高级WINDOWS程序设计技术

王 浩 李 莉 任宇新等 编著

潘金贵 徐殿祥 审校

同济大学出版社

TP316
W210

386160

高级 Windows 程序设计技术

王 浩 李 莉 任宇新 等编著
潘金贵 徐殿祥 审校

TP316
W210

同济大学出版社

JS160/09

内 容 简 介

本书是一本进行 Windows 程序设计的高级参考读物。全书共分为二十五章，包括以下内容：Windows 编程概述、Windows 编程要素简介、基本输出处理、打印、TrueType 字模、键盘和鼠标输入、定时器、内存管理、资源、滚动条、Windows 中的控件、通用对话框、GDI、绘图、位图、位块传输和元文件、文件管理、帮助系统、多文档界面、剪贴板、动态链接库、OLE、系统登录数据库、QuickCase；W 以及 DOS 保护模式接口等。

本书可供计算机专业人员、广大 Windows 用户、各大专院校师生及有关科研人员参考。

责任编辑　王建中
封面设计　李志云

高级 Windows 程序设计技术
王浩　李莉　任宇新　等编著
潘金贵　徐殿祥　审校
同济大学出版社出版
（上海四平路 1239 号）
新华书店上海发行所发行
同济大学印刷厂印刷
开本：787×1092　1/16　印张：50　字数：1280 千字
1996 年 1 月第 1 版　1996 年 1 月第 1 次印刷
印数：1—5000　　定价：52.00 元
ISBN 7－5608－1608－8/TP·173

前　言

目前，Windows 已经售出 1000 万份拷贝，它是目前市场上使用最为广泛的图形用户界面，其市场占有率远远超出了 Macintosh。Windows 的普及主要得益于它使用方便。用户初学 Windows 时，很快就能掌握其使用方法。Windows 操作系统的推出，使得操作计算机（特别是 PC 机）的方法和软件开发方法发生了革命性的变化。而继 Windows 3.0 之后推出的 Windows 3.1，则使这种革命性的变化更为深刻，走上了一个前所未有的新台阶。

Windows 是一个图形窗口操作系统，它是个人机操作系统发展史上的一个里程碑。与 DOS 相比，其功能特点主要表现如下：

- 全新的图形界面。Windows 所定义的图形界面组件目前已成为 PC 机界面的事实上的标准。自 Windows 问世之后，各软件系统纷纷向 Windows 标准靠拢，以期借 Windows 的强大功能，提高自身的性能和价值。
- Windows 3.1 突破了 DOS 内存 640K 的限制，提供了标准模式和 386 增强型模式这两种模式，提供了虚存管理能力。
- 在 Windows 操作系统中可运行 DOS 应用程序。
- 提供了对多任务的并行处理功能，各任务间可方便地进行切换，又可方便地交换信息。
- Windows3.1 中配有多媒体管理器，不仅顺应了多媒体这一潮流，而且较为实用，可望在后续版本中更能有所突破。
- Windows 系统提供了与网络的接口。
- 提供了一系列管理工具，如程序管理器、文件管理器、打印管理器、控制面板等。
- 提供了功能强大的应用程序，如字处理器、画笔软件、终端仿真通信软件等。
- 提供了桌面办公用具，如时钟、卡片、日历、计算器、便笺、记录器、字符图、收录机、多媒体管理器等。
- 为非 Windows 应用程序提供了 PIF 编辑器，为 Windows 应用程序提供了软件开发工具包 SDK。

与 Windows 3.0 相比，Windows 3.1 的新特点主要表现如下：

- 提供了字符图、收录机、多媒体管理器等具有时代特点的桌面办公用具，改进了办公的软环境。
- Windows 3.1 删弃了前期版本中的实模式，将其融入了标准模式及 386 增强型模式之中。
- 带有新版本的 SMARTDRIVE，HIMEM，RAMDRIVE 等辅助工具。
- 在用户界面、操作方法上更为自然、方便，提出了“drag and drop”的新思想。

Windows 3.1 推出后，Windows 才真正像一个操作系统。当系统加电后，进入 DOS 且把计算机系统资源交给 Windows 环境，并受 Windows 管理。在安装了新的 FastDisk 后，Windows 也能管理文件操作，从而使 DOS 只能控制 DOS 应用程序了。

如果系统重新启动，计算机的资源又将由 Windows 管理。Windows 为其环境下运行的应用程序提供多任务、系统管理和资源控制。

Windows 功能强大，为充分发挥其优越性，用户应当把自己的应用程序转向支持 Windows。现在已有很多计算机公司要求为其 PC 机所购买的应用程序必须是可在 Windows 下运行的程序。

本书可供 Windows 用户使用，也可供已熟悉 Windows 的高级程序设计人员使用。对于 DOS/Windows 来说，用 Windows 替代 DOS 的时代已经到来。

参加本书编写的作者有：王浩、李莉、潘金贵、任宇新、韩小琴、刘应木、李阳、何维、王立国、孙朝辉、葛佳、刘秀英、李华、陈越、李燕、朱尽染、姜鸿燕等。潘金贵和王浩同志对全书进行了仔细的修改和统编，并蒙徐殿祥博士审校，值此谨表谢意。

由于水平、时间所限，不妥之处在所难免，欢迎广大读者批评指正。

编著者

1995 年 6 月于南京

目　录

第一章　Windows 编程概述 …… 1
1.1　Windows 综述 …… 1
1.2　对用户的要求 …… 1
1.3　Windows 与 DOS 应用程序 …… 3
1.4　Windows 3.1 略览 …… 4
1.5　Windows 应用程序的组成 …… 7
1.6　本书的编排 …… 9
第二章　Windows 编程要素简介 …… 11
2.1　框图和列表 …… 11
2.2　编译与链接 …… 15
2.3　头文件 …… 16
2.4　WinMain …… 17
2.5　初始化 …… 18
2.6　生成窗口 …… 19
2.7　MSG 结构 …… 21
2.8　消息循环 …… 23
2.9　窗口函数 …… 24
2.10　缺省窗口函数 …… 24
2.11　消息的进一步说明 …… 25
2.12　应用程序的退出 …… 27
2.13　模块定义文件 …… 28
2.14　Windows 程序设计的注意事项 …… 31
第三章　基本输出处理 …… 35
3.1　设备环境 …… 35
3.2　WM_PAINT 消息 …… 37
3.3　输出函数 …… 39
3.4　绘图工具 …… 58
第四章　打印 …… 76
4.1　打印机设置 …… 78
4.2　打印函数 …… 82

4.3 打印机驱动程序 …… 101
第五章 TrueType 字模 …… 123
5.1 TrueType 字模概述 …… 123
5.2 WYSIWYG 的问题 …… 125
5.3 计算机字模技术 …… 129
5.4 其他的 Windows 字模特征 …… 133
5.5 利用 TrueType 字模编程 …… 136
第六章 键盘和鼠标输入 …… 150
6.1 输入消息 …… 150
6.2 键盘输入 …… 150
6.3 鼠标输入 …… 157
第七章 定时器 …… 164
7.1 定时器基础 …… 165
7.2 定时器的用途 …… 167
7.3 使用定时器报告状态 …… 178
7.4 时钟 …… 182
7.5 标准时间 …… 188
7.6 定时器举例 …… 188
第八章 内存管理 …… 193
8.1 概述 …… 193
8.2 动态存储管理 …… 201
8.3 Windows 开销 …… 202
8.4 存储模式 …… 203
8.5 存在的问题 …… 210
8.6 ToolHelp DLL …… 217
第九章 资源 …… 224
9.1 资源的类型 …… 224
9.2 图标 …… 226
9.3 光标 …… 235
9.4 其他单行语句 …… 241
9.5 串表 …… 241
9.6 菜单 …… 242
9.7 键盘加速键 …… 257
9.8 对话框 …… 258
9.9 消息框 …… 273
第十章 滚动条 …… 275
10.1 带滚动条的设计 …… 275

10.2 滚动条示例…… 279
第十一章 Windows 中的控件…… 287
11.1 控件类型概述…… 287
11.2 作为独立窗口的控件…… 289
11.3 按钮…… 293
11.4 控件和颜色…… 303
11.5 静态类…… 312
11.6 滚动条…… 314
11.7 编辑类…… 331
11.8 列表框…… 337
11.9 COMBOBOX 类 …… 343
第十二章 通用对话框…… 351
12.1 概述…… 351
12.2 打开和保存文件…… 353
12.3 查找和替换文本…… 365
12.4 查找和替换对话框…… 370
12.5 字模选择…… 377
第十三章 GDI 简介…… 387
13.1 GDI 基础…… 387
13.2 设备环境…… 388
13.3 映像模式…… 401
第十四章 绘图…… 413
14.1 画点…… 413
14.2 画线…… 413
14.3 画填充区域…… 424
14.4 矩形、区域和剪取 …… 444
14.5 其他 GDI 函数 …… 451
14.6 画图程序示例…… 456
第十五章 位图…… 461
15.1 位图的类型…… 461
15.2 依赖于设备的位图…… 461
15.3 依赖于设备的位图(DIB) …… 474
第十六章 位、块传输和元文件 …… 485
16.1 颜色和位图…… 486
16.2 与设备无关的位图(DIB) …… 486
16.3 GDI 位图对象…… 496
16.4 内存设备环境…… 500

16.5 强大的 BLT …… 501
16.6 元文件 …… 515
第十七章 文件管理 …… 525
17.1 MS-DOS 文件 …… 525
17.2 初始化文件 …… 539
第十八章 帮助系统 …… 552
18.1 Help 应用程序 …… 552
18.2 生成 Help 系统 …… 553
18.3 Help 系统举例 …… 565
第十九章 多文档界面 …… 575
19.1 MDI 应用程序的结构 …… 575
19.2 补充和修改 …… 576
19.3 MDI 示例 …… 581
第二十章 剪贴板 …… 594
20.1 文本格式 …… 595
20.2 位图格式 …… 604
20.3 关于格式的附加知识 …… 614
20.4 剪贴板观察窗 …… 616
第二十一章 动态链接库 …… 623
21.1 远程函数 …… 623
21.2 限制 …… 627
21.3 建立 DLL …… 628
21.4 DLL 示例 …… 632
第二十二章 OLE …… 648
22.1 Windows 下的通信 …… 648
22.2 对象嵌入和链接:OLE …… 652
22.3 OLE 中的客户 …… 655
22.4 对象封装器 …… 697
22.5 OLE 流 …… 701
22.6 服务器 …… 707
第二十三章 系统登录数据库 …… 714
23.1 数据库的结构 …… 714
23.2 登录数据库和 OLE …… 717
23.3 登录数据库和 OLE 示例 …… 718
23.4 文件管理器支持示例 …… 723
23.5 生成新入口 …… 730
第二十四章 QuickCase:W …… 732

24.1 各种开发工具及其用途 …… 732
24.2 使用 QuickCase:W …… 733
第二十五章 DOS 保护模式接口 …… 772
25.1 EMS 仿真 …… 772
25.2 VCPI …… 773
25.3 DPMI …… 773
25.4 DPMI 功能和 31H 中断 …… 774
25.5 DPMI 和 Windows 应用程序 …… 774
25.6 汇总 …… 775

第一章　Windows 编程概述

本章简要描述用户在编写 Windows 应用程序的过程中所必需的知识，包括用户应该备有什么软件以及如何建立一个 Windows 应用程序。

1.1　Windows 综述

Windows 为用户提供了一种多任务、基于图形的窗口环境，在此环境下可运行专门为 Windows 设计的程序。这些程序包括 Microsoft Excel（电子表格和商业图形软件），Microsoft Word（字处理软件），PageMaker（桌面排版系统），Ami（字处理系统）、Designer（绘图软件），Current（个人信息管理软件），TooBook（软件构造工具包）等等。

为 Windows 编写的程序具有一致的外观和命令结构，因此，它比传统的 MS-DOS 程序更加易学易用。用户很容易在不同的 Windows 程序间切换，很容易在程序间交换数据。Windows 提供了一个使用方便、基于图标的程序管理器（用于运行程序）、一个文件管理器（用于维护文件）和一个打印管理器（用于管理打印机队列）。

尽管 Windows 原本用来运行专为它编写的应用程序，但许多为 MS-DOS 编写的程序也能运行于 Windows 之下。当然，这些程序无法利用 Windows 的许多优点，但某些情况下，它们也可以窗口化，并与 Windows 程序并发执行。

Windows 为程序开发人员提供了丰富的内部例程，使程序员可以使用菜单、对话框、滚动条等构造友好的用户界面。Windows 还给出了一种外在的图形程序设计语言，这种语言可以对各种不同的字模进行格式化。程序员可以用一种与设备无关的方式来处理键盘、鼠标、视频显示器、打印机等。

1.2　对用户的要求

阅读本书时，让我们从用户所需的编程经验开始。虽然在无以下知识的情况下用户也能阅读本书，但是我们建议读者尽可能地作进一步的学习，以免误解我们的意图：

- 对于用 C 语言编写程序已有了一些经验。如果用户还未以 C 语言编制过程序的话，那么在这方面有许多参考书可供使用。
- 有基于 PC MS-DOS 方面的经验。
- 写过在 MS-DOS 下运行的应用程序。
- 使用过 Microsoft Windows。

1.2.1 工具

要最有效地运行 Windows 和众多的开发工具，需要以下硬件支持：

- 基于 Intel 80386 微处理器的 IBM 个人机(或兼容机)，配有硬盘、4MB 以上内存、使用 MS-DOS 3.30 或更高版本。
- 图形显示器和视频卡，最好与 IBM VGA(视频图形阵列)或更好的显示器兼容。
- 鼠标。尽管对大多数 Windows 程序来说，鼠标是可选的，但本书中的某些程序需要鼠标支持。

用户将要用到下列软件编写 Windows 应用程序：

- 一个 C 编译器。如果用户是一个有经验的 C 程序员，那么就可能已经对一个编译器有所了解。本书中的源码均采用 Microsoft C 编译器 7.0 版本和 Pressional Development System(PDS)支持的 C 语言。

  用户也可选择使用 Microsoft Quick C for Windows 或 Quick Case:W 开发软件。
- Microsoft Windows Software Development Kit(SDK) 3.1。这个版本包含编写 Windows 应用程序的许多工具。

书中的例子程序采用 Microsoft C 7.0 和 Microsoft Software Development Kit(SDK) 3.1 版本。由 Windows 3.1 来运行已编译的程序。本书还介绍了如何用 Microsoft QuickCase:W 来开发源代码。

C 编译器包括：

- 一个编译器和链接器(CL.EXE 和 LINK.EXE)。这两个程序用于编译源码并且将此源码转变成可执行的程序代码。
- 头文件 DOS.H 和 CONIO.H。头文件包含定义、宏指令和其他重要的编程数据。
- 用于处理不同内存模式的标准库文件：SLIBCE.LIB，MLIBCE.LIB 和 LLIBCE.LIB(代表小、中和大)。

Software Development Kit(SDK)可将一个程序转变成 Windows 应用程序，此 SDK 的功能包括：

- 一个链接器(LINK.EXE 或 LINK4.EXE)和资源编译器(RC.EXE)。
- CodeView，为查找应用程序中的错误而调试应用程序(CVW.EXE)。
- Windows 头文件和库文件。
- 一个字模编辑器(FONTEDIT.EXE)、一个对话框编辑器(DLGEDIT.EXE)和一个图像编辑器应用程序(IMAGEDIT.EXE)。IMAGEDIT 类似于 Microsoft Windows Paintbrush，但是 IMAGEDIT 允许用户在 Windows 应用程序中产生光标、位图和图标等。

1.2.2 安装

我们建议用户先安装 Microsoft C 编译器，再安装 SDK。举例来说，我们将 Microsoft C 安装在 D:\C7.0 目录中，将 SDK 安装在 D:\WINDEV 目录中，再安装如下内容到 AUTOEXEC.BAT 文件中：

```
PATH = C:\DOS;C:\WINDOWS;D:\C700\BIN;D:\WINDEV;D:\WINDEV\INCLUED;D:\
```

```
        WINDEV\INCLUED\SYS
SET LIB=D:\WINDEV\LIB;D:\C700\LIB
SET INCLUED=D:\WINDEV\INCLUED;D:\C700\INCLUED
SET HELPFILES=D:\C700\HELP\*.HLP
SET INT=D:\C700\INI
```

1.3 Windows 与 DOS 应用程序

让我们看看 Windows 应用程序怎样区别于以标准 DOS 为基础的应用程序。我们从开发一个以 DOS 为基础的应用程序的源代码开始(本例使用 C 语言)。

一旦开发人员写下了源代码并且确信代码无错的话,代码就可以通过一个编译程序(CL. EXE)来运行。源码(我们在本例中将称作 HELLO. C)在编译程序中引用头文件(以 .H 为扩展名的文件)。

这些头文件(有时被称作 include 文件,因为 C 语言的 # include 语句在源代码内请求这些文件)包含完整程序所需的信息(例如用于屏幕显示)。

编译器产生一个目标文件(此处命名为 HELLO. OBJ)。

还剩一步:.OBJ 文件和一个库文件(? LIBCEW. LIB)必须由 LINK. EXE 链接以形成一个可执行文件。最后的结果文件是 HELLO. EXE。

图 1.1 说明了一个 DOS 应用程序是怎样编译、并从 C 源码转变为一个可执行文件的:

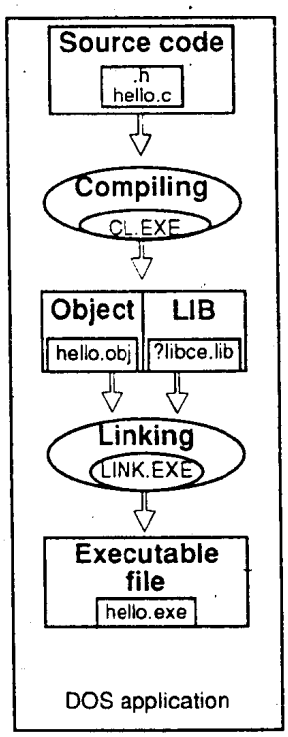


图 1.1 DOS 应用程序怎样从 C 源码转变成可执行文件

生成 Windows 程序时要作同样的处理(编译和链接),除了以下几点不同以外:

(1)一个以 WINDOWS.H 命名的头文件、其他包含文件及源代码被一起引用。WINDOWS.H 包含 Windows 所专用的类型声明、宏和其他信息。

(2)与? LIBCE.LIB 文件由 C 使用不同,Windows 专用程序库由链接器使用(? LIBCEW.LIB 和 LIBW.LIB)。

(3)一个模块定义文件(由 .DEF 扩展名说明)与目标文件程序库链接。模块定义文件定义名称、描述、数据类型和所需的其他资源。

链接好之后还不能在 Windows 下运行,还需要资源,比如位图、字符串、图标和光标等。

资源描述文件(带 .RC 扩展名)使用资源编译器 RC.EXE 来编译。结果文件由 RC.EXE 加到链接好的 .EXE 文件中。

最后结果就是一个与 Windows 3.1 兼容的应用程序。

如果用户未编译资源,略为简单的应用程序可能会在 Windows 3.1 下运行,但可能产生错误。

图 1.2 说明 Windows 应用程序如何从源代码转变为可执行文件。

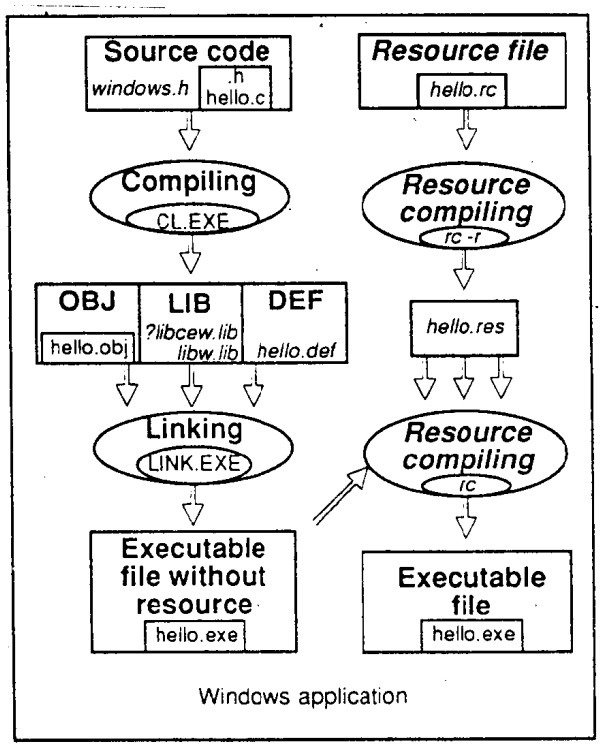


图 1.2 Windows 应用程序如何从源代码转变成可执行文件

1.4 Windows 3.1 略览

Windows 3.1 是在 1992 年春季推出的。Windows 3.1 图形用户接口是一个适于多任务操作的 DOS 用户环境。

1.4.1 Windows 模式

Windows 3.1 有两种不同的模式：标准模式和 386 增强型模式。更早的 Windows 版本有第三种模式，叫做实模式，但 Windows 3.1 不支持这种模式(以后详述)。这两种模式定义如下。

1. 标准模式

标准模式可突破 MS-DOS 的内存限制。在常规内存之上加上一定量的扩展内存，总称为邻接内存。另外，高位内存区域(HMA)的一块内存也加在扩展和常规内存之上。

MS-DOS 应用程序必须在常规内存中运行，正如实模式下运行的 Windows 一样。

2. 386 增强型模式

386 增强型模式只适用于 80386 或 80486 处理器。在此方式下，为每一单独的 DOS 程序创立虚拟 DOS 设备。虚拟存储器用于所有的 Windows 应用程序，也就是说也可使用“页面内存”。

Windows Virtual Memory Manager(VMM)负责对最近未使用的页面和硬盘进行存储和交换。4K 页中的每一页可存在一个临时的或永久性的交换文件中。Windows 给临时文件取名为 WIN386.SWP，永久性文件命名为 386SPART.PAR，并且具有隐含属性和系统属性。

1.4.2 动态链接

如果读者曾进行过 MS-DOS 编程，也许会猜测，Windows 程序是不是通过一个软件中断与 Windows 联系的呢，如 MS-DOS 中断 0x21？有人会猜测，链接程序向 Windows 程序中加入联编，将 Windows 函数调用转换成软件中断。但是这种观点是不对的。Windows 程序与 Windows 的接口是一个“动态链接”过程。

与 MS-DOS 程序一样，Windows 可执行程序具有文件扩展名 .EXE。但是，它使用所谓“新的可执行文件格式”(New Executable file Format)作为 .EXE 格式。每当 Windows 程序调用 Windows 函数时，C 编译程序就产生远程调用的汇编语言代码。.EXE 文件中存在一个表，它用一个动态链接库名和此库中函数的名字或编号(称为序号)来标识被调用的函数。

Windows 本身包含三大类动态链接库。它们是 KRNL286 和 KRNL386(负责内存管理、加载、执行及调度程序)，USER(用户界面和窗口)和 GDI(图形)。这些库包含了大多数 Windows 函数的代码和数据。这三个动态链接库在 Windows 目录的 SYSTEM 子目录中。

在 Windows 程序装入内存后，程序中的远程调用被转换为指向动态链接库中函数的入口，动态链接库也被装入内存。这就是为什么要将所有 Windows 函数定义为 far 的原因：动态链接库中的代码与程序代码不在同一段中。另外，传给 Windows 函数的指针也必须定义为 far，避免与动态链接库自身的代码和数据相混淆。

一般来说，程序员不用担心远程调用和远程指针的使用，因为函数已在 WINDOWS.H 中用远程指针声明为远程函数，C 编译程序会自动进行必要的地址转换。

在链接 Windows 程序以产生可执行文件时，必须使用 Windows 软件开发工具包(SDK)或 Borland C++编译程序中提供的专门“输入库”。这个输入库包含动态链接库名和所有

Windows 函数的序数号。链接程序使用这个信息，在 .EXE 文件中构造一个表，在载入程序时 Windows 用这个表将调用转换为 Windows 函数。

1.4.3 面向对象的程序设计

进行 Windows 程序设计，实际上就是进行一种面向对象的程序设计。这一点在“窗口”中显得最为明显。窗口是我们在 Windows 中使用得最多的对象，该对象也正是 Windows 之所以命名为“Windows”的原因。

窗口是屏幕上的矩形区域。它从键盘或鼠标接受输入，并在其表面显示图形输出。

一个应用程序窗口通常包含程序的标题条、菜单、边框，也许还有一些滚动条。对话框也是一种窗口，不同的是，对话框表面通常包含几个其他窗口，称为“子窗口”。这些子窗口的形式有按钮、单选按钮、复选框、文本输入区、列表框和滚动条。

用户将这些窗口看成屏幕上的对象，可通过按下一个按钮或滚动一个滚动条与这些对象直接交流。有趣的是，程序员的观点与用户的观点极其类似。窗口以“消息”的形式接收窗口的输入，窗口也用消息与其他窗口通信。

对消息的理解将是读者成为 Windows 程序员所必须逾越的障碍之一。

1.4.4 消息驱动

我们可以演示一个在窗口中运行的最基本的字处理程序。在程序窗口的大小改变时，字处理器会重新格式化其中的文本。

很明显，窗口大小改变的细节是由操作系统处理的，程序能够响应这个系统函数。程序是怎样“知道”窗口的大小发生改变的呢？操作系统使用了什么技术将这一信息传递给窗口？在苦思冥想这些问题时，我们的传统编程经验似乎毫无用处。

问题的关键在于理解图形用户界面所使用的体系结构。在 Windows 中，当用户改变窗口的大小时，Windows 给程序发送一条消息指出新窗口的大小。然后，程序就可以调整窗口中的内容，以响应大小的变化。

“Windows 给程序发送消息。”我们希望读者不要对这句话视而不见。它到底表达了什么意思？我们在这里讨论的是程序代码，而不是一个电子邮件系统。操作系统怎么能给程序发送消息呢？

其实，所谓“Windows 给程序发送消息”，是指 Windows 调用程序中的一个函数。该函数的参数描述了这个特定消息。这种位于 Windows 程序中的函数称为“窗口过程”。

1.4.5 窗口过程

无疑，读者对程序调用操作系统的思路是很熟悉的。例如，程序在打开磁盘文件时，就要使用有关的系统调用，读者所不习惯的，可能是操作系统调用程序。而这正是 Windows 面向对象体系结构的一大基石。

程序创建的每个窗口都有相关的窗口过程。这个窗口过程是一个函数，既可以在程序中，也可以在动态链接库中。Windows 通过调用窗口过程来给窗口发送消息。窗口过程根据此消息进行处理，然后将控制返回给 Windows。

更精确地说，窗口通常是在“窗口类”的基础上创建的。窗口类标识了处理窗口消息的窗

口过程。使用窗口类使得多个窗口能基于同一个窗口类，且使用同一个窗口过程。例如，所有 Windows 程序中的所有按钮均基于同一窗口类。这个窗口类与一个处理所有按钮消息的窗口过程(位于 Windows 的 USER.EXE 动态链接库中)联结。

面向对象程序设计中，对象是代码与数据的组合。窗口是一种对象，其代码是窗口过程，数据是窗口过程保存的信息和 Windows 为每个窗口和系统中每个窗口类保存的信息。

窗口过程处理发送给窗口的消息。这些消息经常告知窗口，用户正用键盘或鼠标进行输入。这正是压入按钮窗口知道它被“按下”的奥妙所在。在窗口大小改变，或者窗口表面需要重画时，由其他消息通告窗口。

Windows 程序开始执行后，Windows 为该程序创建一个“消息队列”。这个消息队列用来存放该程序可能创建的各种不同窗口的消息。程序中有一小段代码，叫做“消息循环”，它用来从队列中取出消息，并将它们发送给相应的窗口过程。有些消息直接送给窗口过程，不用放入消息队列中。

若读者注意这段对 Windows 体系结构的简略描述，也许对理解在真实的程序环境中窗口、窗口类、窗口过程、消息队列、消息循环和窗口消息是如何相互配合的会有所帮助。

1.5 Windows 应用程序的组成

许多 Windows 应用程序与本书所列出的所有应用程序一样采用 C 语言。我们曾提到过，为产生一个可执行的应用程序，除源文件以外，还需要其他文件。图 1.3 列出这些增加的文件。在本书中，我们试图采用最通用的文件。

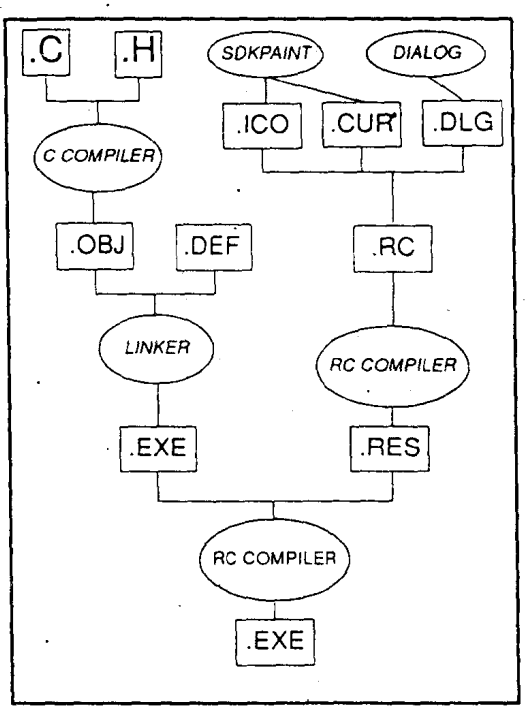


图 1.3 需加到源文件中的增补文件

有些 Windows 应用程序在编译和链接时要加进几个文件。在我们讨论的每个例子中，都将列出所有需要的文件。

1.5.1 书写约定

Windows 程序员应采用一种叫做“匈牙利表示法”的变量命名约定，这是为了纪念传奇性的 Microsoft 程序员 Charles Simonyi。非常简单，变量名以一或多个小写字母开始，这些字母表示变量的数据类型。例如，lpszCmdParam 中的 lpsz 代表指向以 0 结尾的串的长指针。

hInstance 和 hPrevInstance 中的 n 前缀代表“句柄”；nCmdShow 中的 n 前缀代表“数”，通常表示一个整数。WndProc 的两个参数也使用匈牙利表示法：wParam 是一个 UINT（w 代表“字”），lParam 是一个 LONG 类型数据。

在命名结构变量时，可以用结构名的小写作变量名的前缀，或者用作整个变量名。例如，在 HELLOWIN. C 的 WinMain 函数中，msg 变量是 MSG 类型的结构；wndclass 是 WNDCLASS 类型的结构。在 WndProc 函数中，ps 是一个 PAINTSTRUCT 结构，rect 是一个 RECT 结构。

这种表示法能帮助程序员及早发现并避免代码中的错误。由于变量名既描述了变量的作用，又描述了其数据类型，就比较容易避免产生数据类型不匹配的错误。

下表列出了在本书中将要使用的变量前缀：

前缀	数据类型
c	char
by	BYTE（无符号字符）
n	short 或 int
i	int（整数）
x,y	short（分别用作 x 坐标和 y 坐标）
cx,cy	short（分别用作 x 长度和 y 长度；C 代表“count”，即量度）
b	BOOL（int）
w	UINT（无符号整数）或 WORD（无符号字）
l	LONG（long）
dw	DWORD（unsigned long）
fn	function（函数）
s	string（串）
sz	以 0 字节结尾的串

除标准 C 数据类型外，Windows 还有许多自己的类型。Windows 头文件中定义了这些类型，列表如下：

类型	含义
BOOL	布尔量 16 位数值
BYTE	无符号 8 位数值
Char	带符号 8 位数值（也可由 ASCII 字符组成）
DNORD	无符号 32 位数值
far 或 FAR	指示编译器使用一个长指针（32 位地址）
HANDLE	普通句柄（16 位）

HDC	设备上下文句柄(16 位)
HWND	窗口句柄(16 位)
int	带符号 16 位数值
long 或 LONG	带符号 32 位数值
LPINT	指向 int 类型的长指针
LPSTR	指向字符串的长指针
near 或 NEAR	近指针
PINT	指向 int 型的近指针
PSTR	指向字符串的近指针
short	带符号 16 位数值
WORD	不带符号的 16 位数值
void 或 VOID	说明函数不返回值

1.5.2 Windows 程序库

所有 Windows 函数的定义存储在程序库中,这与 C 语言中的运行程序库不同。这些程序是动态链接库(DLLS)。在装载了应用程序后,对 DLL 的引用就是将其链接到应用程序。由于程序库本身是不链接到应用程序的,所以 Dynamic Link Libraries(DLL)可以更新,而不必重写应用程序。

Windows 使用下列动态链接库:

- KERNEL. EXE:负责内存和资源管理以及多任务过程
- USER. EXE:进行窗口管理和输入处理
- GDI. EXE:提供图形界面和所有输出处理

后面我们将详细讨论 DLL。

1.5.3 Windows 函数

由 Windows 提供的所有函数分成三组:Windows Manager 接口函数、Graphics Device 接口函数和 System Service 接口函数。

关键函数

一个 Windows 应用程序源码必须有两个基本函数:

- WinMain 函数:这一函数替代惯用的 C 语言程序中出现的 Main()函数。WinMain 函数代表程序执行的"入口点",并且提供基本操作和参数。
- 主窗口函数:这一函数控制基本的窗口显示并且向窗口传递消息(在下一章对消息将有更多说明)。可给此函数取任何名字,所取的名字应有助于阅读者对于函数的理解(例如,MainWndProc 指示主窗口处理函数)。

1.6 本书的编排

为了逐步引导读者学习 Windows 编程技术,第二章给出了一个极其简单的 Windows 应

用程序范例，以让读者对 Windows 应用程序有一个感性认识。如果读者已对编写 Windows 应用程序具有了初步经验，则可以跳过这一章。接下来讨论了 Windows 应用程序的输出，包括设备环境的基本概念、绘图工具的使用、与打印有关的问题及 Windows 中的 TrueType 字模。第六章和第七章介绍了键盘、鼠标和定时器这三种输入设备与 Windows 应用程序的接口。第八章说明在 Windows 中如何管理内存。后面的章节涉及 Windows 中的资源，其中第九章介绍了各种资源，第十章和第十一章详细讨论了滚动条和各类子窗口控件，第十二章重点介绍了 Windows 和 Windows 应用程序中常用的四种通用对话框，并给出了一个字处理程序范例。第十三章到第十六章涉及 Windows 的图形设备接口。第十七章和第十八章讨论了 Windows 应用程序如何管理文件、如何构造自己的帮助系统。接下来的四章描述了数据交换和链接技术。为了给读者提供更全面的信息，作为对前述内容的补充，第二十三章到第二十五章分别介绍了系统登录数据库、Windows 应用程序开发工具 QuickCase:W 以及与 DOS 保护模式的接口。

下面列出本书的编排目录：

序号	内容	章次
1	Windows 编程入门	一、二
2	输出及字模技术	三、四、五
3	输入	六、七
4	内存管理	八
5	资源管理	九、十、十一、十二
6	图形设备接口(GDI)	十三、十四、十五、十六
7	文件管理	十七
8	帮助系统	十八
9	数据交换与链接	十九、二十、二十一、二十二
10	系统登录数据库	二十三
11	应用程序开发工具 QuickCase:W	二十四
12	DOS 保护模式接口	二十五

第二章　Windows 编程要素简介

本章解释一个 Windows 应用程序的重要成分，将涉及初始化、消息循环、主窗口函数、模块定义文件和 MAKE 文件。

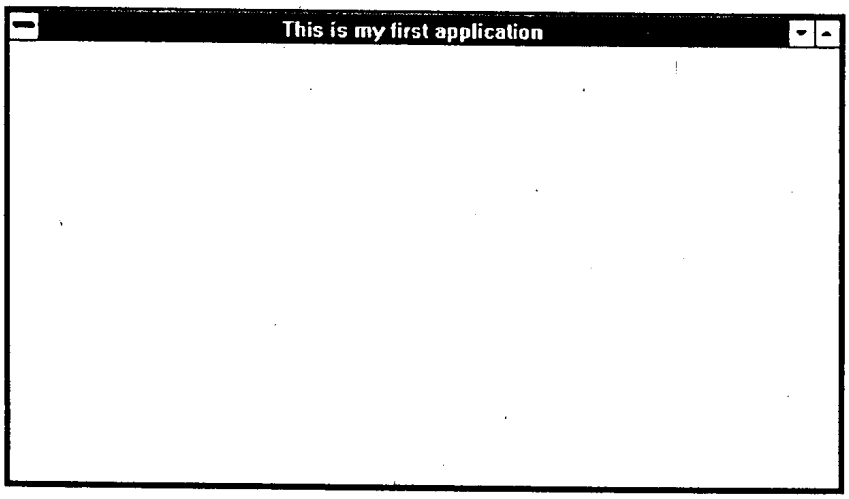


图 2.1

2.1　框图和列表

我们将使用如图 2.2 所示的 WinMain 函数的框图和后面的程序例子作为讨论函数和相关概念的基础。

一个 Windows 应用程序必须有下列两个组成成分：

- WinMain 函数：这是应用程序和主过程的开始。
- 主窗口函数：本章所列的 FIRST.C 源码中，FirstWndProc 是主窗口函数，这个函数处理所有消息（本章后面将详细讨论消息）。

如下所示的源代码在编译和链接后将显示一个窗口，窗口标题包含语句：

"This is my first application"

显示和描述新消息表

新消息	描述
WM_DESTROY	在窗口删除（移动）过程中传送

新函数	简短描述
CreateWindow	生成一个窗口
DefWindowProc	缺省窗口过程
DispatchMessage	向 WndProc 传递消息
GetMessage	从队列中获取消息
PostQuitMessage	生成一条 WM_QUIT 消息
RegisterChous	登录一个类
ShowWindow	显示窗口
TranslateMessage	传送键盘消息
UpdateWindow	给出客户区
WinMain	主过程
新结构	简短描述
MSG	消息数据结构
WNDCLASS	类的数据结构

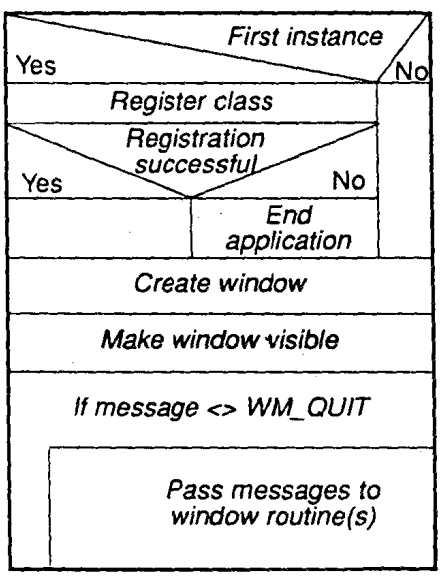


图 2.2　WinMain 函数的框图

源码:FIRST. C

```
/* * FIRST. C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Short Windows application.  FIRST. C displays a window with title bar.  * * /
/* * The title bar states,"This is my first application"  * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows. h"                                    // Include windows. h header file

#if  MSC_VER )= 700
#pragma warning (disable:4028)
#endif
```

```
BOOL FirstInit ( HANDLE );
long FAR PASCAL FirstWndProc( HWND,unsigned,WORD,LONG);

/* * WinMain (main function for every Windows application) * * * * * * * /

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,nCmdShow )
HANDLE hInstance,hPrevInstance;                    // Current & previous instances
LPSTR lpszCmdLine;                                 // Long ptr to string after
                                                   // program name during execution
int nCmdShow;                                      // Specifies the application
                                                   // window's appearance
{
    MSG msg;                                       // Message variable
    HWND hWnd;                                     // Window handle

    if (! hPrevInstance)                           // Initialize first instance
    {
        if (! FirstInit( hInstance ))              // If initialization fails
            return FALSE;                          // return FALSE
    }
/* * Specify appearance of application's main window * * * * * * * * * /

    hWnd = CreateWindow("First",                   // Window class name
        "This is my first application",            // Window caption
        WS_OVERLAPPEDWINDOW,                       // Overlapped window style
        CW_USEDEFAULT,                             // Default upper-left x pos.
        0,                                         // Upper-left y pos.
        CW_USEDEFAULT,                             // Default initial x size
        0,                                         // y size
        NULL,                                      // No parent window
        NULL,                                      // Window menu used
        hInstance,                                 // Application instance
        NULL);                                     // No creation parameters

    ShowWindow( hWnd,nCmdShow );                   // Make window visible
    UpdateWindow( hWnd );                          // Update window

    while (GetMessage(&msg,NULL,0,0))              // Message reading
    {
        TranslateMessage(&msg);                    // Message translation
        DispatchMessage(&msg);                     // Send message to Windows
    }

    return (int)msg. wParam;                       // Return wParam of last message
}

BOOL FirstInit( hInstance )                        // Initialize instance handle
HANDLE hInstance;                                  // Instance handle
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * /
```

```
    WNDCLASS wcFirstClass;                                    // Main window class

    wcFirstClass.hCursor = LoadCursor( NULL,IDC_ARROW );
                                                              // Mouse cursor
    wcFirstClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                              // Default icon
    wcFirstClass.lpszMenuName = NULL;                         // No menu
    wcFirstClass.lpszClassName = "First";                     // Window class
    wcFirstClass.hbrBackground = GetStockObject( WHITE_BRUSH );
                                                              // White background
    wcFirstClass.hInstance = hInstance;                       // Instance
    wcFirstClass.style = 0;                                   // Horizontal & vertical redraw
                                                              // of client area
    wcFirstClass.lpfnWndProc = FirstWndProc;                  // Window function
    wcFirstClass.cbClsExtra = 0 ;                             // No extra bytes
    wcFirstClass.cbWndExtra = 0 ;                             // No extra bytes

    if (! RegisterClass( &wcFirstClass ) )                    // Register window class
        return FALSE;                                         // Return FALSE if registration fails
                                                              // If registration is successful,
    return TRUE;                                              // Return TRUE
}
/* * FirstWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL FirstWndProc( hWnd,message,wParam,lParam )
HWND hWnd;                                                    // Window handle
unsigned message;                                             // Message type
WORD wParam;                                                  // Message-dependent 16 bit value
LONG lParam;                                                  // Message-dependent 32 bit value
{
    switch (message)                                          // Process messages
    {
        case WM_DESTROY:                                      // Send WM_QUIT if window is
            PostQuitMessage(0);                               // destroyed
            break;                                            // End of this message process
        default:                                              // Send other messages to
                                                              // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                            // End of this message process
    }
    return(0L);
}
```

模块定义文件:FIRST.DEF

```
NAME FIRST

DESCRIPTION '1st Windows Application'

EXETYPE WINDOWS
```

```
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS FirstWndProc @1
```

MAKE 文件:FIRST. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: FIRST. EXE

FIRST. OBJ: FIRST. C

FIRST. EXE: FIRST. OBJ FIRST. DEF
    link /NOD/A:16 FIRST,FIRST,nul,libw slibcew,FIRST. def
    rc FIRST. exe
```

2.2 编译与链接

用户必需具备三个文件,即 FIRST. C 源码、FIRST. DEF 模块定义文件及 FIRST. MAK MAKE 文件(这些文件存在配套软盘上)。

从源码直到编译和链接过的应用程序,有许多方法可供采用。在此我们讨论其中两种——在 DOS 提示符下逐行输入命令;使用 MAKE 文件自动生成序列。我们推荐在编译和链接过程中逐行阅读以下信息。但在编译和链接处理时,使用下面描述的 MAKE 方法(最好使用这种方法)。

2.2.1 行指令

为了编译 FIRST. C 源代码,用户应该在 DOS 提示符下输入下列信息:

```
CL -C -Gw -Zp first. c
```

让我们来检查一下此行的每一个组成部分。CL 调用编译器。-C 开关指示源文件只需编译而不进行链接(我们将在以后人工地执行链接操作)。这使得在链接时可向编译过的源文件中增添其他代码。

-Gw 开关在每个由 FAR 定义的函数之前插入入口程序,在其后加上出口程序。这些附加的代码保证数据段和堆栈的使用是正确的,因为 DLL 没有自己的堆栈,只能使用应用程

序的堆栈。

由于用于 Windows 的所有结构都需要压缩，所以用-Zp 开关压缩结构。

-Gs 开关不能检查栈的溢出，它通常需要和上述开关组合起来使用。

用这种方式编译将产生一个目标文件(.OBJ)。现在我们需将模块定义文件(.DEF)和 Windows 专用程序库组合起来形成一个应用程序，需要调用链接器(LINK.EXE 程序)；由下列命令完成这一任务。

```
link/align:16 first,first.exe,,libw+slibcew,first.def
```

LINK 调用链接器，/ALIGN:16 开关使代码和数据段按 16 字节边界对齐。FIRST、FIRST.EXE 是要被链接的 .OBJ 文件名和 .EXE 目标文件名。两个逗号表明不需要映射文件。LIBW+SLIBCEW 确定用户想链接到 FIRST.EXE 上的程序库，FIRST.DEF 确定用户想加到 FIRST.EXE 上的模块定义文件。

最后一步是关键的一步，即通过资源编译器 RC.EXE 处理 FIRST.EXE 文件。

```
rc first.exe
```

到此为止，用户即可将 FIRST.EXE 程序加到相应的程序组中。

2.2.2 MAKE 文件

在编译与链接时，最好使用 MAKE 文件(可从 NMAKE 实用程序中读出)。下列 FIRST.MAK 文件可编译并且链接代码：

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: FIRST.EXE

FIRST.OBJ:FIRST.C

FIRST.EXE: FIRST.OBJ FIRST.DEF
link /NOD/A:16 FIRST,FIRST.nul,libw slibcew,FIRST.def
rc FIRST.exe
```

用户可从配套软盘上取得这个文件，或用 DOS 的 COPY CON 命令进入此文件，还可以使用任何文本编辑器或产生 ASCII 文件的字处理器等进入这一文件，打入下列单词并且按下 Enter 键来编译用户程序：

```
make first.mak
```

MAKE 文件执行所有必需的任务。

2.3 头文件

头文件 Windows.H 包含在 Software Development Kit(SDK)中，它必须包含于每个 Windows 应用程序中。所有 Windows 函数的原型、多种数据类型、消息及其他常数都定义于

这个头文件中。

如果浏览这个文件，用户将大致了解这些定义。经过一些实践后，会发现这些新的数据类型常常比 C 语言中的标准类型更容易阅读和理解。例如，LPSTR 类型是一个指向字符串的远程指针，用 char for * 在 WINDOWS. H 内定义。

定义不同句柄的多种 typedef 也在头文件中，这些句柄用于管理已装载的对象。

Windows 系统在一个内部表中存储有关这些对象的信息，句柄是对此表入口的引用。

除了通常的句柄之外，还存在下列句柄：

句柄	函数
HBITMAP	物理位图句柄
HBRUSH	物理刷句柄
HCURSOR	光标资源句柄
HDC	显示环境句柄
HFONT	物理字模句柄
HICON	图标资源句柄
HMENU	菜单句柄
HPALETTE	逻辑调色板句柄
HPEN	物理画笔句柄
HRGN	逻辑区句柄
HSTR	字符串资源句柄

所有句柄的长度都相同，即数据类型为 WORD。若总使用恰当的句柄类型，那么所列出的程序就便于阅读了。

2.4 WinMain

由于 WinMain 函数是所有应用程序的入口点，一个 Windows 应用程序就必须有一些其他函数。WinMain 函数代替标准 C 程序中的 main()函数。

大部分 Windows 应用程序中，WinMain 初始化窗口类，生成并显示一个窗口。继而产生对消息循环的调用，此调用在应用程序关闭前一直存在。

WinMain 一般都具有同样的结构：

```
int PASCAL WinMain (hInstance,hPrevInstance,lpszCmdLine,CmdShow)
```

上行中各单词的含义如下：

PASCAL：说明从左到右地在堆栈中保存参数，并且由调用函数清除堆栈。

C 语言中通常反过来使用。然而，由于 Windows 直接调用 WinMain 函数，并且习惯于这种约定，就需要关键字 PASCAL。

hInstance，hPrevInstance：这些变量识别 Windows 中刚启动的应用程序。每次启动一个应用程序时，这种初始设置称为一个实例。

如果应用程序重新启动，一个新的 hInstance 将赋给一个新的副本。因此每个副本都是一个新的实例。

这种句柄对应于其他多任务系统中的任务 ID。变量 hPrevInstance 是上一个实例的句

柄。

如果应用程序初次启动，hPrevInstance 的一般值为 NULL。由 Windows 管理这些参数。

对于应用程序，还有一点比较重要，那就是应用程序代码在内存中不是一次性存储的。即使对于多次启动的应用程序来说也是如此。因此，代码必须是可重入的。这是所有 Windows 函数都具备的特点。

代码重入节省了存储空间。每一应用程序实例只有数据段不同。

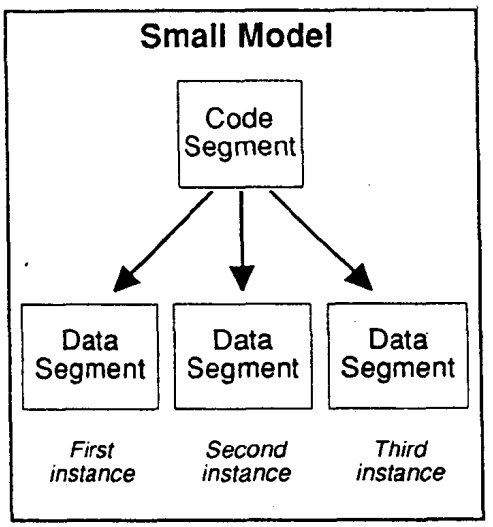


图 2.3　每个应用程序实例的不同数据段

lpszCmdLine：第三个参数是一个远程指针，指向一个以 NULL 结尾的字符串。当程序开始执行时，在此处传送参数。

nCmdShow：这个整型值根据用户的选择来决定应用程序作为一个窗口开始还是作为一个图标开始。如果显示窗口，则将值传递给 ShowWindows 函数。

2.5　初始化

在生成一个窗口前，用户必须登录一个窗口类。为了达到这个目的，用户需要 WINDOW.H 头文件中定义的结构 WNDCLASS。

```
typedef struct tagWNDCLASS {
    LPSTR lpszClassName;                          // Points to class name
    long (FAR PASCAL * lpfnWndProc)();            // Points to Window function
    HANDLE hinstance;                             // Handle of Instance which
                                                  // registers class
    int cbClsExtra;                               // Additional bytes,assigned to
                                                  // class structure
    int cbWndExtra,                               // Additional bytes,assigned to all
                                                  // Windows structures relative to
                                                  // this class
    HICON hIcon;                                  // Handle to icon which represents
                                                  // minimized window
```

```
    HCURSOR hCursor;                    // Handle to cursor displayed in window
    HBRUSH hbrBackground;               // Handle to brush, which determines
                                        // window background color
    LPSTR lpszMenuName;                 // Points to menu name when menu present
    Word style;                         // Style type
} WNDCLASS;
```

窗口类描述基本的窗口参数，这些参数相关于此类窗口并且含有一个指针，此指针指向附属的窗口函数。每个窗口类均被赋给一个窗口函数。由于存在某些预定义的窗口类，所以有一些预定义的窗口函数，如 Button 类等。

在第一个应用程序中，WNDCLASS 结构含有以下值：

lpszClassName："FIRST"；该名字通常与应用程序一致。

lpFnWndProc：FirstWndProc；可以随机选择。

hInstance：登录窗口类的实例句柄。

style：NULL；无使用风格

hbrBackground：GetStockObject(WHITE_BRUSH)；指定白色背景，因为 GetStockObject 返回白色标准刷句柄。

hCursor：LoadCursor(NULL,IDC_ARROW)；标准箭头，由常量 IDC_ARROW 指定，作为光标使用。

hIcon：LoadIcon(NULL,IDI_APPLICATION)；常量 IDI_APPLICATION 指定一个含框架的方框，这是窗口缩为最小时的显示状态。

lpszMenuName：NULL；无菜单。

cbClsExtra：NULL；无额外使用的字节。

cbWndExtra：NULL；无额外使用的字节。

前三个域必须含有恰当的数值，用于选择窗口类名和窗口函数。

一旦提供了 WINDCLASS 结构，就可以使用 RegisterClass 函数执行实际的登录过程，它返回指向结构的指针。

登录成功后，把值 TRUE 返回给主程序。若登录失败，则值为 FALSE。

使用 WinMain 查询返回值，这是因为，如果窗口类无效的话，那么生成一个窗口是不可能的。

2.6 生成窗口

以前面讨论过的初始化信息为基础，让我们看看窗口是如何产生的。

下面我们列出用于生成窗口的标准句法，同时还有 FIRST.C 源码中使用的代码：

```
句法                    例子                        //注释
CreatWindow(            CreatWindow(                //CreatWindow 函数
lpClassName,            "First"                     //确定类名
lpWindowName,           WS_OVERLAPPEDWINDOW         //标题条
dwStyle ,               This is my first application //窗口风格
X,Y,                    CW_USEDEFAULT,0,            //缺省窗口的安放
```

```
nWidth,nHeight,        CW_USEDEFAULT,0,        //X 和 Y 尺寸
hWndParent,            NULL,                   //无父窗口
hMenu,                 NULL,                   //无菜单
hInstance,             hInstance,              //应用程序实例
lpParam);              NULL);                  //无创建参数
```

lpClassName:这个名称必须恰好和 WINDCLASS 结构相匹配,否则不会生成窗口。

lpWindowName: 标题条中出现的文本。

dwStyle: 这是 32 位字段,Windows 将其看作各种信息的开关。这些标志的名称定义于 WINDOWS. H 中,以按位 OR 方式连接。由于高 16 位通常是比较有意义的标志,所以描述如下。

WS_OVERLAPPEDWINDOW 风格出现于列表中。这一风格是 WS_CAPTION、WS_SYSMENU, WS_ MINMIZEBOX, WS_ MAXIMIZEBOX, WS_ THICKFRAME 和 WS_OVERLAPPED 方式的结合。

基于这些风格,窗口有一标题条,一系统菜单,一个最小框和一个最大框。这一窗口称为主窗口,并且可以重新设置大小。

WS_OVERLAPPED 风格用于生成一个主窗口。这类窗口在 WinMain 中一般一次打开,直到应用程序结束。通常用户可在屏幕上改变主窗口的位置和尺寸。

除主窗口外还可生成子窗口和弹出窗口。

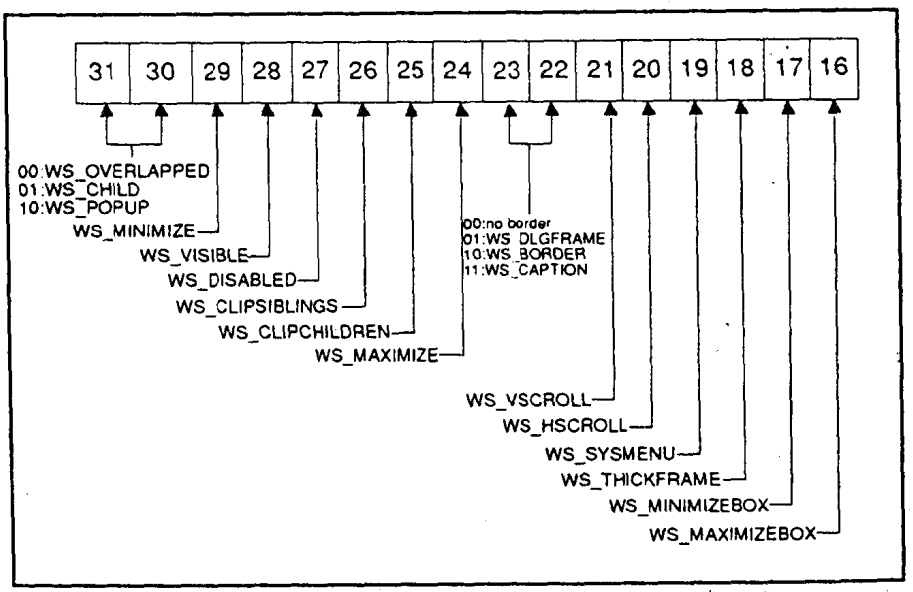


图 2.4 dwStyle 的 32 位字段的高 16 位示图

2.6.1 子窗口

子窗口从属于主窗口。一般在父窗口的客户区内。若父窗口收缩到最小,则与父窗口重叠的子窗口的任何部分都将被截断。

生成一子窗口必须使用 WS_CHILD 风格。子窗口通常用于将父窗口的客户区分割为几

个部分。

2.6.2 弹出式窗口

WM_POPUP 风格产生弹出式窗口。这类窗口分为两组:有父窗口和无父窗口。第一组(有父窗口)也从属于一主窗口。然而,与子窗口不同的是,弹出式窗口有自己的菜单。

第二组(无父窗口)可以不受主窗口的限制而移动。并且,如果主窗口已关闭,这类窗口也不受影响。重叠式窗口与弹出式窗口的唯一不同在于用户不能缩减一个弹出式窗口的尺寸。例如,所有对话框都是弹出式窗口。

