击或击键来取消此屏幕。

方法

扉屏已成为专业应用程序的用户界面整体的一部分，用户也已经习惯看到此屏幕，有时还希望看到一幅漂亮的图形或位图。

在本节中，将讨论图形位图函数、窗口创建函数以及其他几个 Windows 95 API 函数。本节在 Visual C++中生成一个通用的扉屏窗口类，能用来显示存放在硬盘上.BMP 文件中的任何位图。另外，在本节中，还将介绍如何找出位图的大小以及在需要时如何伸展位图（变

大或变小)。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，将显示如图 15-10 所示的扉屏，显示的是 Windows 商标。点击扉屏窗口中的任何地方或主窗口的用户区，则扉屏窗口将消失，如果用户等待 5s，扉屏窗口也会自动消失。

注：如果你使用的是 Visual C++ 4.0，则组件库中包含着可用在你的应用程序中的 Splash Screen 组件，推荐使用 Visual C++ 4.0 中的组件。

实现例子程序的具体步骤如下：

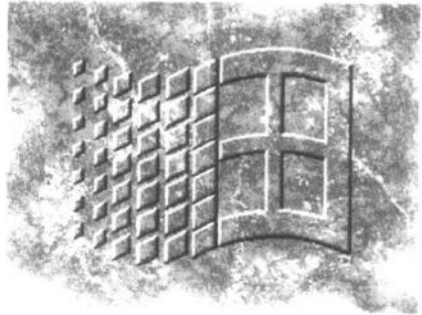


图 15-10　显示 Windows 商标的扉屏

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名新项目文件为 CH157.MAK。

2. 进入 ClassWizard，点击按钮 Add Class。新类命名为 CSplashWnd，并选择 CWnd 为此类的基类，生成此新类。

3. 在 ClassWizard 中，从下拉列表中选择类 CSplashWnd，从对象列表中选择对象 CSplashWnd，从消息列表中选择消息 WM_PAINT，点击按钮 Add Function，在 CSplashWnd 的方法 OnPaint 中添加下列代码：

```
void CSplashWnd::OnPaint ()
{
    CPaintDC dc (this); // device context for painting
    CDC memDC;
    memDC.CreateCompatibleDC (&dc);
    CBitmap * pOld = memDC.SelectObject (bitmap);
    if (pOld == NULL)
        return; // destructors will clean up
    dc.StretchBlt (0, 0, m_winWidth, m_winHeight, &memDC, 0, 0, m_wndWidth,
                m_wndHeight, SRCCOPY);
    memDC.SelectObject (pOld);
    // Do not call CWnd::OnPaint () for painting messages
```

4. 接着，为了创建窗口在类 CSplashWnd 中添加新方法。在文件 SPLSHWND.CPP 中添加下列代码：

```
BOOL CSplashWnd::Create ()

    BITMAP bm;
    bitmap->GetObject (sizeof (BITMAP), &bm);
    // Get the size of the splash window
    m_wndWidth = bm.bmWidth;
    m_wndHeight = bm.bmHeight;
    // Get the size of the screen
```

```
    int screenWidth = GetSystemMetrics (SM_CXSCREEN);
    int screenHeight = GetSystemMetrics (SM_CYSCREEN);

    // If they didn't give us a size, use the bitmap size.

    if ( m_winHeight == 0.0 )
        m_winHeight = m_wndHeight;
    if ( m_winWidth == 0.0 )
        m_winWidth = m_wndWidth;

    // get top/left coord to center the splash window
    int top = (screenHeight - m_winHeight) /2;
    int left = (screenWidth - m_winWidth) /2;

    return CWnd::CreateEx(WS_EX_TOPMOST, "AfxWnd", "", WS_POPUP | WS_VISIBLE,left,
top, m_winWidth, m_winHeight, NULL, NULL);
}
```

5. 在类 CSplashWnd 的构造函数（CSplashWnd:: CSplashWnd）中添加下列代码：

```
CSplashWnd:: CSplashWnd (int resId, double width_ratio, double height_ratio)
{
    bitmap = new CBitmap;
    bitmap-->LoadBitmap ( resId );

    // Get the size of the screen
    int screenWidth = GetSystemMetrics (SM_CXSCREEN);
    int screenHeight = GetSystemMetrics (SM_CYSCREEN);

    m_winWidth = (int) ( (double) screenWidth * width_ratio);
    m_winHeight = (int) ( (double) screenHeight * height_ratio);}
```