X,Y,nWidth,nHeight:

参数 X 和 Y 确定窗口左上角的位置,对无父窗口的窗口和弹出式窗口,原点是屏幕左上角。对有父窗口的弹出式窗口和子窗口,坐标以父窗口的客户区原点为基准。其他参数确定窗口尺寸:CW_USEDEFAULT 值出现在列表中。这个值说明 Windows 应用程序将窗口放置到一个缺省位置且使用缺省尺寸。这些缺省值基于已启动的应用程序的数目。

hWndParent:父窗口句柄。如果用户正在生成一个有父窗口的弹出窗口或子窗口,则必须在此确定父窗口的句柄,以备正确引用之。

hMenu:菜单句柄。假定窗口包含菜单,用户即可在此写出它。当菜单加载时产生这个句柄。

hInstance:实例句柄。用户必须确定窗口所属的实例。这个句柄保证所处理的数据来自正确的数据段。

lpParam:生成窗口时,窗口函数可以使用附加数据。

如果 CreateWindow 能够生成一个窗口,那么 CreateWindow 返回新窗口句柄。许多后继函数需要这个句柄。

如果不能生成一个窗口,则返回的函数值为 NULL。

当 CreateWindow 生成一个窗口时,在屏幕上不会自动显示窗口。首先用 ShowWindow 和 UpdateWindow 函数显示窗口。由于 Windows 不处理工作区,由 ShowWindoww 负责绘制无客户区的窗口。

第二个参数确定窗口作为一个图标出现还是作为一个通常的窗口出现。这一信息设置在 CmdShow 变量中,此变量为 WinMain 的第四个参数。与 CmdShow 不同的常数值(例如 SW_SHOWMINNOACTIVE)可用于任何不是主窗口的窗口。这一常量要求窗口应作为图标显示,且先前活动的窗口应该保持活动状态。

UpdateWindows 提示应用程序(实际上是主窗口函数)刷新自身的工作区(客户区)。在样本应用程序中,窗口背景为白色。

2.7 MSG 结构

MSG 结构对剩下的处理非常重要。这种结构包含来自应用程序消息队列的消息。这些消息由窗口函数处理。

2.7.1 消息队列

Windows 是一个面向消息的系统，即系统内的所有信息交换都是靠传送和接收消息来进行的。

例如，某些输入类型产生消息，这些消息由 Windows 临时存储，交由相应的窗口函数在适当的时候处理。

采用这种方式，应用程序可与来自外界的输入分开。

正如在图 2.5 中见到的一样，消息可来源于键盘、鼠标器、时钟、Windows 系统、另一个应用程序或应用程序本身。

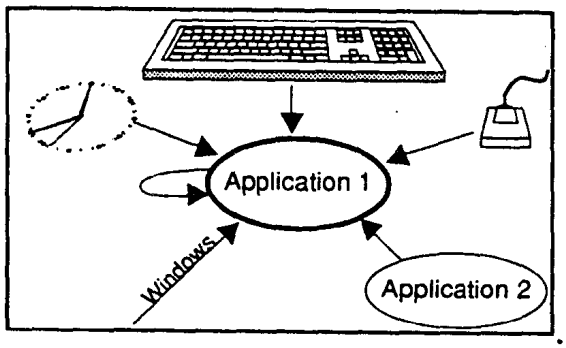


图 2.5 消息的几种来源

Windows 使用消息队列处理所有消息。一种是整个系统的系统消息队列，另一种是每个应用程序的应用程序消息队列。这些队列被看作一个循环缓冲区。

所有鼠标、键盘、时钟事件都由鼠标、键盘或系统驱动程序调用 USER.EXE 库中的函数安放到这个队列中。USER.EXE 库的另一部分从缓冲区中取得消息，传送到已定义的 MSG 结构，并且放置到相应的应用程序消息队列中。

2.7.2 俘获

Windows 必须向当前拥有鼠标的应用程序传递所有鼠标消息(例如，有鼠标“capture”的应用程序)。通常，一旦光标处于窗口中，该窗口就可浮获鼠标消息。Windows 放置所有鼠标消息于窗口的应用程序消息队列中。用户也可以在鼠标离开窗口后使用 SetCapture 函数重新俘获鼠标。

2.7.3 焦点

Windows 总能知道哪个窗口当前占用键盘，这也称为“focus”(焦点)。通常可根据标题条的颜色识别活动窗口和焦点。

用户可通过给定的组合键(Alt+Esc 或 Alt+Tab)或通过鼠标触发一新窗口来向新窗口传递焦点。

当用户按下某键时，Windows 首先在系统消息队列中放置这一事件，然后传递到适当的应用程序消息队列。

焦点和鼠标俘获可随时被赋给不同的窗口。然而，在特定时间只能有一个窗口可以有焦点或鼠标俘获。

现在让我们回到 MSG 结构。此结构由六个区域组成，前四个区域最为重要。应用程序消息队列向 MSG 结构提供数值。

```
typedef struct tagMSG {
    HWND hwnd;
    WORD message;
    WORD wParam;
    LONG lParam;
    DWORD time;
    POINT pt;
} MSG;
```

hWnd：标识接受消息的窗口。一个应用程序的所有实例只使用一个应用程序消息队列。由于实例和窗口的联系通过 CreatWindow 函数建立起来，所以每个窗口句柄也对应一个实例。

message：确定消息。这个域包含消息号。在 WINDOWS. H 头文件中包含消息号的名字。

wParam，lParam：虽然此域可包含扩充的信息，但是是否有效还要依赖于消息本身。例如，在击键消息中，从此域能有效地得知哪一键被按下。

time：确定取得消息的时间。

pt：指定取得消息时鼠标指针的位置。

2.8 消息循环

消息循环支持 MSG 结构。这一循环由 while 指令生成且至少包含两个函数：GetMessage（或 PeekMessage）和 DIspatchMessage。

GetMessage 函数在应用程序消息队列中取得下一消息并且将其放置在 MSG 结构中。因此第一个参数也就是 MSG 变量的地址。其他参数置为 NULL，说明所有消息应接受处理。

如果应用程序消息队列中无消息，应用程序就将 CPU 的控制权移交给 Windows。于是 Windows 可将 CPU 控制权交给下一个应用程序，由应用程序处理其队列。

同时，保留在 GetMessage 函数中的前一个应用程序等待下一次操作。Windows 使用一个链接的任务表管理处于等待状态的应用程序。这种方式称为无优先级多任务处理。

然而，在有优先级时间片机制的多任务处理中，占有 CPU 和需要处理消息的运行任务在一定时间内不被中断，只有在无其他消息供处理时才被中断。

消息的实际处理过程出现在相应的窗口函数而不是 WinMain 中。消息由 DispatchMessage 函数传递给窗口过程。

通过用户调用 RegisterClass 和 CreateWindow 函数时生成的内部结构，Windows 知道哪个窗口函数是正确的。Windows 能够访问类名并从 hWnd 句柄找出正确的窗口过程。这一句柄是 MSG 变量的第一个参数。

消息循环除 GetMessage 和 DispatchMessage 外还可以包含其他函数。通常，TranslateMessage 函数设置在这两个函数之间。这一函数需要处理键盘输入。描述输入的章节将对这个问题作更多的说明。

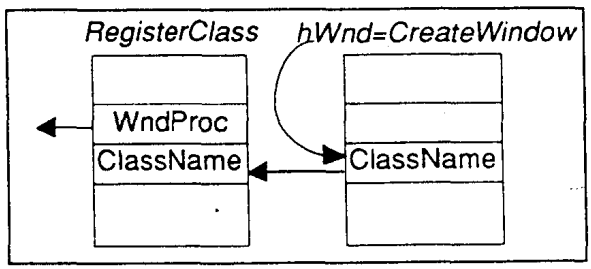


图 2.6　Windows 从 hWnd 中找出正确的窗口句柄

当 GetMesage 的返回值为 FALSE 时结束消息循环。例如，用户关闭应用程序时就将发送 WM_QUIT 消息。有关处理的详细顺序将在后面说明。

2.9　窗口函数

每个 Windows 应用程序必须至少有一个窗口函数。然而，这个函数的规模却可以大不相同。

```
Long FAR PASCAL FirstWndProc(hWnd,message,wParam,lParam)
```

每个窗口函数所传送的参数与 MSG 结构的前四个参数一致，因为消息和附加信息在窗口过程中得到处理。窗口过程一次性地加载到内存中，由一个应用程序的所有实例共用。

Windows 能够由内部结构根据 hWnd 参数确定 hInstance。这一参数使 Windows 能够区分独立的实例。

所有窗口函数必须说明为 FAR PASCAL。与以前在 WinMain 中描述过的一样，PASCAL 说明参数在栈中是从左到右进行放置的，并且由调用函数清除堆栈。使用 FAR 是因为在 Windows 调用窗口过程时产生了一个段间跳转(向另一个段跳转)。

在窗口过程中，用 Switch 指令来处理不同消息，只有应用程序的重要消息才必须接受处理，例如，所有来自鼠标的消息常常就是不重要的。所有不重要的消息都被传送到一个缺省窗口过程。通常对 WM_DESTROY 消息作如下处理：

```
Case WM_DESTROY;
    PostQuitMessage(0);
    break;
```

2.10　缺省窗口函数

```
LONG DefWindowProc(hWnd,Msg,wParam,lParam)
```

这一函数已由 Windows 定义，称为 DefWindowProc。此函数对必须响应的消息作预定的处理。

例如，按下左鼠标按钮通常会激活一个应用程序，鼠标将处于此应用程序的窗口中。实际上，按下左鼠标按钮将产生 WM_LBUTTONDOWN 消息，这一消息由缺省窗口过程处理。缺省窗口过程也可能忽略许多其他消息。

2.11 消息的进一步说明

Windows 中所有的消息可分为 11 组。这些组提供关于消息的附加信息：

- 窗口管理消息
- 初始化消息
- 输入消息
- 系统消息
- 剪接板消息
- 系统信息消息
- 控制消息
- 通知消息
- 滚动条消息
- 无客户区消息
- 多文档界面消息

上述所有消息都以字母“WM”开始，“WM 是“WindowsMessage”的缩写。这些消息适用于任何窗口。然而，也存在只相关于特定窗口的消息。例如，BM_SETCHECK 消息就代表已被激活的一个按钮。

以上所有消息都是有一个值在 0 到 WM_USER-1 之间的系统消息。类似于所有其他消息，WM_USER 也在 WINDOWS. H 头文件中定义。还可生成用户自身的消息系统。自定义消息的十六进制值，必须在 WM_USER 与 7FFFH 之间。

同样地，也存在有队列消息和无队列消息。队列消息设置在应用程序的消息队列中。要获取一个队列消息，可以使用 GetMessage。使用 DispatchMessage 向窗口过程传送一个队列消息。与消息在队列中的情况不同，无队列消息直接传给窗口函数。通常应快速处理的消息都是无队列消息。

例如，WM_CREATE 消息就是一个无队列消息。这一消息是对 GeatWindows 函数的调用而产生的结果，并且是窗口过程中第一个得到处理的消息。用户可同时向应用程序传送有队列和无队列消息。

PostMessage 与 SendMessage 函数负责向一个应用程序传送消息。正如函数名所示，PostMessage 放置消息于消息队列中，SendMessage 向窗口传送消息。这两个函数作为窗口函数有相同的参数格式(下列之一)：

```
SendMessage(hWnd,message,mParam,lParam)—window function
PostMessage(hWnd,message,wParam,lParam)—Application message queue
```

2.11.1 WM_PAINT 消息

WndProc 处理的第一条消息为 WM_PAINT。这条消息在 Windows 程序设计中是很重要的。当窗口客户区域的一部分或全部变为“无效”，以致必须刷新时，它将通知程序。

客户区域怎么会变得无效呢？在最初创建窗口的时候，整个客户区域都是无效的，因为程序还没有在窗口上画什么东西。第一条 WM_PAINT 消息(通常发生在 WinMain 中调用

UpdateWindow 时)指示窗口过程在客户区域上画一些东西。

在用户改变窗口的大小后,客户区域重新变为无效。WndClass 结构的 style 参数设置为标志 CS_HREDRAW 和 CS_VREDRAW。这一风格指示 Windows,在窗口大小改变后,使整个窗口无效;窗口过程接收到一条 WM_PAINT 消息。

在用户令窗口显示为一个图标,然后再次将窗口恢复到以前的大小时,Windows 将不会保存客户区域的内容。在图形环境下,这里涉及的数据量很大。因此,Windows 令窗口无效。窗口过程接收一条 WM_PAINT 消息,并自动恢复其窗口的内容。

在移动窗口以致其相互重叠时,Windows 不保存一个窗口中被另一个窗口所遮盖的内容。在这一部分不再被遮盖之后,它就被标志为无效。窗口过程接收到一条 WM_PAINT 消息,以刷新窗口的内容。

对 WM_PAINT 的处理几乎总是从一个 BeginPaint 调用开始:

```
hdc = BeginPaint (hwnd,&ps);
```

它以一个 EndPaint 调用结束;

```
EndPaint (hwnd,&ps);
```

在这两个调用中,第一个参数都是程序的窗口句柄,第二个参数是指向类型为 PAINTSTRUCT 的结构的指针。PAINTSTRUCT 结构包含一些窗口过程,可以用来刷新客户区域的信息。

在 BeginPaint 调用中,如果客户区域的背景还未被删除,则由 Windows 删除之。它使用用以登录窗口类的 WNDCLASS 结构的 hbrBackground 域中指定的刷子来删除背景。在 HELLOWIN 中,这是一个白色后援刷子。这就意味着,Windows 将通过把窗口背景着为白色来删除窗口背景。BeginPaint 调用令整个客户区域有效,并返回一个"设备环境句柄"。设备环境指的是物理输出设备(如视频显示器)及其设备驱动程序。在窗口的客户区域显示文本和图形需要设备环境句柄。但是不能用从 BeginPaint 返回的设备环境句柄在客户区域之外绘图,读者可以试一试。EndPaint 释放设备环境句柄,使之不再有效。

如果窗口过程不处理 WM_PAINT 消息(这是很罕见的),它们必须被传送给 DefWindowProc。DefWindowProc 只是依次调用 BeginPaint 和 EndPaint,以使客户区域有效。

在调用完 BeginPaint 之后,WndProc 接着调用 GetClientRect:

```
GetClientRect (hwnd,&rect);
```

第一个参数是程序窗口的句柄。第二个参数是一个指针,指向一个名为 rect 的变量,该变量在 WndProc 中定义为 RECT 类型。

RECT 是一个"矩形"结构,在 WINDOWS.H 中定义。它有四个 int 域,分别为 left,top,right 和 bottom。GetClientRect 将这四个域设置为窗口客户区域的尺寸。left 与 top 域通常设置为 0,right 和 bottom 域设置为客户区域的宽度和高度(像素点数)。

WndProc 除了将该 RECT 结构指针作为 DrawText 的第四个参数传送外,不再对它做其他处理:

```
DrawText (hdc,"Hello,Windows!",-1,&rect,
    DT_SINGLELINE|DT_CENTER|DT_VCENTER);
```

DrawText 可输出文本(正如其名字一样)。由于该函数要输出文本,第一个参数是从 BeginPaint 返回的设备环境句柄,第二个参数是要输出的文本,第三个参数为-1,指示文本串

是以字节 0 终结的。

最后一个参数是一系列位标志，它们均在 WINDOWS.H 中定义。标志指示了文本必须显示在一行，水平方向和垂直方向都位于第四个参数指定的矩形中央。一旦客户区域变为无效（正如在改变大小时所发生的情况一样），WndProc 就接收一条新 WM_PAINT 消息。WndProc 通过调用 GetClientRect 获取变化后的窗口的大小，并在新窗口的中央显示文本。

2.11.2 WM_DESTROY 消息

WM_DESTROY 消息是另一条重要消息。这一消息指示，Windows 正在根据用户输入的命令来清除窗口。该消息是用户在程序的系统菜单上选择 Close 或者按下 Alt-F4 时发生的。

可通过调用 PostQuitMessage 以标准方式响应 WM_DESTROY 消息：

```
PostQuitMessage (0);
```

该函数在程序的消息队列中插入一条 WM_QUIT 消息。前面提到过，GetMessage 对于除了 WM_QUIT 之外的从消息队列中取出的所有消息都返回非 0 值。它导致 WinMain 退出消息循环，并终止程序。

2.12 应用程序的退出

在这一部分我们将举例来讨论消息是如何处理和链接的。

正如用户在图 2.7 中所见到的一样，当退出应用程序时（例如，按 Alt+F4 键），Windows 产生 WM_CLOSE 消息。于是在一系列其他消息之后，Windows 将 WM_CLOSE 消息放置于应用程序的消息队列中。

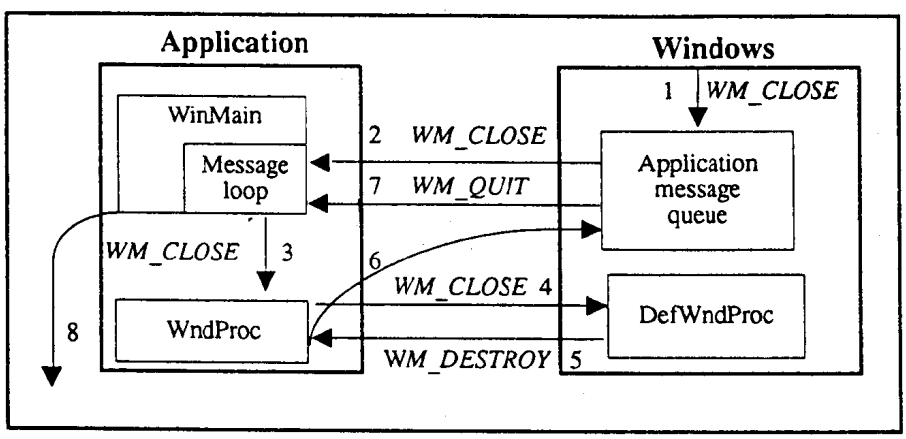


图 2.7 退出应用程序时，Windows 产生 WM_CLOSE 消息且置于消息队列

GetMessage 函数从队列中取得 WM_CLOSE，然后 DispatchMessage 将其传递给窗口过程。在较小的应用程序中，WM_CLOSE 并不接受处理，而是传向缺省窗口过程。

缺省窗口过程调用 DestroyWindows 函数。这一函数删除窗口并且将 WM_DESTROY 消息直接向窗口过程传送而不使用队列。通常，应用程序用 PostQuitMessage 函数来响应 WM_DESTROY。

此函数产生 WM_QUIT 消息，放置于应用程序消息队列中。若 GetMessage 从队列中取出这一消息，GetMessage 的返回值将为 FALSE。

出现这种情况时，消息循环结束，且关闭整个应用程序。

2.13 模块定义文件

除了源文件外，链接程序还需要一个模块定义文件。链接程序用此文件链接已编译的源文件和 Windows 程序库，这样才能生成一个可执行的 Windows 应用程序。

让我们再看看第一个应用程序的模块定义文件：

```
NAME First

DESCRIPTION 'lst Windows Application'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS FirstWndProc @1
```

2.13.1 模块定义文件的关键字

所有模块定义文件包含特定的关键字，确定编译和链接应用程序的过程中所需要的信息。下面给出一套完整的关键字：

关键字	注释
NAME/LIBRARY	模块或 DLL 名称
DESCRIPTION	描述
EXETYPE	Windows 或 OS/2
STUB	"This Program requies Microsoft Windows"
DATA	数据段属性
CODE	代码段属性
HEAPSIZE	属部堆的尺寸
STACKSIZE	局部栈的尺寸
SEGMENT	扩充的代码段
EXPORTS	输出函数
IMPORTS	输入函数

让我们按其在文件中出现的顺序进一步考察这些关键字：

NAME：Windows 应用程序名；否则链接器发出嘟嘟声。

LIBRARY：生成 DLL；与关键字 NAME 不同，使用 LIBRARY（以后详述）。

STUB:WINSTUB. EXE 应用程序已存在,并且当用户试图在 DOS 下启动应用程序时调用它。于是 WINSTUB. EXE 显示句子"This Program requires Microsoft Windows"。在 WINSTUB. EXE 执行后,系统返回到 DOS 提示下。

CODE、DATA:代码和数据段都被定义成 PRELOAD 和 MOVEABLE。由于应用程序是使用 SMALL 模式生成的,所以恰好有一个代码段和一个数据段。若应用程序当前不占用 CPU,并且 Windows 需要用于其他应用程序的内存空间,则 Windows 可在内存中同时移动这些段。CODE 关键字也包括 DISCARDABLE 参数。当内存空间有限时,这个参数确定代码可从内存中移出。

参考图 2.8,观察 Windows 如何分配内存。

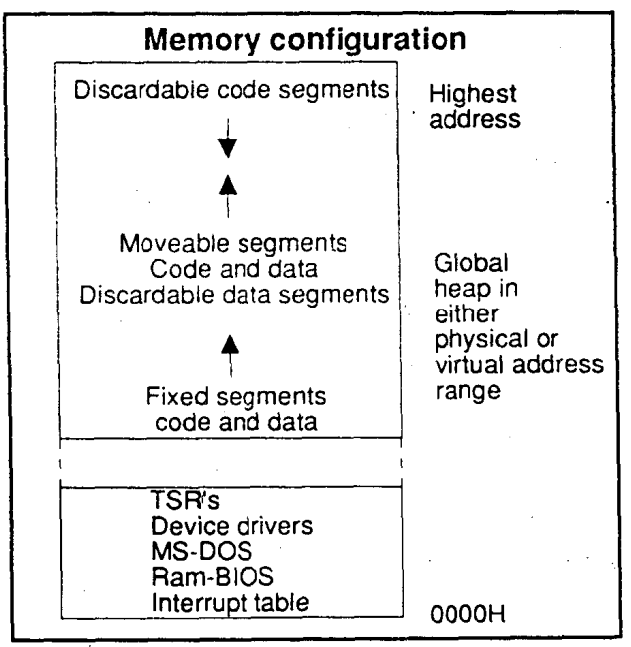


图 2.8 Windows 如何分配内存

如果应用程序分配一个固定段,并且在固定段中,所有其他段都是可移动段,Windows 就尽量靠近堆底部来移动这个固定段。这种移动产生于 Windows 重新排列所有可移动段或可废弃段的时候。

虽然重整内存有助于减少碎片,但是这种处理要花费时间。所以,用户在编程过程中不应过多使用带有 FIXED 属性的段。

DATA 项中的 MULTIPLE 选项确定每次应用程序重新启动时,生成一个新的实例,同时也生成一个新的数据段。一般情况下,每个 Windows 应用程序都使用此选项。由于 DLL 只能有一个数据段,所以使用 SINGLE 来代替 MULTIPLE。

STACKSIZE:每个 Windows 应用程序必须有一个堆栈,其尺寸按字节来确定。堆栈可用作函数自变量的临时存储区。一个小的 Windows 应用程序至少需要使用 4096 个字节。DLL 没有自身的堆栈。

HEAPSIZE:确定局部堆的大小,最少为 256 字节。

EXPORTS:这一关键字定义名称并供用户选择所有将输出的函数的序数。一个函数必须在 Windows 或另一个应用程序调用时输出。在第一个样本应用程序中,输出函数是 FirstWndProc 窗口过程。

由 Windows 调用的所有函数称为回调函数,序数是区分这些函数的另一种方式,这类数字可为任何整型值。

IMPORT:此处用户必须确定应用程序调用的所有函数。这些函数既未在源文件中也未在静态链接到应用程序的程序库中定义。这些函数是由 Dynamic Link Libraries(DLL)定义的。

2.13.2 动态链接库(DLL)

Windows 包含以下动态链接库(DLL):KERNEL.EXE,USER.EXE 和 GDI.EXE。与静态程序库不同,DLL 并不实际链接到应用程序。应用程序本身只包含 DLL 中所定义的函数的交叉引用。仅当应用程序启动和必要时,适当的 DLL 才被装入。由于这些函数代码必须是可重入的,所以,即使在不同应用程序访问同一函数的情况下,DLL 都只装入内存一次。这样可节省 RAM 内存空间。

2.13.3 IMPORTS 和 EXPORTS 示例

我们将使用一个例子来论述关键字 IMPORTS 和 EXPORTS 间的联系。

首先,假定一个 Windows 应用程序(PROG.EXE)正在使用一个名叫 AddString 的函数。这个函数来自一个用户的 DLL(STRLIB.EXE)。这样就使得 DLL 的定义文件和 Windows 应用程序变得重要起来。

用户也可带 IMPORTS 关键字来使用 AddString 函数的序数(例如,IMPORTS STRLIB.4)。但是,如果 DLL 由十个或二十个而不只是由一个函数组成,并且如果 Windows 应用程序想使用所有这些函数,那么就要求多次书写 IMPORTS 关键字。

幸运的是,用户可使用 IMPORT LIBRARY 来解决这个问题。输入程序库包含所请求函数的 DLL 和序数的信息,而不是其函数代码。

Microsoft 输入程序库 LIBW.LIB 含有三个 Windows DLL:GDI.EXE、USER.EXE 和 KERNEL.EXE。使用 IMPLIB.EXE 实用程序生成用户的 Dynamic Link Libraries。

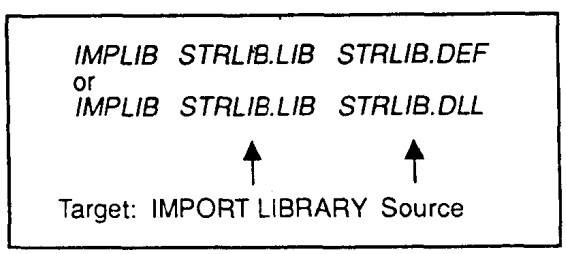


图 2.9

在链接 Windows 应用程序 PROG.EXE 的过程中,用户必须确定输入程序库 STRLIB.LIB 和 LIBN.LIB。当然,用户也可以在定义文件 PROG.DEF 中省略 IMPORTS 关键

字。

2.13.4 动态链接范例

以下例子演示了在 Windows 应用程序运行过程中，如何找到一个来自 DLL 的函数。

每个可执行应用程序都有一个头标。相对于一个平常的 DOS 而言，一个 Windows 应用程序在这个头标中含有扩充信息。例如，所有将要输入的函数都写在链接程序的输入名表中。还有其他表，例如入口表和资源表。

用户首次启动 Windows 应用程序时，扩充信息放置在一个称为 DataBase 的固定内存段中。由于 DLL 也存在同样的情况，所以也适用于 DLL USER. EXE。

当用户启动 FIRST 应用程序时，程序会立即调用 GetMessage 函数。由于用户处于链接过程中，所以将看到程序名和 DLL 的序数而不是函数引用。Windows 将从 Windows 应用程序数据库转移到 USER. EXE 数据库。

动态链接过程中，有一种适用于在 DLL 内定义的所有函数的“弹簧板”类型。用户可以转移到包含 GetMessage 定义的段地址，或者在内存中不存在当前所必要的段的情况下，使用汇编指令 Load Segment。

若使用某两种方法，Windows 就装入这个段，并且将“弹簧板”上的单元内容改为当前装入的段。这样就可分支转移到 GetMessage 函数。

后面将介绍如何编写自己的 DLL。

2.14 Windows 程序设计的注意事项

即使有了对 FIRST. C 的说明，读者对程序的结构和原理可能仍然觉得神秘。在为传统环境编写简单的 C 程序时，整个程序可能包含在 main 函数中。而在 FIRST. C 中，WinMain 只包含了登录窗口类、创建窗口、从消息队列中取出消息和发送消息所必须的代码。

程序的所有实际动作均在窗口过程中发生。在 FIRST. C 中，这些动作不多，它只是在窗口中显示一个文本串。但是在后面的章节中，读者将发现，Windows 程序所做的一切，都是响应发送给窗口过程的消息。这是概念上的主要难点之一，在开始编写 Windows 程序之前，必须先搞清楚。

2.14.1 Windows 应用程序中的调用关系

前面我们提到过，程序员应已熟悉了调用操作系统的思路。例如，C 程序员使用 open 或 fopen 函数打开文件。编译程序提供的库函数具有调用操作系统以打开文件的代码。

但是 Windows 不同，尽管 Windows 有 1000 个以上的函数可供程序调用，但 Windows 也调用用户程序，比如前面定义的窗口过程 WndProc。窗口过程与一个窗口类相联，窗口类是程序调用 RegisterClass 登录的。基于该类所创建的窗口使用这个窗口过程来处理窗口的所有消息。Windows 通过调用窗口过程，从而给窗口发送一条消息。

在第一次创建窗口时，Windows 调用 WndProc。在窗口清除时，Windows 也调用 WndProc。窗口改变大小、移动或变成图标时，从菜单中选择某一项时，滚动滚动条或按下鼠标按钮时，以及窗口客户区必须被刷新时，Windows 都要调用 WndProc。

所有这些调用都以消息的形式进行。在大多数 Windows 程序中，程序的主要部分都用来处理消息。Windows 可以发送给窗口过程大约 140 条不同的消息，它们都以 WM 打头的名字标识，且都在 WINDOWS. H 中定义。

实际上，从程序外调用程序内的例程这一思想，在传统的程序设计中并非闻所未闻。C 中的 signal 函数可以捕获 Ctrl-Break。读者可能有用汇编语言截获硬件中断的经验，也可能在 Microsoft Basic 中使用过 ON 构造之一。Microsoft 鼠标驱动程序提供了一种方法，非 Windows 程序可以用这种方法知道鼠标活动。

在 Windows 中，这种概念扩展为包括一切事件。窗口中发生的一切都以消息的形式传给窗口过程。然后，窗口过程以某种方式响应这条消息，或者将消息传给 DefWindowProc，以进行缺省处理。

在 FIRST. C 中，窗口过程的 wParam 和 lParam 参数除了作为传递给 DefWindowProc 的参数外，不再有其他用处。这些参数给出了关于消息的其他信息。参数的含义与具体消息相关。

让我们来看一个例子。一旦窗口的客户区大小发生改变，Windows 就调用窗口的窗口过程。窗口过程的 hwnd 参数是改变大小的窗口的句柄。message 参数为 WM_SIZE。WM_SIZE 消息的 wParam 参数为常量 SIZENORMAL，SIZEICONIC，SIZEFULLSCREEN，SIZEZOOMSHOW 和 SIZEZOOMHIDE(在 WINDOWS. H 中定义为数值 0 至 4)之一。wParam 参数指示窗口是最大化、最小化、还是隐藏(是另一个窗口最大化的结果)。lParam 参数包含了新窗口的大小。新宽度和新高度均为 16 位值，粘接在一起成为 32 位的 lParam。WINDOWS. H 中提供了帮助程序员从 lParam 中取出这两个值的宏。

有时候，DefWindowProc 处理完消息后会产生其他消息。例如，假设用户运行 FIRST. C，并用键盘或鼠标在系统菜单中选择 Close。DefWindowProc 处理这一键盘或鼠标输入，在检测到用户选择了 Close 选项之后，它给窗口过程发送一条 WM_SYSCOMMAND 消息。WndProc 将这条消息传给 DefWindowProc。DefWindowProc 以给窗口过程发送一条 WM_CLOSE 消息来响应之。WndProc 再次将它传给 DefWindowProc。DefWindowProc 调用 DestroyWindow 来响应这条 WM_CLOSE 消息。DestroyWindow 导致 Windows 给窗口过程发送一条 WM_DESTROY 消息。WndProc 再调用 PostQuitMessage，将一条 WM_QUIT 消息放入消息队列中，以此来响应 WM_DESTROY 消息。这条消息导致 WinMain 中的消息循环终止，然后程序结束。

2.14.2 多任务处理

我们说消息循环内的 GetMessage 调用很重要，这还有一条理由。除了一些必须处理硬件中断(如时钟、键盘、鼠标以及串行端口)的设备驱动程序外，Windows 总是将 FIRST. C 当成在此系统下运行的唯一程序。Windows 不会任意离开 FIRST. C 去运行其他程序。例外是在调用 GetMessage 时，如果 FIRST. C 的消息队列中没有消息等待，而其他程序的消息队列中有一些消息，则 Windows 从 FIRST. C 切换到 FIRST. C。这样就对了，不是吗？

可以这样来考虑：在大多数情况下，当程序调用 Windows 中的函数时，可以期望在处理该函数后，控制会在一定的时间内返回给程序。但是，在调用 GetMessage 时，可能 Windows 还没有返回消息。如果此时程序的消息队列不包含消息，而其他程序的消息队列中有消息，则

Windows 可以利用 GetMessage 调用时由于消息队列为空而引起的空闲，切换到有消息等待的程序。结果是，Windows 使用一种“跳跃式”的多任务。有时程序有很多任务，运行于 Windows 下的所有其他程序就好像在此期间停止了运行一样。

这种特征一般不称为“跳跃式多任务”，而称为“无优先级多任务”。Windows 在程序之间实现多任务是通过切换来实现的。但是 Windows 的做法与传统的多任务系统的做法不同，它不是根据硬件时钟的转动，为每个程序分配一个小时间片。Windows 在程序从消息队列中取出消息时进行多任务切换。

实际过程比我们所讨论的略微复杂一些：Windows 程序还在 PeekMessage 和 WaitMessage 调用时进行切换，但是它们不及 GetMessage 普遍。另外，WM_PAINT 和 WM_TIMER 消息被当成低优先级的消息，所以 Windows 可以在队列中只有 WM_PAINT 和 WM_TIMER 消息的情况下切换到另一程序。

2.14.3 进队消息与不进队消息

我们已经谈到过，Windows 给窗口发送消息，这意味着 Windows 调用窗口过程。但是，Windows 程序也有一个消息循环，它调用 GetMessage 从消息队列中取出消息，并调用 DespatchMessage 将消息发送给窗口过程。

那么，Windows 程序是依次等待消息（类似于普通程序中等价的键盘输入），然后将消息送到某地方去的吗？或者，它是直接从程序外面接收消息的吗？实际上，两种情况都存在。

消息可以分为“进队的”和“不进队的”。进队的消息是由 Windows 放入程序消息队列中的，且在消息队列中取出和发送。不进队的消息在 Windows 调用窗口过程时直接送给窗口。结果是窗口过程获取窗口的所有消息，包括进队的和不进队的。从结构上看，Windows 程序很清晰，因为它们有一个消息处理中心。将进队消息“邮送”给消息队列，而不进队消息“发送”给窗口过程。

进队消息基本上是用户输入的结果，以击键（如 WM_KEYDOWN 和 WM_KEYUP）、击键产生的字符（如 WM_CHAR）、鼠标移动（WM_MOUSEMOVE）和鼠标按钮按动（WM_LBUTTONDOWN）的形式给出。进队消息还包含时钟消息（WM_TIMER）、刷新消息（WM_PAINT）和退出消息（WM_QUIT）。不进队消息则是其他消息。在许多情况下，不进队消息来自进队消息。在窗口过程内将一个不进队消息传给 DefWindowProc 时，Windows 通常通过给窗口过程发送其他消息来处理这条消息。

这一过程显然很复杂，但幸运的是，其中大部分是由 Windows 解决的，不关用户程序的事。从窗口过程的角度来看，这些消息是以一种有序的、同步的方式进出的。窗口过程可以处理它们，也可以不处理。由于这个原因，窗口过程被称做“最后的陷阱”。消息几乎将所有影响窗口的事件告知窗口过程。

不进队消息通常是调用某个 Windows 函数或者调用 SendMessage 显式发送一条消息的结果（消息也可以调用 PostMessage 来放入消息队列）。

例如，在 WinMain 调用 CreateWindow 时，Windows 创建窗口，并在此过程中给窗口过程发送一条 WM_CREATE 消息。在 WinMain 调用 ShowWindow 时，Windows 给窗口过程发送 WM_SIZE 和 WM_SHOWWINDOW 消息。在 WinMain 调用 UpdateWindow 时，Windows 给窗口过程发送一条 WM_PAINT 消息。

消息不同于硬件中断。在窗口过程中处理一条消息时，程序不会被另一个消息所中断。只有在窗口过程调用一个产生新消息的函数时，消息过程才会在函数返回前处理这条消息。

消息循环与窗口过程不是并发运行的。在窗口过程处理进队消息时，它是在 WinMain 中调用 DispatchMessage 的结果。直到窗口过程处理完消息之后，DispatchMessage 才会返回。

但是请注意，窗口过程必须是可重入的。也就是说，Windows 经常调用 WndProc，产生一条用这一消息调用 DefWindowProc 的新消息。这是 Windows 程序需要 8KB 堆栈的理由之一；可以在模块定义文件(.DEF)中定义栈的大小。在大多数情况下，窗口过程的可重入性不会带来任何问题，但是程序员应该意识到这一点。

在许多情况下，窗口过程必须保存它从消息中获取的信息，并在处理另一条消息时使用这些信息。这些信息可以存储在窗口的 static 变量或全局变量中。

当然，读者将在下面几章对此有一个更清楚的理解，因为窗口过程将处理更多的消息。

2.14.4 小结

毫无疑问，读者已从本章得出一条结论：Windows 程序设计与传统环境(如 MS—DOS)下的程序设计大相径庭。不会有人声称 Windows 程序设计很容易。

刚开始学习 Windows 程序设计时，作者决定如以前那样，在学习新操作系统或新语言时，编写一个简单的“打印二进制数”的程序 WINDUMP 来显示文件的内容。在传统的 MS-DOS 环境下，这样的程序涉及到命令行处理、基本的文件 I/O 和屏幕输出格式。但是，这次的 Windows 二进制打印程序变成了一个“怪物”。它要求学习菜单、对话框、滚动条等等。把它作为一个 Windows 程序，这实在是一个错误，要求学习的东西一下子太多了。

这个程序与作者先前编写的二进制打印程序完全不同。WINDUMP 不是从命令行中获取文件名，而是作者给该程序取的名字。提供一个列表框，列出当前目录中的所有文件。WINDUMP 不是以一种简单的电传打字机的方式将输出送到屏幕上，而是具有滚动条，可以移动到文件的任意位置。另一个特殊的优点是，可以运行 WINDUMP 的两个副本，以逐行比较两个文件。

读者应当考虑一下：想不想编写出新颖的、具有创意的用户界面程序(界面中包括菜单、对话框、滚动条和图形)？如果想，则接下来的问题是：是自己编写所有菜单、对话框、滚动条和图形代码，还是利用 Windows 内已有的东西呢？换句话说，是学习使用 1000 种函数调用容易，还是自己编写更容易？是使用消息驱动式体系结构编程容易，还是坚持使用传统模式下的几种不同输入源容易？

如果读者决心自己来编写用户界面，那么最好放下这本书，立即投入工作。乐意用 Windows 来编程的读者，请继续学习在窗口中如何显示和滚动文本及图形。

第三章　基本输出处理

这一章说明 Windows 环境怎样产生输出，讨论显示环境、笔、刷子以及 WM_PAINT 消息，同时列出一些输出函数。在第十三章和第十四章，我们还要继续介绍设备环境和一些图形输出函数。

3.1　设备环境

Windows 中必须有一设备环境产生输出。

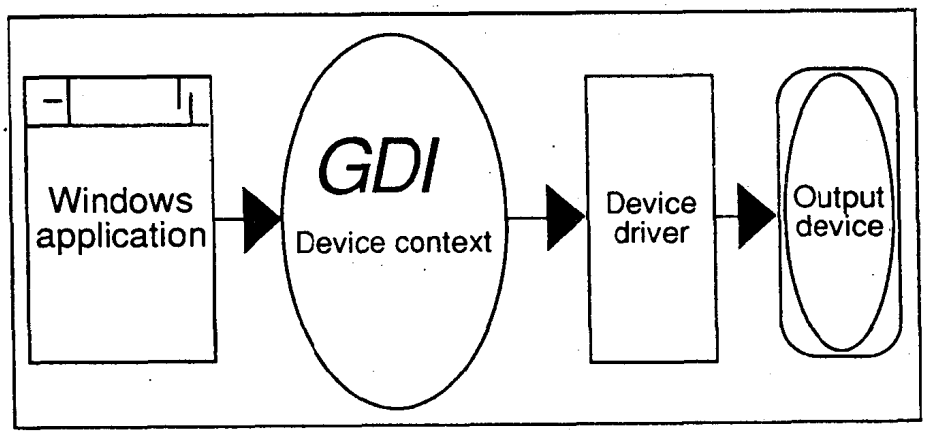


图 3.1　Windows 中的设备环境

设备环境是 Windows 应用程序、设备驱动程序和输出设备(如监视器)之间的连接，Windows 的每个应用程序都能使用 GDI 函数在输出设备上产生输出。

GDI 负责传递对设备驱动程序来说是设备独立的引用，然后驱动程序将这些 GDI 引用翻译成设备独立的操作。

3.1.1　设备环境属性

因此，产生输出的过程总是通过获得设备环境开始。有许多函数能用于这一方面，以后会更详细地讨论这些函数。

检索到的设备环境句柄是一个几乎对所有 GDI 函数都适用的参数。Windows 用设备环境存储某种属性，所以一个设备环境包括某个数据结构，这个数据结构不能被直接访问，必须用 GDI 函数改变单一属性。新的设备环境被填以缺省值。

在设备环境中，背景色属性与 WNDCLASS 结构中定义的不是同一属性。窗口类中的背景是一个刷子。

若有必要，Windows 用这一刷子擦掉客户区（擦除是由 DefWindowProc 中的消息 WM_ERASEBKGND 引起的）。然而，刷子不是设备环境的成分。

欲改变 WNDCLASS 结构中的许多设置（包括背景刷），可使用函数 SetClassWord。当背景模式设置为 OPAQUE 时，背景色就充满字母和直线间的空隙。当背景模式设置为 TRANSPARENT 时，背景色就被忽略。

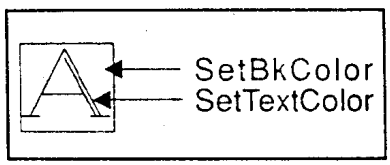


图 3.2　前景和背景色

3.1.2　设备环境类型

设备环境有好几种类型，所用的设备环境类型依赖于指派它的输出设备。显示环境总是指整个窗口或窗口的一部分。设备环境应用于整个设备（例如，打印或整个屏幕）。

内存设备环境是一真实设备环境的备份，定位于内存中。这一设备环境在带有位图工作时使用。每一设备环境类型都有自己的函数偶，由设备环境产生并删除。

设备环境类型	产生和删除的函数偶
裁剪区域	BeginPaint → EndPaint
整个客户区	GetDC → ReleaseDC
整个窗口	GetWindowDC → ReleaseDC
设备 DC	CreateDC → DeleteDC
内存 DC	CreateCompatibleDC → DeleteDC

也可用整个窗口的设备环境绘制系统菜单上的标题条等。因为这一函数是为“粉刷”作用而创建的，所以不推荐将它用于正常绘图。

3.1.3　显示环境类型

除了整个窗口的显示环境外，还有三种显示环境：共享、类和私有。这些显示环境只在客户区产生输出。

共享设备环境对整个 Windows 来说是缺省的设备环境，并且使用最为频繁。当调用 BeginPaint 或 GetDC 时，设备环境连同其句柄将被从五个显示环境缓存中检索出来。

能用这一句柄设置上面提及的属性并产生期望的输出。一旦输出完成，就必须用 EndPaint 或 ReleaseDC 将显示环境返回缓存，所有设置的属性都将丢失。

当释放设备环境以后，直到获得下一个设置环境才能有输出出现。如果只处理一个消息而不是好几个，就应获得并释放共享设备环境。这一点很重要，因为不能确信在释放设备环境时应发生的消息实际会发生。

如果不释放共享设备环境和其他四个缓存中的设备环境，Windows 就会忽略所有对设

备环境的一系列请求。这一缓存不适用于类设备环境和私有设备环境。

与双击鼠标按钮的鼠标消息类似，这些设备环境类型中的两个在 WNDCLASS 结构的风格参数中指明：CS_CLASSDC 和 CS_OWNDC。

类设备环境用于相关于同一 CS_CLASSDC 风格类的所有窗口，既使这些窗口属于不同的实例。在用它工作之前必须获得设备环境。然而，以后不必释放这一设备环境给缓存，因为设备环境不是由缓存生成的。

如果只有一个窗口带有类显示环境工作，则必须保留设置属性，因为 EndPaint 和 ReleaseDC 函数无任何作用。如果同一类的另一个窗口也要访问这个设备环境，则首先必须用 BeginPaint 或 GetDC 获取自己的设备环境句柄。

除了裁剪区和坐标区之外，所有 Windows 为新窗口设置的属性在设备环境中都保持不变，连同其他属性、坐标初值都在显示环境中。它指明窗口的左上角坐标（相对于屏幕的左上角坐标）。

当窗口改变类设备环境中的属性时（例如文本颜色），这一变化影响到后续使用此设备环境的每个窗口。显然这将造成意料不到的后果。

私有显示环境被分配给一个窗口。这一窗口所属的类必须具有 CS_OWNDC 风格。这一设备环境必须只产生一次——它一直存在，直到窗口关闭为止。释放共享设备环境的两个函数不影响私有设备环境。

私有设备环境的缺点是 Windows 必须为每一存储的设备环境保留 800 个字节的空间。

用函数 SaveDC 存储共享显示环境的属性，以便能设置和使用其他属性。以后能用函数 RestoreDC 恢复老环境。然而，不能在这两个函数之间释放显示环境。

3.2 WM_PAINT 消息

在 Windows 应用程序中，应只用 WM_PAINT 消息产生输出。

3.2.1 起源

WM_PAINT 消息的产生有几种方式。无论什么时候 Windows 确定客户区的内容不再是最新了，系统就产生一条 WM_PAINT 消息，并将其放在应用程序消息队列中。逐渐地整个客户区（除了被称作裁剪区的长方形区域外）均变为无效。

Winows 只允许在裁剪区域内绘图。例如，如果试图在客户区的保护部分输出文本，文本将不会出现。因此，Windows 可保证当前区域不被覆写。可用 WM_PAINT 消息检索无效区域并可测试它。

窗口过程获得 WM_PAINT 消息的一种方式是部分窗口已经处于另一窗口下面，并且当前要成为前景窗口。

获得该消息的另一种方式是，用户改变了窗口尺寸，并且两个风格参数 CS_HREDRAW 和 CS_VREDRAW 已在 WNDCLASS 结构中指定。如果这些参数没有设置，WM_PAINT 消息只有在窗口扩大时才会产生。

在其他几种情况下，Windows 总是保留显示器自身的区域，以便以后能恢复它。然而，如果保存失败，Windows 就发送一条 WM_PAINT 消息。例如，当 Windows 移动窗口重迭部分的

对话框或下拉菜单时，就会发生这种情况。

Windows 保存被覆写的显示区域并且以后再恢复它，这还有其他情况。例如，当光标移动且跨越客户区时。

程序员通过使用某个函数（如函数 UpdateWindow）也能产生一条 WM_PAINT 消息，此函数在消息循环中，并在产生第一条 WM_PAINT 消息之前被 WinMain 调用。Windows 立即将这一消息传递给窗口过程作为非队列消息。这就是用 WNDCLASS 结构中的背景刷子绘制客户区的方法。

如前所述，所有输出都应在处理 WM_PAINT 时进行。然而，若某事件发生，比如输出期间遇到按键事件，就需要一个函数在任何时候都能产生 WM_PAINT 消息，InvalidateRect 和 Invalidate 函数就用于完成此工作。

InvalidateRect (hwnd,lpRect,bErase)

hwnd：识别应该修改的窗口。

lpRect：指明一长方形结构包含裁剪区，若整个客户区应重画的话则为 NULL。

bErase：如果此参数置为 TRUE，则在产生新的输出之前擦掉背景；若置为 FALSE，则背景保留。

3.2.2 处理消息

消息 WM_PAINT 总是消息队列中的最后一条消息，所有其他消息都用 GetMessage 先从消息队列中取出，这可防止窗口内容被修改。

消息队列中总有一条 WM_PAINT 消息，如果最后一条消息还未处理，而一条新的消息又到了，那么就从所提供的旧的或新的裁剪区坐标处创建新的无效长方形。

有两个风格参数 CS_VREDRAW（垂直重画）和 CS_HREDRAW（水平重画）设置于 WNDCLASS 结构中，它们确定下面事实：当窗口修改尺寸时是否有消息产生，以及是整个客户区还是只有新的部分设置为无效。

如果 CS_VREDRAW 或 CS_HREDRAW 被置位，那么当用户修改窗口尺寸时，就产生 WM_SIZE 和 WM_PAINT 消息，并且整个客户区是无效的。如果只有窗口被扩大，并且客户区的新部分是无效的，那么就只产生 WM_PAINT 消息。

如果输出是由 WM_PAINT 产生的，那么必须用函数偶 BeginPaint 和 EndPaint 获取和释放显示环境。

函数 BeginPaint 包含参数 hWnd 和 &ps，hwnd 是窗口句柄，整个窗口或其客户区被重画。&ps 参数代表 PAINTSTRUCT 变量的地址；当调用这一函数时，此变量填充当前值。

Paintstruct：

域	说明
hdc	设备环境句柄
fErase	TRUE==擦掉背景
rcPaint	裁剪区域的长方形结构
fRestore	Windows 保留的域
fincUpdate	Windows 保留的域

fgbReserved[16]　　Windows 保留的域

BeginPaint：设备句柄定位在函数 BeginPaint 的第一个参数和返回值中。通常 fErase 设置为 TRUE，以便在产生新的输出之前擦掉背景。欲做到这一点，Windows 产生一条 WM_ERASEBKGND 消息。

WM_REASEBKGND 消息通常传递给缺省窗口过程，这导致背景被 WNDCLASS 结构中指定的刷子绘制；长方形结构是 RECT 数据类型，且定义无效区的左上角和右下角以及客户区的左上角。

EndPaint：设备环境句柄又被释放，且无效区域置为有效。

如果 WM_PAINT 消息传递给 DefWindowProc，将发生下列情况：

```
case WM_PAINT:
    BeginPaint (hWnd,&ps);
    EndPaint (hWnd,&ps);
break;
```

如果在 BeginPaint 和 EndPaint 之间没有调用一个函数，那么先前的无效长方形区域被设置。同样地，Windows 发送另一条消息以擦掉背景。所有不是来自消息 WM_PAINT 的输出也被擦掉。

不能通过 break 语句响应 WM_PAINT 消息来改变这一点。因为 EndPaint 没有被调用，所以无效区域不被设置为有效。Windows 然后又发送 WM_PAINT 消息。结果是，应用程序以及整个系统也许都会挂起。

3.3　输出函数

较简单的输出函数可分成基本的文本函数和图形函数组。所有函数(除 LineDDA 外)都共享第一个参数。我们将在各章节中讨论也能产生输出的位图函数。

3.3.1　文本函数

TextOut 和 DrawText 是使用最频繁的产生输出的文本函数，以后会更详细地讨论它们。也存在其他结构更为复杂的文本函数(如 GrayString)。

BOOL TextOut (hDC,X,Y,lpString,nCount)

参数	数据类型	说明
hDC	HDC	设备环境句柄
X	int	X 坐标文本开始
Y	int	Y 坐标文本开始
lpString	LPSTR	指向文本缓冲区的指针
nCount	int	字符数目

除非先前已由 SetTextAlign 函数指定了环境，否则文本起始点的 X 和 Y 坐标都以逻辑坐标指定。如果使用了缺省映射模式 (MM_TEXT)，则参考点是客户区的左上角。

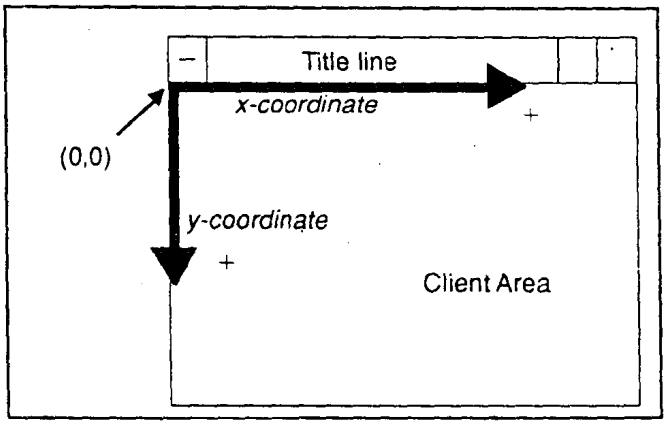


图 3.3

字符串的长度是必要的，因为函数不能识别任何以 NULL 字节结尾的字符串。文本中也不应有任何 ASCII 控制字符，如 Linefeed 或 Tab 等，因为输出时它们不能被正确解释。

同样地，设备环境中当前画笔的位置没有被此函数改变，除非先前已由函数 SetText 指定了环境。

键盘和鼠标样本程序使用了 TextOut 函数。这些例子的输出是一个“黑框”。

```
/* * MouseWndProc: * * * * * * * * * * * * * * * * * */
/* * Main windowfunction: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL MouseWndProc(hWnd,message,wParam,lParam)
    HWND hWnd;
    unsigned message;
    WORD wParam;
    LONG lParam;
{
    static char chTextDown[35];
    static char chTectUp[35];
    static BOOL bCapture = FALSE;
/* * * * * Output parameters * * * * * * */
    HDC hDC;
    PAINTSTRUCT ps;
/* * * * * * * * * * * * * * * * * * */

/* * Check button and movement status,display on screen * * * * * * * */

    switch (message)
    {
        case WM_RBUTTONDOWN:                                    // Right button pressed
            wsprintf(chTextDown,"WM_RBUTTONDOWN: %x,%d,%d",
                wParam,LOWORD(lParam),HIWORD(lParam));
            InvalidateRect(hWnd,NULL,TRUE);
            break;
```

```
        case WM_LBUTTONUP:                                    // Left button released
            wsprintf(chTextUp,"WM_LBUTTONUP: %x,%d,%d",
                wParam,LOWORD(lParam),HIWORD(lParam));
/* * * * * Generate WM_PAINT * * * * * */
            InvalidtaeRect(hWnd,NULL,TRUE);
/* * * * * * * * * * * * * * * * */
            break;
        case WM_LBUTTONDBLCLK:                                // Double-click
            if (bCapture)
            {
                ReleaseCapture();
                bCapture = FALSE;
            }
            else
            {
                SetCapture(hWnd);
                bCapture = TRUE;
            }
            break;
        case WM_MOUSEMOVE:                                    // Beep during mouse movement
            MessageBeep(0);
            break;
/* * * * * * OUTPUT * * * * * * * * */
        case WM_PAINT:                                        // Redraw client area window
            hDC = BeginPaint (hWnd,&ps);
            TextOut(hDC,10,10,chTextDown,strlen(chTextDown));
            TextOut(hDC,10,30,chTextUp,strlen(chTextUp));
            EndPaint(hWnd,&ps);
            break;
/* * * * * * * * * * * * * * * * * */

        case WM_DESTROY:                                      // Destroy window
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,lParam));
            break;
    }
    return(0L);

}
```

文本应该在右鼠标按钮被按下时输出。不是立即由 WM_RBUTTONDOWN 产生输出，而是由函数 InvalidateRect 累积输出。这将产生一条 WM_PAINT 消息，且第二个参数导致整个客户区被标记为无效。

在 WM_PAINT 消息中，显示环境句柄是用 BeginPaint 获得的。然后由函数 TextOut 输出文本。最后，EndPaint 函数释放设备环境，并且将整个客户区置为有效。

int DrawText (hDC,lpString,nCount,lpRect,wFormat)