6. 接着，在类 CSplashWnd 中添加析构函数（CSplashWnd:: ~CSplashWnd），即添加下列代码：

```
CSplashWnd::~ CSplashWnd ()
{
    delete bitmap;
}
```

7. 选择文件 SPLSHWND.H 并做如下修改，修改的行用黑体表示。

```
#ifndef _SPLSHWND_H_
#define _SPLSHWND_H_
/////////////////////////////////////////////////////////////////////////////
//CSplashWnd window

class CSplashWnd : public CWnd
```

```
{
private:
    CBitmap *bitmap;

// Construction
public:
    CSplashWnd (int resId, double width_ratio, double height_ratio);

// Attributes
public:
    virtual BOOL Create ();

// Operations
public:

// Implementation
public:
    virtual ~CSplashWnd ();

protected:
    int m_wndWidth;
    int m_wndHeight;
    int m_winWidth;
    int m_winHeight;

    // Generated message map functions
    //{{AFX_MSG(CSplashWnd)
    afx_msg void OnPaint ();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
#endif
```

8. 从项目列表中选择文件 CH157.CPP 和类 CCh157App，修改类的构造函数(CCh157App::CCh157App)如下，修改的行用黑体表示。

```
CCh157App::CCh157App ()
{
    m_pwndSplash = NULL;
}
```

9. 接着，修改 CCh157App 的方法 InitInstance 如下，添加的所有行用黑体表示。

```
BOOL CCh157App::InitInstance ()
{
    m_dwSplashTime = ::GetCurrentTime ();
```

```
    m_pwndSplash = new CSplashWnd ( "C:\windows\winlogo.bmp", 0.8, 0.8);
    if ( m_pwndSplash->Create () == FALSE ) {
        delete m_pwndSplash;
        m_pwndSplash = NULL;
    }
    else {
        m_pwndSplash->ShowWindow (SW_SHOW);
        m_pwndSplash->UpdateWindow ();
        ASSERT (m_pwndSplash != NULL);
    }

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

    SetDialogBkColor (); // Set dialog background color to gray
    LoadStdProfileSettings (); // Load standard INI file options (including MRU)
```

10. 在类中添加新方法 OnIdle，并在此方法中添加下列代码：

```
BOOL CCh157App::OnIdle (LONG lCount)
{
    // call base class idle first
    BOOL bResult = CWinApp::OnIdle (lCount);

    // then do our work
    if (m_pwndSplash != NULL)
    {
        if (m_pwndSplash->m_hWnd != NULL)
        {
          if (::GetCurrentTime () - m_dwSplashTime >= 50000)
          {
              // timeout expired, destroy the splash window
              m_pwndSplash->DestroyWindow ();
              delete m_pwndSplash;
              m_pwndSplash = NULL;
              m_pMainWnd->UpdateWindow ();
              // NOTE: don't set bResult to FALSE,
              // CWinApp::OnIdle may have returned TRUE
          }
          else
          {
              // check again later...
              bResult = TRUE;
```

```
        }
      }
  }
  return bResult;
}
```

11. 在类中添加新方法 PreTranslateMessage，并在此方法中添加下列代码：

```
BOOL CCh157App:: PreTranslateMessage (MSG * pMsg)
{
  BOOL bResult = CWinApp:: PreTranslateMessage (pMsg);
  if (m _pwndSplash ! = NULL)
  {
     if (pMsg->message == WM _KEYDOWN ||
        pMsg->message == WM _SYSKEYDOWN ||
        pMsg->message == WM _LBUTTONDOWN ||
        pMsg->message == WM _RBUTTONDOWN ||
        pMsg->message == WM _MBUTTONDOWN ||
        pMsg->message == WM _NCLBUTTONDOWN ||
        pMsg->message == WM _NCRBUTTONDOWN ||
        pMsg->message == WM _NCMBUTTONDOWN)
     {
        if (pMsg->hwnd == m _pwndSplash->m _hWnd) m _dwSplashTime -= 5000;
        else
        {
          m _pwndSplash->DestroyWindow ();
          delete m _pwndSplash;
          m _pwndSplash = NULL;
          m _pMainWnd->UpdateWindow ();
        }
     }
  }
  return bResult;
}
```