参数	数据类型	说明
hDC	HDC	设备环境句柄

lpString	LPSTR	文本缓冲区指针
nCount	int	字符个数
lpRect	LPRECT	指向格式化文本区域的指针
wFormat	WORD	格式化方法

可方便地用 DrawText 函数将文本格式化。指向长方形结构的指针作为第四个参数传递。文本被写在长方形中。第三个参数包含值-1 而不是以 NULL 字节结尾的字符串长度；值-1 指明函数应该计算文本长度。

最后一个参数包含不同标志的组合和一个在长方形中居中显示文本的标志。

除非 DT_SINGLELINE 标志被设置，否则硬回车(13)和换行(10)都解释为换行。

DrawText 函数举例

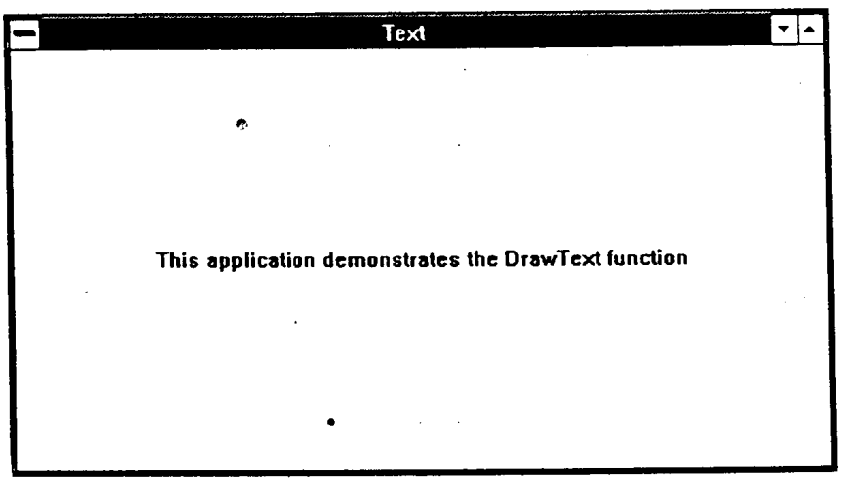


图 3.4

这一样本应用程序测试除 DrawText 函数外的三种情况。这些情况适用于风格参数 CS_HREDRAW 和 CS_VREDRAW。

新消息	说明
WM_PAINT	提示重画后发送

新函数	说明
BeginPaint	为重画准备窗口
EndPaint	标记绘制结束
GetClientRect	拷贝客户区坐标
DrawText	输出字符串

新结构	简要描述
PAINTSTRUCT	包含字符信息

RECT　　矩形结构

源代码:TEXT.C

```
/* * TEXT.C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays text in a window using the DrawText function * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                                    // Include windows.h header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

BOOL TextInit ( HANDLE );                               // Text initialization
long FAR PASCAL TextWndProc( HWND,unsigned,WORD,LONG);

/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                         // Current & previous instances
LPSTR lpszCmdLine;                                      // Long ptr to string after
                                                        // program name during execution
int cmdShow;                                            // Specifies the application
                                                        // window's appearance
{
    MSG msg;                                            // Message variable
    HWND hWnd;                                          // Window handle

    if (! hPrevInstance)                                // Initialize first instance
    {
        if (! TextInit( hInstance ))                    // If initialization fails
            return FALSE;                               // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * */

    hWnd = CreateWindow("Text",                         // Window class name
        "Text",                                         // Window caption
        WS_OVERLAPPEDWINDOW,                            // Overlapped window style
        CW_USEDEFAULT,                                  // Default upper-left x. pos
        0,                                              // Upper-left y pos.
        CW_USEDEFAULT,                                  // Default initial x size
        0,                                              // y size
        NULL,                                           // No parent window
        NULL,                                           // Window menu used
        hInstance,                                      // Application instance
        NULL);                                          // No creation parameters

    ShowWindow( hWnd,cmdShow );                         // Make window visible
    UpdateWindow( hWnd );                               // Update window
```

```
    while (GetMessage(&msg,NULL,0,0))                    // Message reading
    {
        TranslateMessage(&msg);                          // Message translation
        DispatchMessage(&msg);                           // Send message to Windows
    }

    return (int)msg.wParam;                              // Return wParam of last message
}

BOOL TextInit( hInstance )                               // Text initialization
HANDLE hInstance;
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcTextClass;                                // Main window class

    wcTextClass.hCursor = LoadCursor( NULL,IDC_ARROW );
                                                         // Mouse cursor
    wcTextClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                         // Default icon
    wcTextClass.lpszMenuName = NULL;                     // No menu
    wcTextClass.lpszClassName = "Text";                  // Window class
    wcTextClass.hbrBackground = (HBRUSH)GetStockObject( WHITE_BRUSH );
                                                         // White background
    wcTextClass.hInstance = hInstance;                   // Instance
    wcTextClass.style = CS_VREDRAW | CS_HREDRAW;         // Horizontal and.
                                                         // vertical redraw of client area
    wcTextClass.lpfnWndProc = TextWndProc;               // Window function
    wcTextClass.cbClsExtra = 0 ;                         // No extra bytes
    wcTextClass.cbWndExtra = 0 ;                         // No extra bytes

    if (! RegisterClass( &wcTextClass ) )                // Register window class
        return FALSE;                                    // Return FALSE if registration fails

    return TRUE;                                         // If registration is successful,
}                                                        // return TRUE

/* * TextWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL TextWndProc( hWnd,message,wParam,lParam )
HWND hWnd;                                               // Window handle
unsigned message;                                        // Message type
WORD wParam;                                             // Message-dependent 16 bit value
LONG lParam;                                             // Message-dependent 32 bit value
{

    HDC hDC;                                             // Device context handle
    PAINTSTRUCT ps;                                      // PAINTSTRUCT data structure
```

```
    RECT rect;                                          // RECT rectangle structure
    char chText[] = "This application demonstrates the DrawText function";

    switch (message)                                    // Process messages
    {
        case WM_PAINT:                                  // Client area redraw
            BeginPaint (hWnd,&ps);                      // Begin WM_PAINT process
            hDC = ps.hdc;
            GetClientRect(hWnd,&rect);                  // Get client area rectangle
            DrawText( hDC,chText,-1,&rect,DT_CENTER | DT_VCENTER | DT_SINGLELINE);
                                                        // Draw text centered in rectangle
            EndPaint(hWnd,&ps);                         // End WM_PAINT process
            break;                                      // End of this message process
        case WM_DESTROY:                                // Destroy window
            PostQuitMessage(0);
            break;                                      // End of this message process
        default:                                        // Send other messages to default
                                                        // window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                      // End of this message process
    }
    return(0L);
}
```

模块定义文件:TEXT.DEF

```
NAME Text

DESCRIPTION 'DrawText example'

EXETYPE WINDOWS

STUB 'WINSTUB.EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS TextWndProc @1
```

MAKE 文件:TEXT.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
```

```
all: TEXT.EXE

TEXT.OBJ: TEXT.C

TEXT.EXE: TEXT.OBJ TEXT.DEF
    link /NOD/A:16 TEXT,TEXT,nul,libw slibcew,TEXT.def
    rc TEXT.exe
```

运行 NMAKE 实用程序以产生 TEXT 应用程序：

```
nmake text.mak
```

TEXT.EXE 怎样工作

第一种情况下，风格参数 CS_VREDRAW 和 CS_HREDRAW 被使用，且将文本居中显示的长方形由函数 GetClientRect 计算。

此函数决定客户区的当前尺寸，并把该值放到 RECT 结构变量中。客户区的左上角和右下角定义在这一结构中，因为参考点是左上角，所以输入坐标(0,0)。

文本应该水平或垂直地居中。为使标志 DT_VCENTER 起作用，标志 DT_SINGLINE 也必须被指定。

如果改变窗口尺寸，整个客户区就置为无效，这是因为 CS_HREDRAW 和 CS_VREDRAW 的缘故。因此，文本总是显示在客户区中央。

如果窗口尺寸减少得太多，以致使文本在长方形的一行上放不下，部分文本就显示不出来。发生这种情况是因为文本会写在 Windows 所保护的区域中。为确保文本被显示，应使用标志 DT_WORDBREAK 代替标志 DT_SINGLELINE。

在第二种情况下，文本被输出在由 GetClientRect 函数所确定的长方形中。然而，wcTextClass 风格在 WNDCLASS 结构中置为 0。

WM_PAINT 消息只有当窗口被扩大并且只有客户区的新部分被裁剪时才会产生。

如果用鼠标缩减窗口，使最低边缘在文本先前所定位置之上，文本就不再显示出来。

如果稍稍扩大窗口，文本也会消失，因为无效区域与输出区域不匹配。用不同的处理方法，若输出定位于所要画的长方形中，文本也许会再显示出来。

第三种情况是用另一长方形来显示文本。这就是 BeginPaint 输入 ps 变量中的无效区域。

函数 DrawText 的第四个参数现在称作 &ps.rcPaint 而不是 &rect。现在文本被写在裁剪区的中央。

设备环境的一些属性可应用于文本输出。SetTextColor 函数设置文本颜色，用 SetBKMode 和 SetBKColor 决定字符间隙是否应为不同的颜色。

背景色和文本色也可以以 Control Panel 设置的颜色为基准。

若系统色(在此例中指背景色和文本色)被改变，且应用程序接收到 WM_PAINT 消息，那么文本和背景都用新颜色显示。

```
SetTextColor (hDC,GetSysColor (COLOR_WINDOWTEXT));
SetBKColor (hDC,GetSysColor (COLOR_WINDOW));
```

GetSysColor 函数检索用颜色索引所指明的颜色的 RGB 值，每个窗口部分(其颜色由 WINDOWS 控制面板设置)都有一颜色索引。

3.3.2 基本图形函数举例

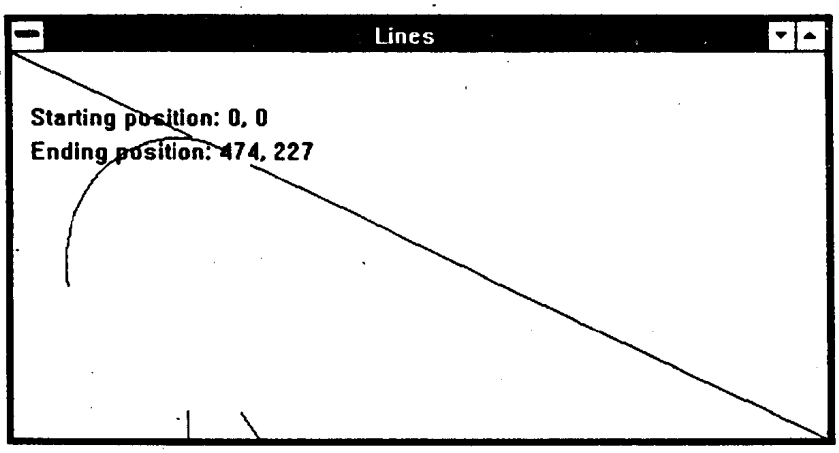


图 3.5

可用基本图形函数画线及封闭图形，如长方形和抛物线。这些函数要求的是逻辑单位坐标，所以所绘制的图形是设备独立的。

根据在设备环境中设置的图形模式，GDI 将逻辑单位转换成物理单位。

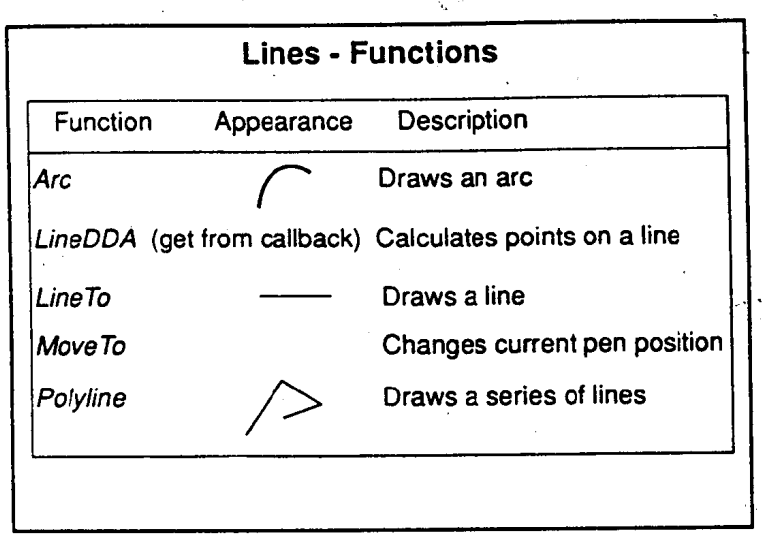


图 3.6 画线函数

函数 LineTo 从当前画笔位置到指定点间画一条直线，指定点不包含在内，这一点代表新的画笔位置。可用 MoveTo 函数改变画笔位置，并且用 GetCurrentPosition 得到位置值。

LineTo 是唯一使用这一位置的函数。LineTo 和 MoveTo 是改变这一位置的函数。

作为第二个参数，Polyline 函数有一个指向数组的指针。这是一个 POINT 数组，并且包含应该相互连接的点。下一参数指明点数。

四个点的 X 和 Y 坐标被传递给 Arc 函数，前两个点定义一长方形；后两点定义弧的起

点和终点。

最后一个函数 LineDDA 更为复杂。作为唯一的输出函数，LineDDA 不要求设备环境句柄。DDA 是“Digital Diferential Analyzer”的缩写，LineDDA 计算直线上两指定点之间的所有点。

这一函数本身不产生输出，而是调用回调函数产生输出。输出不只限于计算点。比如，也可能是椭圆。

新消息	说明
WM_LBUTTONDOWN	画一个弧

新函数	说明
GetCurrentPosition	确定当前画笔位置
LineTo	画线
MoveTo	改变当前画笔位置
Polyline	画一序列现

新宏	说明
MAKEPOINT	把 LONG 变量转换为 POINT
新结构	简要说明
POINT	X 和 Y 坐标结构

源代码：LINES. C

```
/* * LINES.C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Draws lines and figures using Arc,LineTo,MoveTo and PolyLine. * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                                    // Include windows.h header file
#include "string.h"                                     // Include string.h header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

BOOL LinesInit ( HANDLE );
long FAR PASCAL LinesWndProc( HWND,unsigned,WORD,LONG);

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                         // Current & previous instance
LPSTR lpszCmdLine;                                      // Long ptr to string after
                                                        // program name during execution
int cmdShow;                                            // Specifies the application
                                                        // window's appearance
{
    MSG msg;                                            // Message variable
```

```
    HWND hWnd;                                          // Window handle

    if (! hPrevInstance)                                // Initialize first instance
    {
        if (! LinesInit( hInstance ))                   // If initialization fails
            return FALSE;                               // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Lines",                        .// Window class name
        "Lines",                                        // Window caption
        WS_OVERLAPPEDWINDOW,                            // Overlapped window style
        CW_USEDEFAULT,                                  // Default upper-left x pos.
        0,                                              // Upper-left y pos.
        CW_USEDEFAULT,                                  // Default initial x size
        0,                                              // y size
        NULL,                                           // No parent window
        NULL,                                           // Window menu used
        hInstance,                                      // Application instance
        NULL);                                          // No creation parameters

    ShowWindow( hWnd,cmdShow );                         // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while (GetMessage(&msg,NULL,0,0))                   // Message reading
    {
        TranslateMessage(&msg);                         // Message translation
        DispatchMessage(&msg);                          // Send message to Windows
    }

    return (int)msg. wParam;                            // Return wParam of last message
}

BOOL LinesInit( hInstance )                             // Line initialization
HANDLE hInstance;                                       // Instance handle
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */
    WNDCLASS wcLinesClass;                              // Main window class

    wcLinesClass. hCursor = LoadCursor( NULL,IDC_ARROW );
                                                        // Mouse cursor
    wcLinesClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                        // Default icon
    wcLinesClass. lpszMenuName = NULL;                  // No menu
    wcLinesClass. lpszClassName = "Lines";              // Window class
    wcLinesClass. hbrBackground = (HBRUSH)GetStockObject( WHITE_BRUSH );
                                                        // White background
    wcLinesClass. hInstance = hInstance;                // Instance
    wcLinesClass. style = CS_VREDRAW | CS_HREDRAW;      // Horizontal
                                                        // and vertical redraw of client area
```

```
    wcLinesClass. lpfnWndProc = LinesWndProc;                    // Window function
    wcLinesClass. cbClsExtra = 0 ;                               // No extra bytes
    wcLinesClass. cbWndExtra = 0 ;                               // No extra bytes

    if (! RegisterClass( &wcLinesClass ) )                       // Register window class
        return FALSE;                                            // Return FALSE if registration fails
                                                                 // If registration is successful,
    return TRUE;                                                 // return TRUE
}

/* * LinesWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL LinesWndProc( hWnd,message,wParam,lParam )
HWND hWnd;                                                       // Window handle
unsigned message;                                                // Message type
WORD wParam;                                                     // Message-dependent 16 bit value
LONG lParam;                                                     // Message-dependent 32 bit value
{
    HDC hDC;                                                     // Handle to device context
    PAINTSTRUCT ps;                                              // Paint structure declaration
    RECT rect;                                                   // RECT rectangle structure
    static POINT pt;                                             // POINT point structure
    static POINT ptarray[4] = { 100,210,100,250,150,240,130,210 };
    LONG lPoint;
    char chText[50];

    switch (message)                                             // Process messages
    {
        case WM_LBUTTONDOWN:                                     // Left button clicked? Move
            pt = MAKEPOINT (lParam);                             // starting position to
            InvalidateRect(hWnd,NULL,TRUE);                      // new location
            break;                                               // End of this message process
        case WM_PAINT:                                           // Client area redraw
            hDC = BeginPaint (hWnd,&ps);                         // Begin WM_PAINT
                                                                 // Set,display starting position
            MoveTo(hDC,pt. x,pt. y);
            wsprintf(chText,"Starting position: %d,%d",pt. x,pt. y);
            TextOut(hDC,10,30,chText,strlen(chText));
                                                                 // Set,display ending position
            GetClientRect(hWnd,&rect);                           // Get client rectangle
            LineTo(hDC,rect. right,rect. bottom);                // Draw line
            lPoint = GetCurrentPosition(hDC);
            pt = MAKEPOINT(lPoint);
            wsprintf(chText,"Ending position: %d,%d",pt. x,pt. y);
            TextOut(hDC,10,50,chText,strlen(chText));
            pt. x = 0;                                           // Starting position default
            pt. y = 0;
                                                                 // Draw multiple lines ···
            Polyline(hDC,(LPPOINT)&ptarray,(int)4);
```

```
            Arc(hDC,30,50,160,200,140,20,20,140);                    // and arc
            EndPaint (hWnd,&ps);                                     // End WM_PAINT process
            break;                                                   // End of this message process
        case WM_DESTROY:                                             // Destroy window
            PostQuitMessage(0);
            break;
        default:                                                     // Send other messages to
                                                                     // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:LINES. DEF

```
NAME Lines

DESCRIPTION 'Output example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS LinesWndProc @1
```

MAKE 文件: LINES. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: LINES. EXE

LINES. OBJ: LINES. C

LINES. EXE: LINES. OBJ LINES. DEF
    link /NOD/A:16 LINES,LINES,nul,libw slibcew,LINES. def
    rc LINES. exe
```

如下所示运行 NMAKE,以创建 LINES 应用程序:

nmake lines. mak

LINE. EXE 如何工作

应用程序用 LineTo 函数从当前位置到客户区的右下角画线，客户区是借助 GetClientRect 函数计算出来的。画笔位置的缺省值是(0,0)，且存入变量 pt 中。

因此，第一次输出以及所有其他未被按下左鼠标按钮所禁止的输出都是从左上角到右下角画一对角线。终点(也就是当前画笔位置)由 GetCurrentPosition 函数计算并由 TextOut 输出。按下鼠标左按钮将指针所指的点转换成 pt 变量并产生 WM_PAINT 消息。MAKEPOINT 从 lParam 中获得这一点。当处理消息 WM_PAINT 时，MoveTo 函数把当前位置设置为这一点并用 TextOut 输出。最后起点 pt 被擦去，因为当设备环境被释放时，当前画笔位置自动恢复。

三条连线和带有固定点的弧被画出来。如果窗口比指定的坐标小，则表明图片只部分地显示出来。

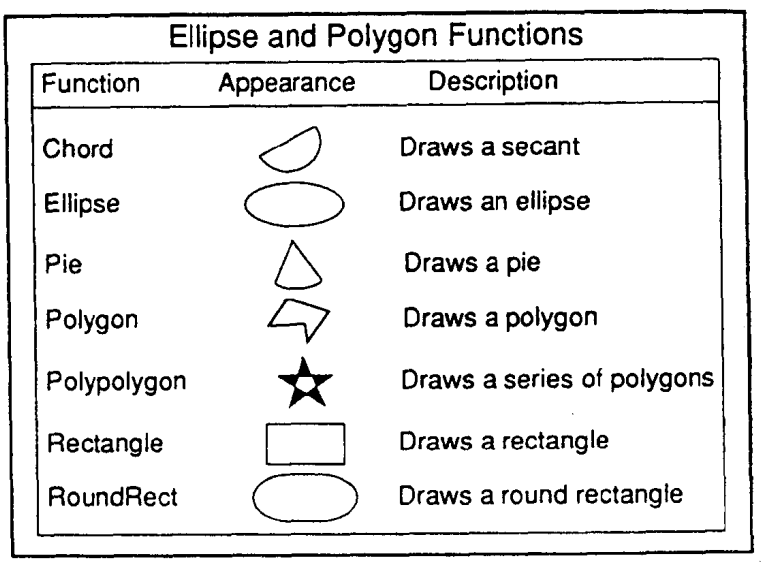


图 3.7　椭圆及多边形函数

除两个多边形函数外，其他函数结构相同。第一个参数通常是设备环境句柄。第二和第三个参数确定左上角。第四和第五个参数确定长方形右下角，此点或被显示，或被用作要画的图片的参考点。

RoundRect 函数(见图 3.8)也需要两个参数定义形成圆角的椭圆的高度和宽度。

另外两个点用 X 和 Y 坐标指定，告知 Chord(图 3.9)在何处与椭圆的部分弧相交。这两个点不必在不可见的长方形中。函数 Pie 和函数 Arc 接受同样的值。

可用两个多边形函数画任何图形，Polygon 函数与 Polyline 函数有相同的参数，除了把起点和终点连接起来之外。

PolyPolygon 函数产生几个闭合多边形，在 ALTERNATE 填充模式下，用当前刷子填充部分多边形。填充模式的当前设置无关紧要。虽然多边形能重叠，但这不是强制性的。

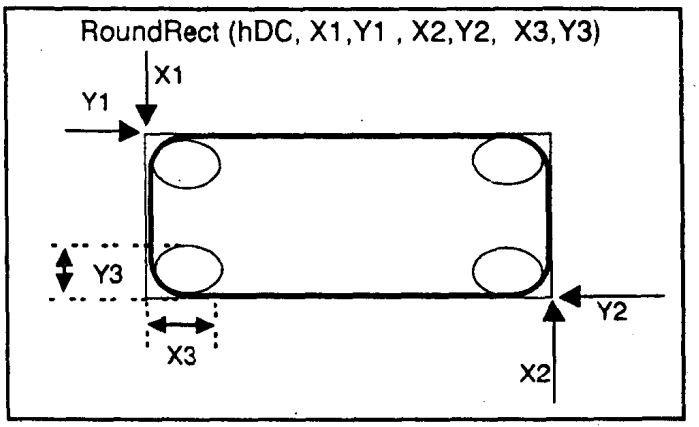


图 3.8 RoundRect 函数

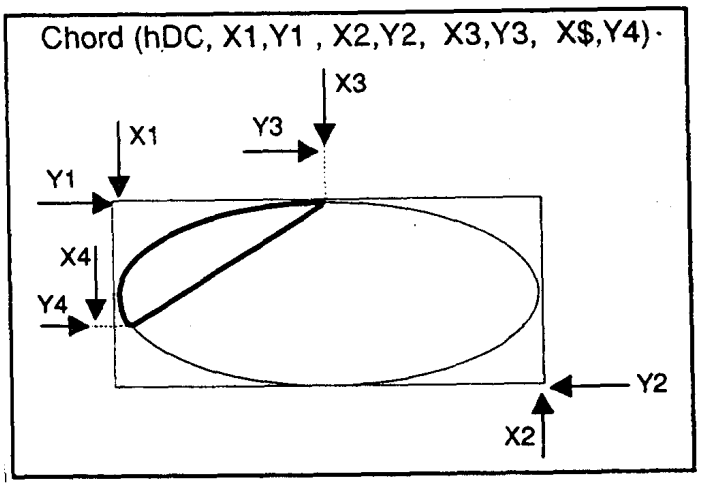


图 3.9 Chord 函数

SetPolyFillMode 函数为多边形设置填充模式。这一函数有两个设置：ALTERNATE 和 WINDING。根据所设置的模式，当前刷子要么填充全部区域，要么只填充封闭多边形的某些区域。

如果要填充所有的区域，就把 WINDING 关键字传给函数作为第二个参数。ALTERNATE 是更复杂的设置。

只有从外部使用奇数条线所能达到的受限区域才被填充。刷子不能绕过交叉部分。

下面的应用程序展示了图形设备接口(GDI)中基本图形函数的某些选择。

当处理 WM_PAINT 消息时，画出各种各样的图形：长方形、椭圆、圆的扇区、切割面和若干个多边形。

像所见到的那样，前五个图形函数的参数很相似。

新函数	简要说明
Chord	画一切面
Ellipse	画一椭圆

Pie	画一饼状图
PolyPolygon	画几个多边形
Rectangle	画一长方形
RoundRect	画一圆角矩形

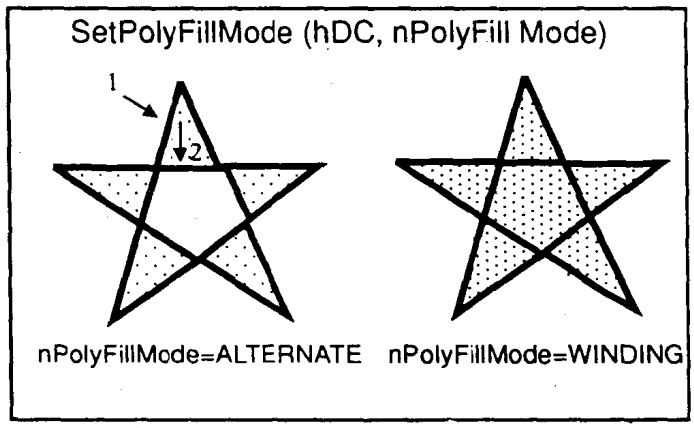


图 3.10 PolyPolygon 函数

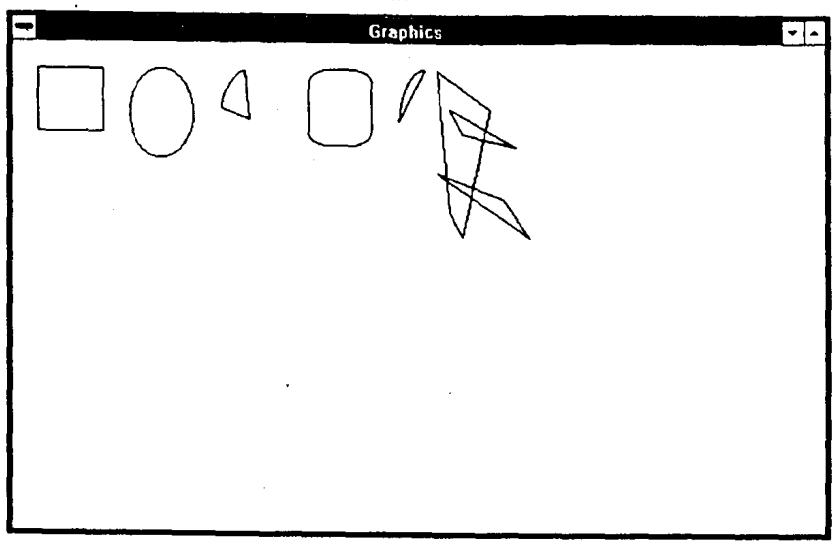


图 3.11 椭圆和多边形输出举例

源代码:GRAPHICS.C

```
/* * GRAPHICS.C * * *
```

```
#endif

long FAR PASCAL GraphicWndProc ( HWND,unsigned,WORD,LONG);
BOOL GraphicInit ( HANDLE );

int PASCAL WinMain ( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                          // Current & previous instance
LPSTR lpszCmdLine;                                       // Long ptr to string after
                                                         // program name during execution
int cmdShow;                                             // Specifies the application
                                                         // window's appearance
{
    MSG msg;                                             // Message variable
    HWND hWnd;                                           // Window handle

    if (! hPrevInstance)                                 // Initialize first instance
    {
        if (! GraphicInit( hInstance ))                  // If initialization fails
            return FALSE;                                // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */
    hWnd = CreateWindow("Graphics",                      // Window class name
        "Graphics",                                      // Window caption
        WS_OVERLAPPEDWINDOW,                             // Overlapped window style
        CW_USEDEFAULT,                                   // Default upper-left x pos.
        0,                                               // Upper-left y pos.
        CW_USEDEFAULT,                                   // Default initial x size
        0,                                               // y size
        NULL,                                            // No parent window
        NULL,                                            // Window menu used
        hInstance,                                       // Application instance
        NULL);                                           // No creation parameters

    ShowWindow( hWnd,cmdShow );                          // Make window visible
    UpdateWindow( hWnd );                                // Update window

    while (GetMessage(&msg,NULL,0,0))                    // Message reading
    {
        TranslateMessage(&msg);                          // Message translation
        DispatchMessage(&msg);                           // Send message to Windows
    }

    return msg. wParam;                                  // Return wParam of last message
}

BOOL GraphicInit( HANDLE hInstance )                     // Timer initialization,
                                                         // instance handle
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */
```

```
    WNDCLASS wcGraphicClass;                                    // Main window class

    wcGraphicClass.style = CS_VREDRAW | CS_HREDRAW;
                                                                // Horizontal and vertical
                                                                // redraw of client area
    wcGraphicClass.hCursor = LoadCursor( NULL,IDC_ARROW );      // Mouse cursor
    wcGraphicClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                                // Default icon
    wcGraphicClass.lpszMenuName = NULL;                         // No menu
    wcGraphicClass.lpszClassName = "Graphics";                  // Window class
    wcGraphicClass.hbrBackground = GetStockObject( WHITE_BRUSH );
                                                                // White background
    wcGraphicClass.hInstance = hInstance;                       // Instance
    wcGraphicClass.lpfnWndProc = GraphicWndProc;                // Window function
    wcGraphicClass.cbClsExtra = 0;                              // No extra bytes
    wcGraphicClass.cbWndExtra = 0;                              // No extra bytes

    if (! RegisterClass( (LPWNDCLASS) &wcGraphicClass ) )       // Register window class
        return FALSE;                                           // Return FALSE if registration fails
    return TRUE;                                                // If registration is successful,
}                                                               // return TRUE

/* * GraphicWndProc * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL GraphicWndProc( hWnd,message,wParam,lParam )
HWND hWnd;                                                      // Window handle
unsigned message;                                               // Message type
WORD wParam;                                                    // Message-dependent 16 bit value
LONG lParam;                                                    // Message-dependent 32 bit value
{
    PAINTSTRUCT ps;                                             // Paint structure declaration
    HDC hDC;                                                    // Device context handle
    static POINT ptarray[] = {330,20,340,130,350,150,370,50,330,20,340,50,350,70,390,80,
                              340,50,330,100,400,150,380,120,330,100 };
    static int iarray[] = {5,4,4 };

    switch (message)                                            // Process messages
    {
        case WM_PAINT:                                          // Client area redraw
            hDC = BeginPaint( hWnd,&ps );                       // Begin WM_PAINT

/* * Draw figures * * * * * * * * * * * * * * * * * * * * * * */
            Rectangle (hDC,20,20,70,70);
            Ellipse (hDC,90,20,140,90);
            RoundRect (hDC,230,20,280,80,30,20);
            Pie (hDC,160,20,210,100,180,20,160,50);
            Chord (hDC,300,20,340,120,320,10,300,60);
            PolyPolygon(hDC,(LPPOINT)&ptarray,(LPINT)iarray,3);
            EndPaint( hWnd,&ps );                               // End WM_PAINT
```

```
            break;                                              // End of this message process
        case WM_DESTROY:                                        // Destroy window
            PostQuitMessage(0);
            break;
        default:                                                // Send other messages to
                                                                // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                              // End of this message process
    }
return(0L);
}
```

模块定义文件:GRAPHICS. DEF

```
NAME Graphics

DESCRIPTION 'Draws ellipses,polygons,etc.'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS
GraphicWndProc @1
```

MAKE 文件:GRAPHICS. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: GRAPHICS. EXE

GRAPHICS. OBJ: GRAPHICS. C

GRAPHICS. EXE: GRAPHICS. OBJ GRAPHICS. DEF
    link /NOD/A:16 GRAPHICS,GRAPHICS,nul,libw slibcew,GRAPHICS. def
    rc GRAPHICS. exe
```

如下所示运行 NMAKE,以创建 GRAPHICS 应用程序:

```
nmake graphics. mak
```

与 Polyline 函数相似，PolyPolygon 函数以一个指向 POINT 结构数组的指针作为第二个参数。不管属于哪个多边形，所有的点都要在这里指定。

下一参数包括一个指向整数数组的指针，它确定多边形的点数。最后一个参数包含这些整数值的总数目。

与 Polygon 函数不同，这些多边形不是自动封闭的（多边形的第一个点必须给定两次）。虽然多边形填充模式 ALTERNATE 生效，但结果不可见，因为缺省模式和背景都是白色的。

3.4 绘图工具

改变某些 GDI“粉刷”工具以改变输出是可能的。笔、刷子、和字模是最重要的绘图工具。

每次输出时不必重置这些工具，因为它们的当前值是设备环境的属性，所以这些值当设备环境获得句柄时可被自动使用。

当以不同属性工作时，应遵循特殊的过程：

- 获得设备环境
- 得到绘图工具
- 在设备环境中选择工具
- 输出函数
- 在设备环境中选中老工具，然后删除第一个工具。

也可在缺省环境下工作或创造自己的工具。所有的缺省工具已在 Windows 系统中。欲设置不同的缺省属性，需要 GetStockObject 函数提供缺省工具的句柄。这一函数的唯一参数是传递过来的一个常量，这一常量设置理想的缺省工具（例如 GetSteckObject（WHITE_BRUSH））。

所有自定义的工具都具有以“Create”开始的函数。这些函数产生新的工具并获得一句柄，这些句柄有下列数据类型：HPEN、HBRUSH、HFONT。

3.4.1 使用工具

因为不需要设备环境以获得工具句柄，所以获取工具和设备环境的过程可以和先前描述的不同。

首先告诉设备环境，它必须重置属性。SelectObject 函数通过获得设备环境句柄以及新工具句柄来做到这一点。这一函数获取当前工具的句柄。

因为设备环境开始和结束时本应有同样的属性，所以必须保留这一句柄。函数 SelectObject 执行后，所有输出都使用新工具。

当所选的刷子、画笔和字模不再使用时，可用同样的函数再选用设备环境中的老工具。DeleteObject 可删除所有非缺省工具，因为只能有有限的句柄存在。设备环境在输出结束时被释放。

3.4.2 画笔

画笔用来画各种类型的线，从简单线(LineTo)到图像(如切面(Chord))。画笔确定三种属性：宽度、颜色和风格。Windows 包含下列缺省画笔：

- BLACK_PEN
- WHITE_PEN
- NULL_PEN

NULL_PEN 能画一没有边缘的图形，如椭圆。若想使线变宽一些或成为点划线，就要用 CreatePen 或 CreatePenIndirect 定义画笔。这些函数和标准函数的唯一不同点是传递的参数不同。

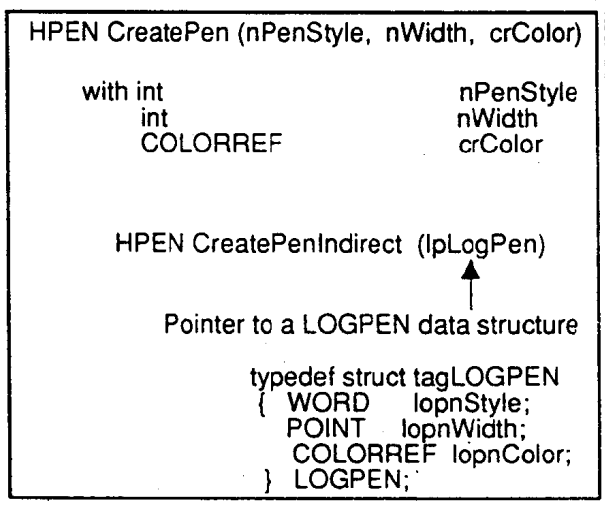


图 3.12 CreatePen 和 CreatePenIndirect 函数

线宽用逻辑单位指定。在 WM_TEXT 映射模式下，这意味着一个像素代表一个逻辑单位。在 LONGPEN 结构中，lopnWidth 的宽度为 POINT 类型。然而，不使用 Y 坐标。

下图列出了不同风格。

Different pen styles

Style	Value	Appearance
PS_SOLID	0	
PS_DASH	1	
PS_DOT	2	
PS_DASHDOT	3	
PS_DASHDOTDOT	4	
PS_NULL	5	
PS_INSIDEFRAME	6	

图 3.13 PEN 的七个不同风格

若设置 PS_INSIDEFRAME 风格并且宽度大于 1，那么对所有图形函数(除多边形和直

线以外),线都画在框内。

用这两个函数和小于或等于 1 的笔宽,使 PS_INSIDEFRAME 风格与 PS_SOLID 等同。能用风格 0,5 或 6 组成较宽的线。

颜色是 COLORREF 数据类型,即长整型。COLORREF 变量能以三种变体来命名颜色。颜色可表示为 RGB 值、一个逻辑调色板入口索引值或调色板相关 RGB 值。

最后两种选择将在各单独的章节中详细讨论。RGB 值可以是十六进制代码或 RGB 宏。

十六进制代码如下:0×00bbggrr,"b"表示蓝,"g"表示绿,"r"代表红。每个初始颜色域定义了颜色的亮度。

值 255(十进制)或 FF(十六进制)表示高亮度,最高字节必须总为 00。

十六进制值	颜色
0×000000FF	纯红
0×0000FF00	纯红
0×00FF0000	纯蓝
0×00000000	黑
0×00FFFFFF	白

RGB 宏提供另一种颜色选择格式。三种初始颜色由能接受值 0 到 255 的字节表示。所表示的颜色序列是红、绿和蓝。设置所有值为 0 时将选择黑色。

如果 GDI 接受一 RGB 值作为函数参数,那么直接传递 RGB 颜色值给输出设备驱动程序。然后输出设备驱动程序选择与理想颜色最接近的颜色。

因为许多颜色输出设备只能显示某几种颜色,当前颜色是由混合像素和能以不同方式出现的显示颜色模拟的。

如果是单色显示器,驱动程序就选择白、黑或各种灰度阴影。在这种情况下,RGB 值的总数大于 765 的颜色为白色。

3.4.3 刷子

刷子是以 8×8 像素表示的位图。此位图在水平和垂直方向重复直到封闭区域内部填满为止。这一区域是由基本的椭圆和多边形函数产生的。

Windows 包含 7 种缺省刷子。用 GetStockObject 获得刷子句柄。

- BLACK_BRUSH
- DKGRAY_BRUSH
- GRAY_BRUSH
- HOLLOW_BRUSH
- LTGRAY_BRUSH
- NULL_BRUSH
- WHITE_BRUSH

用户也可以创建自己的刷子。像画笔一样,函数 CreateBrushIndirect 被传递一个初始结构(LOGBRUSH)。根据刷子的种类,可用四个函数之一指明参数。

有实心刷子、阴影刷子以及包含一个存储于位图中的图案的刷子。位图也可以是一个DIB(Device Independent Bitmap)。

图 3.14 显示出不同种类的阴影刷子。

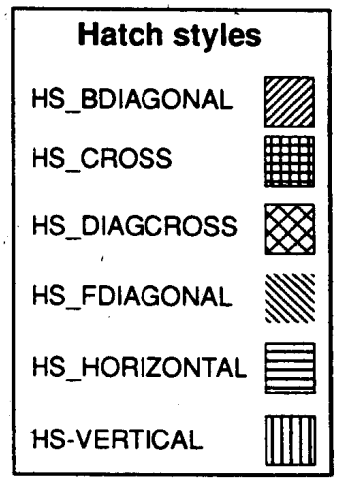


图 3.14 不同种类的阴影刷子

```
HBRUSH CreateSolidBrush (crColor)
with COLORREF crColor: color

HBRUSH CreateHatchBrush (nIndex,crColor)
with int nIndex: Hatch style
     COLORREF crColor: color

HBRUSH CreatePatternBrush (hBitmap)
with HBITMAP hBitmap: Handle to a bitmap

HBRUSH CreateDIBPatternBrush (hPackedDIB,wUsage)
with GLOBALHANDLE hPackedDIB: Handle to global memory object with packed DIB
     WORD wUsage: how the color table should be interpreted
```

与 LOGBRUSH 结构不同,这些函数从不需要所有三个参数,而是只需一个或两个值。所以容易提供这些值,特别是因为 LOGBRUSH 结构的第二个域和第三个域依赖于第一个域的内容。

```
typedef struct tagLOGBRUSH
{
     WORD lbStyle;
     COLORREF LBColor;
     short int lbHatch;
} LOGBRUSH;
```

为了用画笔和刷子作一尝试,下面的应用程序是先前的 GRAPHIC 应用程序的扩展。TOOLS. C 资源代码也包括 ALTERNATE 多边形填充模式。

新函数	说明
CreateHatchBrush	用阴影图案产生一逻辑刷子

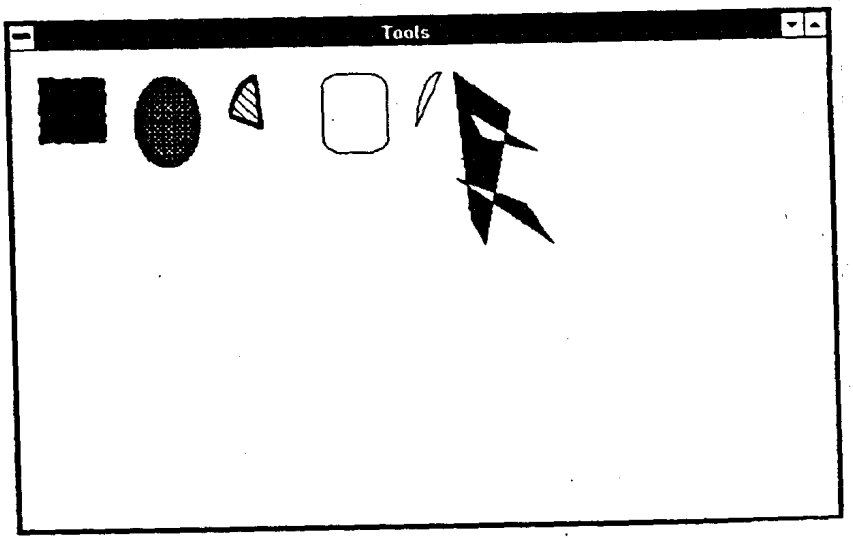


图 3.15　画笔和刷子工具举例

CreatePen	产生逻辑画笔
CreateSolidBrush	产生一实心逻辑刷子
DeleteObject	删除创建的工具
GetStockbject	获得一缺省工具句柄
SelectObject	把设备环境中的工具作为当前工具

源代码:TOOLS. C

```
/* * TOOLS. C * * *
```

```
    HWND hWnd;                                          // Window handle

    if (! hPrevInstance)                                // Initialize first instance
    {
        if (! ToolsInit( hInstance ))                   // If initialization fails
            return FALSE;                               // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Tools",                        // Window class name
        "Tools",                                        // Window caption
        WS_OVERLAPPEDWINDOW,                            // Overlapped window style
        CW_USEDEFAULT,                                  // Default upper-left x pos.
        0,                                              // Upper-left y pos.
        CW_USEDEFAULT,                                  // Default initial x size
        0,                                              // y size
        NULL,                                           // No parent window
        NULL,                                           // Window menu used
        hInstance,                                      // Application instance
        NULL);                                          // No creation parameters

    ShowWindow( hWnd,cmdShow );                         // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while (GetMessage(&msg,NULL,0,0))                   // Message reading
    {
        TranslateMessage(&msg);                         // Message translation
        DispatchMessage(&msg);                          // Send message to Windows
    }

    return msg. wParam;                                 // Return wParam of last message
}

BOOL ToolsInit( HANDLE hInstance )                      // Tools initialization,
                                                        // instance handle

/* * Specify window class * * * * * * * * * * * * * * * * * * * */
{
    WNDCLASS wcToolsClass;                              // Main window class

    wcToolsClass. style = CS_VREDRAW | CS_HREDRAW;// Horizontal and
                                                        // vertical redraw of client area
    wcToolsClass. hCursor = LoadCursor( NULL,IDC_ARROW );
                                                        // Mouse cursor
    wcToolsClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                        // Default icon
    wcToolsClass. lpszMenuName = NULL;                  // No menu
    wcToolsClass. lpszClassName = "Tools";              // Window class
    wcToolsClass. hbrBackground = GetStockObject( WHITE_BRUSH );
                                                        // White background
```

```
    wcToolsClass.hInstance = hInstance;                          // Instance
    wcToolsClass.lpfnWndProc = ToolsWndProc;                     // Window function
    wcToolsClass.cbClsExtra = 0;                                 // No extra bytes
    wcToolsClass.cbWndExtra = 0;                                 // No extra bytes

    if (! RegisterClass( (LPWNDCLASS) &wcToolsClass ) )          // Register
                                                                 // window class
        return FALSE;                                            // Return FALSE if registration fails
                                                                 // If registration is successful,
    return TRUE;                                                 // Return TRUE
}

/* * ToolsWndProc * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL ToolsWndProc( hWnd,message,wParam,lParam )
HWND hWnd;                                                       // Window handle
unsigned message;                                                // Message type
WORD wParam;                                                     // Message-dependent 16 bit value
LONG lParam;                                                     // Message-dependent 32 bit value
{

    PAINTSTRUCT ps;                                              // Paint structure declaration
    HDC hDC;                                                     // Device context handle
    HPEN hPenNew,hPenOld;                                        // Pens
    HBRUSH hBrushNew,hBrushOld;                                  // Brushes

    static POINT ptarray[] = {330,20,340,130,350,150,370,50,330,20,340,50,350,70,390,80,
                              340,50,330,100,400,150,380,120,330,100 };
    static int iarray[] = {5,4,4 };

    switch (message)                                             // Process messages
    {
        case WM_PAINT:                                           // Client area redraw
            hDC = BeginPaint( hWnd,&ps );                        // Begin WM_PAINT

/* * Graphic commands * * * * * * * * * * * * * * * * * * * * */

/* * * * * * * * Rectangle * * * * * * * * * * * * */
            hPenNew = CreatePen( PS_DASHDOT,1,RGB(0,0,255));
            hPenOld = SelectObject( hDC,hPenNew);

            hBrushNew = GetStockObject(GRAY_BRUSH);
            hBrushOld = SelectObject( hDC,hBrushNew);

            Rectangle (hDC,20,20,70,70);

            SelectObject (hDC,hPenOld);
            DeleteObject (hPenNew);
```

```
            SelectObject (hDC,hBrushOld);

/* * * * * * * * * Ellipse * * * * * * * * * * * * * /
            hPenNew = GetStockObject(NULL_PEN);
            hPenOld = SelectObject (hDC,hPenNew);

            hBrushNew = CreateSolidBrush (RGB(255,0,0));
            hBrushOld = SelectObject( hDC,hBrushNew);

            Ellipse (hDC,90,20,140,90);

            SelectObject (hDC,hPenOld);

            DeleteObject( SelectObject (hDC,hBrushOld));

            RoundRect (hDC,230,20,280,80,30,20);

/* * * * * * * * * * Pie * * * * * * * * * * * * /
            hPenNew = CreatePen (PS_SOLID,3,0x00000000);
            hPenOld = SelectObject (hDC,hPenNew);

            hBrushNew = CreateHatchBrush (HS_FDIAGONAL,0x000000FF);
            hBrushOld = SelectObject( hDC,hBrushNew);

            Pie (hDC,160,20,210,100,180,20,160,50);

            DeleteObject( SelectObject (hDC,hPenOld));
            DeleteObject( SelectObject (hDC,hBrushOld));

            Chord (hDC,300,20,340,120,320,10,300,60);

/* * * * * * * * Polygon fill mode * * * * * * * * /
            hBrushNew = GetStockObject (DKGRAY_BRUSH);
            hBrushOld = SelectObject (hDC,hBrushNew);

            PolyPolygon(hDC,(LPPOINT)&ptarray,(LPINT)iarray,3);

            SelectObject (hDC,hBrushOld);

            EndPaint( hWnd,&ps );                                    // End WM_PAINT
            break;                                                 // End of this message process
        case WM_DESTROY;                                             // Destroy window
            PostQuitMessage(0);
            break;                                                 // End of this message process

        default;                                                     // Send other messages to
                                                                   // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                                 // End of this message process
    }
return(0L);
```

```
}
```

模块定义文件:TOOLS. DEF

```
NAME Tools

DESCRIPTION 'Example of drawing tools'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS ToolsWndProc @1
```

MAKE 文件:TOOLS. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: TOOLS. EXE

TOOLS. OBJ: TOOLS. C

TOOLS. EXE: TOOLS. OBJ TOOLS. DEF
    link /NOD/A:16 TOOLS,TOOLS,nul,libw slibcew,TOOLS. def
    rc TOOLS. exe
```

运行 NMAKE 产生 TOOLS 应用程序:

```
nmake tools. mak
```

TOOLS. EXE 如何工作

这一例子展示使用工具的过程,首先获得一句柄以便能选择新工具。定义一画笔并将先前定义的 GRAY_BRUSH 用于长方形。

画一长方形之后,旧工具必须被重新选择,这可使用 SelectObject 函数来完成。然后把所创建的工具用 DeleteObject 删除。其他工具用于绘制椭圆和饼状图。

要演示总用 PolyPolygon 函数显示的 ALTERNATE 多边形填充模式,可用刷子而不是 WHITE_BRUSH。这一点是必要的,因为整个客户区的背景也是白色。

绘图模式

用这个模式指明使用哪种逻辑操作，以对画笔或刷子的像素与目标屏幕像素进行操作。这一逻辑操作也称为 Raster Operation(ROP，光栅操作)。

因为在这种情况下，有两个输入参数(文本色和背景色)，这一模式也叫 Binary Raster Operation 且用 ROP2 Code 指定。在设备环境中，R2_COPYPEN 预定义为绘图模式，这意味着不管背景颜色如何，画笔颜色总是可以接受的。

通过名字可以选择 16 种不同的组合。这些组合可用于提供了四种选择的单色系统中：

- 白色背景、白色画笔
- 黑色背景、白色画笔
- 白色背景、黑色画笔
- 黑色背景、黑色画笔

能用其他十六种不同方式进行组合。

函数 SetROP2 可修改 ROP2 代码。例如，R2_BLACK ROP2 代码只画黑线。

3.4.4 字模

与当前画笔及刷子一样，当前字模在设备环境中。在使用 TextOut 函数用几行文本填充客户区之前，必须确定当前客户区的尺寸和字模大小。

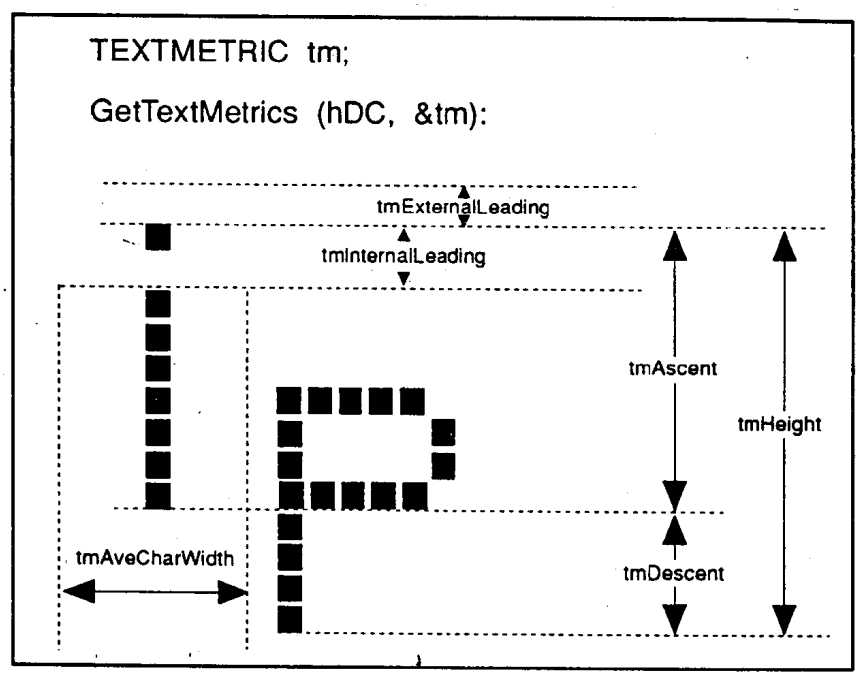


图 3.16 字符结构

GetTextMetrics 函数决定来自设备环境的字符维数并把它们放在 TEXTMETERIC 结构变量中(参看上面的说明)。这一结构包含 20 个域。

所有尺寸都由逻辑单位给出，因此它们依赖于当前图形模式。除了关于字模的信息之外，字符的大小也要指定。

参数 tmExternalLeading 由字模设计者定义，它指定单一文本行间的空隙。

因为许多程序使用系统字模，字符的高度和宽度可由消息 WM_CREATE 决定。

```
static short xChar,yChar;
WM_CREATE:
    hDC = GetDC (hWnd);
    GetTextMetrics (hDC,&tm);
    xChar = tm.tmAveCharWidth;
    yChar = tm.tmHeight + tm.tmExternalLeading;
    ReleaseDC (hWnd,hDC);
    break;
```

客户区的尺寸也是重要的。这一尺寸或由 GetClientRect 函数决定，或由 WM_SIZE 消息决定。WM_SIZE 消息在窗口扩大或缩小时产生。lParam 参数指明新的客户区高度和宽度。