12. 在源文件 CH157.CPP 的顶部添加下列 include 文件行：

```
#include " splshwnd.h"
```

13. 修改类 CCh157App 的头文件如下，添加或修改的行用黑体表示。

```
// ch157.h : main header file for the CH157 application
//
#ifndef __AFXWIN _H __
   #error include 'stdafx.h' before including this file for PCH
#endif
#include " resource.h" // main symbols
/////////////////////////////////////////////////////////////////////////////
```

```
// CCh157App:
// See ch157.cpp for the implementation of this class
//

class CSplashWnd;

class CCh157App : public CWinApp
{
private:
    CSplashWnd *m_pwndSplash;
    DWORD m_dwSplashTime;
public:
    CCh157App ();

// Overrides
    virtual BOOL InitInstance ();
        virtual BOOL OnIdle (LONG lCount);
        virtual BOOL PreTranslateMessage (MSG * pMsg);
// Implementation
    //{{AFX_MSG (CCh157App)
    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 ()
};
//////////////////////////////////////////////////////////////////////////////
```

14. 编译并运行此例子程序。

用法

当启动例子程序时，例子程序创建类 CSplashWnd 的一个实例。函数 PreTranslateMessage 和 OnIdle 则用来检查用户是否按下键、点击鼠标或时间到了(在本节的例子程序中为五秒钟)，以便撤消窗口。

窗口实际上是在类 CSplashWnd 中创建的。构造函数的高度比和宽度比参数用来确定位图占据屏幕的大小，如果这两个参数设置为 0.0，则以位图的原大小来显示位图。方法 Create 首先从可执行程序的资源中装入位图。

一旦装入位图，窗口就以正确的大小被创建（根据位图的大小，或是用户指定的屏幕大小的比率)，位图窗口被置中，并且利用 API 函数 CreateEx 来创建真正的 Windows 95 窗口。

当窗口接收消息 Paint 后，位图被选入窗口的设备描述表，接着将设备描述表绘制到屏幕上，位图以适当的比率被放大或缩小以显示在窗口中。

当所有这些操作结束后，窗口就显示在屏幕上，直到五秒钟过去或用户点击鼠标、或用户按下键后才消失。

15.8 确定应用程序的图标

问题

有的程序员希望能够确定自己的应用程序（或其他的应用程序）在极小化时显示的图标是哪一个，在 Windows 95 中，应用程序在屏幕上显示时的图标和极小化在任务条时的图标有可能是不一样的。

如何利用 Windows 95 API 函数来确定应用程序在不同情况时所显示的是哪一个图标呢？

方法

在 Windows 95 中有两个新的概念，即“大图标”和“小图标”。当应用程序运行时，大图标显示在窗口的左上角，包含 Windows 95 窗口的系统菜单。实际上，在 Windows 95 中还支持第三个图标，即更大的图标（48×48），当应用程序在高分辨率显示器上显示时使用此图标，此时“大”图标就成了“中”图标。

但是，在应用程序极小化时，显示在任务条上的紧挨应用程序名的图标则是另外一个图标，尽管这些图标通常是相同的（或相似的），但它们可以是不同的。

新的 API 函数 SHGetFileInfo 只用在 Windows 95 中，可以用来取回应用程序或动态连接库的图标信息。在本节中，将介绍如何利用 SHGetFileInfo 来获取任一可执行程序的图标信息。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Dialogs 和菜单项 View Program Icons。在弹出的对话框中，键入一个可执行文件名并选择按钮 View，则在此对话框中将显示出此可执行文件的大图标和小图标，如图 15-11 所示。

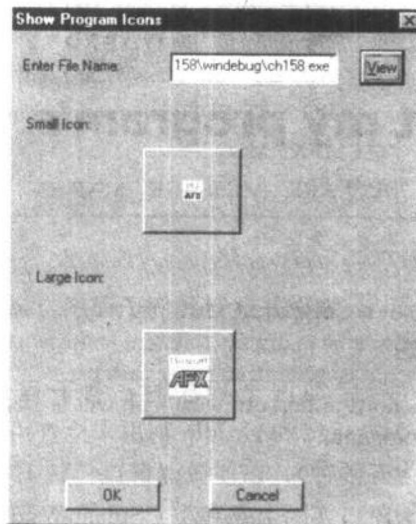


图 15-11　显示应用程序的大图标和小图标的对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH158.MAK。

2. 在资源编辑器中创建新的对话框。添加标题为 Enter File Name：的静态文本域，紧挨静态文本域创建一个编辑域，并在编辑域的右边添加标题为 &View 的按钮。

3. 在对话框中添加两个按钮，标识符分别为 IDC_BUTTON2 和 IDC_BUTTON3，进入 ClassWizard，为刚创建的对话框模板定义对话框类，新类命名为 CPgmIconDlg。

4. 进入 ClassWizard，从下拉列表中选择 CPgmIconDlg，从对象列表中选择对象 IDC_BUTTON1，从消息列表中选择消息 BN_CLICKED，点击按钮 Add Function，新方法命名为 OnViewIcons。在 CPgmIconDlg 的方法 OnViewIcons 中添加下列代码：

```
void CPgmIconDlg:: OnViewIcons ()
```

```
{
    HICON hIconSmall, hIconLarge;
    HIMAGELIST hSysImageList;
    SHFILEINFO shfi;
    char buffer [_MAX_PATH];

    // Get the file name from the edit box

    GetDlgItem (IDC_EDIT1) ->GetWindowText ( buffer, _MAX_PATH );
    if ( ! strlen (buffer) ) {
        MessageBox (" You must enter a file name!", " Error", MB_OK );
        return;
    }
    hSysImageList = (HIMAGELIST) SHGetFileInfo (buffer,
                        0,
                        &shfi,
                        sizeof (SHFILEINFO),
                        SHGFI_SYSICONINDEX | SHGFI_SMALLICON);
    if (hSysImageList)
    {
    hIconSmall = ImageList_GetIcon (hSysImageList,
                                    shfi.iIcon,
                                    ILD_NORMAL);
    // Set the icon in the dialog

    :: SendDlgItemMessage (m_hWnd, IDC_BUTTON2, BM_SETIMAGE,
                        (WPARAM) IMAGE_ICON, (LPARAM) hIconSmall);
    }
    else
    {
    // SHGetFileInfo failed...
    }

    hSysImageList = (HIMAGELIST) SHGetFileInfo (buffer,
                        0,
                        &shfi,
                        sizeof (SHFILEINFO),
                        SHGFI_SYSICONINDEX | SHGFI_LARGEICON);
    if (hSysImageList)
    {
    hIconLarge = ImageList_GetIcon (hSysImageList, shfi.iIcon, ILD_NORMAL);
    // Set the icon in the dialog
```

```
    ::SendDlgItemMessage (m_hWnd, IDC_BUTTON3, BM_SETIMAGE,
                          (WPARAM) IMAGE_ICON, (LPARAM) hIconLarge);
    }
    else
    {
    // SHGetFileInfo failed...
    }
}
```

5. 在资源编辑器中添加标题为 Dialogs 的新主菜单。在菜单 Dialogs 中添加新的菜单项，标题为 View Program Icons，标识符为 ID_VIEW_PGM_ICONS。

6. 进入 ClassWizard，从下拉列表中选择对象 CCh158App，从对象列表中选择对象 ID_VIEW_PGM_ICONS，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新函数命名为 OnViewPgmIcons。在 CCh158App 的方法 OnViewPgmIcons 中添加下列代码：

```
void CCh158App::OnViewPgmIcons ()
{
    CPgmIconDlg dlg;
    dlg.DoModal ();
}
```