```
Static Short xClient,yClient;
WM_SIZE:
    xClient = LOWORD (lParam);
    xClient = HIWORD (lParam);
break;
```

用这些值可以确定可能的行数(yClient/yClient)以及行中可能的字符数(xClient/xClient)。

缺省字模	说明
SYSTEM_FONT	系统字模，即基于 ANSI 字符集的固定间距字模。这一字模总是可以得到的。
OEM_FIXED_FONT	基于 OEM 字符集的固定间距字模，它随系统而改变。
ANSI_FIXED_FONT	基于 ANSI 字符集的固定间距字模。可使用 Courier。其字符比系统字模小。
ANSI_VAR_FONT	基于 ANSI 字符集的可变宽度字模，如果可能的话，Helvetica 或 TimesRoman 字模可被使用。
DEVICE_DEFAULT_FONT	由设备选择的字模。若无字模可选择的话，就用 Courier 字模或系统字模。

在 GDI 中，每个字符都由被称为字符单元的矩形环绕。矩形由固定的行和列组成。

固定间距指字符单元的尺寸相同。这些单元的大小是按比例变化的。

若缺省字模不够用，则可用 CreateFont 或 CreateFontIndirect 创建自己的逻辑字模。

LOGFONT 结构比 LOGPEN 和 LOGBRUSH 结构稍大一些。

LOGFONG 结构

LOGFONT 结构与 TEXTMETRIC 结构很相似，它负责当前物理字模。

当在设备环境中选择逻辑字模时，GDI 选择一物理的、与现有逻辑字模要求匹配最好的字模，这一字模能刚好匹配逻辑字模或完全不同。

GDI 包含一个字模表，此表是在 Windows 被安装并且能用控制面板扩充时建立的。这一

表中的每个入口都描述一个物理字模和它的属性。

每一入口也有一个指针指向相应的字模资源。字模资源是.FON 文件,这一文件包含一或多种尺寸的指定字模。

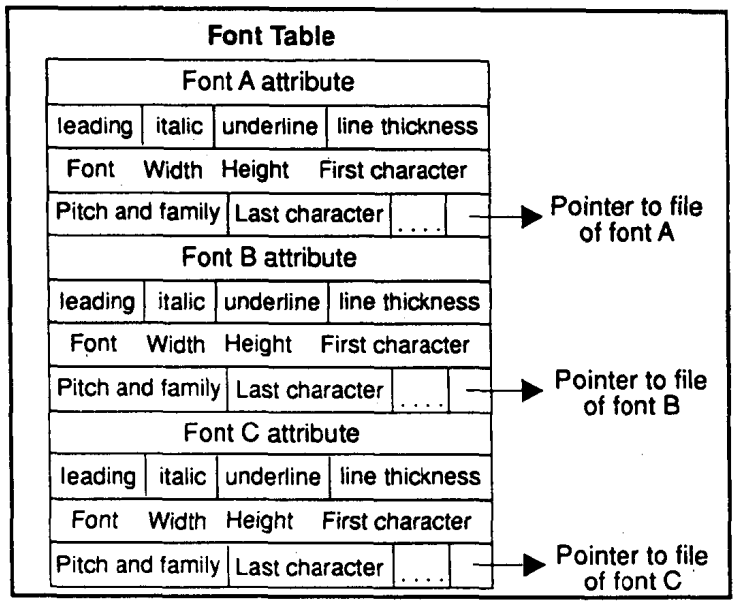


图 3.17 字模表中的主要字模图

当同一物理字模得不到时,偶尔也可使用期望的字模。例如,通常 GDI 能调整字符大小。假定需要一种有十个逻辑单位字符高度的物理字模,而只有五个逻辑单位字符高度的物理字模可得到。

SelectObject 函数能通过使高度加倍来产生期望的字模。Windows 以斜体、黑体、下划线等属性反复使用这个函数,否则,每个属性都需要单独的字模。

当把逻辑字模与字符表中的字模相比较时,GDI 提供没有期望特征的物理字模,这些特征按重要性排列。

如果字符必须加下划线,但字模没有这一属性,就给出一个"处罚点";GDI 选择有最少"处罚点"的物理字模。欲改变属性的重要性,可使用 SetFontMapperWeight 函数。

3.4.5 字模工具举例

下列应用程序产生一个新字模,并且输出有关当前物理字模的信息。

新消息	说明
WM_CREATE	创建窗口时发送

新函数	说明
CreateFont	产生一逻辑字模
GetDC	获得设备环境句柄

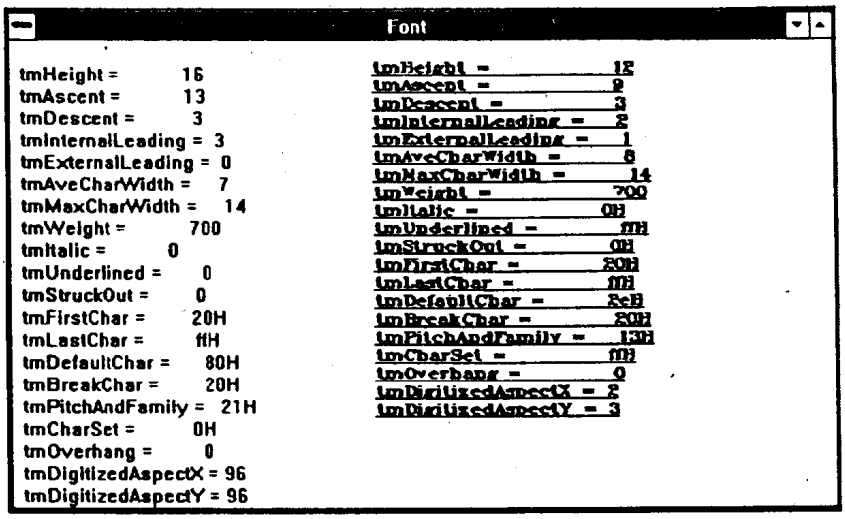


图 3.18

GetTextMetrics	用当前字模值填充缓冲区
ReleaseDC	释放设备环境句柄
新结构	简要说明
TEXTMETRIC	物理字模结构

源代码:FONT.C

```
/* * FONT.C * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays font specifics in default and Times Roman fonts * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                                    // Include windows.h header file
#include "string.h"                                     // Include string.h header file

#if _MSC_VER >=700
#pragma warning (disable:4028)
#endif

long FAR PASCAL FontWndProc ( HWND,unsigned,WORD,LONG);
BOOL FontInit ( HANDLE );

int PASCAL WinMain ( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                         // Current & previous instance
LPSTR lpszCmdLine;                                      // Long ptr to string after
                                                        // program name during execution
int cmdShow;                                            // Specifies the application
                                                        // window's appearance
{
    MSG msg;                                            // Message variable
    HWND hWnd;                                          // Window handle
```

```
    if (! hPrevInstance)                                    // Initialize first instance
    {
        if (! FontInit( hInstance ))                        // If initialization fails
            return FALSE;                                   // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Font",                             // Window class name
        "Font",                                             // Window caption
        WS_OVERLAPPEDWINDOW,                                // Overlapped window style
        CW_USEDEFAULT,                                      // Default upper-left x pos.
        0,                                                  // Upper-left y pos.
        CW_USEDEFAULT,                                      // Default initial x size
        0,                                                  // y size
        NULL,                                               // No parent window
        NULL,                                               // Window menu used
        hInstance,                                          // Application instance
        NULL);                                              // No creation parameters

    ShowWindow( hWnd,cmdShow );                             // Make window visible
    UpdateWindow( hWnd );                                   // Update window

    while (GetMessage(&msg,NULL,0,0))                       // Message reading
    {
        TranslateMessage(&msg);                             // Message translation
        DispatchMessage(&msg);                              // Send message to Windows
    }

    return msg. wParam;                                     // Return wParam of last message
}

BOOL FontInit( HANDLE hInstance )                           // Initialize instance handle
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcFontClass;                                   // Main window class

    wcFontClass. style = CS_VREDRAW | CS_HREDRAW;           // Horizontal and
                                                            // vertical redraw of client area
    wcFontClass. hCursor = LoadCursor( NULL,IDC_ARROW );
                                                            // Mouse cursor
    wcFontClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                            // Default icon
    wcFontClass. lpszMenuName = NULL;                       // No menu
    wcFontClass. lpszClassName = "Font";                    // Window class
    wcFontClass. hbrBackground = GetStockObject( WHITE_BRUSH );
                                                            // White background
    wcFontClass. hInstance = hInstance;                     // Instance
    wcFontClass. lpfnWndProc = FontWndProc;                 // Window function
    wcFontClass. cbClsExtra = 0;                            // No extra bytes
```

```
    wcFontClass.cbWndExtra = 0;                                    // No extra bytes

    if (! RegisterClass( (LPWNDCLASS) &wcFontClass ) )
                                                                   // Register window class
        return FALSE;                                  // Return FALSE if registration fails
    return TRUE;                                  // If registration is successful,return TRUE
}

/* * FontWndProc * * *
```

```
            wsprintf( chtmName[11],"tmFirstChar = %xH",tm.tmFirstChar);
            wsprintf( chtmName[12],"tmLastChar = %xH",tm.tmLastChar);
            wsprintf( chtmName[13],"tmDefaultChar = %xH",tm.tmDefaultChar);
            wsprintf( chtmName[14],"tmBreakChar = %xH",tm.tmBreakChar);
            wsprintf( chtmName[15],"tmPitchAndFamily = %xH",tm.tmPitchAndFamily);
            wsprintf( chtmName[16],"tmCharSet = %xH",tm.tmCharSet);
            wsprintf( chtmName[17],"tmOverhang = %d",tm.tmOverhang);
            wsprintf( chtmName[18],"tmDigitizedAspectX = %d",tm.tmDigitizedAspectX);
            wsprintf( chtmName[19],"tmDigitizedAspectY = %d",tm.tmDigitizedAspectY);

            for (i = 0; i < 20; i++)
                TextOut( hDC,xChar,yChar * (i+1),chtmName[i],strlen(chtmName[i]));

/** Generate new font **********************/

            hFontNew = CreateFont( 12,                                  // lfHeight
                8,                                                      // lfWidth
                0,                                                      // lfEscapement
                0,                                                      // lfOrientation
                700,                                                    // lfWeight
                FALSE,                                                  // lfItalic
                TRUE,                                                   // lfUnderline
                FALSE,                                                  // lfStrikeOut
                OEM_CHARSET,                                            // lfCharset
                OUT_DEFAULT_PRECIS,                                     // lfOutPrecision
                CLIP_DEFAULT_PRECIS,// lfClipPrecision
                DEFAULT_QUALITY,                                        // lfQuality
                VARIABLE_PITCH | FF_ROMAN,// lfPitchAndFamily
                "TmsRmn");                                              // lfFaceName
            hFontOld = SelectObject( hDC,hFontNew);

            GetTextMetrics (hDC,&tmNew);
            xCharNew = tmNew.tmAveCharWidth;
            yCharNew = tmNew.tmHeight + tmNew.tmExternalLeading;

/** Print font parameters in new font **********/
            wsprintf( chtmName[0],"tmHeight = %d",tmNew.tmHeight);
            wsprintf( chtmName[1],"tmAscent = %d",tmNew.tmAscent);
            wsprintf( chtmName[2],"tmDescent = %d",tmNew.tmDescent);
            wsprintf( chtmName[3],"tmInternalLeading = %d",tmNew.tmInternalLeading);
            wsprintf( chtmName[4],"tmExternalLeading = %d",tmNew.tmExternalLeading);
            wsprintf( chtmName[5],"tmAveCharWidth = %d",tmNew.tmAveCharWidth);
            wsprintf( chtmName[6],"tmMaxCharWidth = %d",tmNew.tmMaxCharWidth);
            wsprintf( chtmName[7],"tmWeight = %d",tmNew.tmWeight);
            wsprintf( chtmName[8],"tmItalic = %xH",tmNew.tmItalic);
            wsprintf( chtmName[9],"tmUnderlined = %xH",tmNew.tmUnderlined);
            wsprintf( chtmName[10],"tmStruckOut = %xH",tmNew.tmStruckOut);
            wsprintf( chtmName[11],"tmFirstChar = %xH",tmNew.tmFirstChar);
            wsprintf( chtmName[12],"tmLastChar = %xH",tmNew.tmLastChar);
            wsprintf( chtmName[13],"tmDefaultChar = %xH",tmNew.tmDefaultChar);
            wsprintf( chtmName[14],"tmBreakChar = %xH",tmNew.tmBreakChar);
```

```
            wsprintf( chtmName[15],"tmPitchAndFamily = %xH",tmNew. tmPitchAndFamily);
            wsprintf( chtmName[16],"tmCharSet = %xH",tmNew. tmCharSet);
            wsprintf( chtmName[17],"tmOverhang = %d",tmNew. tmOverhang);
            wsprintf( chtmName[18],"tmDigitizedAspectX = %d",tmNew. tmDigitizedAspectX);
            wsprintf( chtmName[19],"tmDigitizedAspectY = %d",tmNew. tmDigitizedAspectY);

            for (i = 0; i < 20; i++)
                TextOut( hDC,xCharNew + 250,yCharNew * (i+1),
                    chtmName[i],strlen(chtmName[i]));

            DeleteObject( SelectObject( hDC,hFontOld));
            EndPaint( hWnd,&ps );
            break;

        case WM_DESTROY:                                    // Destroy window
            PostQuitMessage(0);
            break;

        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
return(0L);
}
```

模块定义文件:FONT. DEF

```
NAME Font

DESCRIPTION 'Font example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS FontWndProc @1
```

MAKE 文件:FONT. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
```

```
all: FONT.EXE

FONT.OBJ: FONT.C

FONT.EXE: FONT.OBJ FONT.DEF
    link /NOD/A:16 FONT,FONT,nul,libw slibcew,FONT.def
    rc FONT.exe
```

运行 NMAKE 如下产生 FONT 应用程序：

```
nmake font.mak
```

FONT.EXE 如何工作

当创建窗口时，产生 WM_CREATE 消息。这是到达窗口过程的第一条消息。设备环境及包含系统字模作为缺省值的设备环境是可以得到的。GetTextMetrics 函数把有关这个字模的信息放在 TEXTMETRIC 结构类型的 tm 变量中。

为了在客户区的左上角产生多于一行的输出，可把字母的高度和宽度存于 xChar 和 yChar 中，然后释放设备环境。

在消息 WM_PAINT 期间处理文本准备和输出工作。

数组 chtmName 包含 20 个字符串，由 TEXTMETRIC 结构填充，并且能用 FOR 循环和 TextOut 函数一行接一行地输出。

欲比较系统字模，需要一新字模。首先函数 CreateFont 用于产生逻辑字模。参数可自由选择。函数 SelectObject 从字模表中取出与期望特征最匹配的物理字模。

函数 GetTextMetric 确定字符的外观。这些字符在 chtmName 字符串数组中准备好，并且以新字模输出。

在释放显示环境句柄前，旧字模必须放回设备环境，且逻辑字模必须被删除。

第四章　打　印

几乎所有 Windows 应用程序都向用户提供了数据打印、屏幕输出等功能。Windows 的图形设备接口(GDI)负责处理应用程序和打印机之间的通信。因此应用程序不需要特别的打印机配置文件。

面向屏幕和面向打印机的数据输出机制非常相似。两者的设备环境句柄请求过程都是在输出操作发生之前进行的。而且,用于文本和图形输出的是同一输出函数。

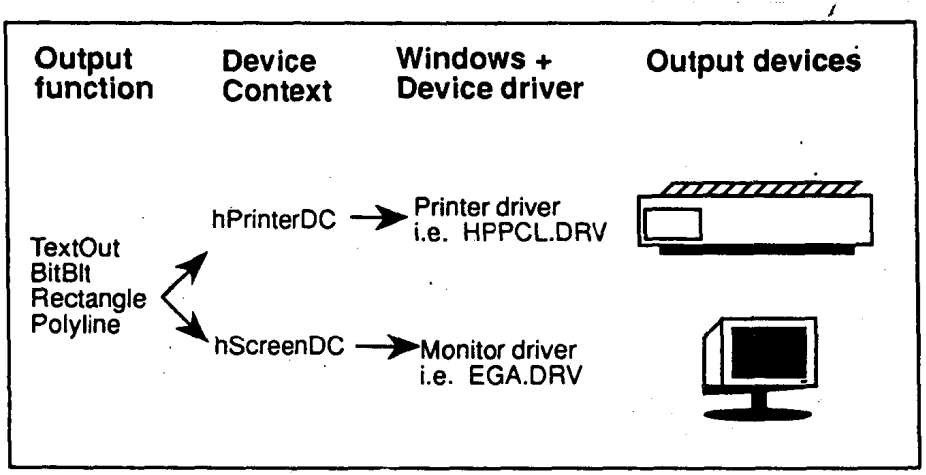


图 4.1　输出设备间的通信

两者之间的差别主要在于发生的错误的类型。例如,就打印机输出而言,可能没有联接打印设备、打印机缺纸或者安装的打印纸尺寸不对。

各功能构件必须相互作用来执行打印进程。除 GDI 和相应的打印机驱动程序外,Windows 的假脱机系统或打印管理器都能正常地激活该进程。在初始化文件 WIN. INI 中有它们的入口设置,如下所示:

```
[windows]
spooler = yes
```

若将假脱机系统选项改为"No",就可终止打印管理器的动作。用户可通过两种方式改变该选项,即直接修改 WIN. INI 文件和取消打印对话框的"使用打印管理器"复选框。

然而,对于用户来说,常使打印管理器处于工作状态是一个良好的习惯。打印管理器确保两道作业的输出彼此独立,并且保证第一道作业在第二道打印作业开始之前完成。打印管理器还支持 Windows 的占先多任务管理,当它在后台运行时,允许其他的 Windows 应用程序继续运行。

打印的基本步骤如下所示(这里仅列出所需的函数,在后面将作进一步的讨论):

(1)创建打印机设备环境(CreateDC)

(2)处理打印请求(StartDoc 和 StartPage)

(3)打印输出(例如 TextOut 或 Ellipse)。

(4)产生换页(EndPage 和 StartPage)。

(5)结束打印请求(EndDoc)

步骤 1 返回打印机设备环境的句柄,调入所需的打印机驱动程序,除非它已调入内存。打印机驱动程序实际上是动态链接库,虽然它们具有 DVR 扩展名。

一旦调用了 StartDoc 函数,GDI 模块就会唤醒打印机驱动程序,该程序执行初始化操作,并向 GDI 返回认可信息,然后 GDI 调入打印管理器。

接着应用程序调用输出函数打印第一页文档。GDI 将调用结果暂时存在 TEMP 目录下。

第一页打印完后,可以调用 Escape 函数的 NEWFRAME 序列或调用 EndPage 函数,打印机驱动程序访问此文件,同时将调用序列翻译为打印机可识别的命令。如果使用的是激光打印机,则翻译为 PostScript 命令。

大多数驱动程序在翻译过程中使用分段技术,通过这种技术将一页分成许多小的区域,或者片段,然后按顺序处理这些片段。一旦元文件全部处理完,GDI 就马上删除该文件。

打印机命令语言的翻译结果顺序存储在 TEMP 目录中。GDI 向同打印管理器发送消息,通知它有一个新的打印作业正处于就绪状态。

一旦应用程序调用 EndDoc 函数结束打印,Windows 就通知其假脱机系统开始工作。

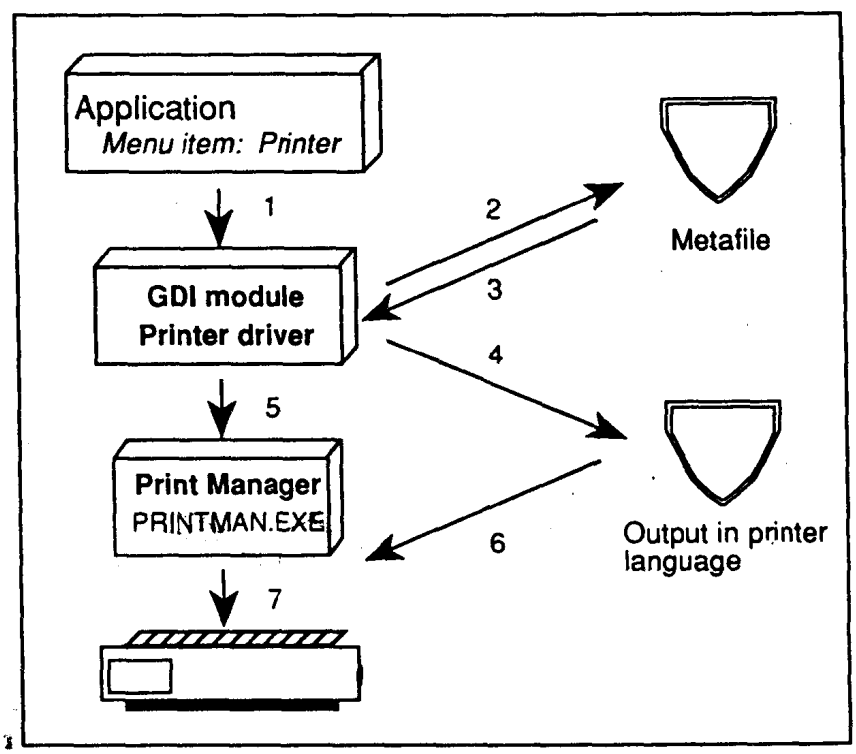


图 4.2 打印进程的内部通信

只有所有打印信息借助 GDI 写到临时文件中以后，应用程序才能处理其他任务。在所有文件打印完毕之前，应用程序都无需等待。

打印管理器必须对文档的打印工作负责。如果出现问题，它可以立即终止打印进程，并删除文档。

4.1 打印机设置

WIN. INI 文件确认了当前所选用的打印机类型。本节将讨论打印机选项以及其他与打印机有关的信息，这些内容都在 WIN. INI 文件中说明。

4.1.1 创建打印机设备环境

当一个应用程序将输出信息送到窗口时，必须有一个函数如 BeginPaint 等向它提供所需的显示环境句柄。如果该句柄是面向全屏幕操作的，就有必要调用 CreateDC 函数，并且字符串“DISPLAY”作为其第一个参数。用户通常使用同样的函数为打印机创建一个设备环境句柄。

CreateDC 函数

```
HDC CreateDC(lpszDriver,lpszDevice,lpszOutput,lpInitData)

LPSTR lpszDriver;                          /* Pointer to driver name */
LPSTR lpszDevice;                          /* Pointer to device name */
LPSTR lpszOutput;                          /* Pointer to port name */
LPVOID llpInitData;          /* Pointer to a DEVMODE structure (optional) */
```

在函数调用前必须设置前三个指针，第四个指针通常设置为 NULL，如果要使用它，就必须先初始化具有 DEVMODE 结构的变量。为访问该结构，应用程序应包含 DRIVINIT. H 文件。打印机名、设备名和端口名可以通过 GetProfileString 函数从初始化文件 WIN. INI 中读取，其[windows]部分包括假脱机系统选项和最新打印机选项，这些选项由用户通过控制面板设置。

用户通过鼠标器按下 Add 按钮后，对话框在其下半部分作进一步显示。在下面的例子中，装入了三种打印机驱动程序，其中 Apple 激光打印机是缺省设置。基于这些设置，WIN. INI 文件存储了下面这些信息：

```
[windows]
device = Apple LaserWrite II NTX,pscript,LPT2:
```

当前打印机的有效设备名是 Apple LaserWrite II NTX，这种 PostScript 打印机连接到 LPT2 并行端口上，这些信息通过调用 GetProfileString 函数从 WIN. INI 文件中读取，该函数根据起始标记“Windows”和关键字“device”识别信息(注意，这些参数都不区分大小写)。

GetProfile 函数

```
char szPrintInfo(80)
GetProfileString("Windows","Device",(LPSTR)"",szPrintInfo,80)
```

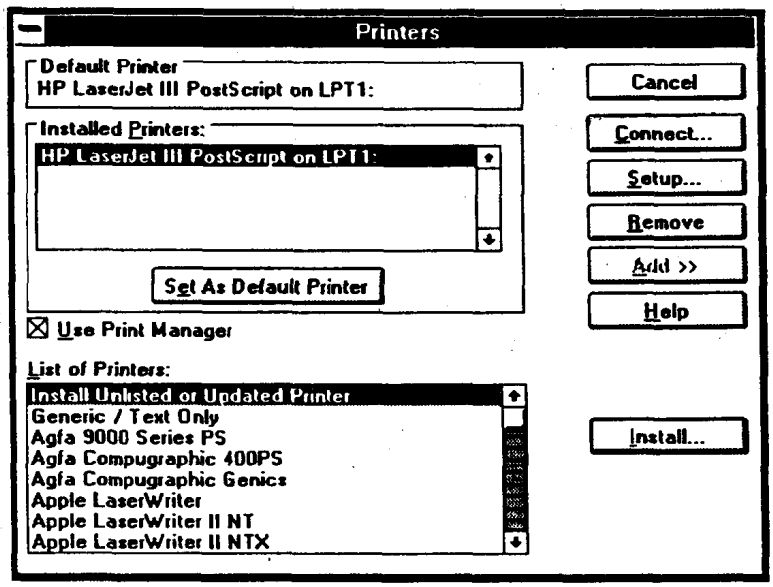


图 4.3　通过控制板设置打印机

该函数将整个字符串都存入 szPrintInfo 变量中，包括子串之间的逗号。程序员可以写一个简短的子程序，通过查找逗号将 szPrintInfo 中的子串分开，然后将它们分别存到不同的新变量中去。

```
lpTemp = lpDevicename = pPrintInfo;
lpDrivername = lpOutputport = 0;
while ( * lpTemp)
{
    if ( * lpTemp ==',')
    {
        * lpTemp++ = 0;
        while ( * lpTemp ==' ')
            lpTemp++;
        if (! lpDrivername)
            lpDrivername = lpTemp;
        else
        {
            lpOutputport = lpTemp;
            break;
        }
    }
    else
        lpTemp++;
}
```

最后，用户可以调用 CreatDC 函数为打印机创建一个设备环境句柄。

```
hPrintDC = CreateDC(lpDrivename,lpDevicename,lpOutputport,(LPSTR) NULL);
```

获取标准打印机的设备环境是程序员经常要面临的任务。Window 3.1 通过使用对话框简化了这个比较复杂的过程。

对话框允许用户进行打印设置。例如文档的哪一部分需要打印等，而且还可以激活另外

一个对话框来改变打印机设置的选择。

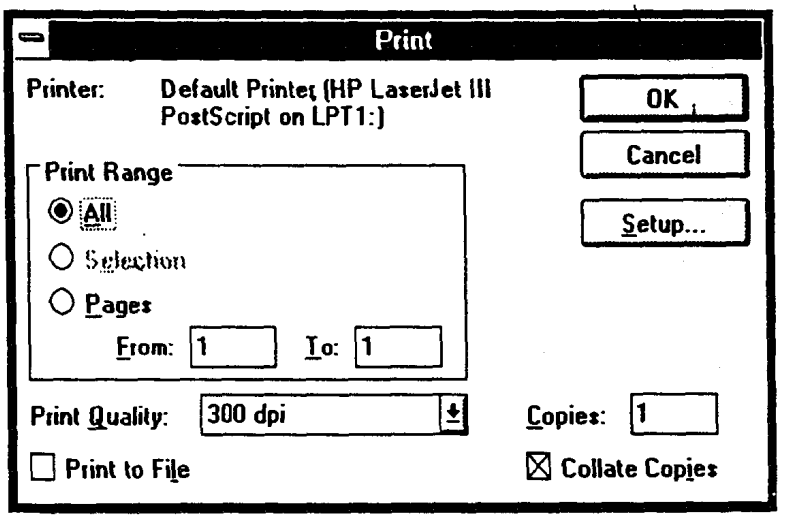


图 4.4 公用对话框

显示对话框必须遵循一定的步骤。首先,用户必须初始化一个 PRINTDLG 结构类型的变量,然后将它传给 PrintDlg 函数,这样才能激活对话框。

这种结构的标志域包括不同的指示符。如果设置了 PD_RETURNDC 指示符,PrintDlg 函数在执行成功后会返回缺省打印机的设备环境句柄,并将之存储在相同结构的 hDC 域中。如果没有提供 PD_RETURNDC 指示符的值,或者是用户需要设备名、驱动程序名和端口名执行另一项任务,那么程序员可以从 PRINTDLG 结构变量中截取返回值。

```
PRINTDLG pd;
HDC hPrintDC;

    pd.Flags = PD_RETURNDC;
    PrintDlg((LPPRINTDLG)&pd);
    hPrintDC = pd.hDC;

LPDEVNAMES lpDevNames;
LPSTR lpszDriverName;
LPSTR lpszDeviceName;
LPSTR lpszPortName;
PRINTDLG pd;
HDC hPrintDC;

PrintDlg((LPPRINTDLG)&pd)

lpDevNames = (LPDEVNAMES)GlpbalLock(pd.hDevNames);
lpszDriverName = (LPSTR)lpDevNames + lpDevNames->wDriverOffset;
lpszDeviceName = (LPSTR)lpDevNames + lpDevNames->wDeviceOffset;
lpszPortName = (LPSTR)lpDevNames + lpDevNames->wOutputOffset;
GlobalUnlock(pd.hDevNames);
GlobalFree(pd.hDevNames);
```

```
hPrintDC = CreateDC(lpszDriverName,lpszDeviceName,lpszPortName,(LPSTR)NULL);
```

在调用 PrintDlg 函数时，变量 pd. DevNames 代表一个全程变量，它包含了 DEVNAMES 结构。

这个结构包括三个特征字符串的偏移地址，程序员可以使用前面所介绍的 GetProfileString

函数从 WIN. INI 文件中读取这三个字符串。

如果 CreateDC 函数未成功地创建设备环境，它会返回一个 NULL 值，这种情况是完全可能发生的。例如，Windows 没有合适的驱动程序时，或 WIN. INI 的[windows]部分将端口设置为“none”时。

4. 1. 2 WIN. INI 文件中与打印机有关的其他信息

CONTROL. INI 文件

当用户在控制面板上按下 Add〉〉按钮想装入新的打印机时，打印机对话框将在其下半部分列出各种可用的打印机名，这些打印机都是从 CONTROL. INI 文件中读取的，该文件在 Windows 的 SYSTEM 子目录下，并且包括多个部分。

[io:device]部分的每一行都包含一种可装入的打印机信息，这种打印机对用户而言是可用的，用户可以任选其中一种。

一旦用户按下 Install…按钮，Windows 就会立刻查找 SYSTEM 子目录下所选打印机的驱动程序，其程序名是[io:device]部分的内容之一。它是在设备名之前声明的。

举一个例子，如果用户预先装入了“Apple LaserWrite”，而且增设了另一 PostScript 打印机，Windows 会立即装入 PSCRIPT. DRV 驱动程序。

大多数驱动程序可为多种打印机服务。如 EPSON9. DRV 文件，它控制所有的 Epson9 针式打印机。

如果 Windows 在 SYSTEM 子目录下没有找到所需要的驱动程序，对话框将提示用户插入一张配置盘。

Windows 从 CONTROL. INF 文件的行首得到磁盘号。

[ports]部分

每个已装入的打印机都与某个端口相连。所有可用端口都在 WIN. INI 文件的[ports]部分声明。

```
[io. device]
6:pscript. DRV,"Apple LaserWriter II NTX","DEVICESPECIFIC"
6:pscript. DRV,"Apple LaserWriter Plus","DEVICESPECIFIC"
7:HPPCL. DRV,7:unidrv. dll,"Apricot Laser","DEVICESPECIFIC"
6:pascript. DRV,"AST TurboLaser/PS","DEVICESPECIFIC"
7:HPPLOT. DRV,"AT&T 435","CONTINUOUSSCALING"
7:CITOH. DRV,7:unidrv. dll,"AT&T 470/475","133,96,72,","75,72,96"
7:oki9ibm. DRV,7:unidrv. dll,"AT&T 473/478","167,120,72","60,72,120"
```

一般情况下，只有在 Windowsc 以 DOS 兼容方式运行于 OS/2 环境下时，才使用上表所列的最后两个端口。在最新的 Windows 版本中，都用“DOS”代替“OS2”(LPT1. DOS、

LPT2. DOS)。

如果打印设备与某个文件相连，用户在执行打印命令时必须提供文件名。这时，输出的内容将写到文件里，而不是打印机上。

Windows 还定义了一个特殊的端口，即空端口。当连接到空端口上时，打印机不产生实际的输出操作。文件的[windows]部分包括此端口名，它在 Windows 装入时就被初始化为“none”。

```
[windows]
NullPort = None
```

[PrintPorts]部分

用户关闭了 Printers 对话框以后，控制面板会把当前打印机的各类信息保存在 WIN. INI 文件的[PrintPorts]部分。

```
[PrinterPorts]
Apple LaserWriter II NTX=pscript,LPT2:,15,90
Apple LaserWriter Plus=pscript,LPT1:,15,90
Epson T-1000=EPSON9,FILE:,15,45
NEC Pinwriter P2200=NEC24pin,LPT1:15,45
```

等号前的字串为打印设备名，后面的是打印机驱动程序名(不包含扩展名)和端口名。

紧接着驱动程序名和端口名的两个数字代表“Device Not Selected”和“Transmission Retry”的超时设置，以秒为计时单位。

所有这些设置都可通过 Connect 对话框进行改动，该对话框紧接在 Printers 对话框后，常用于端口赋值。

[devices]部分

进行程序设计时，通常是从[devices]部分读取所需的设置信息，而不是[PrinterPorts]。这部分的设置信息也是从控制面板上获得的，它包括除起始设置外的所有数据。当然，这些数据与[PrinterPorts]中的完全相同。

```
[devices]
Apple LaserWriter II NTX=pscript,LPT2:
Apple LaserWriter Plus=pscript,LPT1:
Epson T-1000=EPSON9,FILE:
NEC Pinwriter P2200=NEC24pin,LPT1:
```

4.2 打印函数

在 Windows 3.1 之前，必须由 Escape 换码函数处理所有与打印机设备驱动程序的通信任务。

Escape 函数有许多子例程，都被称为换码，因为它们都通过各种不同的函数完成不同的功能，以构成一项打印任务。

但在 3.1 版中，关键的子例程都作为独立函数而存在，这样有利于简化代码。

不过，过去组成 Escape 函数的所有换码序列都与现行版本兼容，如下所示：

新的打印函数	旧的换码序列
AbortDoc	忽略打印请求:ABORTDOC
EndDoc	结束打印请求:ENDDOC
EndPage	结束一页:NEWFRAME
SetAbortProc	设置 Abort 函数:SETABORTDOC
StartDoc	开始处理打印请求:STARTDOC
StartPage	驱动程序准备接收数据:NEWFRAME 和 BANDINFO

4.2.1 起始与结束

Windows 3.1 中,程序员可以调用 StartDoc 函数开始打印文档。这种方式有利于保证整个文档始终保持完整,并且不与其他程序的打印正文相混淆。

```
char szTitle[] = "Example";
DOCINFO di;
HDC hPrintDC;
Error = Escape(hPrintDC,STARTDOC,sizeof(szTitle),szTitle,(LPSTR) NULL)
or
di. cbSize = sizeof(DOCINFO);
di. lpszOutput = szTitle;
di. lpszOutput = NULL;
Error = StartDoc(hPrintDC,&di);
```

DOCINFO 结构是 Start 函数所必需的,它包括下面三个域:

```
typedef struct {
    short obSize;
    LPSTR lpszDocName;
    LPSTR lpszOutput;
}DOCINFO;
```

其中,lpszDocName 域代表的字符串是打印管理器中打印任务的标题,它最多可以有 32 个字符。如果输出内容写到文件中,而不是送到打印机上,则用 lpszDocName 域代表文件名。

一般情况下,应检查函数的返回值,以判断是否要执行打印进程。当函数返回正值时表示 StartDoc 或 Escape 函数执行成功。

EndDoc 函数

Window3.1 中,打印请求的终止是通过调用 Escape 函数或 EndDoc 函数实现的。但是,如果发生了打印错误(例如正在打印文档或即将打印新页),应用程序则不应该调用这些函数。在这种情况下,GDI 模块会预先终止作业。

```
Escape (hPrintDC,ENDDOC,0,NULL,NULL)
EndDoc (hPrintDC);
```

大多数打印请求都要求打印多页文档。程序员有责任保证换页的正常进行,以便文本始终处在栏内。打印纸的尺寸由 GetDeviceCape 函数提供,决定于 HORZRES 和 VERTRES 两个参数。测量值按像素和栅格给出。

```
Pageheight = GetDeviceCaps (hPrintDC,VERTRES);
PageWidth = GetDeviceCaps (hPrintDC,HORZRES);
```

生成换页

打印换页是通过调用 StartPage 和 EndPage 函数实现的，当然也可以采用 NEWFRAME 换码序列调用 Escape 函数达到目的。因此，可设计下面程序生成换页：

```
Error = StartDoc(hPrintDC,&di);
if (Error > 0)
{
    StartPage(hPrintDC);
    Error = EndPage(hPrintDC0;
    if (Error > 0)
        EndDoc(hPrintDC);
}
```

使用 Escape 函数时，程序略有不同：

```
Error = Escape (hPrintDC,STARTDOC,sizeof(sizeof(szTitle),szTitle,NULL);
if ( Error > 0)
{
    Error = Escape (hPrintDC,NEWFRAME,0,NULL,NULL);
    if ( Error > 0)
        Escape (hPrintDC,ENDDOC,0,NULL,NULL);
}
```

错误检查

错误代码是一个整数类型的变量。如果发生了错误，最好通过一个错误处理例程进行检查，并通过一个消息框通知用户，以便用户马上采取相应动作。例如，任务运行时内存不够，用户便可释放内存，并重新处理打印请求。如果是由 Windows 告诉出错信息，那么返回的错误码中 SP_NOTREPORTED 被清除，系统将检查下面的错误位：

错误位	值	意义
SP_ERROR	-1	一般错误
SP_APPABORT	-2	应用程序调用 Abort 函数终止程序
SP_USERABORT	-3	用户通过文件管理器终止程序
SP_OUTOFDISK	-4	磁盘空间不够
SP_OUTOFMEMORY	-5	内存空间不够

Abort 函数

理论上讲，程序员已完全可以设计出一个打印程序了。而实际上，程序员还必须了解许多其他方面的东西以确保程序设计的可靠性。

假设要打印的文档有 20 页，如果仅用上面所描述的函数进行打印，应用程序不得不因长时间的打印操作占用 CPU 时间。在这种情况下，程序往往通过处理单独的消息来完成打印作业。这与用户从菜单中选择“打印”，然后执行打印操作类似，这种方式阻碍了其他应用程序的执行。

而且，这种方式还会引起一个问题，即当打印作业的输出写入到磁盘文件中时，一旦磁盘已满，下一个换页函数（NEWFRAME 列或 EndPage 函数）将返回 SP_OUTOFDISK 错误，于是整个打印操作终止。如果打印管理器有足够时间将当前元文件内容发送到打印机上，然

后删除该文件的话，这种情况一般不会发生。然而，由于用户程序独占 CPU，打印管理器无法提供这种服务来补救可能出错的打印操作。

Abort 函数正是用来解决上述问题的。它必须遵循一定的结构。它通常在程序中被定义为出口函数，其原型声明如下：

```
int FAR PASCAL AbortPro (HDC hPrintDC,INT Code)
```

函数的第一参数是打印机设备环境句柄，它将传给所有打印函数。第二个参数表示一个特定条件。SP_OUTOFDISK 标志就是该参数的一个值，例如，在前面所描述的磁盘空间不足的情况下，此标志将作为参数传递给函数。然而，在一般情况下，不检查该指示符，因为没有它问题也能得到解决。

PeekMessage 函数

有些 Windows 应用程序执行的是很费时的任务，如生成一些素数。写这种程序时，程序员必须按一定方式设计任务的执行，通常是应用程序与其他程序交替运行。它的实现必须借助于 ReekMessage 函数。ReekMessage 和 GetMessage 类似，主要区别将在以后章节中讨论。现在，先看看如何在消息循环内使用 PeekMessage 函数。

```
while(TRUE)
{
    if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))
    {
        if (msg.message == WM_QUIT)
            break;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else
                                        // run time-consuming task
}
```

只要应用程序的消息队列中存有消息，PeekMessage 参数就会从队列中检索出下一个消息，并把它放到 MSG 结构中，以便 DispatchMessage 把它传递给窗口函数，这与 GetMessage 的用法相同。

如果队列为空，应用程序仍将 CPU 的控制权转让给 Windows。当没有其他应用程序占用 CPU 时，它才有可能将 CPU 控制权收回。如果已查明所有其他应用程序的消息队都为空，程序就会重新占用 CPU，因为目前已无其他消息等待处理，PeekMessage 函数的返回值为 FASLE，程序根据第一个 IF 条件语句转移到 ELSE 语句部分。

此时，程序既可以直接处理任务，也可以将消息传给窗口函数，由它来处理。时间的分配亦是如此，既可直接进行，也可通过消息转移给窗口例程执行。

在 GetMessage 循环内，如果从消息队列中检查到 WM_QUIT，应用程序将彻底终止运行，并返回 FALSE 值。因为对于 PeekMessage 函数来说，FALSE 有多种含义，所以在使用该函数时，不必采用另外的方式处理程序的终止请求。也正因为如此，程序中使用 WHILE 循环时，只需在其中加上对 WM_QUIT 消息的特别检查。

Abort 函数从本质上讲不过是一个 PeekMessage 循环而已。没必要重复这一动作。下面的例子描述的就是 WM_QUIT 的处理函数。

```
int FAR PASCAL AbortProc (HDC hPrintDC,int Code)
{
    MSG msg;
    while (PeekMessage(&msg,NULL,0,0,PM_REMOVE))
    {
        DispatchMessage(&msg);
    }
    return(TRUE);
}
```

SetAbortProc 函数

如果想调用 Abort 函数，在启动打印操作前就必须把此函数的有关信息告诉 Windows，因为对于任何一个支持打印操作的函数来说，Windows 都无法保证是否能独立地查找到该函数。要完成这项工作，程序员可以调用 Escpae 函数 SETABORTPROC 换码序列，也可以使用 SetAbortProc 函数。但这两种方式都需要 Abort 函数的瞬时过程地址，调用 MakeProcIntarce 可以达到目的。

```
PARPROC lpfnAbortProc;
lpfnAbortProc =MakeProcInstance(AbortProc,hInst);


                                                        // either
Escape(hPrintDC,SETABORTPROC,NULL,(LPSTR)
    lpfnAbortProc,(LPSTR) NULL);
                                                        // or
SetAvortProc(hPrintDC,lpfnAbortProc);
```

当打印操作进行到最后一页时，程序员可以使用 Escape 函数(…NEWFRAME…)打印新页，此时，完全可能发生磁盘空间不够的问题。在这种情况下，如果没有 Abort 函数的处理，SP_OUTOFDISK 错误消息会终止 Escape 函数的执行。从下面的讨论可看出 Abort 函数是如何帮助解决这一难题的。

打印过程中如果遇到页中断，执行进程将发信号给设备驱动程序，由它把 GDI 指令翻译为依赖于打印机的功能调用，并将之存到一个新文件中。

在时间紧张的过程中，Windwos 总是将这些重复的调用分配给 Abort 函数，由它从队列中取出消息发送给主窗口过程。如果队列为空，Windows 便为其他应用程序释放 CPU 时间。

如果没有消息等待处理，那么凡是用 PeekMessage 函数设计消息循环的程序就轮流占用 CPU 时间。

这些程序也包括打印管理器，它此时可能正将数据从元文件送到打印机上。一旦进行到文件尾，打印管理器会自动删除文件，同时释放磁盘空间。接着，控制面板返回给 Abort 函数，PeekMessage 返回 FALSE 值，而 Abort 函数则返回 TRUE 值。

返回值为正时，进程将告诉 Windows 有可用的磁盘空间，可以继续打印。

如果 Abort 函数给出“return(FALSE)”指令，意味着 Windows 确认打印过程执行完毕。这时，如果再调用 Escape 函数的 NEWFRAME 换页序列，就不会再有错误发生。Escape 函数将返回一个 NULL 值，EndPage 函数则返回错误码 SP_APPABORT(-2)。

EnableWindow 函数

在打印操作完毕之前，程序员必须防止其他可能造成影响的动作发生，如在旧文件正在

写入时调入了新文件。为此，用户可以在打印前锁定窗口，而在打印操作结束后打开它。

```
EnableWindow (hWnd,FALSE);
                                                    // Print
EnableWindow (hWnd,TRUE);
```

程序员可能会觉得奇怪，为何可以在 Abort 函数中调用 PeekMessage 和 DispatchMessage，因为当时那些未激活的窗口可能不发来消息。严格地讲，除 WM_PAINT 消息外，确定没有其他消息送入，而且 WM_PAINT 也得在打印操作之后处理。

4.2.2 Abort 对话框

大多数程序都提供了让用户取消打印操作的功能，通常这是由对话框来实现的。该框包括一些简短的文字说明和 Cancel 按钮。对话框一直保留在屏幕上，直到 Windows 的打印管理器被唤醒，并开始写元文件为止。程序员并不需要按一定模式设计对话框，完全可以采用自己的对话框设计函数。

```
FARPROC lpfnPrintDlgProc;
HWND hDlgPrint;

lpfnPrintDlgProc = MakeProcInstance(PrintDlgProc,hInst);
hDlgPrint = CreateDialog(hInst,"AbortDlg",hWnd,lpfnPrintDlgProc);
```

对话框函数 PrintDlgProc 至少处理两种消息，即 WM_INITDIALOG 和 WM_COMMAND，前者表示将对话框标题用作应用程序标题，两后者则说明 Cancel 按钮已激活。

```
int FAR PASCAL PrintDlgProc(HWND hDlg,unsigned
        msg,WORD wParam,LONG lparam)
{
    switch(msg)
    {
        case WM_INITDIALOG:
            SetWindowText(hDlg,szarAppName);
            break;
        case WM_COMMAND:
            bUserAbort = TRUE;
            EnableWindow (GetParent (hDlg),TRUE);
            DestroyWindow (hDlg);
            hDlgPrint = 0;
            break;
        default:
            retrun FALSE ;
    }
    return TRUE;
}
```

如果用户终止打印操作，Abort 对话框必须消隐，以便处理过程重新使用主窗口。可能有人会觉得奇怪，既然已经有 IsDialogMessage 函数，为何还要这个对话框函数来接收消息？其中原因在以后合适之处再作讨论。

Abort 函数和 Abort 对话框

在进行打印程序设计的过程中，Abort 对话框和对话函数一般不单独使用，而与 Abort

函数(SetAbortProc)一起使用,而且,还用到了全程变量 hDlgPrint 和 bUserAbort,如前面的程序所示。

在 Windows 例程中,对话框句柄被赋给变量 HDlgPrint。这是通过调用 CreateDialog 函数实现的。一旦用户将对话框从屏幕上删除掉,该句柄则被设置为 NULL(见上)。如果 bUserAbort 的值为 TRUE,那么意味着用户已通过对话框终止了打印操作。

注意,不能把 Abort 函数的返回值用于 Abort 对话框的对话函数值相混淆,Abort 函数在执行向对话框发送消息的任务时作为某些扩充。

扩充的 Abort 函数

```
int FAR PASCAL AbortProc (HDC hPrintDC,int Code)
{
    MSG msg;
    while (! bUserAbort && PeekMessage(&msg,NULL,0,0,PM_RECOVE)
    if (! hDlgPrint || IsDialogMessage(hDlgPrint,&msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (! bUserAbort);
}
```

此例多次说明了 Abort 函数的处理方法,以及这两类函数之间的相互作用。对 EndPage 的调用和 NEWFRAME 换码序列的引用都促使 Windows 和打印机驱动程序将指令翻译为打印命令。

另外,GDI 模块也重复地调用 Abort 函数。如果用户按下对话框中的 Cancel 按钮,Abort 函数中的 IsDialogMessage 将转换为 WM_COMMAND 消息发送给对话函数,同时全局变量 bUserAbort 被赋以 TRUE 值。

当 Abort 函数再次激活时,WHILE 循环将终止执行,同时由于返回了 FALSE 值(! bUserAbort),控制权将交还给 Window,FALSE 值还导致 Windows 终止打印操作,EndPage 函数返回错误码 SP_APPABORT(-2)。

如果用户程序使用 Escape 函数的 NEWFRAME 序列,而不用 EndPage,Abort 函数则不返回错误码,但程序仍需检查全程变量 bUserAbort 并终止输出循环,同时调用 ENDDOC 序列终止打印操作。

4.2.3 打印函数举例

用户在菜单中选择打印项后,整个正文才开始打印。当然,用户可以随时取消打印请求。

下面的例子程序使用了新的打印函数(StartDoc 等)以取代 Escape 函数的相应换码序列。

新函数	描述
CreateDC	创建一个打印机设备环境句柄
EnableWindow	改变窗口状态
EndDoc	结束打印请求
EndPage	结束一页

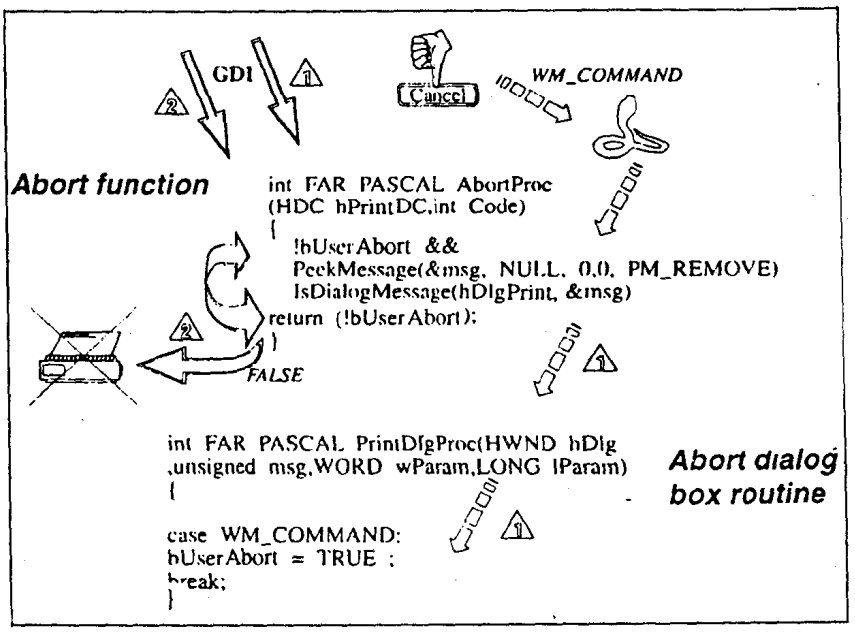


图 4.5　Abort 对话框函数和 Abort 函数之间的相互作用

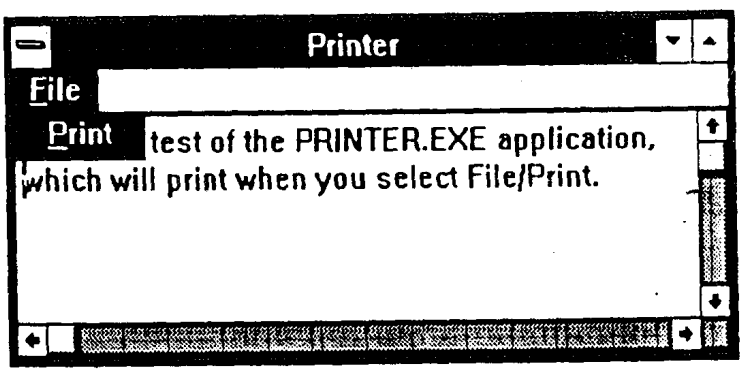


图 4.6　正文输出示例

GetDeviceCaps	获得特定设备的信息
PeekMessage	从队列取出下一条消息
SetAbortProc	定义打印机的 Abort 函数
StartDoc	开始处理打印请求
StartPage	驱动程序准备接收数据
新结构	描述
DOCINFO	StartDoc 函数所需的结构
新消息	描述
EM_GETLINECOUNT	得到一个正文域的行计数
EM_GETLINE	将一行内容存储到缓冲区中

源代码:PRINTER. C

```
/* * PRINTER.C * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates printing under Windows * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "printer.h"                                    // Include PRINTER.H header file

#if _MSC_VER >= 700                                     // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                          // higher? Disable warning 4028
#endif

/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain(HANDLE hInstance,                            // Current instance
        HANDLE hPrevInstance,                                   // Previous instance
        LPSTR lpszCmdLine,                                      // Long ptr to string after
                                                                // program name during execution
        int nCmdShow)                                           // Specifies the application
                                                                // window's appearance
{
    MSG msg;                                                    // Message variable
    HWND hWnd;                                                  // Window handle
    int nError;                                                 // Error integer

    strcpy(szAppName,"Printer");                                // Application name
    hInst = hInstance;                                          // Instance handle

    if (! hPrevInstance)                                        // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                                                // If initialization fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;
        }
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow(szAppName,                              // Window class name
            szAppName,                                          // Window caption
            WS_OVERLAPPEDWINDOW,                                // Overlapped window style
            CW_USEDEFAULT,0,                                    // Default upper-left x,y pos.
            CW_USEDEFAULT,0,                                    // Default initial x,y sizes
            NULL,                                               // No parent window
            NULL,                                               // Window menu used
            hInstance,                                          // Application instance
```

```
            NULL);                                          // No creation parameters

    if(hWnd == NULL)                                        // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,               // then display error window
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );                            // Make window visible
    UpdateWindow( hWnd );                                   // Update window

    while (GetMessage(&msg,NULL,0,0))                       // Message reading
    {
        TranslateMessage(&msg);                             // Message translation
        DispatchMessage(&msg);                              // Send message to Windows
    }
    return (int)msg. wParam;                                // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)                    // Window class registration
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                       // Main window class

    wcClass. hCursor = LoadCursor( NULL,IDC_ARROW );        // Mouse cursor
    wcClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);       // Default icon
    wcClass. lpszMenuName = (LPSTR)"PrintMenu";             // Menu name
    wcClass. lpszClassName = szAppName;                     // Window class
    wcClass. hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
                                                            // White background
    wcClass. hInstance = hInstance;                         // Instance
    wcClass. style = CS_VREDRAW | CS_HREDRAW;
                                                // Horizontal and vertical redraw of client area
    wcClass. lpfnWndProc = PrintWndProc;                    // Window function
    wcClass. cbClsExtra = 0 ;                               // No extra bytes
    wcClass. cbWndExtra = 0 ;                               // No extra bytes

    if (! RegisterClass( &wcClass ) )                       // Register window class
        return FALSE;                                       // Return FALSE if registration fails
    return TRUE;                                            // Return TRUE
}

/* * KeyWndProc * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL PrintWndProc(HWND hWnd,
            unsigned message,
```

```
                WORD wParam,
                LONG lParam )
{
static HWND hEdit;                                          // Window handle
FARPROC lpfnPrintDlgProc,lpfnAbortProc;                     // FAR procedures
HDC hPrintDC;                                               // Device context handle
TEXTMETRIC tm;                                              // Text metrics
short yChar,xChar;                                          // x characters & y characters
int NoLinesPerPage,                                         // Number of lines per page
    Page,                                                   // Page size
    CurrentLine,                                            // Current line number
    LineLength;                                             // Line length
int ErrorCode;                                              // Error code integer
WORD NumberOfLines,Counter;                                 // Number of lines & line counter
DOCINFO di;
char szTheTitle[] = "Print Demonstration";
char szLine[128];

    switch (message)
    {
    case WM_CREATE:                                         // Create edit window
        hEdit = CreateWindow("Edit",NULL,
            WS_CHILD | WS_VISIBLE |
            WS_HSCROLL | WS_VSCROLL |
            WS_BORDER | ES_LEFT |
            ES_MULTILINE,0,0,0,0,
            hWnd,50,hInst,NULL);
        break;

    case WM_SIZE:                                           // Client area size
        MoveWindow(hEdit,0,0,LOWORD(lParam),
            HIWORD(lParam),TRUE);
        break;

    case WM_SETFOCUS:                                       // Set input focus
        SetFocus(hEdit);                                    // to Edit window
        break;

    case WM_COMMAND:                                        // Commands
        switch(wParam)
        {
    case IDM_PRINTOUT:                                      // Print
        hPrintDC = GetPrinterDC();                          // Get printer device context
        if (! hPrintDC)                                     // Can't print? Display error
        {
            MessageBox(hWnd,"Cannot print",NULL,MB_OK | MB_ICONHAND);
            return (NULL);
        }

        bUserCancels = FALSE;                               // User cancelled printing
        EnableWindow(hWnd,FALSE);
```

```
lpfnPrintDlgProc = MakeProcInstance(PrintDlgProc,hInst);
hDlgPrint = CreateDialog(hInst,"AbortDlg",
    hWnd,lpfnPrintDlgProc);
if (! hDlgPrint)
{
    MessageBox(hWnd,"Could not create a Cancel dialog box",
        NULL,MB_OK | MB_ICONHAND);
    EnableWindow(hWnd,TRUE);
    FreeProcInstance(lpfnPrintDlgProc);
    DeleteDC(hPrintDC);
    return (FALSE);
}

lpfnAbortProc = MakeProcInstance(AbortProc,hInst);
SetAbortProc(hPrintDC,lpfnAbortProc);

di.cbSize = sizeof(DOCINFO);
di.lpszDocName = szTheTitle;
di.lpszOutput = NULL;
ErrorCode = StartDoc(hPrintDC,&di);
if (ErrorCode < 0)
{
    PrintError(ErrorCode);
    EnableWindow(hWnd,TRUE);
    FreeProcInstance(lpfnPrintDlgProc);
    FreeProcInstance(lpfnAbortProc);
    DeleteDC(hPrintDC);
    return FALSE;
}

GetTextMetrics(hPrintDC,&tm);
yChar = tm.tmHeight + tm.tmExternalLeading;
xChar = tm.tmAveCharWidth;
Page = GetDeviceCaps (hPrintDC,VERTRES);
NoLinesPerPage = Page / yChar - 1;
NumberOfLines = (WORD)SendMessage(hEdit,EM_GETLINECOUNT,0,0L);
CurrentLine = 1;

StartPage(hPrintDC);
for (Counter = 0; Counter < NumberOfLines; Counter++)
{
    LineLength = (int)SendMessage(hEdit,EM_GETLINE,
        Counter,(DWORD)(LPSTR)szLine);
    TextOut(hPrintDC,xChar,CurrentLine * yChar,
        (LPSTR)szLine,LineLength);
    if (++ CurrentLine > NoLinesPerPage )
    {
        CurrentLine = 1;
        ErrorCode = EndPage(hPrintDC);
        if (ErrorCode < 0)
            break;
```

```
                else StartPage(hPrintDC);
            }
        }
        if (NumberOfLines < (WORD)NoLinesPerPage )
            ErrorCode = EndPage(hPrintDC);

        if (ErrorCode < 0)
            PrintError(ErrorCode);
        else
            EndDoc(hPrintDC);

        if (! bUserCancels)
        {
            EnableWindow(hWnd,TRUE);
            DestroyWindow(hDlgPrint);
        }
        FreeProcInstance(lpfnPrintDlgProc);
        FreeProcInstance(lpfnAbortProc);
        DeleteDC(hPrintDC);
        break;

    default:
        return (DefWindowProc(hWnd,
            message,
            wParam,
            lParam));
        break;

    }
    return(0L);
}

/* * GetPrinterDC * * * * * * * * * * * * * * * * * * * * * */
/* * Creates device context for default printer * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

HDC GetPrinterDC(void)
{
    char pPrintInfo[80];                          // Printable information
    LPSTR lpTemp;
    LPSTR lpPrinter;                                        // Printer
    LPSTR lpPntDriver;                                  // Printer driver
    LPSTR lpPntrInterface;                            // Printer interface

    if (! GetProfileString("windows",             // Profile string of print info
            "Device",
            (LPSTR)"",
            pPrintInfo,80))
    return (NULL);
    lpTemp = lpPrinter = pPrintInfo;
    lpPntDriver = lpPntrInterface = 0;
```

```
    while ( * lpTemp)
    {
        if ( * lpTemp == ',')
        {
            * lpTemp++ = 0;
            while ( * lpTemp == ' ')
                lpTemp++;
            if (! lpPntDriver)
                lpPntDriver = lpTemp;
            else
            {
                lpPntrInterface = lpTemp;
                break;
            }
        }
        else
            lpTemp++;
    }
    return (CreateDC(lpPntDriver,                          // Return printer device context
            lpPrinter,
            lpPntrInterface,
            (LPSTR) NULL));
}

/* * PrintError * * *
```

```
            case SP_APPABORT:                                    // General error
                index = 3;
                break;
            default:
                index = 4;
                break;
        }
        MessageBox(NULL,                                         // Display error message box
            szError[index],
            NULL,
            MB_OK | MB_ICONEXCLAMATION);
    }
}

/* * AbortProc: Abort operation * * * * * * * * * * * * * * * * * */

int FAR PASCAL AbortProc(HDC hPrintDC,int Code)
{
    MSG msg;

    while (! bUserCancels && PeekMessage(&msg,NULL,0,0,PM_REMOVE))
        if (! hDlgPrint || ! IsDialogMessage(hDlgPrint,&msg))
        {
            TranslateMessage(&msg);                              // Message translation
            DispatchMessage(&msg);                               // Send message to Windows
        }
    return (! bUserCancels);                                     // Display cancel message
}

int FAR PASCAL PrintDlgProc(HWND hDlg,
            unsigned msg,
            WORD wParam,
            LONG lParam)
{
    switch(msg)
    {
        case WM_COMMAND:                                         // Exit
            bUserCancels = TRUE ;
            EnableWindow (GetParent (hDlg),TRUE) ;
            DestroyWindow (hDlg) ;
            hDlgPrint = 0 ;
            break ;
        default:
            return FALSE;
    }
    return TRUE ;
}
```