7. 在源文件 CH158.CPP 的顶部添加下列 include 文件行：

```
#include " pgmicond.h"
```

8. 编译并运行此例子程序。

用法

用户在编辑框中键入一个可执行文件名，并点击按钮 View，则例子程序调用函数 SHGetFileInfo 来获取此可执行文件的信息。函数 SHGetFileInfo 是只用于 Windows 95 的新的 API 函数，可返回大量的有关可执行文件的信息。其中之一便是用于显示应用程序的大图标和小图标的图象列表，大图标用在主窗口的左上角，当应用程序极小化在任务条时，小图标显示在应用程序名的旁边。

函数 SHGetFileInfo 根据传送给最后参数的值返回不同的信息。在本节的例子程序中，则用来获取文件中的图象列表，在图象列表中保存的实际上是可显示的图标，可以使用 API 函数 ImageList_GetIcon 来获取某个图标。在本节的例子程序中，重复调用此函数来获取可执行文件的大图标和小图标，接着将这两个图标显示在对话框中。

应该注意的是，在本节的例子程序中显示图标利用的是发送新的 Windows 95 消息 BM_SETIMAGE 给按钮，此消息使得在 Windows 95 中的按钮上显示图标或位图图象更加简单。在 Windows 的早期版本（以及当前的 Windows NT 版本）中，则需要在创建按钮时设置风格位 BS_ICON 来完成此任务。

15.9 显示作为窗口或对话框背景的位图

问题

有的程序员希望在自己的应用程序中以有趣味的位图来代替对话框中的令人厌烦的灰色背景，希望位图在对话框中看起来像墙纸而且并不影响对话框中的控制或静态文本的显示。

许多程序员找不到一个改变窗口背景的简单方法，是否有方法利用 Windows API 函数来改变对话框的背景为某个位图呢？

方法

改变对话框的背景为某个位图并不困难，关键是需要清楚对话框和窗口是如何设置背景颜色的，以及程序员应该如何修改对话框和窗口改变显示的行为。

当 Windows 准备改变对话框背景的颜色时，通常发送两个消息给对话框。第一个消息是 WM _ ERASEBKGND，此消息指示对话框绘制对话框的背景颜色，以“抹去”屏幕上对话框显示区域的任何显示。

第二个消息是 WM _ CTLCOLOR，发送此消息给对话框或窗口来表示 Windows 需要知道对话框中控制的颜色。

在本节中，将重置对消息 WM _ ERASEBKGND 的处理，以便将位图绘制在窗口的背景上。另外，将重置对消息 WM _ CTLCOLOR 的处理，以避免对话框中的控制“剪补”位图。最后的结果是对话框的背景位图绘制在对话框背景上，控制在背景位图的“上面”。

步骤

按照下列步骤实现一个例子程序。运行此例子程序，选择菜单 Dialog 和菜单项 Bitmap Background，将弹出如图 15-12 所示的对话框，显示背景位图和几个控制。

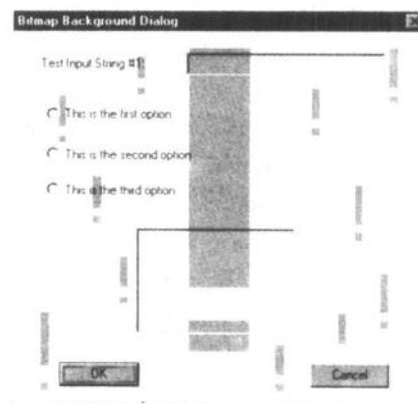


图 15-12　显示背景位图的对话框

实现例子程序的具体步骤如下：

1. 在 Visual C++中，利用 AppWizard 创建新的项目文件，并命名此项目文件为 CH159.MAK。

2. 进入资源编辑器并创建新的对话框模板。在对话框中，添加几个静态文本域和编辑域，以及几个单选按钮和列表框。对话框的实际组成并不重要，只要能够覆盖部分位图就可以了。

3. 选择 ClassWizard，为刚创建的对话框模板创建对话框类，新类命名为 CBitmapBkgdDlg。

4. 在资源编辑器中创建新的位图。在本例子程序中，位图只是一个大的惊叹号，周围有一些小的惊叹号，你可以发挥自己的艺术天才来创建任意的位图，当你的杰作完成时，设置此位图的标识符为 IDB _ BITMAP1 并保存资源文件，退出资源编辑器。