模块定义文件:PRINTER. DEF

```
NAME PRINTER
DESCRIPTION 'PRINTER'
```

```
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS PrintWndProc @1
    AbortProc @2
    PrintDlgProc @3
```

资源描述文件:PRINTER.RC

```
#include "printer.h"
#include "windows.h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW,"Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END

PrintMenu MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&Print", IDM_PRINTOUT
    END
END

AbortDlg DIALOG 40,40,120,40
    CAPTION "Abort"
    STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE
    BEGIN
        CTEXT "Cancel printing",-1,4,6,120,12
        DEFPUSHBUTTON "Cancel", IDCANCEL,44,22,32,14,WS_GROUP
    END
```

头文件:PRINTER.H

```
#define PRINTING
#include <windows.h>
#include <string.h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define IDM_PRINTOUT 100

char szString[128];                    /* Variable for loading resource text */
char szAppName[20];                    /* Window class name */
```

```
HANDLE hInst;
BOOL bUserCancels;
HWND hDlgPrint;

int nCwRegisterClasses(HANDLE);
HDC GetPrinterDC(void);
void PASCAL PrintError(int);
long FAR PASCAL PrintWndProc( HWND,unsigned,WORD,LONG);
int FAR PASCAL AbortProc(HDC,int);
int FAR PASCAL PrintDlgProc(HWND,unsigned,WORD,LONG);
```

MAKE 文件:PRINTER. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES =
H = PRINTER. H

all: PRINTER. EXE

PRINTER. RES: PRINTER. RC $ (RESFILES)

PRINTER. OBJ: PRINTER. C $ (H)

PRINTER. EXE: PRINTER. OBJ PRINTER. RES PRINTER. DEF
    link /NOD/A:16 PRINTER,PRINTER,nul,libw slibcew,PRINTER. def
    rc PRINTER. res PRINTER. exe
```

按下述方式运行 NMAKE,创建 PRINTER 应用程序:

```
nmake printer. mak
```

在包含 WINDOWS. H 文件之前,必须在头文件 PRINT. H 中加上命令 # define PRINTING,这样才能使用打印函数如 StartDoc 等。

如果用户想输入正文,则可以在主窗口中开辟一个编辑区,其大小由 WM_SIZE 消息指定。运行时,焦点始终对准该正文区,以便用户随时进行编辑工作。

用户在菜单中选择了打印命令以后,系统首先从 WIN. INI 文件中读取当前缺省打印机配置,这项工作由专门提供给应用程序的 GetPrinDC 函数完成。然后,锁定窗口以防用户干扰打印操作。

Abort 函数和对话函数的瞬间过程地址都写入了定义文件,而且需要时也是从该文件中读取,这样有助于用 SetAbortProc 函数定义 Abort 函数以及显示对话框。

如果有错误发生,必须中断已执行的任务,否则打印作业只好用 StartDoc 函数重新启动。

```
di. cbSize = sizeof(DOCINFO);
di. lpszDocName = szTitle;
```

```
di.lpszOutput = NULL;
Errorcode = StartDoc(hPrintDC,&di);
```

下面的几行代码用于取得打印纸尺寸和字模的维数，以便计算每页所需行数，同时发送EM_GETLINECOUNT 消息给正文域以确定要打印的行数。

由 StartPage 函数开始首页打印工作，它利用 FOR 循环按行打印。如果输出多页文档，可在调用 EndPage 后再调用 StartPage 生成换页。如果打印过程中没有错误发生，则调用 EndDoc 终止打印进程。

```
Linecount = (WORD)SendMessage(hEdit,EM_GETLINECOUNT,0,OL);

StartPage(hPrintDC);
for (Counter = 0; Counter<Linecount;Counter++)
{
    LineLength = (int)SendMessage(hEdit,WM_GETLINT,Counter,(DWORD)(LPSTR)szLine);
    TextOut(hPrintDc,xChar,aktLine * yChar,(LPSTR)szLine,Linelength);
    if (++ curLine >LinesPerPage )
    {
        aktLine = 1;
        Errorcode = EndPage(hPrintDC);
        if (Errorcode < 0)
            break;
        else StartPage(hPrintDc);
    }
}
if (Lonecount < (WORD)LinesPerPage)
    Errorcode = EndPage(hPrintDC);
if (Errorcode < 0)
    PrintError(Errorcode);
else
    EndDoc(hPrintDC);
```

如果用户在对话框中按下了 Cancel 按钮以取消输出，那么程序员必须在对话框中将全程变量 bUserAbort 设置为 TRUE 值，同时由 Abort 函数通知 GDI 的终止模块以结束进程，这种情况下错误码为 SP_APPABORT。

如果打印过程由于某个实际错误而终止，而不是因为用户的干预，那么系统将在执行完对话函数的剩余任务后进行错误检查，这些任务包括清除对话框，重新打开主窗口等。

```
if (! bUserAbort)
{
    EnableWindow(hWnd,TRUE);
    DestroyWindow(hDlgPrint);
}
```

如果运行中有错误发生，系统可以向用户发送消息框，当然必须保证 Windows 还没有这么做，也就是说，此时 SP_NOTREPORTED 已设置了值。

4.2.4 分段打印

分段是一种技术，它将整个图形页分为许多小片段，然后将它们分别送至打印机打印。

通常，应用程序本身不使用分段功能，由 GDI 模块和打印机驱动程序实施这种技术。然而，在程序中使用分段技术却有两大优点。

首先，调用 GDI 中可对所给片段进行操作的函数能提高打印速度。因为这种方式的操作不必访问元文件。其次，将整体的磁盘存储改为分段存储，这在打印大的位图时尤其具有重要意义。

GetDeviceCaps 函数

程序员可以利用参数 RASTERCAPS 调用 GetReviceCaps 函数，从而确定设备是否支持分段，如下所示，

```
if (GetDeviceCaps (hPrintDC,RASTERCAPS) & RC_BANDING)
// Printer supports banding technique
```

程序员在这个问题的处理上不能和帮助系统混淆。对 RC_BANDIGH 位的解释指出，当该位被置位时，表示设备需要分段技术支持。不过，这句话指的是 GDI 模块，而不是针对程序员。该位将告诉 Windows，是否打印设备能独立执行分段技术。

如果上面的调用返回 TRUE 值，则程序员可以使用分段技术，不过并非一定要用。例如，PostScript 打印机不需要 GDI 模块的帮助，因而不用设置 RC_DANDING 位。

NEXTBAND

如果想在程序中执行分段功能，程序员必须定义一个 RECT 类型的变量接收下一段的尺寸值，该尺寸是通过调用 Escape 函数的 NEXTBAND 序列得到的，尺寸大小用设备坐标（像素）表示。不考虑打印设备环境设置的坐标映射模式，而由专门的 DPtoLP 函数将之转换为逻辑坐标。

另外，还可以调用 GDI 函数实现分段。虽然它有可能激活所有输出函数，但由于仅仅将当前段设置为裁剪区域，因而这种可能性比较小。

而且，除该段外，所有输出都被忽略了。如果想得到下一个段，可以再次调用 NEXTBAND。如此重复循环，直到函数返回一个四个值皆为空矩形的结构，这意味着打印已进行到页尾。

```
DOCINFO di;
HDC hPrintDC;
RECT rect;

//Printing a page with banding
di. cbSize = sizeof(DOCINFO);
di. lpszDocName = szTitle;
di. lpszOutput = NULL;
if (StartDoc(hPrintDC,&di)>0 && Escape(hPrintDC,NEXTBAND,0,NULL,(LPSTR)&rect)>0)
{
    while (! IsRectEmpty(&rect)
    {
        DPtoLP(hPrintDC,&rect,2);
        // Call GDI functions
        Escape( hPrintDC,NEXTBAND,0,NULL,(LPSTR)&rect)
    }
}
Escape( hPrintDC,NEWFRAME,0,OL,OL);
```

```
EndDoc( hPrintDC);
```

4.3 打印机驱动程序

最早引进 Windows 的时候，几乎没有什么可用的打印机驱动程序。但由于打印对于多数用户来说是一项基本要求，因此缺乏合适的驱动程序一直是众人评判的话题。

然而，随着 Windows 3.1 日益为人们所接受，这种状况也有了很大改进。目前，几乎所有打印机都有一个 Windows 驱动程序。这一切即使对制造商来说也是很有利的，不过驱动程序最频繁的使用者还是 Windows 本身。

这些打印机驱动程序的文件都以 DRV 为扩展名，并以动态链接库方式执行。这使得多个应用程序有可能访问同一驱动程序，并且可以几乎同时执行打印操作。

打印驱动程序的根本任务是将与设备无关的 GDI 函数调用翻译为一系列命令和数据，而且这些命令和数据必须是打印机可理解的。另外，大多数驱动程序都允许用户改变打印机设置，如打印纸规格等。

因为系统所支持的打印机特征总是随机器种类的不同而互有差异，所以使用对话框改变设置的任务一般不交给 Windows 或应用程序，而是由驱动程序完成，例如图 4.7 所示的对话框声明，且相关代码就是调用 PSCRIPT.DRV 文件的结果。

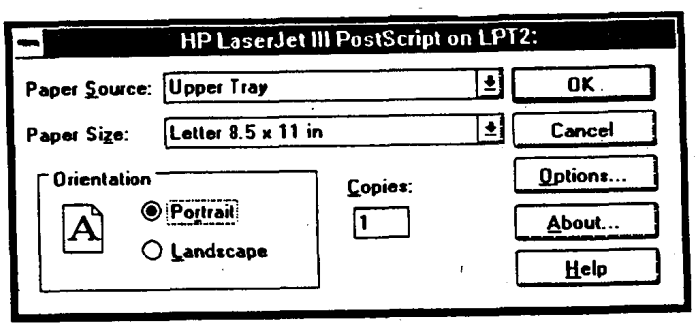


图 4.7 HP 激光打印机选项

如果用户确认了所改变的打印机设置，大多数打印机驱动程序会将新值写入 WIN.IN2 文件中。每一种如果只采用缺省设置，则文件中无输入信息。

```
[Apple LaserWriter II NTX.LPT2]
advflag=664
feed1=13
feed4=9
orient=2

[NEC Pinwriter P2200,LPT1]
Orinetation=2
Print Quality=360
Y Resolution=360
```

4.3.1 打印纸格式

由于打印机类型不同(点阵、激光、菊花轮)，打印机声明部分的结构也互不相同。但下例

中,两部分都含有一个与打印纸格式有关的关键字,前一部分为"orient",后一部分为"Orientation"。在这两种情况下,值为 1 都表示肖像格式,值为 2 则代表风景格式。

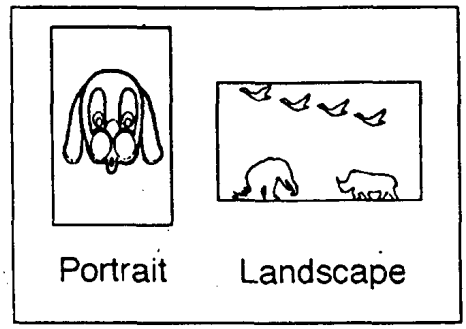


图 4.8 两种可能的打印纸格式

Windows 3.1 中定义了另一种对话框,用于指定打印机设置,被称为公用对话框。应用程序负责显示这种对话框,并处理用户响应,包括将适当输入存入 WIN.IN 文件中。

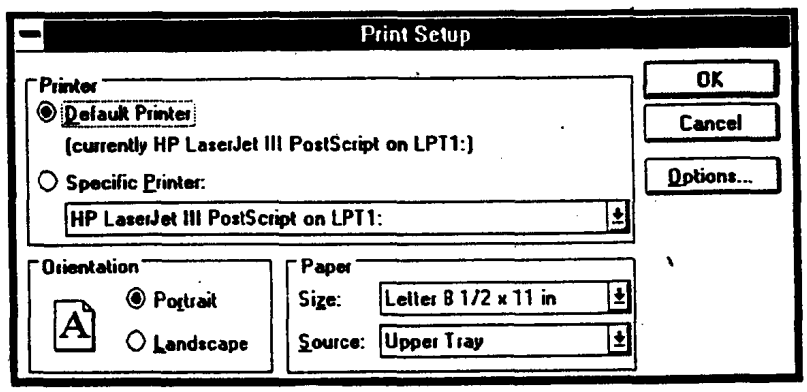


图 4.9 装入 Apple 激光打印机的公用对话框

4.3.2 DeviceMode 函数

程序员除了使用公用对话框外,可能还要进行应用程序与打印机驱动程序间的通信。为了在屏幕上显示驱动程序的对话框,程序员必须从驱动程序的动态链接库(如 PSCRIPT.DRV)中调用 DeviceMode 函数。几乎所有的 Windows 打印驱动程序都把它当作出口声明来执行。其原型定义如下:

```
Void FAR PASCAL DeviceMode(hWnd, HANDLE hLibrary, LPSTR szDeviceName,
                           LPSTR szPort)
```

该函数的第二个参数代表驱动程序的文件句柄。因为从技术上讲每个驱动程序都是一个动态链接库,所以主要由 LoadLibrary 函数调入参数信息。

驱动程序名通常是从 WIN.INI 文件中获得的,该文件在[windows]部分声明当前缺省打印机,而在[devices]部分声明其他打印机。

下例中,用 lpDriver 变量存储驱动程序名,它也是从 WIN.INI 中读取的,不过必须将 DRV 扩展名加到后面,以便调入动态链接库:

```
lstrcpy(szDriverFile, lpDriver);
lstrcat(szDriverFile, ".DRV");
hDriver = LoadLibrary(szDriverFile);
if (hDriver < 32)
```

```
return(FALSE);
```

GetProcAddress 函数

下面这段程序使用 GetProcAddress 函数获得驱动程序中 DeviceMode 函数的地址，并由其第二个参数指定名字或所需函数的序数。

```
FARPROC lpfnDevMode
lpfnDevMode = GetProcAddress (hPriver,"DEVICEMODE");
```

程序员可以用 PROC_OLDDEVICENAME 代替 DEVICEMODE，该值在 PRININIT. H 文件中定义，这个文件必须包含到程序中：

```
#define PROC_OLDDEVICEMODE MAKEINTRESOURCE (B)
```

如果打印机驱动程序没有提供 DeviceMode 函数，GetProcAddress 将返回值。若返回值为正，程序员可以使用该地址间接调用 DeviceMode 函数。

```
(*lpfnDevMode) (hWnd,hDrive,lpPrinter,lpPort)
```

指针 lpPointer 指向诸如“AppleLaserWriter II NIX”的字符串，而 lpPort 则指向“LPT2:”这样的端口名，这两个参数的值都从 WIN. INI 文件中读取，同时在 WIN. INI 文件中说明的驱动程序名则用于装入库。

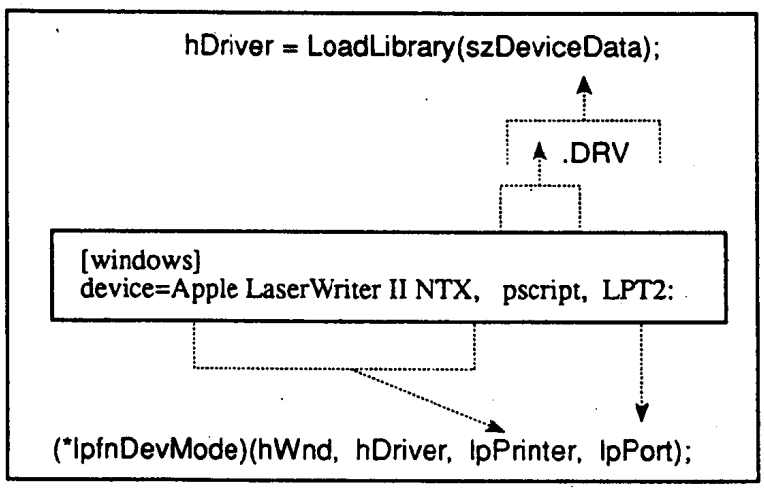


图 4.10 供 DeviceMode 函数使用的 WIN. INI 文件信息

调用 DeviceMode 函数的结果是在屏幕上显示对话框，以使用户改变打印机设置。下面看看究竟打印机驱动程序包括哪些过程。

驱动程序的内部处理过程

首先，根据当前打印机设置填充 DEVMODE 结构的域值，并对指定的端口赋值。DEVMODE 结构不是在 WINDOWS. H 中定义的，而是在附加的头文件 PRIVINIT. H 中定义。

Windows 在内存中保存了打印机设置。如果该设置尚未存储，Windows 会从 WIN. INI 中查找。若文件中没有，则将 DEVMODE 结构所填充的标准值显示在对话框中。

接下来，用户可按需要改变设置值，也可以用鼠标器选择 OK 按钮或 Cancel 按钮关闭

话框。按 OK 后，进程会把设置信息写到 DEVMODE 结构中以及 WIN. INI 中的适当部分。另外，可调用 SetEnvironment 函数将改变后的设置保存在内存中，供 GDI 模块访问。

这时，先前应用于端口的所有环境设置都被删除，进程向所有的主窗口和弹出窗口发送下一条 WM_DEVMODECHANGE 消息，并将变化通知其他活动程序，用 lParam 参数索引打印机名。

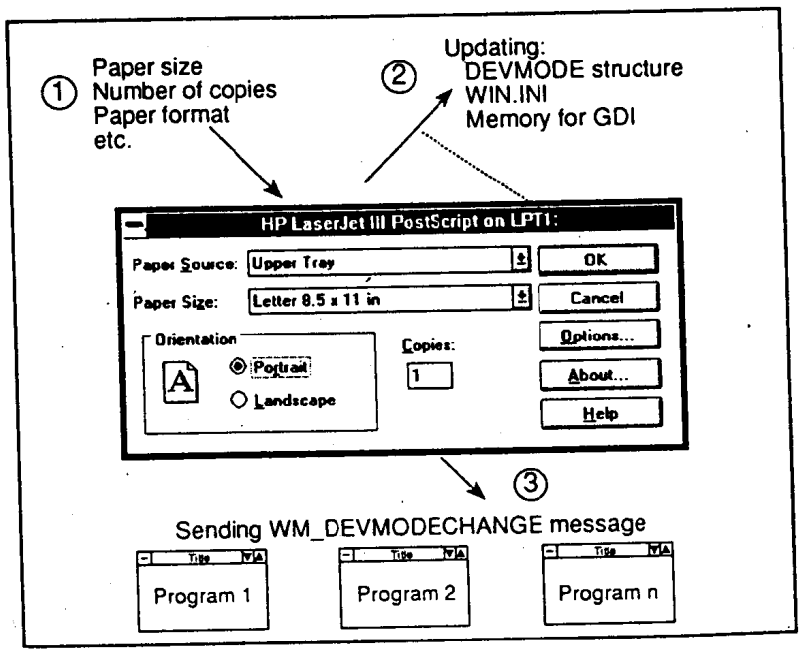


图 4.11　DeviceMode 函数中设置驱动程序的执行

最后打印机驱动程序从屏幕上删除对话框，同时将控制权交还给调用 DeviceMode 函数的应用程序。

WM_DEVMODECHARNGE 消息

下面的程序重新处理 WM_DEVMODECHANGE 消息，变量 lpPort 指向字符串“LPT1”。

```
LPDEVMODE lpEnv;
HANDLE hEnv;
WORD nMax;

case WM_DEVMODECHANGE:
    wMax = GetEnvironment ( lpPort,NULL,0);
    hEnv = GlobalAlloc(GMEM_MOVEAB
    lpEnv = (LPDEVMODE)GlobalLock(hEnv);
    GetEnvironment(lpPort,(LPSTR)lpEnv,wMax);
    // Processing of values in DEVMODE structure,
    // referenced by the pointer lpEnv
    // e. g. lpEnv->dmDeviceName
    GlobalUnlock(lpEnv);
    GlobalFree(hEnv);
```

```
    break;
```

此段代码用于确定新的打印机设置，其中包括 DEVMODE 结构，因为此结构包括某种特定信息，如它可能随长度而变化，所以该结构的所有大小都必须事先确定。程序员可调用 GetEnviroment 函数达到这一目的，调用时第二个和第三个参数为空。

该函数的返回值用于分配所需的缓冲区空间。可以用 lpEnv 指针第二次调用 GetEnviroment，同时将打印机的设置信息填入缓冲区，占用长度为 wMax 的区域。

Windows SDK 3.1 也可应用这种方式。Windows 3.1 的帮助系统揭示了 DEVMODE 结构的某些变化，特定设备的信息已不再是该结构的直接子域，因为已无法通过 dmDriverData 域引用它，只能把它当作附加成分。

不过，为 DEVMODE 结构分配的空间都是相同的，因此每种打印机都可以用 sizeof (DEVMODE)决定内存大小。

DEVMODE 结构

下面是关于 DEVMODE 结构的定义，可以在 Windows 3.1 的头文件 DRIVIHIT.H 中找到它。

```
typedef struct _devicemode {
    char dmDeviceName[CCHDEVICENAME];
    WORD dmSpecVersion;
    WORD dmDriverVersion;
    WORD dmSize;
    WORD dmDriverExtra;
    DWORD dmFields;
    short dmOrientation;
    short dmPaperSize;
    short dmPaperLength;
    short dmPaperWidth;
    short dmScale;
    short dmCopies;
    short dmDefaultSource;
    short dmPrintQuality;
    short dmColor;
    short dmDuplex;
    short dmYResolution;
    short dmTTOption;
} DEVMODE;
```

结构中的前五个域是头标信息，包括打印机驱动程序的一般信息，如设备名和版本号等，后面的域则指明了将在对话框中显示的变量设置。

为了增强可读性，大多数域值都作为常量在 DRIVINIT.H 文件中定义，如 dmOrientation 域，可使用值 DMORIENT_PORTRAIT(1)或 DMORIENT_LANDSCAPE(2)。

4.3.3 DeviceMode 函数应用举例

本程序展示用于打印机设置的公对话框与驱动程序函数 DeviceMode 之间的差别，这一差别将影响对话框的显示结果和消息处理。

图 4.12 显示的是当前缺省打印机的设置。

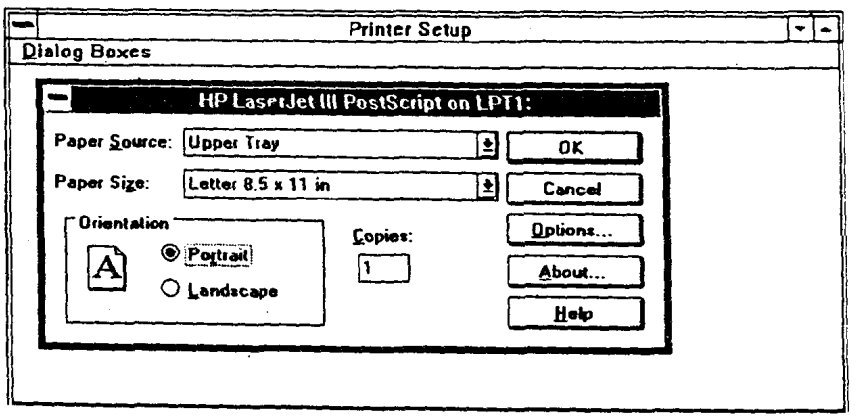


图 4.12

新函数	描述
DeviceMode	打印机驱动程序调用对话框
GetProcAddress	获取模块中某函数的地址

新消息	描述
WM_DEVMODECHANGE	打印机环境已改变
WM_WININICHANGE	WIN. INI 的内容改变

源代码:DEVMODE. C

```
/* * DEVMODE. C * * *
```

```
    strcpy(szAppName,"DevMode");                          // Application name
    hInst = hInstance;                                    // Instance handle

    if (! hPrevInstance)                                  // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                                          // If registration fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;
        }
    }

/* * Specify appearance of application's main window * * * * * * * * * */
    hWnd = CreateWindow(szAppName,                        // Window class name
            "Printer Setup",                              // Window caption
            WS_OVERLAPPEDWINDOW,                          // Overlapped window style
            CW_USEDEFAULT,0,                              // Default upper-left x,y pos.
            CW_USEDEFAULT,0,                              // Default initial x,y sizes
            NULL,                                         // No parent window
            NULL,                                         // Window menu used
            hInstance,                                    // Application instance
            NULL);                                        // No creation parameters

    if(hWnd == NULL)                                      // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,             // then display error window
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );                          // Make window visible
    UpdateWindow( hWnd );                                 // Update window

    while (GetMessage(&msg,NULL,0,0))                     // Message reading
    {
        TranslateMessage(&msg);                           // Message translation
        DispatchMessage(&msg);                            // Send message to Windows
    }
    return (int)msg. wParam;                              // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)                  // Window class registration
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * * */
    WNDCLASS wcClass;                                     // Main window class
```

```
    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );                    // Mouse cursor
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);                   // Default icon
    wcClass.lpszMenuName = (LPSTR)"SettingMenu";                       // Menu name
    wcClass.lpszClassName = szAppName;                                 // Window class
    wcClass.hbrBackground =(HBRUSH)GetStockObject(WHITE_BRUSH);        // White background
    wcClass.hInstance = hInstance;                                     // Instance
    wcClass.style = CS_VREDRAW | CS_HREDRAW;                           // Horizontal and vertical
                                                                       // redraw of client area
    wcClass.lpfnWndProc = WndProc;                                     // Window function
    wcClass.cbClsExtra = 0 ;                                           // No extra bytes
    wcClass.cbWndExtra = 0 ;                                           // No extra bytes

    if (! RegisterClass( &wcClass ) )                                  // Register window class
        return FALSE;                                                  // Return FALSE if registration fails
    return TRUE;                                                       // Return TRUE
}

/* * WndProc * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL WndProc(HWND hWnd,
            unsigned message,
            WORD wParam,
            LONG lParam )
{
    HDC hDC;                                                           // Device context handle
    PAINTSTRUCT ps;                                                    // Paint structure
    HANDLE hDriver;                                                    // Driver handle
    FARPROC lpfnDevMode;                                               // Device mode FAR procedure
    static PRINTDLG pd;
    static char szDriverFile [16];
    static char szDevMode[30];
    static BOOL bDevMode = FALSE;

    switch (message)
    {
        case WM_CREATE:                                                // Create window
            pd.lStructSize = sizeof(PRINTDLG);
            pd.hwndOwner = hWnd;
            if (! DefaultPrinter())                                    // Can't find default printer
                MessageBox(hWnd,"No printer found",NULL,MB_OK);
            break;

        case WM_COMMAND:                                               // Commands
            switch(wParam)
            {
                case IDM_STANDARD:                                     // Standard setup dialog box
                    pd.hDevMode = NULL;                                // Device mode
                    pd.hDevNames = NULL;                               // Device names
                    pd.Flags = PD_PRINTSETUP;                          // Flags for Print Setup
```

```
                pd.nCopies = 1;                                    // Number of copies (1)
                PrintDlg((LPPRINTDLG)&pd);
                break;
            case IDM_DEVMODE:                                      // Device mode call
                lstrcpy(szDriverFile,lpPntDriver);
                lstrcat(szDriverFile,".DRV");
                hDriver = LoadLibrary(szDriverFile);
                if (hDriver < 32)
                    return(FALSE);
                lpfnDevMode = GetProcAddress(hDriver,"DEVICEMODE");
                if (lpfnDevMode != NULL)
                    (*lpfnDevMode)(hWnd,hDriver,lpPrinter,lpPntrInterface);
                FreeLibrary(hDriver);
                break;
            default:
                return (DefWindowProc(hWnd,message,
                    wParam,lParam));
                break;
        }
        break;
    case WM_WININICHANGE:                                          // Change to WIN.INI?
        if ((lstrcmp((LPSTR)lParam,"windows")) || (lParam == NULL))
        {
            if (! DefaultPrinter())
                MessageBox(hWnd,"No printer found",NULL,MB_OK);
        }
        break;

    case WM_DEVMODECHANGE:                                         // Change in device mode?
        lstrcpy(szDevMode,(LPSTR) lParam);
        bDevMode = TRUE;
        InvalidateRect(hWnd,NULL,TRUE);
        break;

    case WM_PAINT:                                                 // Paint structure
        hDC = BeginPaint(hWnd,&ps);
        if (bDevMode)
        {
            TextOut(hDC,10,10,szDevMode,strlen(szDevMode));
        }
        EndPaint(hWnd,&ps);
        break;

    case WM_DESTROY:                                               // Exit
        PostQuitMessage(0);
        break;

    default:
        return (DefWindowProc( hWnd,message,wParam,
                lParam ));
        break;
```

```
    }
    return(0L);
}

BOOL DefaultPrinter(void)                                    // Default printer
{
    static char pPrintInfo[80];
    LPSTR lpTemp;

    if (! GetProfileString("windows","Device",(LPSTR)"",pPrintInfo,80))
        return (FALSE);
    lpTemp = lpPrinter = pPrintInfo;
    lpPntrInterface = lpPntDriver = 0;
    while ( * lpTemp)
    {
        if ( * lpTemp == ',')
        {
            * lpTemp++ = 0;
            while ( * lpTemp == ' ')
                lpTemp++;
            if (! lpPntDriver)
                lpPntDriver = lpTemp;
            else
            {
                lpPntrInterface = lpTemp;
                break;
            }
        }
        else
            lpTemp++;
    }
    return (TRUE);
}
```

模块定义文件:DEVMODE. DEF

```
NAME DevMode
DESCRIPTION 'DevMode Function'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS WndProc @1
```

资源描述文件: DEVMODE. RC

```
#include "devmode. h"
```

```
#include "windows.h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW,"Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END

SettingMenu MENU
BEGIN
    POPUP "&Dialog Boxes"
    BEGIN
        MENUITEM "&Print Setup…",IDM_STANDARD
        MENUITEM "&Device Mode…",IDM_DEVMODE
    END
END
```

头文件：DEVMODE.H

```
#include <windows.h>
#include <string.h>
#include "commdlg.h"

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define IDM_STANDARD 100
#define IDM_DEVMODE 101

char szString[128];                          /* Variable for loading resource text */
char szAppName[20];                                  /* Window class name */
HANDLE hInst;

LPSTR lpPrinter;
LPSTR lpPntDriver;
LPSTR lpPntrInterface;

int nCwRegisterClasses(HANDLE);
BOOL DefaultPrinter(void);
long FAR PASCAL WndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件：DEVMODE.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = DEVMODE.H
LIBS_EXT = commdlg.lib
```

```
all: DEVMODE.EXE

DEVMODE.RES: DEVMODE.RC $(RESFILES)

DEVMODE.OBJ: DEVMODE.C $(H)

DEVMODE.EXE: DEVMODE.OBJ DEVMODE.RES DEVMODE.DEF
    link /NOD/A:16 DEVMODE,DEVMODE,nul,libw slibcew $(LIBS_EXT),DEVMODE.def
    rc DEVMODE.res DEVMODE.exe
```

按下述方式运行 nmake，创建 DEVMODE 应用程序：

```
nmake devmode.mak
```

程序开始于对缺省打印机名和驱动程序名的访问，DefaultPrinter 例程从 WIN.INI 文件中读取这些信息，所有变量名都在[windows]部分利用关键字[device]声明。

本章前面的 PRINT 程序也使用了同样的例程，不过这里却没有用到打印机设备环境句柄。三个变量 lpDriver、lpPrinter 和 lpPort 定义为全程变量，以便窗口函数也能对它进行访问。

如果 Windows 应用程序改变了 WIN.INI 文件的某个设置，它会立即通知其他正在运行的程序，以使它们采取相应的动作。这一切都是通过发送 WM_WININICHANGE 消息实现的，它被送到所有高层窗口。

参数 lParam 通常指定 WININI 文件中改变了的部分。如果有多处被修改，这个参数则被设为 NULL 值。因此，在检查消息时，对于确定新的缺省打印机的两种可能性，程序员都要予以考虑。

```
case WM_WININICHANGE:                                    // Change to WIN.INI?
    if ((lstrcmp((LPSTR)lparam,"windows")) || (lparam == NULL))
    {
        if (! DefaultPrinter())
            MessageBox(hWnd,"No printer found",NULL,MB_OK);
    }
    break;
```

例子程序具有包含两个选项的菜单，用户可以通过它进行选择要显示的对话框。对于公用对话框，PRINTDLG 结构的某些域必须首先填充。

hDevNames 和 hDevMode 被设置为空，以便在屏幕上同时显示对话框。由于 PrintDlg 函数可同时调用两个与打印机有关的对话框，因此程序中必须给出 PD_PRINTSETUP 标志值。

```
case IDM_STANDARD:
    pd.hDevMode = NULL;
    pd.hDevNames = NULL;
    pd.Flags = PD_PRINTSETUP;
    pd.nCopies = 1;
    PrintDlg((LPPRINTDLG)&pd);
    break;
```

如果不考虑当前缺省打印机设置，公用对话框一般来说都是相同的。相比之下，第二个菜单选项并不总是产生同样的打印机设置，因为当前任何打印机驱动程序所提供的对话框都是活动的。

另外，还应考虑到个别打印机的特性。扩展名 .DRV 必须加到驱动程序名后，以使 LoadLibrary 函数调用。如果返回值大于 32，表明调用不成功。若返回正值，则可以使用函数

名和合适的序号调用 GetProcAddress 函数，从而获得驱动程序中 DevcieMode 函数的地址。

在获取 DeviceMode 的地址后，程序可采用四个参数间接调用它。这四值定义为窗口句柄、打印机驱动程序的动态链接库、打印机名和驱动程序名。

通过这个调用，驱动程序才能显示与打印机设置有关的对话框。

```
case IDM_DEVMODE:
    lstrcyp(szDriverFile,lpPntDriver);
    lstrcat(szDriverFile,".DRV")
    hDriver = LoadLibrary(szDriverFile);
    if (hDriver < 32)
        return(FALSE);
    lpfnDevMode = GetProcAddress(hDriver,"DEVICEMODE");
    if (lpfnDevmode ! = NULL)
        ( * lpfnDevMode)(hWnd,hDriver,lpPrinter,lpPntrInterface);
    FreeLibrary(hDriver);
    break;
```

4.3.4 ExtDeviceMode 函数

在打印机驱动程序中，除 DeviceMode 外还要执行一些函数，如 DeviceMode 一样，只有成功地调用 LoadLibrary 和 GetProcAddress 后，才能引用该函数。如 DeviceCapabilities 函数，允许程序检查某些特殊打印机的详细情况，如版本号和打印纸的最大尺寸等。本节主要讨论 DeviceCapabilities 和 DeviceMode，这两个函数都涉及到 DEVMODE 结构，该结构前面已经讨论过。

为调用 ExtDeviceMode，必须有一个为 Windows 3.x 版本写的打印机驱动程序。ExtDeviceMode 函数的功能比 DeviceMode 强，如果程序员引用的驱动程序有这种函数，那么可以按与 DeviceMoe 相同的方式获得地址。

```
case WM_DEVMODECHANGE:
    lstrcpy(szDevMode,(LPSTR) lparam);
    bDevMode = TRUE;
    InvalidateRect(hWnd,NULL,TRUE);
    break;
```

头文件 PRIVINIT.H 包括 ExtDeviceMode 函数和指针函数的原型。

```
LPFNDEVMODE lpfnDevMode;

hDriver = LoadLibrary(szDriverFile);
if (hDriver < 32)
        return(FALSE);
lpfnDevMode = GetProcAddress(hDriver,"EXTDEVICEMODE");
lpfnDevMode = GetProcAddress(hDriver,PROC_EXTDEVICEMODE);
```

除了 DeviceMode 函数所需的四个参数(窗口句柄、驱动程序句柄、设备名和端口)外，该函数还需要四个新参数，从类型定义可知，有两个指向 DEVMODE 结构的指针。

在新的打印机驱动程序中调用 DeviceMode 实际上是激活了 ExtDeviceMode 函数的变体，下面调用实际与调用 DeviceMode 完全相同。

```
lpfnDevMode (hWnd,hDriver,NULL,lpPrinter,lpPort,NULL,NULL,
    DM_IN_PROMPT|DM_OUT_DEFAULT)
```

在标志 DM_IN_PROMT 的基础上，程序显示熟悉的对话框供用户改变设置。但是程序

还需 DM_OUT_DEFAULT，以便在对话框关闭之后，将 WM_DEVMODECHANGE 消息发送给所有主窗口，同时更新 WIN. INI 文件的输入。

如果在显示对话框之前就已经改变了当前打印机环境，则程序需要包含新值的 DEVMODE 结构和 DM_IN_BUFFER 标志，而结构中与头标信息有关的五个域则必须被初始化。

```
DEVMODE dmIN;

lstrcpy(dmIn.dmDeviceName,lpPrinter);
dmIn.dmSpecVersion = DM_SPECVERSION;
dmIn.dmDriverVersion = 0;
dmIn.dmDriverExtra = 0;
dmIn.dmSize = sizeof(DEVMODE);
```

dmFields 域指定了将要改变的参数。例如，如果这些参数只是一些复制份数和调整因子，dmFields 将按下述方式设置。

```
dmIn.dmFields = DM_COPIES|DM_SCALE;
```

要显示的新值被传给合适的域，其他域则被忽视了，因为指示符未在 dmFields 中定义。

```
dmIn.dmCopies = 20;
dmIn.dmScale = 80;
```

下面将调用 ExtDeviceMode 函数：

```
lpfnDevMode(hWnd,hDriver,NULL,lpPrinter,
    lpPort,(LPDEVMODE)&dmIn,
    NULL,DM_IN_PROMPT| DM_IN_BUFFER)
```

当然，一般情况下，程序员总想保存用户的选择以便程序使用，进而启动程序。要达到这一目的，必须设置 DM_OUT_BUFFER 标志，并提供另一个 DEVMODE 结构接收新的设置。

```
DEVODE dmIn,dmOut;

wResult = lpfnDevMode(hWnd,hDriver,(LPDEVMODE)&dmOut,lpPrinter,lpPort,
    (LPDEVMODE)&dmIn,NULL,DM_IN_PROMPT| DM_IN_BUFFER |
    DM_OUT_BUFFER);
```

一旦指定了标志 DM_IN_PROMPT，程序员便可以接收返回值 IDOK 或 IDCANCEL，当然这依赖于用户选择的是哪个按钮。第 7 个参数过去始终为空值，现在程序员可以用它来指定一个不同于 WIN. INI 的初始化文件。

文件指定后，所有设置的读写都面向该文件，而不是 WIN. INI。如果程序员手头上有一段 Windows 3. 0 代码，则可看到，作为最后一个参数传递的四种标志的名字与 Windows 3. 1 程序不同。

下面对两种版本在这方面的差异作了一些比较。

Windows 3. 0 标志	Windows 3. 1 标志
DM_MODIFY	DM_IN_BUFFER
DM_PROMPT	DM_IN_PROMPT
DM_COPY	DM_OUT_BUFFER
DM_UPDATE	DM_OUT_DEFAULT

在 Windows 3. 0 中，程序也可用公用对话框代替 ExtDeviceMode 进行打印机设置，这和

上例类似。

程序员还可以将指示符 PD_RETURNDEFAULT 传递给 PRINTDLG 结构的 Flags 域，同时返回一个填充后的 DEVMODE 结构。

如果一个全程变量包含 DEVMODE 结构，而且已经有初始值，则程序员可以用 hDevMode 域来解释它。

4.3.5 ExDeviceMode 函数应用举例

ExtDeviceMode 可用于将某些初始值传递给打印机设置对话框，这些值都来源于用户。程序必须通过再次调用才能处理和显示用户所做的改动。

当用户按下 OK 按钮时，这些设置仍可写到主窗口中。

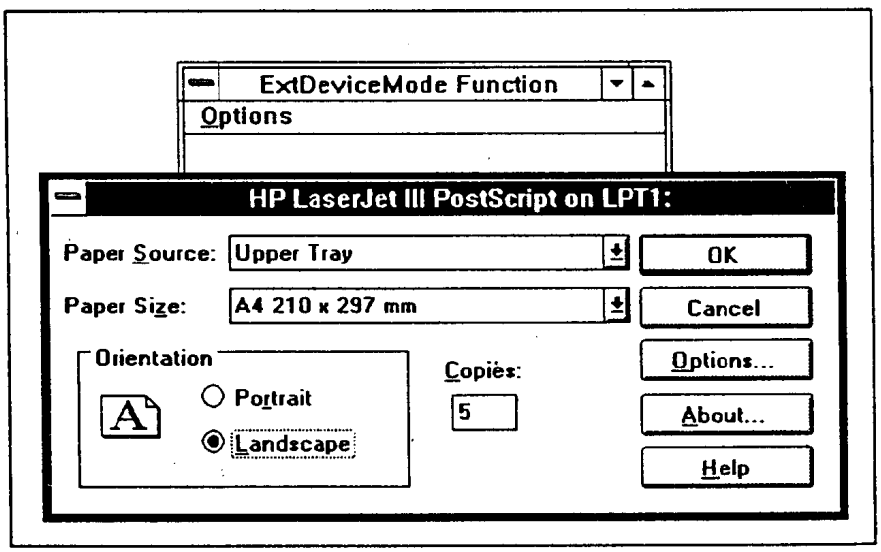


图 4.13 打印机驱动程序中 ExtDeviceMode 函数举例

新函数	描述
ExtDeviceMode	DeviceMode 函数的升级版本

源代码：EXTDEVM. C

```
/* * EXTDEVM. C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates access to external device mode * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "extdevm. h"                                    // Include EXTDEVM. H header file

#if _MSC_VER >= 700                                      // Running Microsoft C 7. 00 or
#pragma warning (disable:4028)                           // higher? Disable warning 4028
#endif

/* * WinMain (function for every Windows application * * * * * * * * * /
```

```
int PASCAL WinMain(HANDLE hInstance,                    // Current instance
        HANDLE hPrevInstance,                           // Previous instance
        LPSTR lpszCmdLine,                              // Long ptr to string after
                                                        // program name during execution
        int nCmdShow)                                   // Specifies the application
                                                        // window's appearance
{
    MSG msg;                                            // Message variable
    HWND hWnd;                                          // Window handle
    int nError;                                         // Error integer

    strcpy(szAppName,"ExtDevM");                        // Application name
    hInst = hInstance;                                  // Instance handle

    if (! hPrevInstance)                                // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                                        // If registration fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
            return nError;
        }
    }
/** Specify appearance of application's main window *********/

    hWnd = CreateWindow(szAppName,                      // Window class name
        "ExtDeviceMode Function",                       // Window caption
        WS_OVERLAPPEDWINDOW,                            // Overlapped window style
        CW_USEDEFAULT,0,                                // Default upper-left x,y pos.
        CW_USEDEFAULT,0,                                // Default initial x,y sizes
        NULL,                                           // No parent window
        NULL,                                           // Window menu used
        hInstance,                                      // Application instance
        NULL);                                          // No creation parameters

    if(hWnd == NULL)                                    // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,            // Then display error window
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );                        // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while (GetMessage(&msg,NULL,0,0))                   // Message reading
    {
```

```
        TranslateMessage(&msg);                                  // Message translation
        DispatchMessage(&msg);                                   // Send message to Windows
    }
    return (int)msg.wParam;                                      // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)                         // Window class registration
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                            // Main window class

    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );              // Mouse cursor
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);             // Default icon
    wcClass.lpszMenuName = (LPSTR)"SettingMenu";                 // Menu name
    wcClass.lpszClassName = szAppName;                           // Window class
    wcClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); // White background
    wcClass.hInstance = hInstance;                               // Instance
    wcClass.style = CS_VREDRAW | CS_HREDRAW;                     // Horizontal and vertical
                                                                 // redraw of client area
    wcClass.lpfnWndProc = WndProc;                               // Window function
    wcClass.cbClsExtra = 0 ;                                     // No extra bytes
    wcClass.cbWndExtra = 0 ;                                     // No extra bytes

    if (! RegisterClass( &wcClass ) )                            // Register window class
        return FALSE;                                            // Return FALSE if registration fails
    return TRUE;                                                 // Return TRUE
}

/* * WndProc * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL WndProc(HWND hWnd,
        unsigned message,
        WORD wParam,
        LONG lParam )
{
    HDC hDC;                                                     // Device context handle
    PAINTSTRUCT ps;                                              // Paint structure
    HANDLE hDriver;                                              // Driver handle
    static char szDriverFile [16];
    static BOOL bDevMode = FALSE;
    static DEVMODE dmOut;
    DEVMODE dmIn;
    char szDevMode[30];
    WORD wDevResult;
    LPFNDEVMODE lpfnDevMode;

    switch (message)
    {
```

```
case WM_CREATE:                                          // Create window
    dmOut.dmOrientation = DMORIENT_LANDSCAPE;            // Landscape format
    dmOut.dmCopies = 5;                                  // Number of copies
    dmOut.dmPaperSize = DMPAPER_A4;                      // DIN A4 paper size
    break;
case WM_COMMAND:                                         // Commands
    switch(wParam)
    {
    case IDM_DEVMODE:
        if (! DefaultPrinter())
            MessageBox(hWnd,"No printer found",NULL,MB_OK);
        lstrcpy(szDriverFile,lpPntDriver);
        lstrcat(szDriverFile,".DRV") ;
        hDriver = LoadLibrary(szDriverFile);
        if (hDriver < 32)
            return(FALSE);
        lpfnDevMode = (LPFNDEVMODE)GetProcAddress(hDriver)
        if (lpfnDevMode ! = NULL)
        {
            memset(&dmIn,0x00,sizeof(DEVMODE));
            lstrcpy(dmIn.dmDeviceName,lpPrinter);
            dmIn.dmSpecVersion = DM_SPECVERSION;
            dmIn.dmDriverVersion = 0;
            dmIn.dmDriverExtra = 0;
            dmIn.dmFields = DM_ORIENTATION | DM_COPIES | DM_PAPERSIZE;
            dmIn.dmSize = sizeof(DEVMODE);
            dmIn.dmOrientation = dmOut.dmOrientation;
            dmIn.dmCopies = dmOut.dmCopies;
            dmIn.dmPaperSize = dmOut.dmPaperSize;

            wDevResult = lpfnDevMode(hWnd,                // List device result
                hDriver,
                (LPDEVMODE)&dmOut,
                lpPrinter,
                lpPntrInterface,
                (LPDEVMODE)&dmIn,
                NULL,
                DM_IN_PROMPT|
                DM_IN_BUFFER |
                DM_OUT_BUFFER);
            if (wDevResult == IDOK)
                bDevMode = TRUE;
            else
                bDevMode = FALSE;
            InvalidateRect(hWnd,NULL,TRUE);
        }
        FreeLibrary(hDriver);
        break;
    default:
        return (DefWindowProc(hWnd,message,
            wParam,lParam));
```

```
                break;
            }
            break;
        case WM_PAINT:                                          // Paint structure
            hDC = BeginPaint(hWnd,&ps);
            if (bDevMode)
            {
                wsprintf(szDevMode,"Number of copies = %d",dmOut.dmCopies);
                TextOut(hDC,10,10,szDevMode,strlen(szDevMode));
                lstrcpy(szDevMode,"Paper format = ");
                if (dmOut.dmOrientation==DMORIENT_PORTRAIT)
                    lstrcat(szDevMode,"Portrait");
                else
                    lstrcat(szDevMode,"Landscape");
                TextOut(hDC,10,30,szDevMode,strlen(szDevMode));
                wsprintf(szDevMode,"Paper Size = %d",dmOut.dmPaperSize);
                TextOut(hDC,10,50,szDevMode,strlen(szDevMode));
            }
            EndPaint(hWnd,&ps);
            break;
        case WM_DESTROY:                                        // Exit
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,
                    message,
                    wParam,
                    lParam ));
            break;
    }
    return(0L);
}

BOOL DefaultPrinter(void)                                       // Default printer
{
    static char pPrintInfo[80];
    LPSTR lpTemp;

    if (! GetProfileString("windows","Device",(LPSTR)"",pPrintInfo,80))
        return (FALSE);
    lpTemp = lpPrinter = pPrintInfo;
    lpPntrInterface = lpPntDriver = 0;
    while ( *lpTemp)
    {
        if ( *lpTemp == ',')
        {
            *lpTemp++ = 0;
            while ( *lpTemp == ' ')
                lpTemp++;
            if (! lpPntDriver)
                lpPntDriver = lpTemp;
```

```
            else
            {
                lpPntrInterface = lpTemp;
                break;
            }
        }
        else
            lpTemp++;
    }
    return (TRUE);
}
```

模块定义文件:EXTDEVM. DEF

```
NAME ExtDevM
DESCRIPTION 'ExtDeviceMode Function'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS WndProc @1
```

资源描述:EXTDEVM. RC

```
#include "extdevm. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW,"Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END

SettingMenu MENU
BEGIN
    POPUP "&Options"
    BEGIN
        MENUITEM "&ExtDeviceMode…",IDM_DEVMODE
    END
END
```

头文件:EXTDEVM. H

```
#include <windows. h>
#include <string. h>
#include <drivinit. h>

#define IDS_ERR_CLASS 20
```

```
#define IDS_ERR_WINDOW 21
#define IDM_DEVMODE 101

char szString[128];                                    /* Variable for loading resource text */
char szAppName[20];                                    /* Window class name */
HANDLE hInst;

LPSTR lpPrinter;
LPSTR lpPntDriver;
LPSTR lpPntrInterface;

int nCwRegisterClasses(HANDLE);
BOOL DefaultPrinter(void);
long FAR PASCAL WndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:EXTDEVM. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = EXTDEVM.H

all: EXTDEVM.EXE

EXTDEVM.RES: EXTDEVM.RC $(RESFILES)

EXTDEVM.OBJ: EXTDEVM.C $(H)

EXTDEVM.EXE: EXTDEVM.OBJ EXTDEVM.RES EXTDEVM.DEF
    link /NOD/A:16 EXTDEVM,EXTDEVM,nul,libw slibcew,EXTDEVM.def
    rc EXTDEVM.res EXTDEVM.exe
```

按下述方式运行 NMAKE;创建 EXTDEVM 应用程序。

```
nmake extdevm.mak
```

如果用户要执行此程序,必须清楚所用的打印机驱动程序是否包含 ExtDeviceMode 函数,例如,"Apple Laserwriter II NIX"打印机的驱动程序 DSCRIPT. DRV 程序员可通过它进行检测。

本例的应用程序来自程序 DEVMODE。一旦使用 ID 值 IDM_DEVMODE 选择了菜单项,屏幕上就马上显示出对话框,虽然目前它尚未被 ExtDeviceMode 激活。

第一次调用后,复制份数域包含值 5,Orientation 被设置为风景格式,打印纸尺寸规格为 DIN A4。程序利用 WM_CREATE 消息把这些值写到 DEVMODE 类型的变量 dmOut 中,并作为输出缓冲区。

```
case WM_CREATE:
    dmOut.dmOrientation = DMORIENT_LANDSCAPE;
```

```
        dmOut.dmCopies = 5;
        dmOut.dmPaperSize = DMPAPER_A4;
        break;
```

在调用 ExtDeviceMode 函数之前，设置参数由输出缓冲区写到输入缓冲区，以便显示对话框中上次输入的值 dmIn。dmFields 域定义了所包含的属性。另外，还必须提供五个头标信息域的值。

```
memset(&dmIn,0x00,sizeof(DEVMODE));
lstrcpy(dmIn.dmDeviceName,lpPrinter);
dmIn.dmSpecVersion = DM_SPECVERSION;
dmIn.dmDriverVersion = 0;
dmIn.dmDriverExtra = 0;
dmIn.dmFields = DM_ORIENTATION | DM_COPIES | DM_PAPERSIZE;
dmIn.dmSize = sizeof(DEVMODE);
dmIn.dmOrientation =dmOut.dmOrientation;
dmIn.dmCopies = dmOut.dmCopies;
dmIn.dmPaperSize = dmOut.dmPaperSize;

wResult = lpfnDevMode(hWnd,hDriver,(LPEDVMODE)&dmOut,lpPrinter,lpPort,
(LPDEVMODE)&dmIn,NULL,DM_IN_PROMPT| DM_IN_BUFFER | DM_OUT_BUFFER);
if (wResult == IDOK)
    bDevMode = TRUE;
else
    bDevMode = FALSE;
InvalidateRect(hWnd,NULL,TRUE);
```

用户使用 OK 按钮关闭对话框以后，三个可修改的值应在主窗口中显示。这样，应在 WM_PAINT 消息中赋值的 Bool 变量才会被设置为 TRUE 值。

第五章　TrueType 字模

5.1　TrueType 字模概述

过去的 Windows 版本(3.0 版以下)都使用两种类型的字模:光栅字模和向量字模,但这两种字模类型都有一定的缺陷。

在 Windows 中,向量字模的显示是很慢的,而且没有特别引人之处。光栅字体是按位图存储的,放大时容易变得粗糙,因为它们是按某种给定的分辨率设计的。而且,如果需要特定属性的字模,如黑体、斜体等,GDI 模块就必须采用合成的方式。

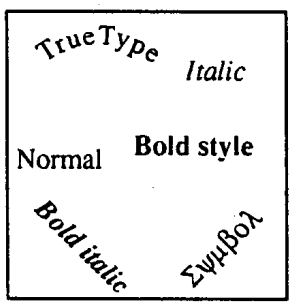


图 5.1　不同字模的字母显示

Windows 3.1 的 TrueType 字模集包括正常字模、黑体、斜体、黑斜体。每种字模都存在各自的文件中,而且这种字模比老版本的字模更容易转换。

5.1.1　TrueType 的来源

TureType 是 Apple 公司和 Microsoft 公司合作开发的结果。最初,开发者只是想打破 PostScript 的发明者 Adobe 在该邻域的垄断地位,因为在推出 TrueType 字模之前,凡是想提供高质量屏幕或打印机字模的公司都不得不付给 Adobe 版权费。

为打破这种垄断地位,Apple 公司计划发展其自己的字模(命名为 Royal 字体),并把 Royal 字模的格式对开发者公开。

在 Microsoft 购买了 PostScript 的版权后,Apple 和 Microsoft 在开发新字模上达成一致。目标是开发一种与 PostScript 兼容的字模语言,并且输出要比现行的高质量屏幕和打印字模快。

TrueType 技术的思路是相当优秀的,因为其应用领域面向大众。任何一个字模设计者

都可以使用 TrueType 格式开发其自己的字模。

5.1.2 TrueType 字模的优点

TrueType 字模比起早期的 Windows 字模来说，有很多优越之处，它们可以任意缩放，但其尺寸始终保持合适的比例和任意弯曲度。

打印机和屏幕可使用同种字模，屏幕字模甚至可以旋转。它们的显示机制也是一致的，而不考虑到文档类型、打印机或其他硬件设备的配置。

通过控制面板，可以在一个列表框中显示已装入的所有字模。一旦选择了 TrueType 字模，在其后将显示样本正文，因为字模对屏幕和打印机都是合法的，所以仅显示一行。

其他的字模也可以按字模文件的定义以任意尺寸显示。

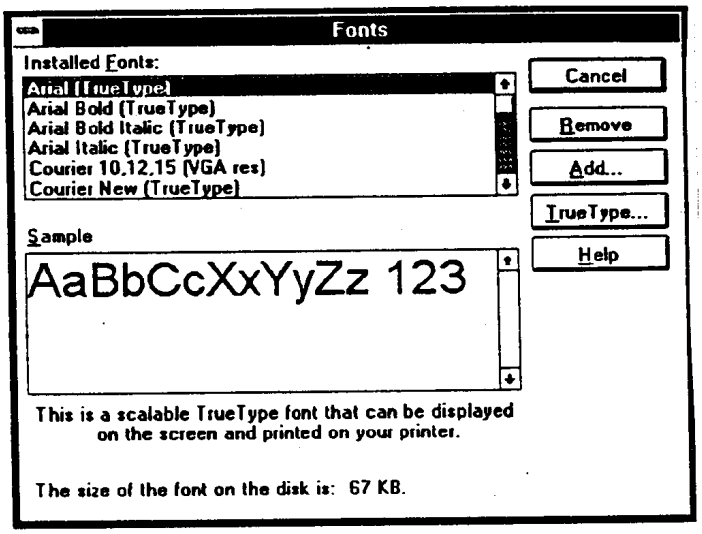


图 5.2

如果用户按下了 TrueType 按钮，表明用户已有权使用 TrueType 字模，或关闭该字模的使用权；同时也能指定只能在应用程序中显示唯一一种 TrueType 字体。这在某种程度上给了用户更优越的进行字模选择的控制权。

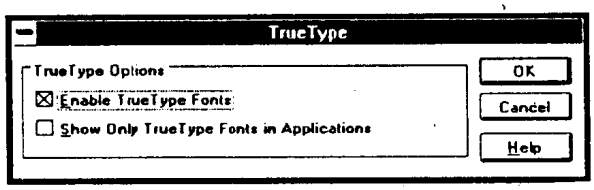


图 5.3

5.1.3 WIN.INI 中的[TrueType]部分

用户激活的设置项一般在对话框关闭之后存储在初始化文件 WIN.INI 中。[TrueType] 部分的输入格式如下：

[TrueType]

nonTTCaps=0
TTEnable=1
TTOnly=0
TrueType files

Windows 有 13 个 TrueType 字模文件，包括下面几种字模类型：

字模集	字模名	字模类型
Arial	Arial Bold	Sans-Serif, proportional
	Arial Italic	Sans-Serif, proportional
	Arial Bold Italic	Sans-Serif, proportional
Courier New	Courier New Bold	Serif, non-proportional
	Courier New Italic	Serif, non-proportional
	Courier New Bold Italic	Serif, non-proportional
Symbol	Symbol	No variants
TimesNewRoman	TimesNewRoman Bold	Serif, proportional
	TimesNewRoman Italic	Serifen, proportional
	TimesNewRoman Bold Italic	Serifen, proportional

对于 TrueType 字模集，Windows 并没有综合出特别的属性。因为，TrueType 集的每个成员都有其自身的轮廓计算方法。为确保字模中字模高度的一致性，其斜体字是真正的斜体。

图 5.4 给出了 13 种 TrueType 字模的例子，它们是以 10×10 点阵显示的。

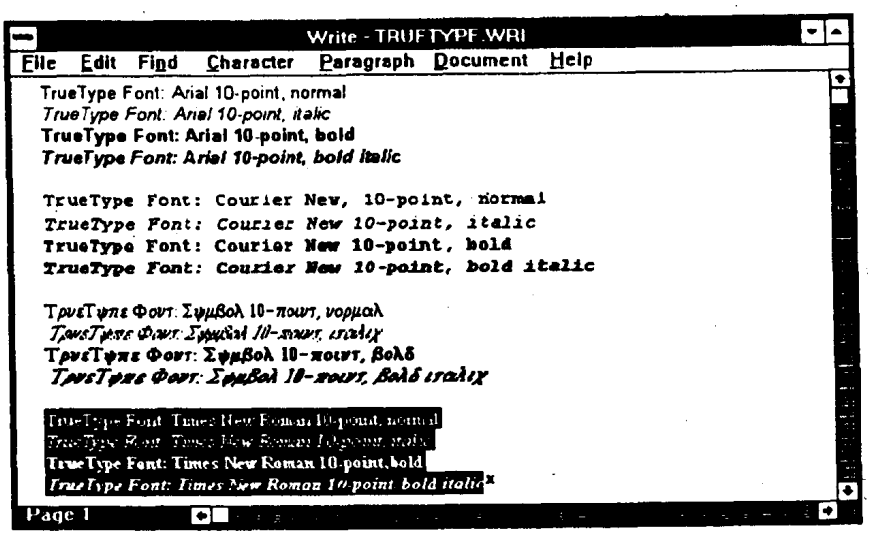


图 5.4　13 种 TrueType 字模按 10×10 点阵显示

5.2　WYSIWYG 的问题

与光栅字模相比，TrueType 字模的优点表现得更为明显。目前市场上的每种软件包几乎都标上这么一句流行语言“WYSIWYG”(What You See Is What You Get)。然而实际上，很少有达到它所说的那种境界的。

WYSIWYG 意味着打印机的输出效果和屏幕输出相同。如果真是这样，系统只需保证打

印的文档能在监视器上正常显示，因为打印机与屏幕完全匹配。但是因为屏幕和打印机有不同的分辨率，即使是 TrueType 字模，要达到完全匹配也是不大可能的。

按一般用法，WYSIWYG 可以用于屏幕输出和打印机输出时的行中断和页中断，当然必须首先保证二者的行数、页数相同。要达到这一点，必须先计算一行所占的空间，然后调整两字符间距。这样，可以减少某些合入误差，譬如每个字符半个像素及每行半个像素。

虽然有上面的处理方式，但过去的 Windows 字模却从未将 WYSIWYG 的要求变为现实，原因在于 Windows 使用的是光栅字模，并将它们显示到屏幕上。

5.2.1 光栅字模与 TureType 对比

为了更深入地了解有关字模的技术，最好先考察一下光栅字模与 TureType 字模的差别。

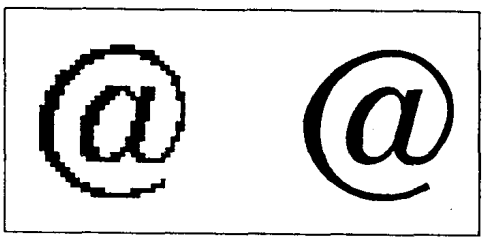


图 5.5 光栅字模与 TureType 对比

由于办公及桌面印刷系统的软件日趋复杂，在字模和文档的创建与操作方面，用户的要求越来越高。

CorelDraw 应用程序在字模的成形方面具有强大的功能。字母几乎可以任意放缩，而且屏幕和打印机的输出效果相同。也许有人会觉得奇怪，为什么一个绘图程序能提供如此大的功能，而像 Microsoft Windows Write 这样的字处理程序却不能？

这主要是因为 Write（作为 Windows 的应用程序）使用的是 Windows 提供的字模，而 CorelDraw 却有它自己的字模。到 Windows 3.0 版，Windows 都是把单个字符作为一个光栅图像显示到屏幕上（注意，上面的讨论都不包括向量字模，因为它们有某些缺陷，而且很少使用）。

每种光栅字模都可以看作由许多独立像素组成的图像，如 Paintbrush 图像（见图）。由于它是由许多固定的点（像素）构成的，因此，在放大图像时不可能没有失真，这样，每种尺寸的字模都需要有自己的光栅字符集。不过，这种方式要占用大量的存储空间，因此 Windows 可直接支持的一定尺寸字模的数目受到限制。

1. 屏幕显示对照

在 Write 中，系统给进行正文编辑的用户提供了多种尺寸的可选字模。然而，对应的字模文件中却没有包括多少，如 TimesRoman 字模文件就只提供了八点和十二点的字模。

如果在 Write 中直接使用这两种字模，屏幕显示和打印则非常接近，然而，一旦用户选择了不同尺寸的字模，Write 就必须将光栅图像放大，以便生成字符。

放大处理包括将当前光栅图像拷贝到一个合适的放大显示矩阵中。不过这样容易导致

斜轮廓线参差不齐，即会出现锯齿，也称作阶梯效应。相比之下，如果按 TrueType 格式生成一定精度的字符，无论在屏幕上或打印机上，都比较平滑。

2. 用光栅字符编辑的正文

下面的例子将使我们更清楚地了解 Windows 中的文本，它先用光栅字模显示，然后用 TrueType 字模。虽然内容相同，但从屏幕效果可以看到，两种类型字模的差别。

正文“TrueType”最初是在 Write 中使用 TimesRoman 字模生成的，没有使用任何 TrueType 字模，因为这种字模已经通过控制面板的字模集窗口锁定了。

在按下 TrueType 按钮后，系统显示 TrueType 对话框，用户清除了多余的复选框，然后再次启动 Windows。图 5.6 是用 110 点的 TimesRoman 字模生成的“TrueType”字符串。

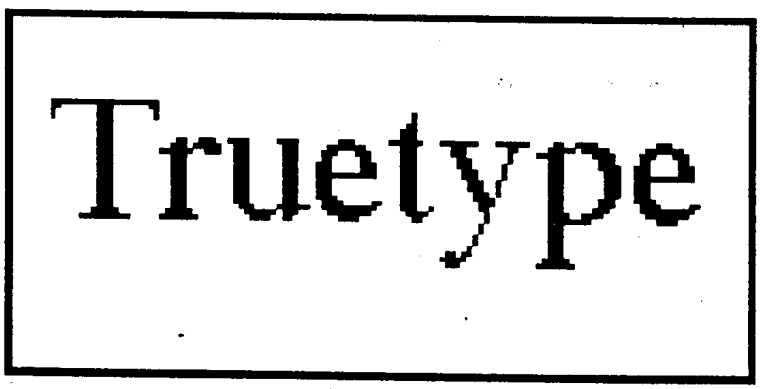


图 5.6　屏幕上的光栅字符

显然，这种字符有锯齿。下面授权使用 TrueType 字模，用 TrueType 字模 TimesRomanNew 进行同样的测试，图 5.7 显示的是输出结果。

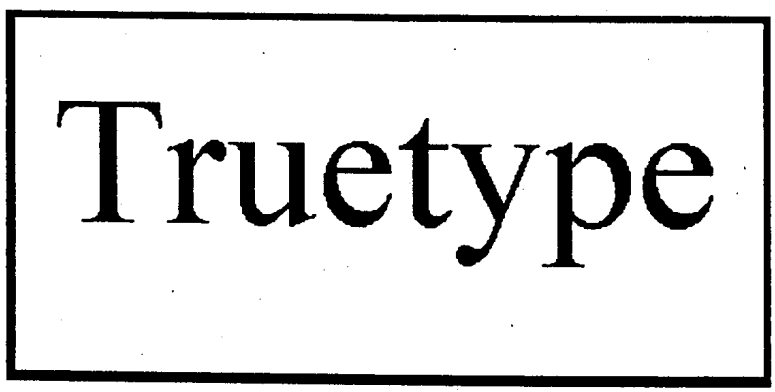


图 5.7　屏幕上的 TureType 字模

注意这两种字模在质量和尺寸上的差别。虽然它们都是在 Windows 3.1 下生成的，而且使用的都是 110 点的 TimesRoman 字模。不过这样不能说明已经满足了 WYSIWYG 的要求，下面我们将看到屏幕显示和打印机输出是如何保持一致的。

3. 打印机输出

这种测试比前面介绍的要困难一些，因为这实际上是用打印输出与屏幕显示作比较，因此需要一台 PostScript 打印机和 CorelDraw 应用程序。测试时，将 Write 正文屏幕转储按 TrueType 字模和光标字符的格式下装到 CorelDraw 中。

为说明每个正文的来源，附加上标题“Screen”和“PostScript”。接着，可在 CorelDraw 中使用 Tomonto 字模（一种 TimesRoman 字模）输入“TrueType”，并放大 CorelDraw 样本字模以匹配 TrueType 字模。

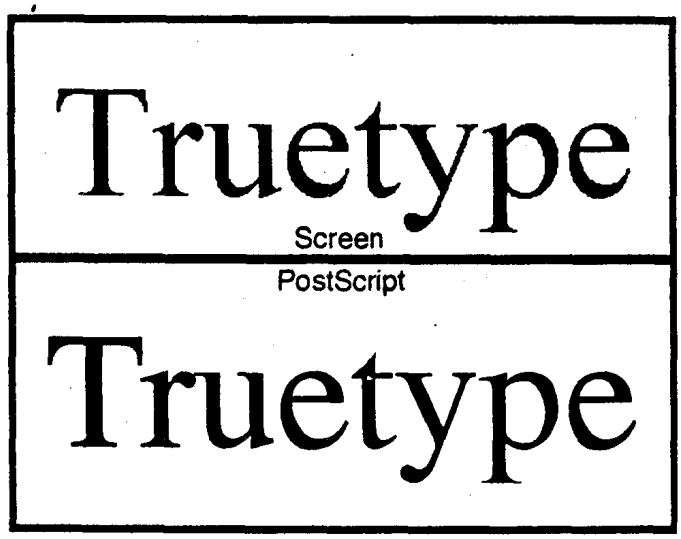


图 5.8　PostScript 打印机上输出 TrueType 字符

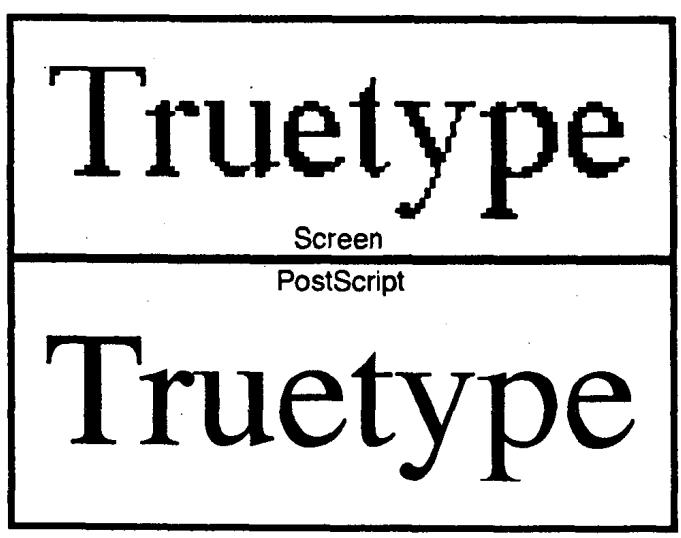


图 5.9　光栅字模的 PostScript 打印机输出

5.2.2　TrueType = WYSIWYG

屏幕和打印机的输出效果基本上是相同的，但仍有细微差别，尤其是屏幕显示时

TureType 字模的边缘仍有少量毛刺。

图 5.9 对 TrueType 字模和光栅字符做了一个比较。

5.3 计算机字模技术

5.3.1 光栅字模

目前，大多数输出设备都利用光栅图像显示字模和图形。这种图像通常被称为位图，它的计算机处理和输出都是非常快的，因为它无需进行耗时的计算。前面已经讨论了这种格式的特征，为保证图像质量，不同类型、尺寸的字模及各种输出设备都应该有自己的光栅图像。但从存储空间角度考虑，一个简单的字处理程序要达到这一点是不容易的，不过许多图形应用程序(如 CorelDraw)都有一套自己的字模。

5.3.2 向量字模

前面的测试表明，光栅字模在放大时往往会出现失真。而向量字模却较好地解决了这个问题，这种字模在过去的 Windows 版本中也提供了。

这种格式中，每个字符都是由许多直线段构成的，并且线段都用数学公式描述。许多字母可以用这种方式，但涉及到曲线处理时向量格式就不是那么令人满意了。

而且，虽然向量字模在放大时能保持其光滑度，但这对黑体字很不适合，因为形成一定宽度的笔划需要很多向量直线。

正是由于这个因素，使得 Windows 的向量字模并不被人们广泛接受。另一方面，PostScript 技术作为光栅字模和向量字模的结晶，却越来越受人喜爱。

5.3.3 轮廓字模

除了光栅字模和向量字模字，还有一种字模类型，一般被称为轮廓字模。这种格式仅仅用数学公式描述字符的外部轮廓线。它和向量字模有些接近，但却是之后填充轮廓线的内容。

而且，这种方式不仅适用于直线，也可用于由曲线生成的字符轮廓。用一句技术术语，这种连接字符所用的计算方法被称为 BeziEr 函数。

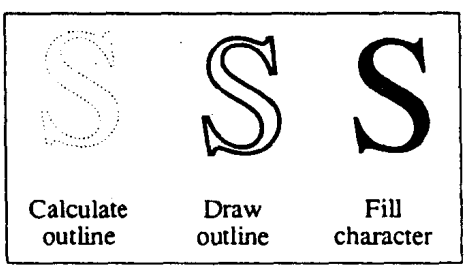


图 5.10 生成一个轮廓字模的字符

Adobe 使用这种处理方法，用命名为 Adobe Type 1 的方法在屏幕上显示了 PostScript 字

模。

5.3.4 TrueType 字模

与前面所描述的轮廓字模相比，TrueType 也描述了字符的轮廓，通常称为雕刻字，但它与 Bezier 函数无关，只是一系列轮廓线上的区域。

这些区域由一些直线段和曲线段构成。曲线都是由多个孤立点拟合而成的，通常使用两次 B 样条进行拟合计算。

拟合时，有些点在曲线上，而有些则可能落到曲线外，如图 5.11 所示。直线则通过两点的连接构成。

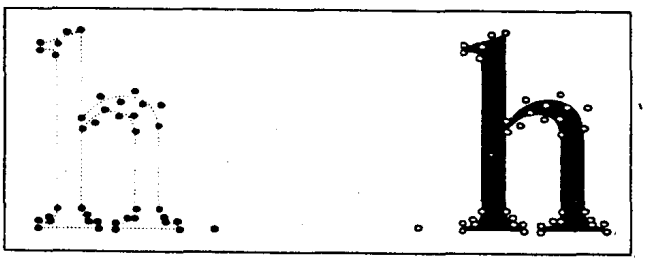


图 5.11 TrueType 字符的轮廓

字符轮廓是通过设置和连接许多离散点形成的，所有点都必须由一定的方案来计数。如果按升序计数点方向生成曲线(如图 5.12)，则被填充的区域总是在计数值的右边。

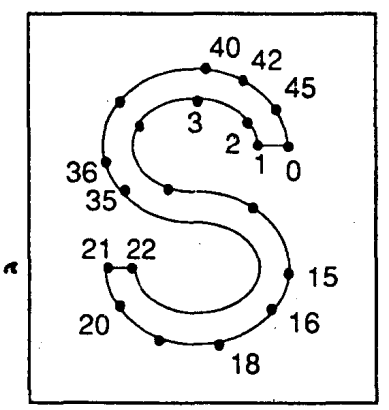


图 5.12 雕刻字的点计数

字模单元(FUnits)

TrueType 文件描述了字模单元(FUnits)中点的位置，该单元被定义为一个 em 方阵的最小可测量单元，该方阵的边界可以想象为一个二维的栅格，其内部空间用于放大或校准雕刻字。为了简化某些字符的操作，其部分轮廓也可以留在方阵之外。

栅格的右上角坐标是(0,0)，每个点的位置值都在－16384 FUnits 到＋16384 FUnits 的范围内。选择不同的分辨率，可寻址的范围还可以减少。粗细程度通常被定义为每 em 方阵(upem)所占的单元数，它是由字模设计者决定的。

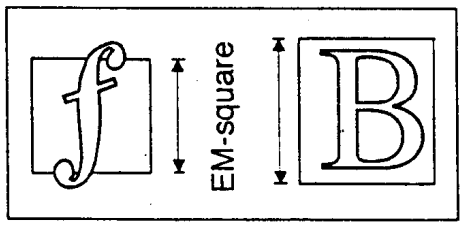


图 5.13　em 方阵

设计者同时还要确定字模单元本身的尺寸，如果可能的话，单元最好设计为能将方阵分为一定数目的区域，该数为 2 的平方。一旦满足这个条件，字模的放大计算也就方便多了。

Windows 所用的 TrueType 字符是按 2048×2048 的布局来设计的，该值通常出现在 OUTLINETEXTUETRIC 结构的 otmEMSquare 域中，而由 GetOutlineTextMetrics 函数调用。

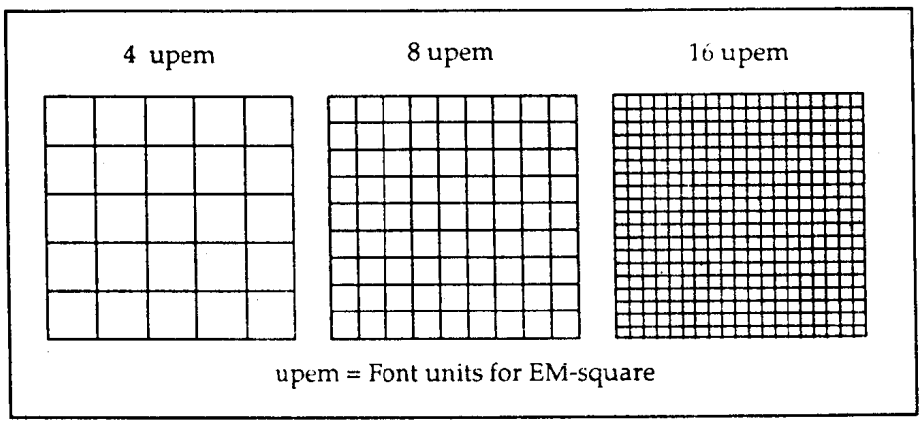


图 5.14　不同单元的 EM 方阵

由上图可以看出，字模单元(FUnit)总是与 em 方阵有关，而且没有固定尺寸。其大小随着 em 方阵的大小而变动。

一个给定字模中保持不变的量通常不是点大小，而是每个 em 方阵单元(upem)，这样安排的优点在于雕刻字的点总是具有相同的 FUnit 坐标。

例如，图 5.15 中字母 B 的左上角坐标总是(13,20)。

每英寸有多少个点

输出设备的分辨率一般由每英寸所占的点数(像素数)来决定。在一个 em 方阵中，像素的数目依赖于设备的分辨率。分辨率越高，字符的精度也越高。

当应用程序存取一个 TrueType 字模时，通常由系统内部使用轮廓线生成一个字符的位图，大小按要求给定，这种算法处理叫作线性化。

所创建的位图存储在内存中，以便在以后调用时无需重新计算即可用于显示。显示这种内存中的光栅图像和一般光栅字模一样快。

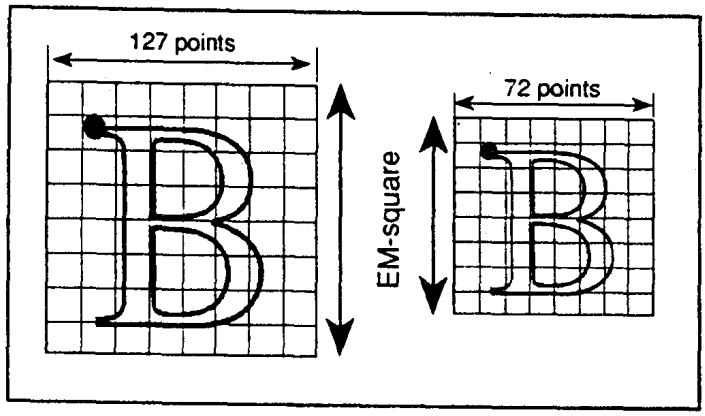


图 5.15

上面提到的算法处理必须先检查是否引入了失真因素，以确保排版时缩放字模正常。

TrueType 指令

TrueType 使用自己的语言描述和显示字符，其指令超过 100 条。与汇编指令相似，它们是通过字符放大时保持显示来体现其运行机制的。

如果要开发自己的 TrueType 字模，用户必须详细地了解这些指令。因为许多指令对生成和输出 TrueType 字符来说都是必不可少的，因此字模文件一般都很大。

创建 TrueType 字模的指南见 Microsoft 公司的 TrueType 字模规范，其字模也可用于 Macintosh。

5.3.5 文本格式

Windows 的字符包括一些可见成分，如雕刻字；同时也有许多不可见部分，如字符单元。如果使用 CreateFont 函数或 CreateFontIndirect 创建逻辑字模，则程序员必须给 lfHeight 域提供所需的尺寸。此时，正值代表字符单元的大小，负值则表示雕刻字的尺寸。

标准字模高度由点来定义，代表七十二分之一英寸。这个点的大小也就是字符单位的大小。通常当文本在屏幕上显示时，用于放大的测试单位是逻辑英寸，它可比一般的物理单位大 40%。这样，12×12 点阵的字符组成的正文在窗口显示时要比打印纸上的大，为了提高字符的清晰度，屏幕分辨率通常比打印机分辨率低。如果需要的话，具有物理英寸大小的 TrueType 字模也可用于屏幕显示，这是通过调用 GetOutlineTextMetrics 函数实现的。

字模宽度

创建逻辑 TrueType 字模时，必须在 LOGFONT 结构的 ifWidth 域中指定所需的字体宽度值，该值与高度值完全无关，因而设计时无需改变高度就可改变字体。

前导和尾缀

“前导”通常用于表示两种测试值，一种是字符单元基线至上边的距离；另一种叫印刷前导，代表最高的字母，在 TrueType 字模中通常是小写“f”。

与此类似，“尾缀”意味着字母在基线之下的部分或者字模中最短的字母，一般是小写“g”。这两个印刷体值都通过调用 GetOutlineTextMetrics 函数获得。

该函数返回结构 OUTLINETEXTMETRIC，它包括 otmAscent 和 otmDescent 域。

如果程序员过去用过函数 GetTextMetrics 和 TEXTMETRIC 结构，就可以熟悉一下其他的正文格式，如 tmInterralLeading 域，它指定了雕刻字上部边缘线间的距离和字符单元的大小。

图 5.16 显示了部分格式的例子，前缀“tm”代表该域在 TEXTMETRIC 结构中定义，而“otm”则表示 OUTLINETEXTMETRICS 中定义了该域。

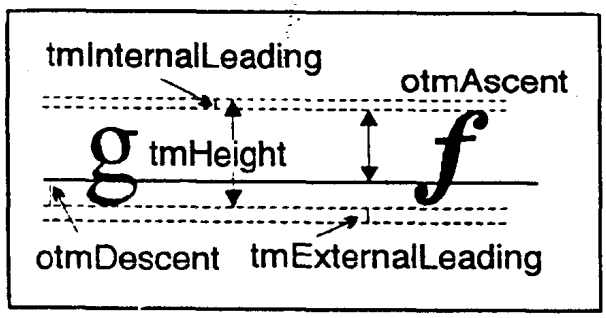


图 5.16

ABC 宽度

为校定字符宽度，TrueType 字模使用了一个叫作 ABC 字符分隔处理的方法，这比仅用字符单元宽度要精确得多。

不过，这可能带来一些麻烦，尤其当老的 Windows 程序使用 TrueType 字模重复正文输出的时候，问题就更为突出。如果仅使用正常宽度，同行中前一个雕刻字的右边位置可能出错，该行的起始位置也可能发生偏移。

其结果是在处理 WM_PAINT 消息时，新的输出正文无法直接覆盖旧的输出。

新的 ABC 字符分隔方法不仅包括字符宽度，雕刻字(B)宽度，而且包括了(A)之前和(C)之后的字符的空间，全部宽度是三个测量值的和。

A 和 C 的宽度也可假设为负值，因为严格地讲它们定义的是视觉上的字符边界。这些附加值允许程序员提供给用户一个更好的正文图像。

图 5.17 显示的是小写“f”和“g”之间的 ABC 宽度，它通常用于 TrueType 字体的排版规范。

小写字母“g”的三个宽度都是正值，而“f”的 A，C 值都为负，这些值可通过 GetCharABCWidths 函数和 ABC 结构获得，该函数仅用于 TrueType 字模。

5.4 其他的 Windows 字模特征

除了 TrueType 字模外，Windows 3.1 还有一些与字模有关的特征。在讨论 TrueType 字模装入之前，我们先对这些特征作一些说明。

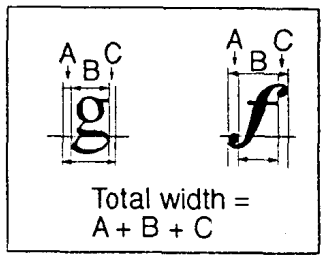


图 5.17　小写“g”和“f”的 ABC 宽度

特征字符的映像

如果要使用字处理程序编辑正文，而且该正文包括一些特征字符(如数学公式符号)，这肯定会给程序员带来一些麻烦，因为键盘上找不到这些字符。

然而在 Windows 中可以按 Alt 和一些数字代码的组合键输入这种字符。该过程需要使用一个字符表作为数字代码与特征字符的映像。

Windows 3.1 提供了一种字符映像解决这个问题，应用程序可以显示任何一个变体字符集，并允许程序将字符转移到文档中去。

当前所选字符的数字代码同样还在字符映像窗口的底端显示。

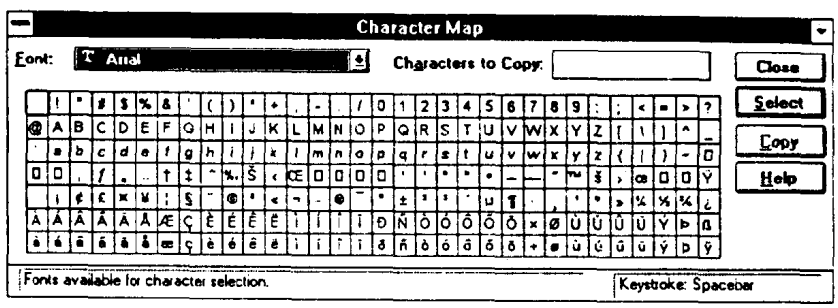


图 5.18　字符映像

当用户通过鼠标选择了所需字符时，窗口中会显示一个放大的字符图像。用户可以再按一下鼠标或按下 Select 按钮将字符移到要拷贝的字段中。

在用 Copy 命令将字符放到剪贴板之前，用户可以按这种方式选择单个字符或多个字符。然后，通过剪贴板将字符插入到 Write 文档中。Windows 及其他应用程序都能访问剪贴板。

WIN.INI 中的新说明部分

光栅字模过去称之为“Tms Rmn”和“Hew”，新版本则更名为“MS Serif”和“MS Sans Serif”，但保留了过去的名称，使用旧名的应用程序无需修改。在 WIN.INI 文件中增加了新的说明部分，称为[FontSubstitutes]，它包括下列内容：

[FontSubstitutes]
Helv=MS Sans Serif
Tms Rmn=MS Serif
Times=Times New Roman
Helvetica=Arial

当应用程序申请"Helv"字模时，Windows 字模管理程序引用该说明部分，并用"MS Sans Serif"自动换名。另外，在调用 EnumFonts 和 EnumFontFarilies 函数时也引用了这段说明。

5.4.1 新的光栅字模集

虽然从理论上讲，TrueType 字模可减少到 8×8 点阵以下。但在这种情况下最好还是使用 Windows 3.1 提供的新的小号字模。这些字模字母非常容易读取，虽然它们都很小，同时相应的字模文件都以"Small"开头，如 SMALLLF.FON。

5.4.2 TrueType 字模构成的新字符集

TrueType 字符集

虽然 TrueType 字模的输出质量很高，但与过去的 Windows 和 DOS 相比，其字符集大得多。TrueType 字符集总共包括 332 个字符，有趣的是，TrueType 字符集与 Apple 的类似，但却将 Apple 的 logo 字符排除在外。

装入 TrueType 字模

如果硬盘有足够的空间，Windows 随时都能装入多达 1170 个 TrueType 字模，不过 GDI 却强加了一些限制。

选择一个 TrueType 字模

如果某些字模同名(如 Courier)，那么 Windows 就首先选择 TrueType 字模，因为 Microsoft 公司建议只要有可能就使用 TrueType 字模。

如果调用 SelectObject 函数时需要逻辑字模的最佳物理近似值，字模管理程序通常也选择 TrueType 字模。在进行这一处理时，它按下面的几种情况进行管理：

- 如果所需字模名不在字模表中，那么系统将按所需尺寸、衬线和比例用 TrueType 字模替换。
- 如果字模名指的是光栅字模，但却不在所需的点阵尺寸之列，则用该尺寸的 TrueType 字模替换。
- 如果字模名指的是具有黑体或斜体属性的光栅字模，或者是多种属性的综合体，系统也用同属性的 TrueType 字模替换它。

为保持兼容性，在涉及到创建字库这个问题时，用户现有的应用程序无需修改，因为构成 CreateFont 函数的许多子例程都没有什么改动。

TrueType 和打印机

TrueType 字模能在任何打印机上打印，只要 Windows 有该打印机的驱动程序。GDI 模块

将 TrueType 字符的光栅图像或已填充了的多边图形发送到打印机上。如果打印机驱动程序(如 PSCRIPT. DRV)可驱动 Windows 的 PostScript 打印机,那么该驱动程序将把字符转换成为 PostScript 格式,它使用的是 Bezier 曲线。

5.5 利用 TrueType 字模编程

Windows 3.1 设计了一些新的函数和结构实现 TrueType 字模的各种操作。本节讨论这些函数,并给出 TrueType 的应用实例。

5.5.1 新的函数和结构

GetTextMetrics 函数在早期的 Windows 版本中也存在,它用于确定所指定的字模是否为 TrueType。如果是,则将 TEXTMETRIC 结构的 tmPitchandFamily 域的第二位置为 1。

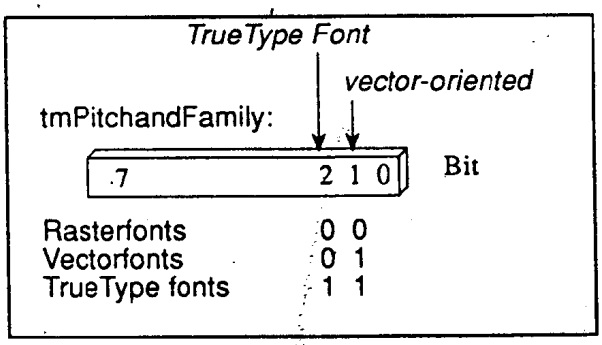


图 5.19 TEXTMETRIC 结构中指定的 TrueType 字模

因为 TEXTMETRIC 和 LOGFONT 结构在许多方面都很相似,因此后者的 lfPitchandFamily 域也能用于 TrueType 字模。设置该域的第二位可以确保由 CreateFont 或 CreateFontIndirect 创建的字模一定是 TrueType 字模。

GetRasterizerCaps 函数

在将 TrueType 字模用于程序中之前,必须先调用 GetResterizerCaps 函数以确定 TrueType 字模是否能用,起码要判断出是否已装入了 TrueType 字模。该函数有两个参数,第二个参数指向结构 RASTERIZER_STATUS,该结构将填充调用结果信息。

```
typedef struct _RASTERIZER_STATUS
{
    short nSize;
    short wFlags;
    short nLanguageID;
} RASTERIZER_STATUS;
```

函数执行后,检查 wFlags 域的 TT_AVAILABLE 或 TT_ENABLED 以确定当前状态。

```
Number = sizeof(RASTERIZER_STATUS);
GetRasterizerCaps(&rStatus,Number);
if rStatus. wFlags == (TT_AVAILABLE | TT_ENABLED))
```

```
{
    //TrueType fonts can be used
}
```

CreateScalableFontResource 函数

由于 Windows 不能直接解释原始的 TrueType 文件(TTF)格式,因此必须增加一些附属文件,它仿校 .FON 文件并具有 .FOT 扩展名;每个 FOT 文件都有一个简短的头文件说明该文件,它包括许多有用的 TrueType 字模信息。这种文件与所说明的文件同名,不过扩展名为 .TTF。

CreateScalableResource 函数最早是由 Windows 的 Setup 程序使用的,它创建一个 FOT 文件,并指向所需的 TrueType 文件。然后这种 TrueType 字模文件按正常方式由 AddFontResource 函数传给系统。

```
BOOL CreateScalableFontResource(hdc,lpszResourceFile,lpszFontFile,
    lpszCurrentPath)
```

该函数的第二个参数是一个指向将要生成的 FOT 文件的指针。第三个参数则指向 TrueType 文件(TTF)本身,最后一个参数表明 TTF 文件的路径。

OUTLINETEXTMETRIC 结构

该结构以及 TEXTMETRIC 结构都是由 TrueType 字模提供域值,这些值通过调用 GetOutlineTextMetrics 函数输入到当前设备环境中。

```
DWORD GetOutlineTextMetrics(hdc,cbData,lpOTM)

HDC hdc;                                          /* Handle of device context */
WORD cbData;                                      /* Szie of data to be returned */
LPOUTLINETEXTMETRIC lpOTM;      /* Pointer to a variable of OUTLINETEXTMETRIC structure */
```

因为要获得数据的正确值,因此 GetOutlineTextMetrics 函数必须一开始就一次调用成功,该函数的第二个参数是 OUTLINETEXTMETRIC 结构的大小,第三个参数为 NULL,返回值用于保存内存空间。GlobalLock 函数传递一个指向该区域的指针,然后将其传到 GetOutLineTextMetrics 函数中,以便存放 TrueType 字模的值。

```
Size = (WORD)GetOutlineTextMetrics(hDC,sizeof(OUTLINETEXTMETRIC),NULL);
hMim = GlobalAlloc(GMEM_MOVEABLE,wSize);
lpMem= (LPOUTLINETEXTMETRIC)GlobalLock(hMem);
GetOUtlineTextMetrics(hDC,wSize,lpMem);
                                        // now the values can be accessed; for example,
otmAscent =lpMem->otmAscent;
otmtmHeight = lpMem->otmTextMetrics. tmHeight;
                                                  // when pertinent data has been processed
GlobalUnlock(hMem);
GlobalFree(hMem);
```

OUTLINETEXTMETRIC 结构

下表所列的是 OUTLINETEXTMETRIC 结构,后面将详细讨论其结构域。

字段	数据类型	描述
otmSize	WORD	结构的字节长度
otmTextMetrics	TEXTMETRIC	TEXTMETRIC 结构
otmFiller	BYTE	显示结构是字对齐还是字节对齐
otmPanoseNumber	PANOSE	原始字模格式(如斜体和下划线)
otmfsSelection	WORD	指出字模是否存在
otmfsType	WORD	指定字模是否授权
otmsCharSlopeRise	WORD	斜角光标
otmsCharSlopeRun	WORD	斜角光标
otmEmSquare	WORD	em 方阵大小
otmAscent	WORD	从基线开始的上行距离
otmDescent	WORD	基线上下行距
otmLineGap	WORD	印刷线间距
otmCapEmHeight	WORD	支持
otmXHeight	WORD	支持
otmrcFontBox	RECT	字模周围的边界框
otmMacAscent	WORD	用于 Macintosh 的上行字
otmMacDesent	WORD	用于 Macintosh 的下行字
otmMacLineGap	WORD	用于 Macintosh 的行间距
otmusMinimumPPEM	WORD	字模的最小尺寸
otmptSubscriptSize	POINT	推荐的下标大小
otmPtSubcriptOffSet	POINT	下标位置
otmptSuperscriptOffsetSize	POINT	上标大小
otmksStrikeOutSize	POINT	上标位置
otmsUnderScoreSize	WORD	计划笔划宽度
otmsStrikeOutPosition	WORD	笔划位置
otmsUnderScoreSize	WORD	下划线宽度
otmsUnderScorePosition	WORD	下划线位置
otmpFamilyName	PSTR	字模族名指针
otmpFaceName	PSTR	字型名指针
otmpStyleName	PSTR	字模格式名指针
otmpFullName	PSTR	字模全名指针

otmfsType 域对于程序来说是非常重要的,因为它支持字模与文档的链接,以便能传送到另一个系统。如果清除了该域的第一位,则字模可以修改。否则,字模只能读取,而且不能永久地装入到系统中。

每种 TrueType 字模都有最小尺寸,以便于读取在 OUTLINETEXTMETRIC 结构中的相应域为 otmusMinimumPPEM(每 em 矩形区的像素数)。

TrueType 字模文件有一个特别的编号,称为 Panose 号,它帮助应用程序根据所给的规范选取最合适的字模。该编号是由 10 种不同属性的评估值集合生成的,PANOSE 结构就包

括表示上面 10 种属性的域，其任务是由 OUTLINETEXTMETRIC 结构的 otmPanoseNumber 域完成的。

EnumFontFamilies 函数

程序员可以使用 EnumFonts 函数来确定哪种字模可应用于所给设备。该函数访问一个由应用程序提供的回调函数，结果信息放在 LOGFONT 和 TEXTMETRIC 结构中。不过在新版本中使用新的 EnumFontFamilies 函数取代它。因为它包括了一些附加功能，如合并字模格式名。

EnumFonts 函数仅仅识别在同一字模集中的字模，换句话说，只有正体、黑体、斜体、黑斜体等属性，而附加字模如变形黑体却无法实现，因为它们在不同的字符集中。

TrueType 字模在一定程度上按不同方式进行组织。一个字模集(像 Courier 和 Arial)只有字模格式属性编号。函数 EnumFontFamilies 则统计所给字模集中的所有字符，甚至格式不同于前面所列的那些字模也可以识别。

回调函数

和 EnumFonts 一样，EnumFontFamilies 函数也调用回调函数，它定义如下：

```
int FAR PASCAL FontFunc(lpLogFont,lpTextMetrics,nFontType,lpData)
LPLOGFONT lpLogFont;                          //Pointer to LOGFONT structure
LPTEXTMETRICS lpTextMetrics;                  // Pointer to
                                              // TEXTMETRIC structure
short nFontType;                              // Font type
LPSTR lpData;                                 // Pointer to additional data
```

参数 nFontType 指出字模是否 TrueType。如果，则设置 SCALABLE、FONTTYPE 位。

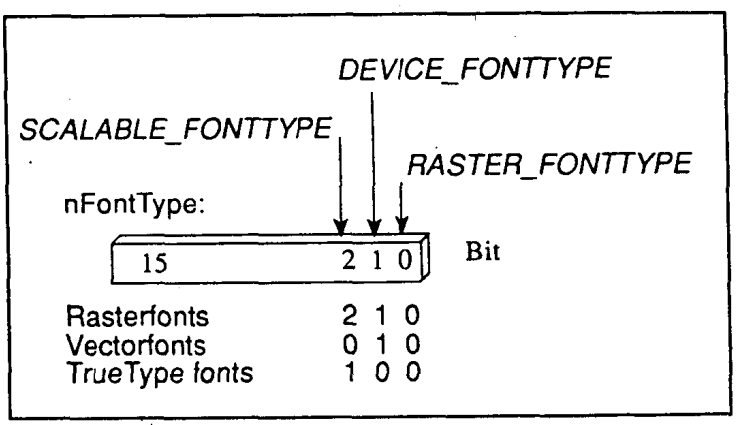


图 5.20 参数 nFontType 的内容

为了找出所有对系统有用的字模集，EnumFontFamilies 函数可按下面的方式调用：

```
EnumFontFamilies (hDC,NULL,lpEnumCallBack,(LPSTR) &.aFontCount);
```

回调函数的瞬间过程地址必须首先由 MakeProcInstance 函数创建，回调函数将字模编号输入到变量中，其他地址由最后一个参数指定。

回调函数仅仅适合于 TrueType 集，其他的字模，它都不予统计。这些字体集名是从 LOGFONT 结构中获得的，它代表回调函数的第一个参数，它们存储在 szFontname 全程字符串数组中，可被通用的窗口函数访问。

```
BOOL FAR PASCAL EnumCallBack(LPIOGFONT lpLogFont,LPTEXTMETRIC lpTextMetric,
        short FontType,LPSTR lpFontCount)
{
    int far * lpiCount = (int far * ) lpFontCount;

    if(FontType & SCALABLE_FONTTYPE)
    {
        lstrcpy(szFontname[ * lpiCount],
            (LPSTR)lpLogFont->lfFaceName);
        ( * lpiCount)++;
    }
    return TRUE;
}
```

在某些应用程序中，参数 nFontType 的零位是由 EnumFonts 的回调函数检查的，以避免使用向量字模。当然也可以通过应用程序禁止使用 TrueType 字模。

GetGlyphOutline 函数

调用 GetGlyphOutline 函数可以得到指定的 TrueType 字符的轮廓。程序员可以指定是否把轮廓作为雕刻字位图返回，或者使用与字模单元有关的曲线像素。如果字符要旋转，程序员必须调用其位图表示，同时定义旋转矩阵。

```
DWORD GetGlyphOutline(hdc,wChar,wFormat,lpMetrics,
        cbBuffer,lpBuffer,lpMatrix)

HDC hdc;                          /* Handle of device context */
WORD wChar;                       /* desired character */
WORD wFormat;                     /* desired data format */
LPGLYPHMETRICS lpMetrics;         /* Pointer to a GLYPHMETRICS structure */
DWORD cbBuffer;                   /* Size of subsequent buffer */
LPSTR lpBuffer;                   /* Pointer to data buffer */
LPMAT2 lpMatrix;                  /* Pointer to a MAT2 Structure */
```

GetCharABCWidths 函数

GetCharABCWidths 函数获得一个 TrueType 字符的 A，B，C 宽度，正如本章前半部分讨论的那样。字模中几乎每一个字符都有不同的宽度。甚至一个字符集中，同一字母也可能有不同的宽度，如 Arial 和 Arial Bold。下面的调用将返回字符“W”的宽度。

```
GetCharABCWidths (hDC,"W","W",&abc);
```

GetCharABCWidths 函数将返回的宽度值写到一个 ABC 结构类型的变量中，其地址已由参数给出，宽度 A 和 C 的值为负。

```
typedef struct _ABC
{
    short abcA;
    WORD abcB;
    short abcC;
```

} ABC;

5.5.2 TrueType 字模应用实例

下面的应用程序说明了某些新的 TrueType 字模函数的使用方法。首先，它确定系统是否装有 TrueType 字模。如果有，确定其数目。结果显示在窗口中，能通过 Font/Redraw 菜单项调入。

如果已找到 TrueType 字模，则选择第一种字模作为显示正文的字模。用户能修改显示正文，可以按 B 键(改为黑体)、I 键(改变斜体)、S 键(印刷体)、U 键(加下划线)。

一旦按下了某个键，应用程序将显示键值所代表的字母的 A，B，C 宽度值。如果想显示开始出现过的正文，可选择 Font/Redraw 菜单项。

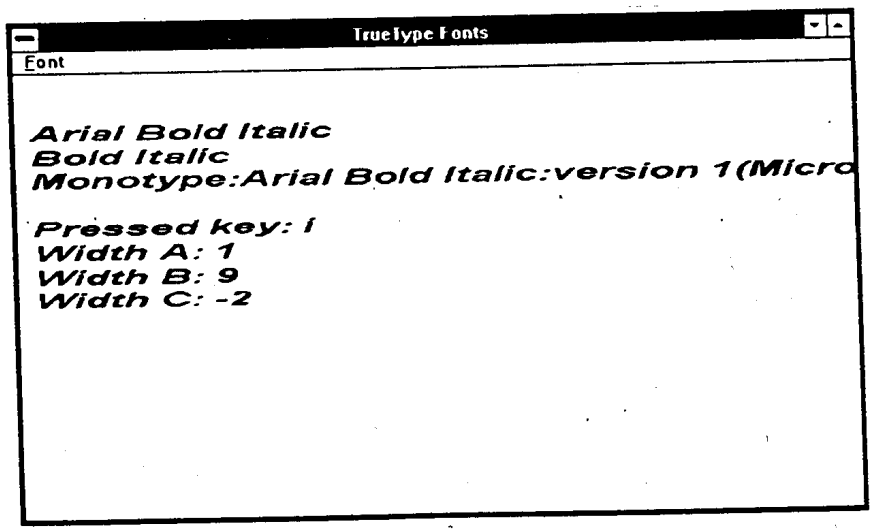


图 5.21 TrueType 字模实例

新函数	描述
EnumFontFamilies	找出可用于某设备的字模
GetABCWidths	给出 TrueType 字模的 ABC 宽度
GetOutLineTextMetrics	提出关于 TrueType 字模的信息
GetRasterizerCaps	查明 TrueType 字模是否可用
新结果	描述
OUTLINETEXTMETRIC	由 GetOutlineTextMetrics 函数提供
QASTERIZER_STATUS	由 GetRasterizerCaps 函数提供

源代码：TTFONT. C

```
/* * TTFONT. C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates TrueType fonts in Windows * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "ttfont. h"                    // Include TTFONT. H header file
```

```
#if _MSC_VER >= 700                          // Running Microsoft C 7.00 or
#pragma warning (disable:4028)               // higher? Disable warning 4028
#endif

int PASCAL WinMain(HANDLE hInstance,                    // Current instance
    HANDLE hPrevInstance,                               // Previous instance
    LPSTR lpszCmdLine,                                  // Long ptr to string after
                                                        // program name during execution
    int nCmdShow)                                       // Specifies the application
                                                        // window's appearance
{
  MSG msg;                                              // Message variable
  HWND hWnd;                                            // Window handle
  int nError;                                           // Error integer

  strcpy(szAppName,"TTFont");                           // Application name
  hInst = hInstance;                                    // Instance handle

  if (! hPrevInstance)                                  // Initialize first instance
  {
    if ((nError=nCwRegisterClasses(hInstance)) ==-1)
    {
                                                        // If registration fails
      LoadString(hInstance,IDS_ERR_CLASS,
        szString,sizeof(szString));
      MessageBox(NULL,szString,NULL,
        MB_ICONEXCLAMATION);
      return nError;
    }
  }

/** Specify appearance of application's main window **/
  hWnd = CreateWindow(szAppName,                        // Window class name
    "TrueType Fonts",                                   // Window caption
    WS_OVERLAPPEDWINDOW,                                // Overlapped window style
    CW_USEDEFAULT,0,                                    // Default upper-left x,y pos.
    CW_USEDEFAULT,0,                                    // Default initial x,y sizes
    NULL,                                               // No parent window
    NULL,                                               // Window menu used
    hInstance,                                          // Application instance
    NULL);                                              // No creation parameters

  if(hWnd == NULL)                                      // If window handle = NULL
  {
    LoadString(hInstance,IDS_ERR_WINDOW,                // then display error window
      szString,sizeof(szString));
    MessageBox(NULL,szString,NULL,
      MB_ICONEXCLAMATION);
    return IDS_ERR_WINDOW;
  }
```

```
  ShowWindow( hWnd,nCmdShow );                          // Make window visible
  UpdateWindow( hWnd );                                 // Update window

  while (GetMessage(&msg,NULL,0,0))                     // Message reading
  {
    TranslateMessage(&msg);                             // Message translation
    DispatchMessage(&msg);                              // Send message to Windows
  }
  return (int)msg. wParam;                              // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)                // Window class registration
{
/* * Specify window class * * * * * * * * * * * * * * * * * * */

  WNDCLASS wcClass;                                     // Main window class

  wcClass. hCursor = LoadCursor( NULL,IDC_ARROW );      // Mouse cursor
  wcClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);     // Default icon
  wcClass. lpszMenuName = (LPSTR)"FontMenu";            // Menu name
  wcClass. lpszClassName = szAppName;                   // Window class
  wcClass. hbrBackground =
    (HBRUSH)GetStockObject(WHITE_BRUSH);                // White background
  wcClass. hInstance = hInstance;                       // Instance
  wcClass. style = CS_VREDRAW | CS_HREDRAW;
                                                        // Horizontal and vertical redraw of client area
  wcClass. lpfnWndProc = WndProc;                       // Window function
  wcClass. cbClsExtra = 0 ;                             // No extra bytes
  wcClass. cbWndExtra = 0 ;                             // No extra bytes

  if (! RegisterClass( &wcClass ) )                     // Register window class
    return FALSE;                                       // Return FALSE if registration fails
  return TRUE;                                          // Return TRUE
}

/* * WndProc * * * * * * * * * * * * * * * * * * * * * * * * * */
/* *. Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL WndProc(HWND hWnd,
     unsigned message,
     WORD wParam,
     LONG lParam )
{
  FARPROC lpEnumCallBack;                               // FAR procedure
  static int aFontCount = 0;                            // Font count integer
  char szText[30];                                      // Text string
  HDC hDC;                                              // Device context handle
  PAINTSTRUCT ps;                                       // Paint structure
  static char szFaceName[LF_FACESIZE],                  // Typeface name
    szStyleName[LF_FACESIZE],                           // Style name for font
```

```
  szFullName[LF_FULLFACESIZE];                          // Full name for font
static BOOL bExist,                                     // Existence of font = FALSE
  bOutline,                                             // Outline for font = FALSE
  bWeight = FALSE;                                      // Font weight = FALSE
int i,Number;                                           // Number variables

/* * Font variables,controlling sizes,names,etc. * */
RASTERIZER_STATUS rStatus;
static LOGFONT TTFont;
static HFONT hFont;
HFONT hOldF;
short yChar;
static short otmtmHeight,otmtmExLdg,FChWidthA,FChWidthB;
TEXTMETRIC tm;
WORD wSize;
HANDLE hMem;
LPOUTLINETEXTMETRIC lpMem;
LPSTR lpFaceName;
ABC abc;
static WORD FChWidthC,wKeyChar;

switch (message)                                        // Process messages
{
  case WM_CREATE:
    lpEnumCallBack = MakeProcInstance(EnumCallBack,hInst);
    hDC = GetDC(hWnd);
    EnumFontFamilies(hDC,NULL,lpEnumCallBack,(LPSTR)&aFontCount);
    ReleaseDC(hWnd,hDC);
    FreeProcInstance(lpEnumCallBack);
    InvalidateRect(hWnd,NULL,TRUE);
    bExist = TRUE;