5. 进入 ClassWizard，从下拉列表中选择 CBitmapBkgdDlg，从对象列表中选择对象 CBitmapBkgdDlg，从消息列表中选择消息 WM _ INITDIALOG，点击按钮 Add Function，在 CBitmapBkgdDlg 的方法 OnInitDialog 中添加下列代码：

```
BOOL CBitmapBkgdDlg:: OnInitDialog ()
{
    CDialog:: OnInitDialog ();
    CBitmap * pBmpOld;
    RECT rectClient;
    VERIFY ( m _brush = (HBRUSH) GetStockObject ( HOLLOW _BRUSH ) );
```

```
    VERIFY ( m _Bitmap.LoadBitmap ( IDB _BITMAP1 ) );

    m _Bitmap.GetObject ( sizeof (BITMAP), &m _bmInfo );
    GetClientRect ( &rectClient );
    m _size.cx = rectClient.right;
    m _size.cy = rectClient.bottom;
    m _pt.x = rectClient.left;
    m _pt.y = rectClient.top;
    CClientDC dc (this);
    VERIFY ( m _dcMem.CreateCompatibleDC ( &dc ) );
    VERIFY ( pBmpOld = m _dcMem.SelectObject ( &m _Bitmap ) );
    VERIFY ( m _hBmpOld = (HBITMAP) pBmpOld->GetSafeHandle () );
    return TRUE; // return TRUE unless you set the focus to a control
}
```

6. 接着，在 ClassWizard 中，从对象列表中选择 CBitmapBkgdDlg，从消息列表中选择消息 WM _CTLCOLOR，点击按钮 Add Function，在 CBitmapBkgdDlg 的方法 OnCtrlColor 中添加下列代码：

```
HBRUSH CBitmapBkgdDlg:: OnCtlColor (CDC * pDC, CWnd * pWnd, UINT nCtlColor)
{
    // Have text and controls be painted smoothly over bitmap
    // without using default background color
    pDC->SetBkMode ( TRANSPARENT );
    return m _brush;
}
```

7. 接着，在 ClassWizard 中，从对象列表中选择 CBitmapBkgdDlg，从消息列表中选择消息 WM _DESTROY，点击按钮 Add Function，在 CBitmapBkgdDlg 的方法 OnDestroy 中添加下列代码：

```
void CBitmapBkgdDlg:: OnDestroy ()
{
    CDialog:: OnDestroy ();

    // Select old bitmap into memory dc (selecting out circle bitmap)
    // Need to create a temporary pointer to pass to do this
    ASSERT ( m _hBmpOld );
    VERIFY ( m _dcMem.SelectObject ( CBitmap:: FromHandle (m _hBmpOld) ) );

    // Need to DeleteObject () bitmap which was loaded
    m _Bitmap.DeleteObject ();
}
```

8. 编辑 CBitmapBkgdDlg 的消息映射如下，添加的新行用黑体表示。

```
BEGIN _MESSAGE _MAP (CBitmapBkgdDlg, CDialog)
    // { {AFX - MSG - MAP (CBitmapBkgdDlg)
```

```
    ON_WM_ERASEBKGND ()
    ON_WM_CTLCOLOR ()
    ON_WM_DESTROY ()
    //}} AFX_MSG_MAP
    END_MESSAGE_MAP ()
```

9. 在 CBitmapBlgdDlg 的源文件 BITMAPBK.CPP 中添加下列新方法：

```
BOOL CBitmapBkgdDlg::OnEraseBkgnd (CDC * pDC)
{
    pDC->StretchBlt ( m_pt.x, m_pt.y, m_size.cx, m_size.cy, &m_dcMem, 0, 0, m_bmInfo.
bmWidth-1, m_bmInfo.bmHeight-1, SRCCOPY );
    return TRUE; // No more background painting needed
    }
```