    Number = sizeof(RASTERIZER_STATUS);
    GetRasterizerCaps(&rStatus,Number);
    if (rStatus.wFlags == (TT_AVAILABLE | TT_ENABLED))
    {
      memset(&TTFont,0x00,sizeof(LOGFONT));
      TTFont.lfHeight = 0;
      TTFont.lfWidth = 15;
      TTFont.lfWeight = 700;
      TTFont.lfItalic = TRUE;
      strcpy(TTFont.lfFaceName,szFontname[0]);
      hFont = CreateFontIndirect(&TTFont);
    }
    break;
  case WM_COMMAND:                                      // Commands
    switch(wParam)
    {
    case IDM_EXIST:                                     // Does this exist?
      InvalidateRect(hWnd,NULL,TRUE);
      bExist = TRUE;
```

```
      break;
    default:
      return (DefWindowProc(hWnd,message,
          wParam,lParam));
      break;
    }
    break;
  case WM_CHAR:                                          // Display characters
    Number = sizeof(RASTERIZER_STATUS);
    GetRasterizerCaps(&rStatus,Number);
    if (rStatus.wFlags == (TT_AVAILABLE | TT_ENABLED))
    {
      switch (wParam)
      {
      case 's':                                          // S for strikeout
        TTFont.lfStrikeOut = (BYTE)! TTFont.lfStrikeOut;
        break;
      case 'u':                                          // U for underline
        TTFont.lfUnderline = (BYTE)! TTFont.lfUnderline;
        break;
      case 'i':                                          // I for italic
        TTFont.lfItalic = (BYTE)! TTFont.lfItalic;
        break;
      case 'b':                                          // B for bold
        if (bWeight)
        {
          TTFont.lfWeight = FW_BOLD;
          bWeight = FALSE;
        }
        else
        {
          TTFont.lfWeight = FW_NORMAL;                   // Otherwise revert to normal
          bWeight = TRUE;
        }
        break;
      }
      if ((wParam=='s')|(wParam=='u')|(wParam=='i')|(wParam=='b'))
      {
        DeleteObject(hFont);
        hFont = CreateFontIndirect(&TTFont);
      }
      wKeyChar = wParam;
      hDC = GetDC(hWnd);
      hOldF = SelectObject(hDC,hFont);
      GetCharABCWidths(hDC,wParam,wParam,&abc);
      FChWidthA = abc.abcA;
      FChWidthB = abc.abcB;
      FChWidthC = abc.abcC;

      wSize=(WORD)GetOutlineTextMetrics(hDC,sizeof(OUTLINETEXTMETRIC),NULL);
      hMem = GlobalAlloc(GMEM_MOVEABLE,wSize);
```

```
        lpMem = (LPOUTLINETEXTMETRIC)GlobalLock(hMem);
        GetOutlineTextMetrics(hDC,wSize,lpMem);
        SelectObject(hDC,hOldF);
        ReleaseDC(hWnd,hDC);
        otmtmHeight = lpMem->otmTextMetrics.tmHeight;
        otmtmExLdg = lpMem->otmTextMetrics.tmExternalLeading;
        lpFaceName = (char far *)lpMem+(WORD)lpMem->otmpFaceName;
        lstrcpy(szFaceName,lpFaceName);
        lpFaceName = (char far *)lpMem+(WORD)lpMem->otmpStyleName;
        lstrcpy(szStyleName,lpFaceName);
        lpFaceName = (char far *)lpMem+(WORD)lpMem->otmpFullName;
        lstrcpy(szFullName,lpFaceName);
        GlobalUnlock(hMem);
        GlobalFree(hMem);
        InvalidateRect(hWnd,NULL,TRUE);
        bOutline = TRUE;
      }
      else                                          // No TT fonts? Display msg box
        MessageBox(hWnd,                                    // Window handle
          "No TrueType fonts available",
          "TTFONT",                                         // Window caption
          MB_OK |                                           // OK button
          MB_ICONHAND);                                     // Hand or stop sign
      break;
    case WM_PAINT:                                          // Paint window
      hDC = BeginPaint(hWnd,&ps);
      if (bExist)
      {
        hOldF = SelectObject(hDC,hFont);
        GetTextMetrics(hDC,&tm);
        yChar = tm.tmHeight+tm.tmExternalLeading;
        wsprintf(szText,"Number of TrueType families: %d",aFontCount);
        TextOut(hDC,10,2 * yChar,(LPSTR) szText,strlen(szText));
        for (i=0;i<aFontCount;i++)
        {
          TextOut(hDC,20,4 * yChar+ i * yChar,szFontname[i],strlen(szFontname[i]));
        }
        SelectObject(hDC,hOldF);
        bExist = FALSE;
      }
      if (bOutline)
      {
        hOldF = SelectObject(hDC,hFont);
        yChar = otmtmHeight + otmtmExLdg;
        TextOut(hDC,10,2 * yChar,(LPSTR) szFaceName,strlen(szFaceName));
        TextOut(hDC,10,3 * yChar,(LPSTR) szStyleName,strlen(szStyleName));
        TextOut(hDC,10,4 * yChar,(LPSTR) szFullName,strlen(szFullName));

        wsprintf(szText,"Pressed key: %c",wKeyChar);
        TextOut(hDC,10,6 * yChar,(LPSTR) szText,strlen(szText));
        wsprintf(szText,"Width A: %d",FChWidthA);
```

```
            TextOut(hDC,10,7 * yChar,(LPSTR) szText,strlen(szText));
            wsprintf(szText,"Width B: %d",FChWidthB);
            TextOut(hDC,10,8 * yChar,(LPSTR) szText,strlen(szText));
            wsprintf(szText,"Width C: %d",FChWidthC);
            TextOut(hDC,10,9 * yChar,(LPSTR) szText,strlen(szText));
            bOutline = FALSE;
            SelectObject(hDC,hOldF);
          }
          EndPaint(hWnd,&ps);
          break;
        case WM_DESTROY:                                          // Exit
          PostQuitMessage(0);
          break;
        default:
          return (DefWindowProc( hWnd,message,wParam,
              lParam ));
          break;
      }
      return(0L);
}

/ * * EnumCallBack FAR procedure * * * * * * * * * * * * * * * * * * /

BOOL FAR PASCAL EnumCallBack(LPLOGFONT lpLogFont,LPTEXTMETRIC lpTextMetric,
  short FontType,LPSTR lpFontCount)
{
  int far * lpiCount = (int far * ) lpFontCount;

  if(FontType & SCALABLE_FONTTYPE)
  {
    lstrcpy(szFontname[ * lpiCount],(LPSTR)lpLogFont->lfFaceName);
    ( * lpiCount)++;
  }
  return TRUE;
}
```

模块定义文件:TTFONT. DEF

```
NAME TTFONT
DESCRIPTION 'TrueType Fonts'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS WndProc @1
    EnumCallBack @2
```

资源描述:TTFONT. RC

```
#include "ttfont. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END

FontMenu MENU
BEGIN
    POPUP "&Font"
    BEGIN
        MENUITEM "&Redraw", IDM_EXIST
    END
END
```

头文件:TTFONT. H

```
#define NOMINMAX
#include <windows. h>
#include <string. h>
#include <stdlib. h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define IDM_EXIST 100

#ifndef SCALABLE_FONTTYPE
#define SCALABLE_FONTTYPE TRUETYPE_FONTTYPE
#endif

char szString[128];                                    /* Variable for loading resource text */
char szAppName[20];                                    /* Window class name */
HANDLE hInst;
char szFontname[10][30];

int nCwRegisterClasses(HANDLE);
long FAR PASCAL WndProc(HWND,unsigned,WORD,LONG);
BOOL FAR PASCAL EnumCallBack(LPLOGFONT,LPTEXTMETRIC,short,LPSTR);
```

MAKE 文件: TTFONT. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
```

```
H = TTFONT.H

all: TTFONT.EXE

TTFONT.RES: TTFONT.RC $(RESFILES)

TTFONT.OBJ: TTFONT.C $(H)

TTFONT.EXE: TTFONT.OBJ TTFONT.RES TTFONT.DEF
    link /NOD/A:16 TTFONT,TTFONT,nul,libw slibcew,TTFONT.def
    rc TTFONT.res TTFONT.exe
```

按下列方式启动 NMAKE，以创建 TTFONT 应用程序：

```
nmake ttfont.mak
```

首先，应用程序使用一个字模集新成员，Windows 将 TrueType 字模格式转化为位图，并将之放入内存。这样，以后对这些图像的调用可直接进行，速度比开始时要快得多。

如果 TrueType 字模可用，则回调函数接收的第一个字模为缺省字模。字模宽度的缺省值为 15，应用程序使用的缺省字模为黑体和斜体。

```
TTFont.lfWidth = 15;
TTFont.lfWeight = 700;
TTFont.lfItalic = TRUE;
strcpy(TTFont.lfFaceName,szFontname[0]);
hFont = CreateFontIndirect(&TTfont);
```

每次按下键后，程序检查 TrueType 字模是否可用。如果 TrueType 字模存在，应用程序则显示一个消息框，应用程序检查 B,K,U,S 键，因为它们说明了显示字模的变化。当用户按下某个键时，相应的显示过程从静态变量中接收相应字母的宽度值，并且显示宽度值。

```
GetCharABcWidths(hDC,wParam,wParam,&abc);
WidthA = abc.abcA;
WidthB = abc.abcB;
WidthC = abc.abcC;
```

在按键之后，OUTLINETEXTMETRIC 结构传入不同的字模名，并允许字模变化。

```
lpFaceName = (char far *)lpMem+ (WORD) lpMem->otmpFaceName
lstrcpy(szFaceName,lpFaceName);
lpFaceName = (char far *)lpMem+ (WORD) lpMem->otmpStyleName
lstrcpy(szStyleName,lpFaceName);
lpFaceName = (char far *)lpMem+ (WORD) lpMem->otmpFullName
lstrcpy(szFullName,lpFaceName);
```

下面的 WM_PAINT 消息显示字模名和 ABC 宽度。如果用户选择 TrueType 菜单项，则在窗口中显示可用字模，程序随之启动。OUTLINETEXTMETRIC 结构提供了起始线位置。GetTextMetrics 函数必须在开始时调用以获得数据。

```
otmtmHeight = lpMem->otmTextMetrics.tmHeight;
otmtmExLdg = lpMem->otmTextMetrics.tmExternalLeading;
yChar = otmtmHeight + otmtmExLdg;
```

或

```
GetTextMetrics(hDC,&tm);
yChar = tm.tmHeight+tm.tmExternalLeading;
```

第六章　键盘和鼠标输入

这一章描述用于输入显示的消息，然后看看当按下某个键或使用鼠标时所发生的情况，也讨论如何访问定时器。

6.1　输入消息

用户无论何时按下某个键、移动鼠标器或按下鼠标按钮，Windows 都会发送输入消息给应用程序。同样地，Windows 也产生消息以响应定时器输入。这里有各种各样的消息：

消息类型	描述
键盘	用户从键盘输入
字符	键盘输入转换为字符代码
鼠标	用户以鼠标输入
菜单	用户用菜单输入
滚动条	用户用滚动条输入
定时器	通过系统定时器输入

键盘、鼠标和定时器消息都是由硬件中断产生的。Windows 把这些消息放在系统的消息队列中，再从这个队列传送到适当的应用程序消息队列里。

字符、菜单和滚动条消息是对非客户区所产生的鼠标或键盘动作的响应，或者是由解释键盘消息而产生的。

6.2　键盘输入

在 Windows 中，应用程序不负责收集键盘输入，而是由 Windows 系统完成这一任务。系统用不同的消息通知用户动作给应用程序。

6.2.1　键盘消息

用户一旦按下或释放一个键，Windows 就会基于键盘驱动程序的信息产生键盘消息。这一消息最后到达当前具有焦点的应用程序。SetFocus 函数可把焦点指派到特定的窗口，GetFocus 函数可检索窗口句柄。

消息	事件
WM_KEYDOWN	按键
WM_KEYUP	释放键

WM_SYSKEYDOWN　按下系统键

WM_SYSKEYUP　松开系统键

消息 WM_SYSKEYDOWN 和 WM_SYSKEYUP 对于 Windows 比对特殊窗口函数更为重要，它们由 Alt 键和其他键(例如 Alt+Tab 或 Alt+F4)联合产生。在某些实例中这些消息得到处理。

然而，如果这些消息被处理，必须传递给缺省的窗口过程。否则，譬如当用户按下 Alt+Tab 时将不能切换到其他应用程序，这将使基本操作不可能执行。

当不按下 Alt 键时会出现另外两个消息。如果不在窗口函数中处理而是传递给 DefWndProc，将无任何影响，因为 Windows 不处理这两条消息。

每一个被按下或被松开的键的虚拟键代码定位在各个键盘消息的 wParam 参数中。虚拟键代码对于某个键来说是一个设备独立值。

频繁使用的字母和数字键在 WINDOWS.H 头文件中都定义了一个名字，这些名字以 VK、VK_F2 或 VK_INSERT 开头。

参数 wParam 划分为五个区域：重复值、OEM 扫描码、相关码、早期键状态和转换码，然而这些数值通常并不使用。

这四个键盘消息没有指出是否按下 Shift 键。但有一个函数可用于决定每个虚拟键状态：

```
int Get Keystate (nVirKey)
```

nVirtKey：虚拟键代码(例如 'A' 或 VK_SHIFT)

int：如果检测到一个负值(假设最高位置为 1)，那么就按下了特殊键。由于需要保持同步，应该使用只带键盘消息的函数。

6.2.2 字符消息

许多应用程序是随着字符消息而不是随着键盘消息的处理而工作。TranslateMessage 函数用于这一转换，并且必须置于消息循环中。

函数必须产生一个新消息，称为 WM_CHAR 或 WM_SYSCHAR——无论何时检测到 WM_KEYDOWN 或 WM_SYSKEYDOWN 消息。这一新消息放在应用程序的消息队列中。它借助于 GetMessage 和 DispatchMessage 函数被发送给窗口过程。参数 wParam 很有用，因为它包含能用于开关指令的 ANSI 字符代码。

下例表示按下 Shift 键和 1 键产生一个大写字符的过程。然而，为 Shift 键而产生的消息是消息 WM_KEYDOWN 和 WM_KEYUP，而不是消息 WM_CHAR。

消息	wParam
WM_KEYDOWN	虚拟键 VK_SHIFT
WM_KEYDOWN	虚拟键“1”
WM_CHAR	ANSI 码“1”
WM_KEYUP	虚拟键“1”
WM_KEYUP	虚拟键 VK_SHIFT

Windows 应用程序在正常情况下使用一个扩展字符集，称为 ANSI 字符集。这一字符集与十六进制 20H 和 7EH 之间的 ASCII 字符集相匹配。

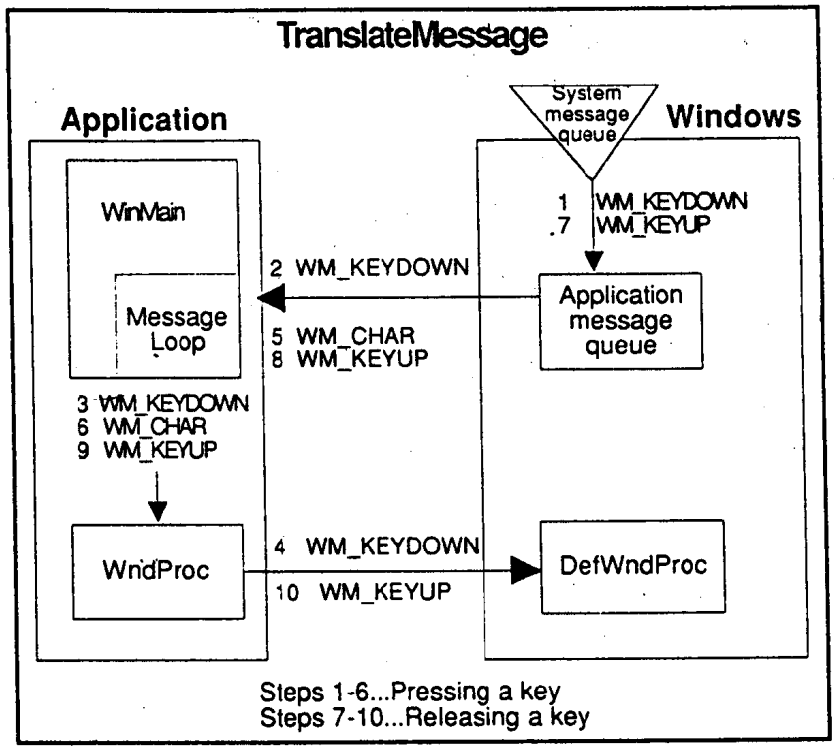


图 6.1 TranslateMessage 函数在应用程序处理字符消息而非键盘消息前须置于消息循环中

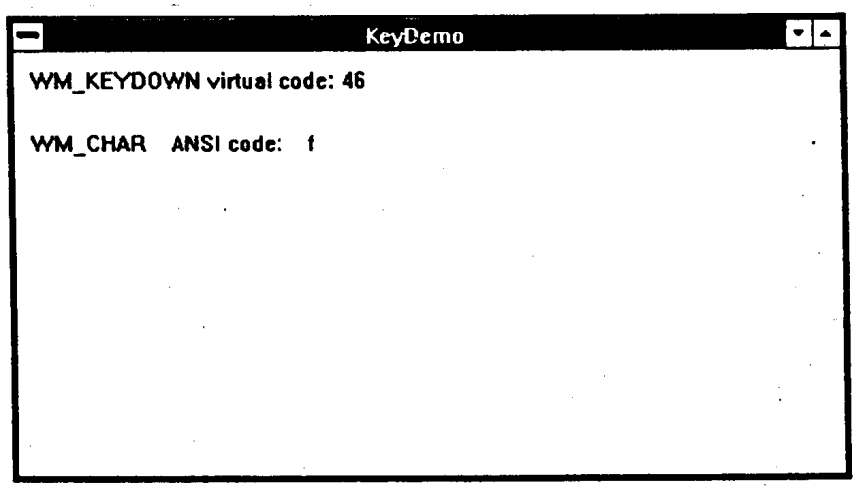


图 6.2

6.2.3 键盘举例

KEYDEMO.EXE 应用程序响应每个键的按下和松开动作以及消息 WM_CHAR。在客户区中，屏幕显示出适当的消息和定位于 wParam 中的值。

新消息	描述
WM_KEYDOWN	按下键时发送
WM_KEYUP	松开键时发送
WM_CHAR	原于 WM_KEYDOWN 的解释
新函数	**说明**
wsprintf	在缓冲区中格式化并存储字符和值

源代码:KEYDEMO. C

```
/* * KEYDEMO. C * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays WM_KEYDOWN and WM_KEYUP messages as virtual and ANSI codes * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows. h"                                    // Include WINDOWS. H header file
#include "string. h"                                     // Include STRING. H header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

BOOL KeyInit ( HANDLE );                                 // Key initialization
long FAR PASCAL KeyWndProc( HWND,unsigned,WORD,LONG);

/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,wCmdShow )
    HANDLE hInstance,hPrevInstance;                      // Current & previous instances
    LPSTR lpszCmdLine;                                   // Long ptr to string after
                                                         // program name during execution
    int wCmdShow;                                        // Specifies the application
                                                         // window's appearance
{
MSG msg;                                                 // Message variable
HWND hWnd;                                               // Window handle

    if (! hPrevInstance)                                 // Initialize first instance
    {
        if (! KeyInit( hInstance ))                      // If initialization fails
            return FALSE;                                // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("KeyDemo",                       // Window class name
            "KeyDemo",                                   // Window caption
            WS_OVERLAPPEDWINDOW,                         // Overlapped window style
            CW_USEDEFAULT,                               // Default upper-left x pos.
            0,                                           // Upper-left y pos.
            CW_USEDEFAULT,                               // Default initial x size
            0,                                           // y size
```

```
            NULL,                                   // No parent window
            NULL,                                   // Window menu used
            hInstance,                              // Application instance
            NULL);                                  // No creation parameters

    ShowWindow( hWnd,wCmdShow );                    // Make window visible
    UpdateWindow( hWnd );                           // Update window

    while (GetMessage(&msg,NULL,0,0))               // Message reading
    {
        TranslateMessage(&msg);                     // Message translation
        DispatchMessage(&msg);                      // Send message to Windows
    }
    return (int)msg. wParam;                        // Return wParam of last message
}

BOOL KeyInit( hInstance )                           // Key status
HANDLE hInstance;                                   // Instance handle
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */

WNDCLASS wcKeyClass;                                // Main window class

    wcKeyClass. hCursor = LoadCursor( NULL,IDC_ARROW );
                                                    // Mouse cursor
    wcKeyClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                    // Default icon
    wcKeyClass. lpszMenuName = NULL;                // No menu
    wcKeyClass. lpszClassName = "KeyDemo";          // Window class
    wcKeyClass. hbrBackground = GetStockObject( WHITE_BRUSH );
                                                    // White background
    wcKeyClass. hInstance = hInstance;              // Instance
    wcKeyClass. style = CS_VREDRAW | CS_HREDRAW;
                                                    // Horizontal and vertical redraw of client area
    wcKeyClass. lpfnWndProc = KeyWndProc;           // Window function
    wcKeyClass. cbClsExtra = 0 ;                    // No extra bytes
    wcKeyClass. cbWndExtra = 0 ;                    // No extra bytes

    if (! RegisterClass( &wcKeyClass ) )            // Register window class
        return FALSE;                               // Return FALSE if registration fails
                                                    // If registration is successful,
    return TRUE;                                    // Return TRUE
}

/* * KeyWndProc * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL KeyWndProc( hWnd,message,wParam,lParam )
    HWND hWnd;                                      // Window handle
    unsigned message;                               // Message type
```

```
    WORD wParam;                                    // Message-dependent 16 bit value
    LONG lParam;                                    // Message-dependent 32 bit value
{
    static char chTextDown[35];                     // Declare TextDown 35 characters
    static char chTextUp[35];                       // Declare TextUp 35 characters
    static char chTextChar[35];                     // Declare TextChar 35 characters
    static BOOL bDown = FALSE;                      // Set Down default to FALSE
    static BOOL bUp = FALSE;                        // Set Up default to FALSE
    static BOOL bChar = FALSE;                      // Set Char default to FALSE

    HDC hDC;                                        // Device context handle
    PAINTSTRUCT ps;                                 // PAINTSTRUCT data structure

/* * React to keyboard status * * * * * * * * * * * * * * * * * */

    switch (message)                                // Process messages
    {
        case WM_KEYDOWN:                            // Process if key pressed
            wsprintf(chTextDown,"WM_KEYDOWN virtual code: %x",wParam);
            InvalidateRect(hWnd,NULL,TRUE);         // Redraw client area
            bDown = TRUE;                           // Key pressed? Make bDown TRUE
            break;                                  // End of this message process
        case WM_KEYUP:                              // Process if key released
            wsprintf(chTextUp,"WM_KEYUP virtual code: %x",wParam);
            InvalidateRect(hWnd,NULL,TRUE);         // Redraw client area
            bUp = TRUE;                             // Key released? Make bUp TRUE
            break;                                  // End of this message process
        case WM_CHAR:                               // Process if key ANSI character
            wsprintf(chTextChar,"WM_CHAR ANSI code: %c",wParam);
            InvalidateRect(hWnd,NULL,TRUE);         // Redraw client area
            bChar = TRUE;                           // Character key? Make bChar TRUE
            break;                                  // End of this message process
        case WM_PAINT:                              // Redraw client area window
            hDC = BeginPaint (hWnd,&ps);            // Begin WM_PAINT process
            if (bDown)                              // Key pressed?
            {                                       // Display text…
                TextOut(hDC,10,10,chTextDown,strlen(chTextDown));
                bDown = FALSE;                      //… and reset bDown
            }
            if (bUp)                                // Key released?
            {                                       // Display text…
                TextOut(hDC,10,30,chTextUp,strlen(chTextUp));
                bUp = FALSE;                        //… and reset bUp
            }
            if (bChar)                              // Was key a character?
            {                                       // Display text…
                TextOut(hDC,10,50,chTextChar,strlen(chTextChar));
                bChar = FALSE;                      //… and reset bChar
            }
            EndPaint(hWnd,&ps);                     // Finish processing WM_PAINT
            break;                                  // End of this message process
```

```
        case WM_DESTROY:                                    // Send WM_QUIT if window is
            PostQuitMessage(0);                                          // destroyed
            break;                                          // End of this message process
        default:                                                  // Send other messages to
                                                             // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                          // End of this message process
    }
    return(0L);
}
```

模块定义文件:KEYDEMO. DEF

```
; KEYDEMO. DEF

NAME KeyDemo

DESCRIPTION 'Keyboard example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS KeyWndProc @1
```

MAKE 文件:KEYDEMO. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: KEYDEMO. EXE

KEYDEMO. OBJ: KEYDEMO. C

KEYDEMO. EXE: KEYDEMO. OBJ KEYDEMO. DEF
    link /NOD/A:16 KEYDEMO,KEYDEMO,nul,libw slibcew,KEYDEMO. def
    rc KEYDEMO. exe
```

运行 NMAKE 产生应用程序 KEYDEMO:

```
nmake KEYDEMO. MAK
```

加入了四条新消息，这些消息在应用程序中处理。这里不详细讨论消息 WM_PAINT，现在所需理解的是已准备好的文本在哪里显示。输出由函数 InvalidateRect 开始。

同一函数以不同的参数用于其他三条消息 WM_KEYDOWN，WM_KEYUP 和 WM_CHAR。函数 wsprintf 格式化并存储字符于缓冲区中(此例中每一消息都有自己的缓冲区)。wsprintf 处理带有 Param 数值的文本，以便文本以后能被显示。每条消息都有各自的布尔变量，在消息被处理时设置为 TRUE。当消息输出时，此变量被求值并被删除。

6.3 鼠标输入

用户几乎能在所有 Windows 应用程序中使用鼠标。Windows 支持带有 1，2 或 3 个按钮的鼠标器。使用带有参数 SM_MOUSEPRESENT 的函数 GetSystemMetrics 确定鼠标是否存在。若返回值是 TRUE，则鼠标已被设置。

当硬件鼠标未被设置时，可能显示出鼠标光标(指针)。Windows 有一内部计数器来决定光标是否显示。当计数器大于或等于 0 时可看见鼠标光标。

如果鼠标被设置，则计数器的初始值置为 0；否则置为－1。函数 ShowCursor 能使计数器的值增加或减少。

```
int showcursor (bshow)
```

bshow：若此参数为 TRUE，“显示值”增 1；如果此参数值为 FALSE，“显示值”减 1。

int：返回新“显示值”

既然光标被所有 Windows 应用程序所用，它应该在离开客户区或窗口丢失焦点之前设置成初始状态。

6.3.1 鼠标消息

一旦用户移动鼠标光标，Windows 就发送消息 WM_MOUSEMOVE。每个鼠标按钮也对应三条消息，表明按钮是否按下和松开。没有与 WM_CHAR 等价的消息。

然而，能导致按两次鼠标按钮。这种情况发生在用户迅速按下、松开，然后再按下按钮时，这一过程必须在系统定义的双击按钮时间内发生。

消息	事件
WM_MOUSEMOVE	鼠标在窗口中移动
WM_LBUTTONDOWN	按下鼠标左按钮
WM_LBUTTONUP	松开鼠标左按钮
WM_MBUTTONDBCLK	双击左按钮
WM_MBUTTONDOWN	按下中间按钮
WM_MBUTTONUP	松开中间按钮
WM_MBUTTONDBCLK	双击中间按钮
WM_RBUTTONDOWN	按下右按钮
WM_RBUTTONUP	松开右按钮

WM_RBUTTONDBCLK　双击右按钮

除了这些鼠标消息外，还有 11 个与消息区相关的其他消息，通常在窗口过程中被忽视，所有这些消息都以 WM_NC 开头。

参数 WParam 和 IParam 对上述所有鼠标消息都有同样的结构，wParam 包括一个位屏蔽，指明鼠标按钮、Shift 键和 Ctrl 键当前的状态。IParam 包含鼠标光标的 X 和 Y 坐标，总是指向客户区的左上角。用 LOWORD 和 HIWORD 获得 X 和 Y 的值。

WParam：

位屏幕	意义
MK_CONTROL	按下 CONTROL 键时设置
MK_LBUTTON	按下左鼠标按钮时设置
MK_MBUTTON	如果中间鼠标按钮按下就设置
MK_RBUTTON	若右鼠标按钮按下则设置
MK_SHIFT	若 SHIFT 键按下则设置

IParam：

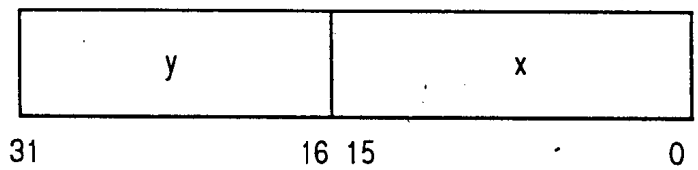


图 6.3

欲改变箭头光标位置，可使用 SetCursorPos 函数。新位置的 X 和 Y 坐标指的是整个屏幕的左上角而不是客户区的左上角。用函数 ClientToScreen 和函数 ScreenToClient 可把一个点从一个坐标系统转到另一坐标系统。

Windows 总是发送鼠标消息给具有鼠标焦点设置的窗口。鼠标光标也在此窗口中。第二种选择是，负值能用作 X 和 Y 坐标，因为所指的位置（客户区的左上角）不变。

SetCapture 在重要的操作（如装载应用程序等过程）中使用。既然鼠标在整个系统中存在，那么操作一完成就应该使用函数 ReleaseCapture 释放鼠标焦点。否则，其他应用程序将不能使用鼠标。

也不能在系统框上通过按鼠标按钮退出当前应用程序，因为在非客户区没有任何鼠标消息，然而仍需使用消息 WM_NCLBUTTONDBLCLK 退出应用程序。

若当 WNDCLASS 结构完成时形参恢复了 CS_DBLCLKS 标志，Windows 将只发送双击按钮消息给窗口过程。

用函数 SetDoubleClickTime 可改变双击按钮的时间，新的时间在整个系统中有效，而不只在特殊的应用程序中有效。

下面是双击左鼠标按钮的消息序列。

序列	消息
第一个消息	WM_LBUTTONDOWN

第二个消息　　WM_LBUTTONUP
第三个消息　　WM_LBUTTONDBLCLK
第四个消息　　WM_LBUTTONUP

6.3.2 鼠标举例

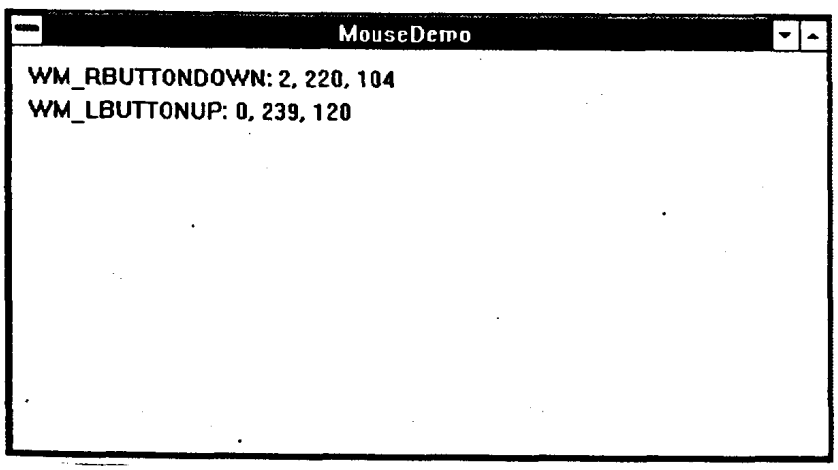


图 6.4

当按下右鼠标按钮并松开左鼠标按钮时此例将显示文本(与在键盘举例中显示的文本相似)。每条 WM_MOUSEMOVE 消息指示机器发出蜂鸣声。当双击左鼠标按钮时,鼠标焦点被设置并被释放。

新消息	说明
WM_MOUSEMOVE	鼠标移动时发送
WM_RBUTTONDOWN	右鼠标按钮按下时发送
WM_LBUTTONUP	左鼠标按钮松开时发送
WM_LBUTTONDBLCLK	双击左鼠标按钮时发送
新函数	说明
SetCapture	所有鼠标输入发送到特殊窗口
ReleaseCapture	鼠标焦点被释放
MessageBeep	产生嘟嘟声
新宏	简要说明
HIWORD	检索"long integer"值的最高值字
LOWORD	检索"long integer"值的最低值字

源代码:MOUSE_BT.C

```
/* MOUSE_BT.C ********************************/
/* Reads mouse button status and cursor position using WM_RBUTTONDOWN, */
/* WM_LBUTTONUP and WM_MOUSEMOVE. Button status and cursor position at */
/* the time of that status is displayed in the window. Beeping indicates */
```

```
/* * mouse movement. * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                                    // Include windows.h header file
#include "string.h"                                     // Include string.h header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

BOOL MouseInit ( HANDLE );                              // Mouse initialization
long FAR PASCAL MouseWndProc( HWND,unsigned,WORD,LONG);

/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,nCmdShow )
    HANDLE hInstance,hPrevInstance;                     // Current & previous instances
    LPSTR lpszCmdLine;                                  // Long ptr to string after
                                                        // program name during execution
    int nCmdShow;                                       // Specifies the application
                                                        // window's appearance
{
MSG msg;                                                // Message variable
HWND hWnd;                                              // Window handle

    if (! hPrevInstance)                                // Initialize first instance
    {
        if (! MouseInit( hInstance ))                   // If initialization fails
            return FALSE;                               // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("MouseDemo",                    // Window class name
           "MouseDemo",                                 // Window caption
           WS_OVERLAPPEDWINDOW,                         // Overlapped window style
           CW_USEDEFAULT,                               // Default upper-left x pos.
           0,                                           // Upper-left y pos.
           CW_USEDEFAULT,                               // Default initial x size
           0,                                           // y size
           NULL,                                        // No parent window
           NULL,                                        // Window menu used
           hInstance,                                   // Application instance
           NULL);                                       // No creation parameters

    ShowWindow( hWnd,nCmdShow );                        // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while (GetMessage(&msg,NULL,0,0))                   // Message reading
    {
        TranslateMessage(&msg);                         // Message translation
```

```
        DispatchMessage(&msg);                          // Send message to Windows
    }
    return (int)msg.wParam;                             // Return wParam of last message
}

BOOL MouseInit( hInstance )                             // Mouse status
HANDLE hInstance;                                       // Instance handle
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcMouseClass;                              // Main window class

    wcMouseClass.hCursor = LoadCursor( NULL,IDC_ARROW );
                                                        // Mouse cursor
    wcMouseClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                        // Default icon
    wcMouseClass.lpszMenuName = NULL;                   // No menu
    wcMouseClass.lpszClassName = "MouseDemo";           // Window class
    wcMouseClass.hbrBackground = GetStockObject( WHITE_BRUSH );
                                                        // White background
    wcMouseClass.hInstance = hInstance;                 // Instance
    wcMouseClass.style = CS_VREDRAW | CS_HREDRAW | CS_DBLCLKS;
                                                        // Horizontal and vertical redraw of client area,
                                                        // and check for double-clicks
    wcMouseClass.lpfnWndProc = MouseWndProc;            // Window function
    wcMouseClass.cbClsExtra = 0 ;                       // No extra bytes
    wcMouseClass.cbWndExtra = 0 ;                       // No extra bytes

    if (! RegisterClass( &wcMouseClass ) )              // Register window class
        return FALSE;                                   // Return FALSE if registration fails
                                                        // If registration is successful,
    return TRUE;                                        // Return TRUE
}

/* * MouseWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL MouseWndProc( hWnd,message,wParam,lParam )
    HWND hWnd;                                          // Window handle
    unsigned message;                                   // Message type
    WORD wParam;                                        // Message-dependent 16 bit value
    LONG lParam;                                        // Message-dependent 32 bit value
{
    static char chTextDown[35];                         // Declare TextDown 35 characters
    static char chTextUp[35];                           // Declare TextUp 35 characters
    static BOOL bCapture = FALSE;                       // Set Capture default to FALSE
    HDC hDC;                                            // Device context handle
    PAINTSTRUCT ps;                                     // PAINTSTRUCT data structure
```

```
/* * Check button and movement status,display on screen * * * * * * * */

    switch (message)                                          // Process messages
    {
        case WM_RBUTTONDOWN:                                  // Process if r. button pressed
            wsprintf(chTextDown,"WM_RBUTTONDOWN: %x,%d,%d",
                    wParam,LOWORD(lParam),HIWORD(lParam));
            InvalidateRect(hWnd,NULL,TRUE);// Redraw client area
            break;                                            // End of this message process
        case WM_LBUTTONUP:                                    // Process if l. button released
            wsprintf(chTextUp,"WM_LBUTTONUP: %x,%d,%d",
                    wParam,LOWORD(lParam),HIWORD(lParam));
            InvalidateRect(hWnd,NULL,TRUE);// Redraw client area
            break;                                            // End of this message process
        case WM_LBUTTONDBLCLK:                                // Double-click
            if (bCapture)                                     // bCapture sent?
            {
                ReleaseCapture();                             // Release mouse capture &
                bCapture = FALSE;                             // set bCapture to FALSE
            }
            else                                              // If no bCapture sent
            {
                SetCapture(hWnd);                             // Set mouse capture &
                bCapture = TRUE;                              // set bCapture to TRUE
            }
            break;                                            // End of this message process
        case WM_MOUSEMOVE:                                    // Beep during mouse movement
            MessageBeep(0);                                   // Sound beep
            break;                                            // End of this message process
        case WM_PAINT:                                        // Redraw client area window
            hDC = BeginPaint (hWnd,&ps);                      // Begin WM_PAINT process
                                                              // Display text...
            TextOut(hDC,10,10,chTextDown,strlen(chTextDown));
            TextOut(hDC,10,30,chTextUp,strlen(chTextUp));
            EndPaint(hWnd,&ps);                               //...and end WM_PAINT
            break;                                            // End of this message process
        case WM_DESTROY:                                      // Send WM_QUIT if window is
            PostQuitMessage(0);                               // destroyed
            break;                                            // End of this message process
        default:                                              // Send other messages to
                                                              // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                            // End of this message process
    }
    return(0L);
}
```

模块定义文件:MOUSE_BT. DEF

```
; MOUSE_BT. DEF

NAME MouseDemo
```

```
DESCRIPTION 'Mouse input example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS
    MouseWndProc @1
```

MAKE 文件:MOUSE_BT. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES =

all: MOUSE_BT. EXE

MOUSE_BT. OBJ: MOUSE_BT. C

MOUSE_BT. EXE: MOUSE_BT. OBJ MOUSE_BT. DEF
    link /NOD/A:16 MOUSE_BT,MOUSE_BT,nul,libw slibcew,MOUSE_BT. def
    rc MOUSE_BT. exe
```

运行 NMAKE 产生 MOUSE_BT 应用程序:

```
nmake mouse_bt. mak
```

MOUSE_BT. EXE 如何工作

消息 WM_RBUTTONDOWN 和 WM_LBUTTONUP 的编程逻辑、包括输出都能被接受。只有布尔变量被忽略了。IParam 的值划分为 HIWORD 和 LOWORD,以获得 X 和 Y 的坐标。

只要窗口设置为捕获状态且鼠标一移动,消息 WM_MOUSEMOVE 就产生了。函数 MessageBeep 响应这一消息。这表示这一消息频繁出现。消息队列中一次只包含一条 WM_MOUSEMOVE 消息。WNDCLASS 结构中的风格参数必须扩展,以适应任何 WM_LBUTTONDBCLK 消息的出现。

```
WcMouseClass. style = …|CS_DBLCLK
```

当这一消息被处理时,布尔变量被设置,这一变量表示鼠标捕获当前是否被设置。函数 SetCapture 接收唯一的参数,即包含鼠标捕获的窗口的句柄。函数 ReleaseCapture 不需任何参数。

第七章　定时器

当先前设置的时间间隔已满时，Windows 的定时器是另一种通知应用程序的输入设备。SYSTEM. DRV 驱动程序在 Windows 中负责处理硬件定时中断。定时器能用于各种各样的任务：除时钟程序之外，定时器能提高大型任务中多任务环境的效率。

每个应用程序尽可能快地返回 CPU 对 Windows 的控制。因此，较长的过程通常分成单独的部分，在 WM_TIMER 消息期间，一次只运行一部分。

处理并行或串行端口时也需要 WM_TIMER 消息。既然端口不产生自己的消息，那么 WM_TIMER 消息必须用于驱动数据端口。

当指定间隔的时间过去时，定时器会周期性地通知应用程序。用户程序将时间间隔告诉 Windows，然后，Windows 给用户程序发送周期性地发生的 WM_TIMER 消息，以表示时间间隔到了。

开始时，Windows 定时器似乎不如键盘和鼠标设备重要，对许多应用程序都是这样。但是，定时器比用户所想象的要重要得多。它不仅仅只用于计时程序（如 Windows CLOCK），Windows 所提供的 CALENDAR、CONTROL、PANEL、SPOOLER、TERMINAL 和 WRITE 等程序也要用到定时器。下面列举了 Windows 定时器的一些应用，有些是明显的，有些可能并不是那么明显：

- 计时——Windows 中的 CLOCK 和 CONTROL 程序都要显示当前时间。定时器告诉程序何时更新时钟。
- 唤醒——Windows CALENDAR 程序使用定时器触发一个预置的报警信号。
- 多任务处理——Windows 是一个非占先调度的多任务环境，程序尽快地把控制返回给 Windows 是非常重要的。若程序必须做大量的处理，它可以把作业分成小块，在接收到 WM_TIMER 消息时才处理一块作业。
- 实现“自动存盘”功能——定时器提示 Windwos 程序在指定时间过去后把用户的工作保存到磁盘上。
- 步进移动——游戏中的图形对象或计算机辅助教学程序中的连续显示需要按指定的速率来处理。使用定时器可消去由于微处理器速度的变化而造成的不一致。
- 终止程序的演示版本——例如，一些程序的演示版本在其开始之后的 30 min 后结束。如时间已到，定时器就会通知应用程序。
- 使用串行或并行通信——不像 Windows 中的其他输入设备，串行或并行通信口不产生消息。程序需要输入时必须查询它们，定时器能告诉程序何时去查询（定时器用于查询的另一方面涉及由 PeekMessage 调用建立的消息循环。

本章还讨论了定时器用于 Windows 编程的其他方面的问题。其中最主要的一个是“回

调”函数。这些函数的处理方式有点古怪，其中不止一次用到定时器。本章还讨论了程序不能访问定时器时产生的问题——产生这个问题的原因是 Windows 只维护了有限数目的定时器。解决这个问题是很重要的，但是下面提出的方法也可用于其他程序的错误处理。最后提供了一个示例程序，它处理如 Windows 颜色的使用、称为“弹出式”窗口类型的使用、强制应用程序使用图标、获取 Windows 中可用的自由内存总量、在 Windows 程序中使用浮点数、访问 WIN. INI 文件以获取关于国际时钟与数据格式等非定时器问题。

7.1 定时器基础

调用 SetTimer 函数可以为 Windows 程序分配一个定时器。SetTimer 含有一个指定时间间隔范围为 1 ms 到 65535 ms 的变量。这个值指示 Windows 把 WM_TIMER 消息发送给程序的速率。例如，若间隔为 1000 ms，则 Windows 每秒给程序发送一条 WM_TIMER 消息。

当程序开始使用定时器时，它调用 KillTimer 函数停止定时器消息。在处理 WM_TIMER 消息时，通过调用 KillTimer 函数可编程一个“瞬间”定时器。KillTimer 调用清除任何悬挂 WM_TIMER 消息的消息队列，从而程序在调用 KillTimer 之后就不会接收到漂泊的 WM_TIMER 消息。

Windows 在每一时刻只允许 32 个定时器有效。若 32 个定时器都已分配完，SetTimer 就返回 NULL。使用定时器的 Windows 程序必须考虑处理此问题的方法。

7.1.1 SYSTEM. DRV 与定时器

Windows 定时器是 IBM PC 硬件和 ROM BIOS 构造的定时器逻辑的一个相当简单的扩展。PC 的 ROM BIOS 初始化 Intel 8259 定时器芯片来产生硬件中断 08H。这个中断有时称为“定时器滴答”中断。中断 08H 每 54.925 ms 产生一次，或大约每秒 18.2 次。BIOS 使用中断 08H 更新存于 BIOS 数据区的“时间”值。MS-DOS 使用这个值计算当前时间。

Windows 目录下 SYSTEM 子目录中的 SYSTEM. DRV 驱动程序处理硬件定时器中断。SYSTEM. DRV 在初始化期间设置一个新的中断 08H 向量地址，在 Windows 终止之前恢复原来的向量地址。SYSTEM. DRV 中的中断 08H 例程在做自己的处理之前调用原来的中断 08H 处理程序，从而要求此中断的其本系统函数仍可正常工作。

当 SYSTEM. DRV 接收到中断 08H 时，它调用 Windows USER 模块中的一个例程，此例程递减由 Windows 应用程序设置的每个定时器的计数。当计数器减到零时，USER 把 WM_TIMER 消息放入应用程序的消息队列中，并把计数器置为原来的值。

因为 Windows 应用程序从正常消息队列中取到 WM_TIMER 消息，从而用户程序在进行其他处理时不必担心被偶然的 WM_TIMER 消息中断了程序。在这个方面，定时器类似于键盘和鼠标。驱动程序处理异步硬件中断事件，Windows 把这些事件翻译成有次序、结构化、系列化的消息。

SYSTEM. DRV 不企望对 IBM PC 的 8259 定时器进行重新编程。Windows 定时器有如基于 PC 的定时器一样的的 54.925 ms 的分辨率。此事实有两个重要的潜在影响：

- 使用单个定时器时，Windows 应用程序不能接收速率快于每秒 18.2 次的 WM_TIMER 消息。

• 在 SetTimer 调用中指定的时间间隔总是截尾为时钟滴答的整数倍。例如，1000 ms 的间隔除以 54.925 ms，得到 18.207 个时钟滴答，截尾后为 18 个时钟滴答，它实际上为 989 ms。对每个小于 55 ms 的间隔，每个时钟滴答都产生一条 WM_TIMER 消息。

在用户的 Windows 程序中不要试图截取 ROM BIOS 时器中断，请使用 Windows 定时器。

7.1.2 定时器消息

Windows 系统中有 16 个定时器可以同时启动，可用函数 SetTimer 启动某一定时器。根据所指明的时间，产生 WM_TIMER 消息。如果函数检索到 NULL，则所有定时器都被指派，并且一个消息也不产生。对每个应用程序，若带有定时器，则应该提供返回值并立即作出响应；SetTimer 有四个参数：

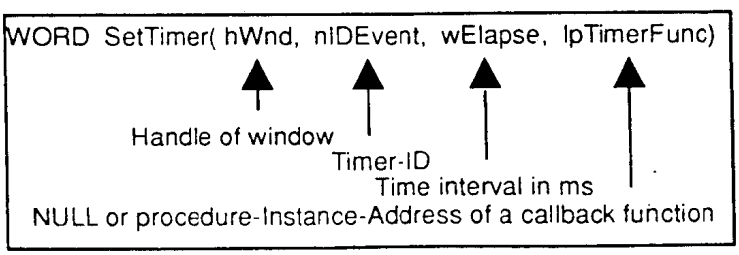


图 7.1 SetTimer 的四个参数

第三个参数指明时间间隔(以 ms 计)，因为定时器依赖于硬件定时器，消息 WM_TIMER 每秒钟发生的次数不能超过 18.2 次。在指定的时间间隔后消息也可能不会到达窗口过程。

这是由延迟造成的。比如消息是来自另一使用 CPU 较长时间的应用程序，因为消息 WM_TIMER 在应用程序队列中的优先级较低，并且 WM_TIMER 消息队列中一次只能有一个消息，所以如果没有及时从消息队列中移出就有可能丢失。

第四个参数通常设为 NULL。在这种情况下，WM_TIMER 消息传递给正常的窗口函数。也可包括回调函数的返回地址。这样使得回调函数而不是窗口过程接收到 WM_TIMER 消息。首先通过使用函数 MakeProcInstance 确定必要的地址。

随着一条 WM_TIMER 消息，消息产生所需的 ID 值在 wParam 中，lParam 参数没有使用。

设置的定时器一直运行，直到被函数 KillTimer 终止。因为一个应用程序可启动多个定时器，每条属于这个定时器的 WM_TIMER 消息都是由函数 KillTimer 从消息队列中取出的。

7.1.3 非异步定时器消息

为 IBM PC 及其兼容机编写的非 Windows 程序能通过截获中断 08H 或中断 1CH(由 BIOS 08H 中断处理程序调用的软中断)来使用定时器滴答中断。当硬中断产生时，当前运行的程序被挂起，控制传递到硬中断处理程序。在中断处理程序完成后，再把控制返回到被中断的程序。

用硬件键盘和鼠标中断一样，硬件定时器滴答中断有时也称为异步中断，因为它对所中

断的程序来说是随机生成的(实际上对定时器中断来说,使用等时中断比使用异步中断更加准确,因为中断是以相等的时间间隔发生的。但是对其他处理来说,中断仍然是异步的)。

虽然SYSTEM.DRV驱动程序也处理异步中断——08H时钟滴答,但是Windows送到应用程序的WM_TIMER消息并非是异步的。WM_TIMER消息放在正常的消息队列之中,与它消息有先后关系。因此,若在SetTimer调用中指定间隔为100 ms,则不能保证程序每1000 ms或989 ms会接收到一条WM_TIMER消息。若其他程序的运行时间超过1s,在此期间,用户程序将接收不到任何WM_TIMER消息。只有在其他程序把控制交给Windows(由调用GetMessage、PeekMessage或WaitMessage)时,用户程序才能从队列中取到下一条WM_TIMER消息。

使用Windows中的CLOCK程序很容易演示这一点。若另一个程序正忙于绘图作业而不放弃控制,CLOCK将会停止。当CLOCK得到控制时,它就跳过一步以校正时间。实际上,Windows对WM_TIMER的处理很像对WM_PAINT消息的处理。这两种消息的优先级都较低。若一个程序的消息队列中只含有WM_PAINT或WM_TIMER消息,而另一个程序的消息队列中含有非WM_PAINT或WM_TIMER的消息时,Windows将把控制传给后一个程序。

WM_TIMER还在另一方面同WM_PAINT相似;Windows不能持续在消息队列中装入多条WM_TIMER消息,而是将多条WM_PAINT组合成单条消息。因此,应用程序不能一次接收多条这样的消息,尽管可能在短时间内得到两条WM_TIMER消息。应用程序不能确定这种处理所导致的WM_TIMER消息"丢失"的数目。

当CLOCK再次得到控制后就跳前一步校正时间,并不是因为它一次得到几条WM_TIMER消息。CLOCK必须得到实际时间,然后再设置自己。WM_TIMER消息仅提示它应当更新。程序本身不能通过统计WM_TIMER消息数来保持时间。

为方便起见,下面在讨论时钟时,将使用"每秒得到一次WM_TIMER消息"的概念。但是这些消息并非精确的时间滴答中断。

7.2 定时器的用途

若想在整个程序期间使用定时器,那么从WinMain函数中或在处理WM_CREATE消息时调用SetTimer,并调用KillTimer响应WM_DESTROY消息。在WinMain中设置定时器提供了当定时器不存在时的最方便的错误处理。根据调用SetTimer时使用的参数,可以用以下三种方法之一使用定时器。

7.2.1 把WM_TIMER消息发送给应用程序的窗口过程

这是最方便的一种方法,它让Windows把WM_TIMER消息发送到应用程序的窗口过程。SetTimer调用的格式如下:

```
SetTimer (hwnd,1,wMsecInterval,NULL);
```

第一个参数是其窗口过程将接收WM_TIMER消息的窗口的句柄。第二个参数是定时器ID,它是一个非0数,在本例中假定为1。第三个参数是以ms为单位指定时间间隔的WORD类型(16位无符号整数)的值。当取最大值(65355)时大约每分钟传送一次WM_

TIMER 消息。

下列调用可以在任何时刻停止 WM_TIMER 消息(即使正在处理 WM_TIMER 消息):

```
KillTimer (hwnd,1);
```

此函数的第二个参数是 SetTimer 调用中所用的同一定时器。在终止程序之前,用户应当响应 WM_DESTROY 消息停止任何活动的定时器。当窗口过程接收到 WM_TIMER 消息时,wParam 参数等于定时器的 ID 值(上述情形为 1),lParam 为 0。如果需要设置多个定时器,则对每个定时器都使用不同的定时器 ID。wParam 的值将随传递到窗口过程的 WM_TIMER 消息的不同而不同。为了使程序具有更好的可读性,使用 #define 语句定义不同的定时器 ID:

```
#define TIMER_SEC 1
#define TIMER_MIN 2
```

使用两个 SetTimer 调用就可设置两个定时器:

```
SetTimer (hwnd,TIMER_SEC,1000,NULL);
SetTimer (hwnd,TIMER_MIN,6000,NULL);
```

WM_TIMER 逻辑如下:

```
case WM_TIMER:
    switch (wParam)
    {
        case TIMER_SEC:
            [once-per-second processing]
            break;
        case TIMER_MIN:
            [once-per-minute processing]
            break;
    }
    return 0;
```

若要把已存在的定时器设置为不同的时间间隔,那么停止该定时器并再次调用 SetTimer。假定定时器 ID 为 1,则代码如下:

```
KillTimer (hwnd,1);
    SetTimer (hwnd,1,wMsecInterval,NULL);
```

wMsecInterval 参数是以 ms 为单位的新时间间隔。Windows CLOCK 应用程序使用这个方法把定时器的间隔从 1000 ms 改为变成图标时的 60000 ms。从而当其为图标时,CLOCK 只需每分钟而不是每秒钟更新一次时钟。当从图标改为窗口时,CLOCK 又把定时器间隔改回到 1000 ms。

没有定时器时的处理

Windows 在任意时刻只允许 32 个活动定时器。若没有可用定时器,SetTimer 就返回 NULL 值。程序在没有定时器时应当有正确运行的能力。若程序真的需要定时器而不能获得定时器时,则没有其他选择而只有终止程序。若是从 WinMain 中调用 SetTimer,则通过从 WinMain 中返回 FALSE 就可终止程序。

假定用户需要一个 1000 ms 的定时器。在调用 CreateWindow 之后,在消息循环之前可能有这么一条语句:

```
if (! SetTimer (hwnd,1,1000,NULL))
```

```
        return FALSE;
```

这种终止方法不够友好。用户退出后却不知道为何不能装入应用程序。使用 Windows 消息框显示一条信息是一种友好的方法，实现起来也相当容易。

消息框是一种在显示器中间显示的弹出式窗口。消息框有一标题条而无缩放区。标题条通常含有应用程序的名称。消息框中包含一条信息和一、二或三个按钮（为 OK，Retry，Cancel，Yes，No 等一些组合）。消息框还含有一个预定义图标：小写 i（它代表“information”）、一个感叹号、一个问号或一个句号。当运行 Windows 时，可以看到许多消息框。

下面一段代码在 SetTimer 分配一个定时器失败时产生一个消息框：

```
if (! SetTimer (hwnd,1,1000,NULL))
{
    MessageBox (hwnd,
        "Too many clocks or timers!",
        "Program Name",
        MB_ICONEXCLAMATION|MB_OK);
    return FALSE;
}
```

此消息框如图 7.2 所示。当按下 Enter 键或按下 OK 按钮时，WinMain 由于返回 FALSE 而终止。

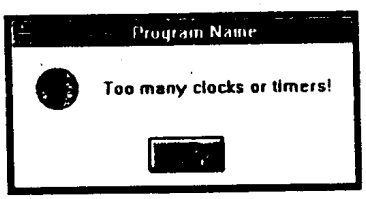


图 7.2 “友好终止”的消息框

缺省时，消息框是“应用程序模式”窗口，这意味着用户在应用程序继续运行之前要响应消息框。不过，通过按 Alt－Tab、Alt－Esc 或在另一程序窗口按下鼠标按钮，就可以切换到其他应用程序。