10. 从工程列表中选择头文件 BITMAPBK.H，在此文件中做下列修改，用黑体表示。

```
// bitmapbk.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CBitmapBkgdDlg dialog

class CBitmapBkgdDlg : public CDialog
{
protected:
    CDC m_dcMem; // Compatible Memory DC for dialog
    CBitmap m_Bitmap; // Bitmap to display
    HBITMAP m_hBmpOld; // Handle of old bitmap to save

    HBRUSH m_brush; // Handle of background brush

    BITMAP m_bmInfo; // Bitmap Information structure
    CPoint m_pt; // Position for upper left corner of bitmap
    CSize m_size; // Size (width and height) of bitmap
// Construction
public:
    CBitmapBkgdDlg (CWnd * pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA (CBitmapBkgdDlg)
    enum { IDD = IDD_DIALOG1 };
        // NOTE: the ClassWizard will add data members here
    //}} AFX_DATA

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

```
    // Generated message map functions
    //{{AFX_MSG (CBitmapBkgdDlg)
    afx_msg HBRUSH OnCtlColor (CDC * pDC, CWnd * pWnd, UINT nCtlColor);
    afx_msg void OnDestroy ();
    virtual BOOL OnInitDialog ();
    virtual BOOL OnEraseBkgnd (CDC * pDC);
    //}} AFX_MSG
    DECLARE_MESSAGE_MAP ()
};
```

11. 进入资源编辑器，在菜单 IDR_CH159TYPE 中添加新的菜单，标题为 Dialog。在菜单 Dialog 中添加新的菜单项，标题为 Bitmap Background，标识符为 ID_BITMAP_BKGD，退出资源编辑器，保存资源文件。

12. 进入 ClassWizard，从下拉列表中选择对象 CCh159App，从对象列表中选择对象 ID_BITMAP_BKGD，从消息列表中选择消息 COMMAND，点击按钮 Add Function，新函数命名为 OnBitmapBkgd。在 CCh159App 的方法 OnBitmapBkgd 中输入下列代码：

```
void CCh159App:: OnBitmapBkgd ()
{
    CBitmapBkgdDlg dlg;
    dlg.DoModal ();
}
```

13. 在源文件 CH159.CPP 的顶部添加下列行：

```
#include " bitmapbk.h"
```

14. 编译并运行此例子程序。

用法

当 Windows 初始化对话框时，它发送消息 WM_ERASEBKGND 给对话框的窗口句柄。程序员可以捕捉此消息，以便在应用程序中抹去对话框的背景。在本节中，首先捕捉此消息，接着调用 API 函数 StretchBlt 来将位图（从资源文件中装入）拷贝到对话框的背景上。

在对话框的方法 OnCtrlColor 中，通过设置背景模式为透明来确保对话框中的控制不会“剪补”位图，从而使得位图看起来好像是绘制在对话框中的，没有静态控制的背景所引起的空白。

机械工业出版社 华章图文信息有限公司 计算机图书可供书目

书名	定价
美国当代计算机职业培训系列教程	
个人计算机第一本入门书	26.00
DOS6.XX 轻松入门	28.00
Windows3.1（中文版）轻松入门	23.00
Windows95 轻松入门	25.00
Access for Windows95 轻松入门	26.00
Excel for Windows95 轻松入门	30.00
PowerPoint For Windows95 轻松入门	34.00
Word for Windows95 轻松入门	34.00
Visual Basic 程序设计轻松入门	42.00
Visual Basic4 程序设计轻松入门	27.00
C 语言程序设计轻松入门	39.00
C++程序设计轻松入门	58.00
计算机病毒剖析大全	16.00
计算机网络轻松入门	30.00
Internet for Windows95 轻松入门	15.00
Netware4.1 轻松入门	26.00
最新多媒体电脑实用系列丛书	
多媒体电脑内存管理技巧	28.00
多媒体电脑超级 CD-ROM 技术大全	28.00
多媒体电脑使用大全	52.00
多媒体电脑虚拟现实技巧	64.00
多媒体电脑教育综艺大观	26.00
多媒体电脑影视技术大全	49.00
光盘运作与应用技巧	15.00
中文 Windows95 使用技巧	39.00
计算机图解大全系列	
个人计算机图解大全	42.00
中文版 Excel for Windows95 图解大全	42.00
中文版 Word for Windows95 图解大全	42.00
中文版 Microsoft Office for Windows95 图解大全	42.00
Internet 图解大全	42.00
计算机软件开发与程序设计系列丛书	