当然可以再给用户提供一个机会，在关闭使用定时器的其他应用程序之后，再成功地装入用户应用程序，此时代码如下：

```
while (! SetTimer (hwnd,1,1000,NULL))
    if (IDCANCEL == MessageBox (hwnd,
        "Too many clocks or timers!",
            "Program Name",
            MB_ICONEXCLAMATION|MB_RETRYCANCEL))
return FALSE;
```

这个消息框如图 7.3 所示，它有标为 Retry 和 Cancel 的两个按钮。若用户选择 Cancel，MessageBox 函数返回一个等于 IDCANCEL 的值，程序终止。若用户选择 Retry，SetTimer 被再次调用。

示例程序

下面是一个使用定时器的示例程序。这个称为 BEEPER1 的程序把定时器的时间间隔设

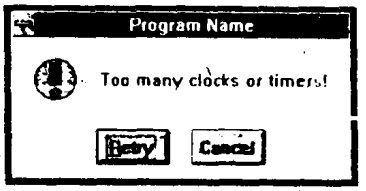


图 7.3　提供一个选择的消息框

置为 1s。当接收到 WM_TIMER 消息时，它把客户区的颜色在红蓝之间切换，并调用 MessageBeep 函数发出蜂鸣声(虽然 MessageBeep 是 MessageBox 类的相关函数，但它经常用于各种蜂鸣目的。MessageBeep 的 UINT 参数可以是任意值)。BEEPER1 在 WinMain 函数中设置定时器，在 WndProc 窗口过程中处理 WM_TIMER 消息。在处理 WM_TIMER 消息期间，BEEPER1 调用 MessageBeep，转换 bFlipFlop 的值，使窗口无法产生 WM_PAINT 消息。在处理 WM_PAINT 消息期间，BEEPER1 通过调用 GetClientRect 获得存放窗口尺寸的 RECT 结构，通过调用 FillRect 获得窗口颜色。

```
# BEEPER1.MAK make file

beeper1.exe : beeper1.obj beeper1.def
     $(WINLINK) beeper1,beeper1,NUL, $(WINLIB),beeper1
     rc -t beeper1.exe

beeper1.obj : beeper1.c
     $(WINCC) beeper1.c

/*----------------------------------------
BEEPER1.C -- Timer Demo Program No. 1
----------------------------------------*/

#include <windows.h>

#define ID_TIMER 1

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
     {
          static char szAppName[] = "Beeper1" ;
          HWND hwnd ;
          MSG msg ;
          WNDCLASS wndclass ;

          if (! hPrevInstance)
          {
               wndclass.style = CS_HREDRAW | CS_VREDRAW ;
               wndclass.lpfnWndProc = WndProc ;
               wndclass.cbClsExtra = 0 ;
               wndclass.cbWndExtra = 0 ;
```

```
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName,"Beeper1 Timer Demo",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,CW_USEDEFAULT,
            CW_USEDEFAULT,CW_USEDEFAULT,
            NULL,NULL,hInstance,NULL) ;

    while (! SetTimer (hwnd,ID_TIMER,1000,NULL))
        if (IDCANCEL == MessageBox (hwnd,
            "Too many clocks or timers!",szAppName,
            MB_ICONEXCLAMATION | MB_RETRYCANCEL))
        return FALSE ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static BOOL fFlipFlop = FALSE ;
    HBRUSH hBrush ;
    HDC hdc ;
    PAINTSTRUCT ps ;
    RECT rc ;

    switch (message)
    {
        case WM_TIMER:
            MessageBeep (0) ;

            fFlipFlop = ! fFlipFlop ;
            InvalidateRect (hwnd,NULL,FALSE) ;

            return 0 ;
```

```
        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;

            GetClientRect (hwnd,&rc) ;

            hBrush = CreateSolidBrush (fFlipFlop ? RGB(255,0,0) :
                     RGB(0,0,255)) ;
            FillRect (hdc,&rc,hBrush) ;
            EndPaint (hwnd,&ps) ;
            DeleteObject (hBrush) ;
            return 0 ;
        case WM_DESTROY:
            KillTimer (hwnd,ID_TIMER) ;
            PostQuitMessage (0) ;
            return 0;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; BEEPER1.DEF module definition file
NAME BEEPER1
DESCRIPTION 'Timer Demo Program No. 1 (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

因为BEEPER1通过发声来指示它接收到一条WM_TIMER消息，所以装入BEEPER1并在Windows中实现其他一些活动，就可很好地理解WM_TIMER消息的不稳定性。例如，移动或缩放BEEPER1窗口。这将导致程序进入“模态消息循环”。Windows通过在Windows内部消息循环中俘获所有消息而非通过用户程序的消息循环来禁止对移动或缩放操作的任何干扰。通过循环到达程序窗口的大少数消息都被丢弃，这就是BEEPER1停止蜂鸣的原因。当完成了移动与缩放之后，用户将会注意到BEEPER1不能取得它所丢弃的所有WM_TIMER消息，尽管前两条消息的间隔可能少于1s。

Windows中的颜色

Windows使用一个无符号长(32位)整数值来表示一种颜色。如图7.4所示，最低三个字节指定范围为0至255的红、绿、蓝值。这导致调色板可能有224(大约16000000)种颜色。

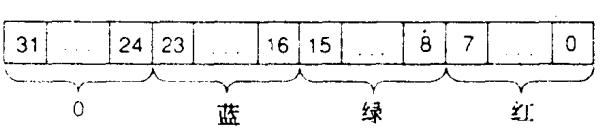


图7.4 32位颜色值

这个无符号长整数经常被称为“RGB”颜色。WINDOWS.H头文件提供了几个处理RGB

颜色值的宏。WINDOWS.H 中的 RGB 宏有三个分别表示红、绿、蓝值的参数，它把三个值组成无符号长整数：

```
#define RGB (r,g,b) ((DWORD) (((BYTE) (r)|\
    ((WORD)(g) << 8))|\
    (((DWORD)(BYTE)(b)) << 16)))
```

从而值：

```
RGB (255,0,255)
```

为 0x00FF00FF，它是洋红色的 RGB 颜色值。当所有三个参数都设置为 0 时，颜色为黑色。当所有三个参数都设置为 255 时，颜色为白色。GetRValue、GetGValue、GetBValue 宏从无符号长整数 RGB 颜色值中抽取出无符号字符格式的单色值。当使用把 RGB 颜色返回给用户程序的 Windows 函数时，所有这些宏使用起来非常方便。

Windows 使用的通用视频显示适配器是 VGA 或 EGA。在 Windows 使用的显示器分辨率中，这两种显示器都能显示 16 种不同的颜色(一些“Super VGA”卡在 Windows 下能显示 256 种不同颜色，虽然 Windows 通常只使用其中的 20 种颜色。对其他 256 种颜色的使用要求使用 Windows 调色板管理程序)。Windows 通过“高频振荡”能显示其他颜色，它创建一个组合不同纯颜色的像素模式。

并非所有红、绿、蓝字字节的组合都能产生不同的高频振动模式。例如，在彩色 EGA 或 VGA 上，红、绿、蓝值通常递增 4 才能产生一种不同的高频振动模式。从而这些适配器有 218 或(262144)种高频振动颜色。

BEEPER1 使用 FillRect 函数给客户区着色。FillRect 的第一个参数是设备环境句柄，第二个参数是指向 RECT 结构的指针，第三个参数是指向“刷子”的句柄。刷子是 Windows 用于填充一个区域的图形对象。刷子可以是实颜色或者由不同隐线或图案组合而成。

BEEPER1 调用 CreateSolidBrush 创建一个实颜色刷子。唯一的参数是 RGB 颜色值。依赖于 fFlipFlop 的值，BEEPER 把这个参数设置为 RGB(255,0,0)或 RGB(0,0,255)，前者为红色，后者为蓝色。

刷子是一个图形对象。如果创建了一个刷子，那么在用完之后必须删除它。在调用 FillRect 之后，BEEPER1 调用 DeleteObject 删除刷子。

7.2.2 把 WM_TIMER 消息发送到通常的窗口过程

设置定时器的第一种方法是把 WM_TIMER 消息发送到通常的窗口过程。本方法是让 Windows 直接把定时器消息发送给用户程序的另一个函数。

接收这些定时器消息的函数称为“回调”函数。这是一个在用户程序之中但由 Windows 调用的函数。用户先告诉 Windows 此函数的地址，然后 Windows 调用此函数。这看起来也很熟悉，因为程序的窗口过程实际上也是一类回调函数。当登录窗口时要将函数的地址告诉 Windows，当发送消息给程序时，Windows 调用此函数。不过，对非窗口过程的回调函数的处理有些不同。

SetTimer 并非是唯一使用回调函数的 Windows 函数。CreateDialog 和 DialogBox 函数使用回调函数处理对话框中的消息；由几个 Windows 函数(EnumChildWindows、EnumFonts、EnumObjects、EnumProps 和 EnumWindows)把枚举信息传递给回调函数，还有几个不那么常

用的函数(QrayString、LineDDA、SetResoorceHandler 和 SetWindowsHook)也要求回调函数。对刚开始用 Windows 编程的人员来说,回调函数是使系统挂起的重要原因。可能会出现一些古怪的事情。下面我们首先介绍如何使用回调函数,然后讨论这样做的原因。

像窗口过程一样,回调函数也必须定义为 FAR PASCAL _export,因为它是由 Windows 从程序的代码段外调用的。回调函数的参数和回调函数的返回值依赖于回调函数的目的。同定时器有关的回调函数中,输入参数同窗口过程的输入参数一样。定时器回调函数返回 UINT 值给 Windows。

我们把以下回调函数称为 TimerProc,它仅处理 WM_TIMER 消息。

```
UINT FAR PASCAL _export TimerProc (HWND hwnd,UINT message,
        UNIT wParam,LONG lParam)
{
    [process WM_TIMER message]
    return 0;
}
```

hwnd 输入参数是在调用 SetTimer 时指定的窗口句柄。Windows 仅把 WM_TIMER 消息送给 TimerProc,从而 message 参数总是等于 WM_TIMER。wParam 的值为定时器 ID,lParam 的值为系统时间。

前面已提到过,用第一种方法设置定时器时要求下面格式的 SetTimer 调用:

```
SetTimer (hwnd,wMsecInterval,NULL);
```

当使用回调函数处理 WM_TIMER 消息时,SetTimer 的第四个参数由回调函数的远程地址所取代。

但事实并非如此。因为作为 SetTimer 第四个参数传递到 Windows 的远程地址并非程序中此函数的地址。它由从 Windows 中的 MakeProcInstance 函数获得的远程地址所取代。为了使用 MakeProcInstance,首先要定义一个指向一个函数的远程指针作为参数。对此定义可使用 WINDOWS. H 中的标识符 FARPROC:

```
FARPROC lpfnTimerProc;
```

在 WinMain(或者是对每个实例只执行一次的程序段)中,调用 MakeProcInstance。MakeProcInstance 的参数是 TimerProc 的地址和 hInstance 的值。MakeProcInstance 返回一个保存在 lpfnTimerProc 中的远程指针:

```
lpfnTimerProc = MakeProcInstance ((FARPROC) TimerProc,hInstance);
```

现在调用 SetTimer 时可使用 lpfnTimerProc 的值:

```
SetTimer (hwnd,1,wMsetIntervel,lpfnTimerProc);
```

示例程序

首先看一段示例代码就会了解这些东西是如何组合在一起的,然后再进行一步讨论 MakeProcInstance。在功能上,除 Windows 发送一个定时器消息给 TimerProc 而非 WndProc 之外,如下所示的 BEEPER2 程序与 BEEPER1 是相同的。为了告诉 C 编译器 TimerProc 是一个函数,从而调用 MakeProcInstance 时我们可以使用此函数名;我们在程序的顶部同 WndProc 一起声明。注意,程序对每个实例都调用 MakeProcInstance。

```
# BEEPER2. MAK make file
```

```
beeper2.exe : beeper2.obj beeper2.def
    $(WINLINK) beeper2,beeper2,NUL, $(WINLIB),beeper2
    rc -t beeper2.exe

beeper2.obj : beeper2.c
    $(WINCC) beeper2.c
/*----------------------------------------
BEEPER2.C -- Timer Demo Program No. 2
----------------------------------------*/

#include <windows.h>

#define ID_TIMER 1

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;
UINT FAR PASCAL _export TimerProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
    static char szAppName[] = "Beeper2" ;
    FARPROC lpfnTimerProc ;
    HWND hwnd ;
    MSG msg ;
    WNDCLASS wndclass ;

    if (! hPrevInstance)
    {
        wndclass.style = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc = WndProc ;
        wndclass.cbClsExtra = 0 ;
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName,"Beeper2 Timer Demo",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    lpfnTimerProc = MakeProcInstance ((FARPROC) TimerProc,hInstance) ;

    while (! SetTimer (hwnd,ID_TIMER,1000,lpfnTimerProc))
```

```
        if (IDCANCEL == MessageBox (hwnd,
            "Too many clocks or timers!",szAppName,
            MB_ICONEXCLAMATION | MB_RETRYCANCEL))
    return FALSE ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    switch (message)
    {
        case WM_DESTROY:
            KillTimer (hwnd,ID_TIMER) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

UINT FAR PASCAL _export TimerProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    static BOOL fFlipFlop = FALSE ;
    HBRUSH hBrush ;
    HDC hdc ;
    RECT rc ;

    MessageBeep (0) ;
    fFlipFlop = ! fFlipFlop ;

    GetClientRect (hwnd,&rc) ;

    hdc = GetDC (hwnd) ;
    hBrush = CreateSolidBrush (fFlipFlop ? RGB(255,0,0) : RGB(0,0,255)) ;

    FillRect (hdc,&rc,hBrush) ;
    ReleaseDC (hwnd,hdc) ;
    DeleteObject (hBrush) ;

    return 0 ;
}
```

```
; BEEPER2. DEF module definition file

NAME BEEPER2

DESCRIPTION 'Timer Demo Program No. 2 (c) Charles Petzold, 1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

正确处理回调函数

下面是关于回调函数的两条规则。用户程序中由 Windows 调用的任意函数处理如下：

(1)该函数应当定义为 FAR PASCAL _expert。

(2)送给 Windows 的函数地址是 MakeProcInstance 的返回值(这条规则不适用于 RegisterClass 调用中作为窗口类结构的一部分传递到 Windows 的窗口过程。此时 Windows 本身处理 MakeProcInstance 要求)。

这些规则源于 Windows 1.0 的时代，那时 Windows 大量运行于 8088 和 80286 微处理器的实模式下。对运行同一程序的几个实例的 Windows 来说(这些实例有相同的代码段和各自的数据段)，这些规则是很有必要的。Windows 也要求这一点，它才能在内存中移动代码段和数据段。下面是两条规则的实际结果：

(1)当把函数定义为 FAR 类型，并用_Gw(Windows)标志编译时，编译器在函数中插入特殊的起始与结束代码。在汇编语言中，起始代码使用 MOV AX,DS 指令把 AX 寄存器设置为 DS 的值。然后保存 DS 的值(PUSH DS)并用 MOV DS,AX 指令使 DS 等于 AX 的值。在函数的结尾，结束代码从堆栈中弹出原来的 DS 值。

(2)若 FAR 函数也是输出的(即使用_export 关键字定义的函数)，那么 Windows 在把代码装入内存时，用 MOV AX,DS 指令取代函数顶部由 NOP(空操作)起始的指令。

(3)MakeProcInstance 创建一小段代码——称为"thunk"。从 MakeProcInstance 返回的远程地址即是此 thunk 的地址。thunk 把数据段的段地址装入 AX，并跳转到该函数。然后函数从 AX 中装入 DS。

MakeProcInstance 要求 hInstance 参数。对每个实例必须使用 MakeProcInstance 创建一个不同的 thunk，因为每个实例都需要自己的数据段。对于一个特定的回调函数，每个实例的 thunk 都连接到同一函数地址(因为所有实例都使用相同的代码段)，但每个 thunk 把 AX 置为不同的数据段——该实例的数据段。当 Windows 在内存中移动数据段时，它必须修改该实例的 thunk，从而 thunk 把 AX 设为一个新的数据段地址。thunk 本身总是位于一个不能移动的内存中。

更复杂的情形是在用户程序中对 FAR 函数的引用——如传递到 MakeProcInstance 调用的地址和 thunk 跳转到的地址——甚至不是用户程序中函数的地址。在 thunk 和实际函数之间还有一个小例程。若此例程还未被装入内存或已从内存中删除，则把代码段装入内存。

这些规则暗示用户不能在其程序中直接调用远程输出函数。例如，用户或许想用下面的语句模拟定时器消息：

```
TimerProc (hwnd,WM_TIMER,1,0L);                    // WRONG !!!
```

这条语句好像没错，但实际上是错的，TimerProc 的前缀把 DS 设为 AX 的值，但 AX 的值不缺定，它很可能不是程序数据段的段地址。若需要从用户程序中直接调用一个输出函数，则可使用 MakeProcInstance 返回的远程指针。

```
(*lpfnTimerProc) (hwnd,WM_TIMER,1,0L);             // RIGHT
```

此代码调用 thunk，thunk 在分支到函数之前把 AX 的值设置为正确的数据段。

近几年来情况又有了一些变化。Windows 3.0 支持保护模式下的 Windows 操作和 Windows 应用程序。Windows 3.1 不再支持实模式，保留了这个趋势。虽然运行在保护模式下的应用程序打破了这些规则，但最好还是继续观察它们。

7.2.3 由函数处理 WM_TIMER 消息

设置定时器的第三种方法类似于第二种方法。它要求有一个由 MakeProcInstance 创建的指向处理 WM_TIMER 消息的函数的指针。不过 SetTimer 的 hwnd 参数设为 NULL，第二个参数(定时器 ID)不同。最后，此函数返回定时器 ID。

```
nTimerID = SetTimer (NULL,0,wMsecInterval,lpfnTimerProc);
```

若没有可用的定时器，从 SetTimer 返回的 nTimerID 值为 NULL。

killTimer 的第一个参数(通常是窗口过程)也必须是 NULL。定时器 ID 必须是 SetTimer 的返回值：

```
KillTimer (NULL,nTimerID);
```

传递到 TimerProc 定时器函数的 hwnd 参数也为 NULL。wParam 参数为定时器 ID，lParam 参数为 lpfnTimerProc，同第二种方法一样。

本方法很少使用。若在用户程序中有一系列不同时刻的 SetTimer 调用，而不希望跟踪用户使用的是哪个定时器，则使用此方法是很方便的。

在介绍了如何使用 Windows 定时器之后，就可以介绍一些有用的定时器程序了。

7.3 使用定时器报告状态

Windows 定时器的应用之一是周期性地更新屏幕上显示的状态报告。程序在接收到 WM_TIMER 消息之前放弃控制，从而不过多的使用处理时间。下面所示的 FREEMEM 程序显示了 Windows 中的可用自由内存量。自由内存量每秒更新一次，并同程序管理器和文件管理器的 About 框中显示的数值一致。FREEMEM 可以告诉用户 Windows 是否快用完内存。当测试一个新的 Windows 程序时，FREEMEM 能指示程序是如何分配和释放内存的。

```
# FREEMEM.MAK make file

freemem.exe : freemem.obj freemem.def
    $(WINLINK) freemem,freemem,NUL,$(WINLIB),freemem
    rc -t freemem.exe

freemem.obj : freemem.c
    $(WINCC) freemem.c
```

```
/ * ----------------------------------------
FREEMEM. C -- Free Memory Display Program
---------------------------------------- * /

#include (windows. h)
#include (stdio. h)

#define ID_TIMER 1

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "FreeMem" ;
     HDC hdc ;
     HWND hwnd ;
     MSG msg ;
     TEXTMETRIC tm ;
     WNDCLASS wndclass ;

     if (hPrevInstance)
          return FALSE ;

     wndclass. style = CS_HREDRAW | CS_VREDRAW ;
     wndclass. lpfnWndProc = WndProc ;
     wndclass. cbClsExtra = 0 ;
     wndclass. cbWndExtra = 0 ;
     wndclass. hInstance = hInstance ;
     wndclass. hIcon = NULL ;
     wndclass. hCursor = LoadCursor (NULL,IDC_ARROW) ;
     wndclass. hbrBackground = GetStockObject (WHITE_BRUSH) ;
     wndclass. lpszMenuName = NULL ;
     wndclass. lpszClassName = szAppName ;

     ! RegisterClass (&wndclass) ;

     hwnd = CreateWindow (szAppName,"Free Memory",
               WS_OVERLAPPEDWINDOW,
               CW_USEDEFAULT,CW_USEDEFAULT,
               CW_USEDEFAULT,CW_USEDEFAULT,
               NULL,NULL,hInstance,NULL) ;

     hdc = GetDC (hwnd) ;
     GetTextMetrics (hdc,&tm) ;
     ReleaseDC (hwnd,hdc) ;

     if (4 * tm. tmAveCharWidth ) GetSystemMetrics (SM_CXICON) ||
               2 * tm. tmHeight ) GetSystemMetrics (SM_CYICON))
     {
          MessageBox (hwnd,"Icon size too small for display!",
```

```
            szAppName,MB_ICONEXCLAMATION | MB_OK) ;
        return FALSE ;
    }

    if (! SetTimer (hwnd,ID_TIMER,1000,NULL))
    {
        MessageBox (hwnd,"Too many clocks or timers!",
            szAppName,MB_ICONEXCLAMATION | MB_OK) ;
        return FALSE ;
    }

    ShowWindow (hwnd,SW_SHOWMINNOACTIVE) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    static DWORD dwFreeMem,dwPrevMem ;
    static RECT rect ;
    char cBuffer [20] ;
    HDC hdc ;
    PAINTSTRUCT ps ;

    switch (message)
    {
        case WM_TIMER:
            dwFreeMem = GetFreeSpace (0) ;
            if (dwFreeMem ! = dwPrevMem)
                InvalidateRect (hwnd,NULL,TRUE) ;
            dwPrevMem = dwFreeMem ;
            return 0 ;
        case WM_SIZE:
            GetClientRect (hwnd,&rect) ;
            return 0 ;
        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;
            DrawText (hdc,cBuffer,
                sprintf (cBuffer,"%. 2f megs",
                    dwFreeMem / 1024. 0 / 1024. 0),
                &rect,DT_WORDBREAK) ;
            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_QUERYOPEN:
```

```
            return 0 ;
        case WM_DESTROY:
            KillTimer (hwnd,ID_TIMER) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; FREEMEM. DEF module definition file

NAME FREEMEM

DESCRIPTION 'Free Memory Display (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

因为 FREEMEM 不需要很多的显示空间，所以我们将其设计成在 Windows 屏幕底部显示的图标。

图 7.5 FREEMEM 图标

7.3.1 图标的使用

同大多数 Windows 程序不一样，FREEMEM 以图标的形式运行，它在图标中显示需要显示的一切内容，但不能展开成正常的窗口。大多数 Windows 应用程序有一个在窗口类结构中指定的静态图形图标。这些图标通常是由 Windows 软件开发工具箱所提供的 Image Editor 实用程序创建的。所以我们已经在程序中使用了预定义的图标。另一方面，用户可以用下面的语句在窗口类结构中指定 NULL 图标：

```
wndClass. hIcon = NULL;
```

NULL 图标意味着由应用程序负责绘制图标。当需要绘出图标时，Windows 把 WM_PAINT 消息送到应用程序，从而在程序运行时可以修改图标的形状。Windows 中的 CLOCK 程序就是使用这种技术显示时钟的，即使窗口是一个图标时亦然。NULL 图标实际上是一个小窗口，用户可以像在正常窗口的客户区域中一样在此窗口中进行绘制。若用户需要知道何时应用程序成为一个图标，则可从 WM_ SIZE 消息的 wParam 参数中获知这一消息。例如，当 CLOCK 成为图标时，它从时钟中删除了第二根指针，并把定时器的间隔从 1s 改为 1 min。程序调用 IsIconic 函数也能确定这一窗口是否为图标。

FREEME 在图标中显示两行文本。这在 Windows 使用的大多数常见的显示器上都能运行，但是在某些使用较大系统字模的高分辨率卡上可能不能运行。因为 FREEME 在这些视

频卡上不能正确地显示。所以它检查系统字模的大小和图标的大小。若图标尺寸太小，它就使用消息框告诉用户图标太小。

7.3.2 图标的强制使用

在进入 WinMain 消息循环之前，大多数应用程序执行函数：

```
ShowWindow (hwnd,nCmdShow);
```

nCmdShow 参数作为 WinMain 的参数传递给程序。

如果在文件管理器中运行一个程序(从 File 菜单中选择 Run，当光标在该程序名上时按 Enter 或连按二次鼠标按钮)，nCmdShow 设置为 SW_SHOWNORMAL。若装入一个图标形式的应用程序(在 Run 对话框中选择 Run Minimized 选项，或当光标在程序名上时按下 Shift—Enter)，则 nCmdShow 置为 SW_SHOWMINNOACTIVE。应用程序一般不需要画出图标，而只需把这个参数送给 ShowWindow。

不过，并不是一定要在 ShowWindow 中使用 nCmdShow 参数。FREEMEM 使用语句：

```
ShowWindow (hwnd,SW_SHOWMINNOACTIVE);
```

这就强制窗口以图标的形式出现，而不管 nCmdShow 的值。活动程序仍然是活动的。

使用这个技术可以实现一些小技巧，若用户希望某一应用程序首先以全屏幕的形式出现，则使用：

```
ShowWindow (hwnd,SW_SHOWMAXIMIZED);
```

7.3.3 自由内存

当 Windows 把 WM_TIMER 消息发送给 FREEMENT 时，FREEMEM 必须决定自由内存量。像程序管理器和文件管理器中的 About 框一样，FREEMEM 调用带有一个 0 参数的 GetFreeSpace 来得到自由内存量。在 Windows 3.0 中，GetFreeSpace 函数是新增的。对 Windows 3.0 以前的版本，Windows 程序使用 GlobalCompact 来获得系统中的最大的连续自由内存块，这个值不如自由内存总量那么有用。

如果自由内存量由于最近一个 GetFreeSpace 的调用而发生了改变，则 FREEME 使客户区无法产生 WM_PAINT 消息。FREEMEM 调用 DrawText 处理 WM_PAINT 消息，此函数是用于简单的字上卷文本的函数。FREEMEM 把以字节为单位的自由内存量转换为以 MB 为单位的浮点值，Sprintf 以二位十进制格式来保存这个值。

7.4 时 钟

时钟是定时器最明显的应用，虽然数字时钟曾经流行一时，但是模拟形式仍占主流。在 Windows 中已经有一个模拟时钟。虽然 CLOCK 程序有一个数字时钟设置，但是我们还要编写一个更好的数字时钟程序——因为它提供了定时器应用的一个有趣的例子。下面所示为 DIGCLOCK 程序，它创建一个弹出式窗口，此窗口位于显示器右下角的图标区。此程序显示星期、时间和日期。

```
# DIGCLOCK.MAK make file
```

```
digclock.exe : digclock.obj digclock.def
     $(WINLINK) digclock,digclock,NUL, $(WINLIB),digclock
     rc -t digclock.exe

digclock.obj : digclock.c
     $(WINCC) digclock.c
/*----------------------------------------
DIGCLOCK.C -- Digital Clock Program
----------------------------------------*/

#include <windows.h>
#include <time.h>

#define ID_TIMER 1

#define YEAR (datetime->tm_year % 100)
#define MONTH (datetime->tm_mon + 1)
#define MDAY (datetime->tm_mday)
#define WDAY (datetime->tm_wday)
#define HOUR (datetime->tm_hour)
#define MIN (datetime->tm_min)
#define SEC (datetime->tm_sec)

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;
void SizeTheWindow (short *,short *,short *,short *) ;

char sDate[2],sTime[2],sAMPM[2][5] ;
int iDate,iTime ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "DigClock" ;
     HWND hwnd;
     MSG msg;
     short xStart,yStart,xClient,yClient ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = NULL ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = NULL ;
          wndclass.lpszClassName = szAppName ;
```

```
        RegisterClass (&wndclass) ;
    }

    SizeTheWindow (&xStart,&yStart,&xClient,&yClient) ;

    hwnd = CreateWindow (szAppName,szAppName,
        WS_POPUP | WS_DLGFRAME | WS_SYSMENU,
        xStart, yStart,
        xClient,yClient,
        NULL,NULL,hInstance,NULL) ;

    if (! SetTimer (hwnd,ID_TIMER,1000,NULL))
    {
        MessageBox (hwnd,"Too many clocks or timers!",szAppName,
        MB_ICONEXCLAMATION | MB_OK) ;
        return FALSE ;
    }

    ShowWindow (hwnd,SW_SHOWNOACTIVATE) ;
    UpdateWindow (hwnd);

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam;
}

void SizeTheWindow (short * pxStart, short * pyStart,
        short * pxClient,short * pyClient)
{
    HDC hdc ;
    TEXTMETRIC tm ;

    hdc = CreateIC ("DISPLAY",NULL,NULL,NULL) ;
    GetTextMetrics (hdc,&tm) ;
    DeleteDC (hdc) ;

    * pxClient = 2 * GetSystemMetrics (SM_CXDLGFRAME) + 16 * tm. tmAveCharWidth ;
    * pxStart = GetSystemMetrics (SM_CXSCREEN) - * pxClient ;
    * pyClient = 2 * GetSystemMetrics (SM_CYDLGFRAME) + 2 * tm. tmHeight ;
    * pyStart = GetSystemMetrics (SM_CYSCREEN) - * pyClient ;
}

void SetInternational (void)
{
    static char cName [] = "intl" ;

    iDate = GetProfileInt (cName,"iDate",0) ;
    iTime = GetProfileInt (cName,"iTime",0) ;
```

```
    GetProfileString (cName,"sDate", "/",sDate, 2) ;
    GetProfileString (cName,"sTime", ":",sTime, 2) ;
    GetProfileString (cName,"s1159","AM",sAMPM [0],5) ;
    GetProfileString (cName,"s2359","PM",sAMPM [1],5) ;
}

void WndPaint (HWND hwnd,HDC hdc)
{
    static char szWday[] = "Sun\0Mon\0Tue\0Wed\0Thu\0Fri\0Sat" ;
    char cBuffer[40] ;
    RECT rect ;
    short nLength ;
    struct tm * datetime ;
    time_t lTime ;

    time (&lTime) ;
    datetime = localtime (&lTime) ;

    nLength = wsprintf (cBuffer," %s %d%s%02d%s%02d \r\n",
            (LPSTR) szWday + 4 * WDAY,
        iDate == 1 ? MDAY : iDate == 2 ? YEAR : MONTH,(LPSTR) sDate,
        iDate == 1 ? MONTH : iDate == 2 ? MONTH : MDAY, (LPSTR) sDate,
        iDate == 1 ? YEAR : iDate == 2 ? MDAY : YEAR) ;

    if (iTime == 1)
        nLength += wsprintf (cBuffer + nLength," %02d%s%02d%s%02d ",
        HOUR,(LPSTR) sTime,MIN,(LPSTR) sTime,SEC) ;
    else
    nLength += wsprintf (cBuffer + nLength," %d%s%02d%s%02d %s ",
        (HOUR % 12) ? (HOUR % 12) : 12,
        (LPSTR) sTime,MIN,(LPSTR) sTime,SEC,
        (LPSTR) sAMPM [HOUR / 12]) ;

    GetClientRect (hwnd,&rect) ;
    DrawText (hdc,cBuffer,-1,&rect,DT_CENTER | DT_NOCLIP) ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    HDC hdc ;
    PAINTSTRUCT ps;

    switch (message)
    {
        case WM_CREATE :
            SetInternational () ;
            return 0 ;
        case WM_TIMER :
            InvalidateRect (hwnd,NULL,FALSE) ;
            return 0 ;
```

```
        case WM_PAINT :
            hdc = BeginPaint (hwnd,&ps) ;
            WndPaint (hwnd,hdc) ;
            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_WININICHANGE :
            SetInternational () ;
            InvalidateRect (hwnd,NULL,TRUE) ;
            return 0 ;
        case WM_DESTROY :
            KillTimer (hwnd,ID_TIMER) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; DIGCLOCK. DEF module definition file

NAME DIGCLOCK

DESCRIPTION 'Digital Clock (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

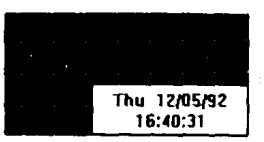


图 7.6 DIGCLOCK 窗口

上面列出的所有程序都把 CreateWindow 函数的第三个参数设置为窗口风格 WS_OVERLAPPEDWINDOW。DIGCLOCK 使用窗口风格：

WS_POPUP | WS_DIGFRAME | WS_SYSMENU

这就创建了一个带有对话框和系统菜单的称为“弹出式”的窗口风格。弹出风格通常用于对话框和消息框，应用程序很少使用。DIGCLOCK 还使用了 ShowWindow 调用的另一变体：

ShowWindow (hwnd,SW_SHOWNOACTIVATE);

通常情况下，运行一个程序时该程序就成为活动窗口。SW_SHOWNOACTIVATE 告知 Windows，那个装入 DIGCLOCK 的程序应一直保持为活动窗口。不过，在该窗口上按鼠标按钮或者按 Alt—Tab 或 Alt—Esc 也能使 DIGCLOCK 成为活动窗口。虽然 DIGCLOCK 没有系统菜单框，但是当 DIGCLOCK 活动时按 Alt—Spacebar 仍可访问系统菜单。若用户选择 Move，则可以用键盘移动窗口。

7.4.1 弹出式窗口的定位和缩放

DIGCLOCK 弹出式窗口定位在显示器的右下角。窗口必须能够容纳 2 行，每行 16 个字符。DIGCLOCK. C 中的 SizeTheWindow 过程决定在 CreateWindow 调用中使用的正确参数。一般情况下，程序在第一次创建窗口之前不能获得文本的大小，因为它需要用窗口句柄来得到设备环境句柄。DIGCLOCK 使用 CreateIC 来获得屏幕设备环境句柄的信息以解决上述问题。此函数类似于 CreateDC，但是它用于从设备环境中获取信息。文本尺寸同 GetSystemMetrics 中可用的消息相组合就可推导出最初的起始位置及窗口尺寸。

7.4.2 获取日期和时间

在 WndPaint 函数中，DIGCLOCK 使用 C 函数 time 和 localtime 来决定当前日期与时间。localtime 函数把我们需要的所有信息都放入一个结构中，在程序顶端定义的几个宏使 wsPrintf 调用具有更强的可读性（在 Windows 程序中要避免使用 MS-DOS 或 ROM BIOS 函数，而使用 Windows 函数或 C 运行库）。

7.4.3 国际化考虑

Windows 包括国际化支持。在 Windows 安装期间创建的 WIN. INI 文件含有以[intl]开头的部分。它列出了日期、时刻、货币和数字的格式。可以用以下三种格式来显示日期：月－日－年、年－月－日或日－月－年。这三个数字间的分隔符可以是斜杠、破折号、句号或其他任意字符。也可以用 12 小时或 24 小时的格式显示时间，时、分、秒之间通常用冒号或句号分开。

DIGCLOCK 中的 SetInternational 函数使用 Windows 中的 GetProfileInt（对整数）和 GetProcfileString（对字符串）函数从 WIN. INI 文件中取得格式信息。这些调用必须含有 Windows 不能在 WIN. INI 中找到所需值时的缺省值。SetInternational 在一个全局变量中保存这个值，这个全局变动量与 WIN. INI 中的字符串同名。WndPaint 函数使用 WIN. INI 获得的值来格式化日期与时刻的显示，然后调用 DrawText 在窗口中显示这两行文本。

DIGCLOCK 的窗口过程一接收到 WM_TIMER 消息，它就使窗口无法产生 WM_PAINT 消息。但是 WndProc 在接收到 WM_WININICHANGE 消息时也使窗口无效。任意一个修改 WIN. INI 的应用程序都把 WM_WININICHANGE 消息发送给所有活动窗口。若 WIN_INI 的[intl]部分被修改，则 DIGCLOCK 就会接到通知并得到新的通用信息。装入 DIGCLOCK 和 Windows 中的 CONTROL PANEL 程序，选择国际化图标，然后修改日期格式、日期分隔符、时刻格式或时刻分隔符，用户就会看到这种修改所带来的变化。按下 Enter，CONTROL PANEL 更新 WIN. INI 文件，DIGCLOCK 的显示反映了这种变化。

当窗口过程接收到 WM_WININICHANGE 消息时，它使用

```
InvalidateRect (hwnd,NULL,TRUE);
```

使窗口无效。当 DIGCLOCK 接收到 WM_TIMER 消息时，它使用

```
InvalidateRect (hwnd,NULL,FALSE);
```

使窗口无效。最后一个参数为 TRUE 时，它告诉 Windows 在绘制窗口之前擦除背景色；为 FALSE 时，告诉 Windows 简单地在当前背景上绘制窗口。处理 WM_TIMER 消息时我们使用

FALSE，因为这种方法减少了显示器的闪烁。用户可能一点也不明白为什么要使用 TRUE 值。

当处理 WM_WININICHANGE 时 TRUE 是有必要的，因为若把时间格式从 12 小时改为 24 小时，则所显示的字符长度会发生变化。不过，由 WM_TIMER 消息导致的最大变化是两个字符——如 12/31/87 的下一天是 1/1/88——那么 WndPaint 显示的格式化字符串由于在长度与比例字模方面的变化而留下两个空格。

DIGCLOCK 还处理 WM_TIMECHANGE 消息，它将系统时间与日期发生变化这一消息通知应用程序。DIGCLOCK 因 WM_TIMER 消息而每秒更新一次，实际上没有必要那么做，处理 WM_TIMECHANGE 消息对每分钟更新一次的时钟更为敏感。

7.5 标准时间

若用户正在阅读《Windows 程序员参考手册》，很可能对 DIGCLOCK 中不使用 Windows GetCurrentTime 函数而感到不可理解。其实，GetCurrentTime 告知的是“Windows 时间”而非实际时间。这是一个从当前 Windows 会话开始之后经历的时间(以 ms 为单位)。GetCurrentTime 经常用于计算从 GetMessageTime 返回的时间差。如果要处理一条消息，以判定它在消息队列中直到处理时已存留多长时间，则可以使用这两个调用。

7.6 定时器举例

此应用程序试图启动两个定时器。若应用程序启动成功，则显示一窗口。如果 Windows 中已有足够多的定时器在运行，消息框就通知用户这一事实。用户然后就要选择是重新开始一定时器的启动还是终止此应用程序。

一旦定时器启动，此定时器每秒发一次嘟嘟声，而其他嘟嘟声大约每五秒两次。这些次数是大致相当的并受两个因素的影响：

(1)PC 的运行速度；

(2)除 TIMER.EXE 外打开的应用程序的个数。

新消息	说明
WM_TIMER	在设置的时间间隔满时发送
新函数	**简要说明**
SetTimer	产生一系统定时器事件
KillTimer	删除指明的定时器事件
MessageBox	创造一带有指定文本的窗口

源代码：WINTIMER.C

```
/* * WINTIMER.C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Starts two timers which instruct the computer to beep at one second &. * * /
/* * approx. five second intervals respectively. Intervals vary with the * * /
/* * number of applications active! * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
#include "windows.h"                                    // Include windows.h header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

#define SEC_1 1                                         // Define first SEC_ variable
#define SEC_5 2                                         // Define second SEC_ variable

BOOL TimerInit ( HANDLE );                              // Timer initialization
long FAR PASCAL TimerWndProc( HWND,unsigned,WORD,LONG);

/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
    HANDLE hInstance,hPrevInstance;                     // Current & previous instances
    LPSTR lpszCmdLine;                                  // Long ptr to string after
                                                        // program name during execution
    int cmdShow;                                        // Specifies the application
                                                        // window's appearance

{
MSG msg;                                                // Message variable
HWND hWnd;                                              // Window handle

    if (! hPrevInstance)                                // Initialize first instance
    {
        if (! TimerInit( hInstance ))                   // If initialization fails
            return FALSE;                               // return FALSE
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Timer",                        // Window class name
            "Timer",                                    // Window caption
            WS_OVERLAPPEDWINDOW,                        // Overlapped window style
            CW_USEDEFAULT,                              // Default upper-left x pos.
            0,                                          // Upper-left y pos.
            CW_USEDEFAULT,                              // Default initial x size
            0,                                          // y size
            NULL,                                       // No parent window
            NULL,                                       // Window menu used
            hInstance,                                  // Application instance
            NULL);                                      // No creation parameters

    while (! SetTimer(hWnd,SEC_1,1000,NULL))            // Too many
                                                        // timers (16) running?
        if (IDCANCEL == MessageBox(hWnd,"2 timers too many started",
                "Timer",MB_RETRYCANCEL))                // ERROR
            return FALSE;                               // Return FALSE
```

```
    while (! SetTimer(hWnd,SEC_5,5000,NULL))                    // Too many
                                                                 // timers (16) running?
        if (IDCANCEL == MessageBox(hWnd,"1 timer too many started",
                "Timer",MB_RETRYCANCEL))                         // ERROR
            return FALSE;                                        // Return FALSE

    ShowWindow( hWnd,cmdShow );                                  // Make window visible
    UpdateWindow( hWnd );                                        // Update window

    while (GetMessage(&msg,NULL,0,0))                            // Message reading
    {
        TranslateMessage(&msg);                                  // Message translation
        DispatchMessage(&msg);                                   // Send message to Windows
    }

    return (int)msg. wParam;                                     // Return wParam of last message
}

BOOL TimerInit( hInstance )                                      // Timer status
HANDLE hInstance;                                                // Instance handle
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */
    WNDCLASS wcTimerClass;                                       // Main window class

    wcTimerClass. hCursor = LoadCursor( NULL,IDC_ARROW );
                                                                 // Mouse cursor
    wcTimerClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
                                                                 // Default icon
    wcTimerClass. lpszMenuName = NULL;                           // No menu
    wcTimerClass. lpszClassName = "Timer";                       // Window class
    wcTimerClass. hbrBackground = GetStockObject( WHITE_BRUSH );
                                                                 // White background
    wcTimerClass. hInstance = hInstance;                         // Instance
    wcTimerClass. style = CS_VREDRAW | CS_HREDRAW;
                                                                 // Horizontal and
                                                                 // vertical redraw of client area
    wcTimerClass. lpfnWndProc = TimerWndProc;
                                                                 // Window function
    wcTimerClass. cbClsExtra = 0 ;                               // No extra bytes
    wcTimerClass. cbWndExtra = 0 ;                               // No extra bytes

    if (! RegisterClass(&wcTimerClass ) )                        // Register window class
        return FALSE;                                            // Return FALSE if registration fails
                                                                 // Return FALSE if registration
    return TRUE;                                                 // Return TRUE
}

/* * TimerWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
long FAR PASCAL TimerWndProc( hWnd,message,wParam,lParam )
    HWND hWnd;                                          // Window handle
    unsigned message;                                   // Message type
    WORD wParam;                                        // Message-dependent 16 bit value
    LONG lParam;                                        // Message-dependent 32 bit value
{
/* * Perform timer task * * * * * * * * * * * * * * * * * * * * */
    switch (message)                                    // Process messages
    {
        case WM_TIMER:                                  // Timers running
            if (wParam == SEC_1)                        // Beep once every second
                MessageBeep(0);
            if (wParam == SEC_5)                        // Beep twice every five seconds
            {
                MessageBeep(0);
                MessageBeep(0);
            }
            break;                                      // End of this message proces
        case WM_DESTROY:                                // Destroy window
            KillTimer(hWnd,SEC_1);                      // Disable one second timer
            KillTimer(hWnd,SEC_5);                      // Disable five second timer
            PostQuitMessage(0);
            break;                                      // End of this message process
        default:                                        // Send other messages to
                                                        // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                      // End of this message process
    }
    return(0L);
}
```

模块定义文件:WINTIMER. DEF

```
; WINTIMER. DEF

NAME WinTimer

DESCRIPTION 'Timer example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5120

EXPORTS TimerWndProc @1
```

MAKE 文件:WINTIMER. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =

all: WINTIMER.EXE

WINTIMER.OBJ: WINTIMER.C

WINTIMER.EXE: WINTIMER.OBJ WINTIMER.DEF
    link /NOD/A:16 WINTIMER,WINTIMER,nul,libw slibcew,WINTIMER.def
    rc WINTIMER.exe
```

WINTIMER. EXE 怎样工作

在这一应用程序中,主过程 WinMain 被初次扩充。窗口出现之前,应用程序用函数 SetTimer 启动两个定时器。因为第一个参数是窗口句柄,所以函数引用必须发生在 CreateWindow 之后。

第二个参数是定时器的 ID 值,它在应用程序开始时定义,时间用 ms 指定。

如果所有 16 个定时器都已指派了,以致应用程序不能启动两个定时器,那么消息框将会出现。消息框是包括标题和文本的简单窗口,也可选择性地使用按钮。

函数 MessageBox 的返回值表示按下哪个按钮。该框有两个按钮:Cancel 和 Retry。

如果选择 Cancel,整个 Windows 应用程序结束,如果选择 Retry,应用程序再用 SetTimer 试图启动定时器。也可应用第二个定时器,与第一个定时器的区别仅在于它每隔 5s 产生一条 WM_TIMER 消息而不是 1s。

新消息 WM_TIMER 被插入到窗口过程中。wParam 包括适当定时器的 ID 值,定时器被用于调用函数 MessageBeep 一次或两次。

应用程序关闭之前,这两个定时器必须被删除。否则,这些定时器不能再被其他应用程序使用,而且可得到的定时器的数量将减少。

第八章　内存管理

8.1　概　述

在 Windows 3.0 中，Windows 依靠系统的内存多少和处理器类型，可在三种可能的模式下运行。这些操作模式分别称为实模式、标准模式和 386 增强型模式。

在 3.1 版中，只有 386 增强型方式和标准模式有效。这样可保持与 2.0 版的向下兼容性。

早期在实模式下运行的 Windows 应用程序将不能在 Windows 中运行。

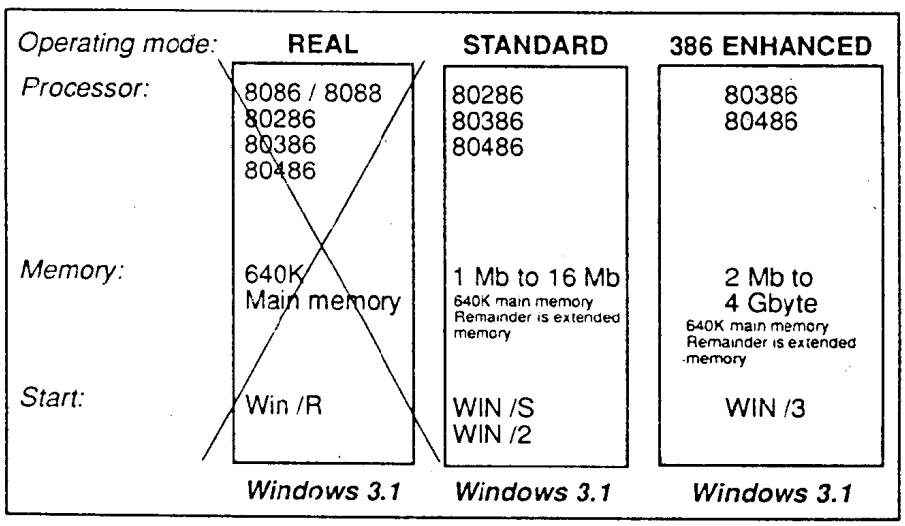


图 8.1　Windows 操作模式图

当用户仅仅在 DOS 提示下打入 WIN 命令来启动 Windows 时，Windows 将根据计算机的配置来决定使用哪种模式。

用户从 Program Manager 中选择 Help/Abont Program Manager…来决定使用哪一模式。对话框显示当前模式以及自由内存量。

对编程人员而言，GetWinFlags 函数可确定是否在保护模式下运行。以下例子具体说明 GetWinFlags 函数的典型使用方法。

许多标准应用程序使用同样可靠的方法来确信 Windows 下启动的某个应用程序处于保护模式下，否则程序将立即终止。

```
if( ! (LOBYTE(GetVersion()) == 3)
    && ( (GetWinFlags() & WF_PMODE) == WF_PMODE) )
{
    /* Windows is not running in protected mode under version 3 */
    /* display error message */
    MessageBox( hWnd,"No Protected Mode !!",
            "KoalaDb",MB_SYSTEMMODAL);
    return (FALSE);
}
```

GetWinFlags 函数返回几个数据项，这些数据项经过恰当的筛选后，说明 Windows 环境的当前操作状态。返回值由 32 位组成，各个独立的位代表不同种类的数据项。

```
DWORD GetWinFlags()
```

正如上面的例子一样，返回值与一个常量进行“AND”，并且用于 TRUE 或 FALISE 状态的读出。下表显示可获得的信息。

值	含义
WF_80×87	系统包含 Intel 数字协处理器
WF_CPU086	系统 CPU 是一个 8086 处理器
WF_CPU186	系统 CPU 是一个 80186 处理器
WF_CPU286	系统 CPU 是一个 80286 处理器
WF_CPU386	系统 CPU 是一个 80386 处理器
WF_CPU486	系统 CPU 是一个 80486 处理器
WF_ENLIANCED	Windows 运行于 386 增强型模式下
WF_LARGEFRAME	Windows 运行于 EMS 大帧内存装配下
WF_PMODE	Windows 运行于保护模式下
WF_SMALLFRAME	Windows 运行于 EMS 小帧内存装配下
WF_STANDARD	Windows 运行于标准模式下

GetVersion 函数可获得 Windows 的主版本号和次版本号(在 3.1 版中，3 是主版本号)。返回值的低位字节给出主版本号，高位字节给出次版本号。

8.1.1 Windows 3.0 实模式

在实模下，Windows 3.0 中许多部分的运行类似于 Windows 2.10 和 2.11 版本。只使用常规内存，即成功地启动 Windows 至少必须有 384K 可用内存。

用户也许想依靠所使用的应用程序，甚至在 80286 和更新的处理器中也能在 Windows 3.0 实模式下运行。

许多早期在 2.X 版本下开发的 Windows 应用程序与标准模式及 386 增强模式下的 Windows 存储管理技术不兼容。但是，要记住，在 Windows 3.1 版本中，实模式是不可用的。

8.1.2 标准模式

在启动 Windows 的过程中使用/S 或/2 开关时，扩展内存将加到有效的常规内存上，并且作为邻接内存来使用。

如果用户正互斥地工作于 Windows 应用程序，并且有足够多的内存可用，那么用户也

希望能在 80386 处理器上使用这一模式，即使在某种程度上 Windows 在标准模式下的运行快于 80386 增强型模式下的运行。用于 MS-DOS 的应用程序只能使用常规内存。

8.1.3 386 增强模式

在 386 增强模式下，许多 DOS 应用程序可以并行使用“时间片”处理技术来启动和运行。每个应用程序有其自身的存储区域。

为 Windows 所写的程序甚至可访问硬盘上的某个区域，并且将其作为内存来使用。为了正确访问这一虚拟内存，Windows 提供了一个专用的内存管理程序，称为虚拟内存管理器(VMM)。

VMM 将整个内存分为 4K 大小的段或页，并且基于“最近最少使用”算法，在需要的时候与硬盘上的一个文件进行交换。这一处理叫做页交换。

8.1.4 VMM

虚拟内存管理器保留一张页表，包含对于内存或磁盘上的每一页的引用。对于每一页，表中同时也包含一个标志，说明自此页从磁盘上(访问属性)装入以来是否已被访问。并且，如果已访问过，则说明该访问是否为写访问。

当一新页必须从磁盘上读出，并且没有足够的可用内存时，运行于 386 增强型方式下的 Windows 将检查保存在页中的这两种属性，并且在必要时清除属性。相关的页不能移到磁盘上，直到两种标志均被清除为止。

通过使用 Windows 3.1 控制面板，用户可选择是用一个临时文件还是用一个永久文件来保存硬盘上的交换页。一个永久性的交换文件实际上由两个文件组成，这两个文件自动命名为 386SPART. PAR 和 SPART. PAR，并且自动赋以系统属性和隐含属性。

用户只有在两个交换文件能保持邻接时才可扩大交换文件的尺寸。临时文件 WIN386. SWP 是一个独立文件，且能正常地扩大。许多用户推荐使用永久性的交换文件。除了其他便利之外，这允许 Windows 在内存比较紧张时运行得更快。

8.1.5 内存类型

在讨论以上提到的多种操作模式的存储管理时，我们将讨论三种不同类型的内存：

- 常规内存
- 扩充内存
- 扩展内存

1. 常规内存

常规内存占用内存的头 640K。所有运行于 DOS 下的应用程序完全限制在这一区域。

640K 边界也是运行于实模式或标准模式 Windows 下的 DOS 应用程序的内存限制。在 386 增强型模式下，许多 DOS 应用程序可占用超过这一边界的虚拟内存。

2. 扩充内存

在使用 8086/8088 处理器的机器上，用户不能扩充 640K 的常规内存区。一种称为“银

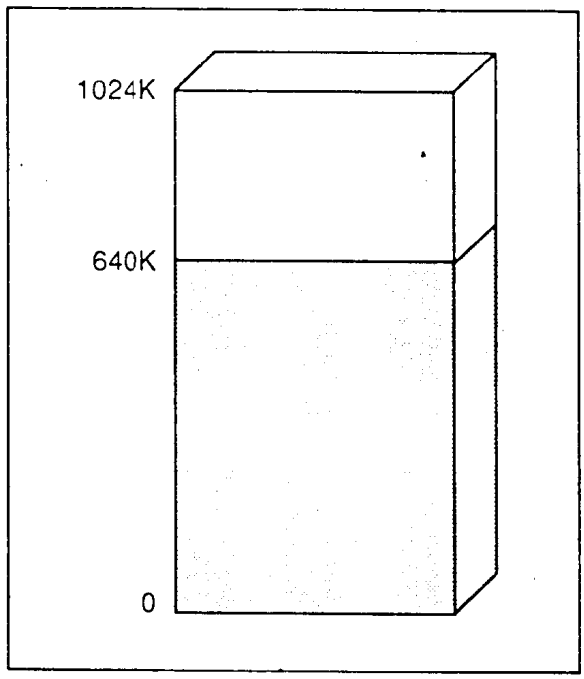


图 8.2　常规内存中的 640K 边界

行切换"的专用技术却允许使用称为扩充内存的附加内存。

扩充内存由许多 Windows 应用程序使用。这些应用程序是为早期 Windows 2.X 版本所编写的。在实模式下，Windows 3.0 只访问由两个不同的 LIM 标准(LIM 3.2 和 LIM 4.0)定义的扩充内存。

640K 与 1M 间的有效区域用作扩充内存，同时需要一个附加的内存卡和一个专用的驱动程序。

此驱动程序称作扩充内存管理器(EMM)，它负责处理内存卡与 CPU 间的通信。

根据 LIM 3.2 标准，此驱动程序在 640K 与 1M 间建立一块 64K 大小的区域(称为一帧)用于切换。

页帧分成 4 张邻接的 16K 页，这些页准许将扩充内存的 4 个 16K 块作为物理页。

当还需要附加内存卡上的又一逻辑页时，这一逻辑页将替代物理页帧内的某一页。在处理过程中，640K 与 1M 间的数据并不从 EMS 卡物理地拷贝到该区域。

更确切地说，相关寄存器中的地址只是作简单的调整，以便页帧引用扩充内存中的 4 页。之后，应用程序便可访问这些页上的数据了。

定义 LIM 4.2 标准是因为 LIM 3.2 不支持多任务。这一标准提供了"银行切换区"，这一区域可高达 64 页而不是 4 页，因而总计为 1M。

附加说明一点，在 LIM 4.0 中，页帧不必由邻接的地址空间组成。

3. 扩展内存

扩展内存可由标准模式和 386 增强型模式下的 Windows 使用，此处 Windows 运行于处

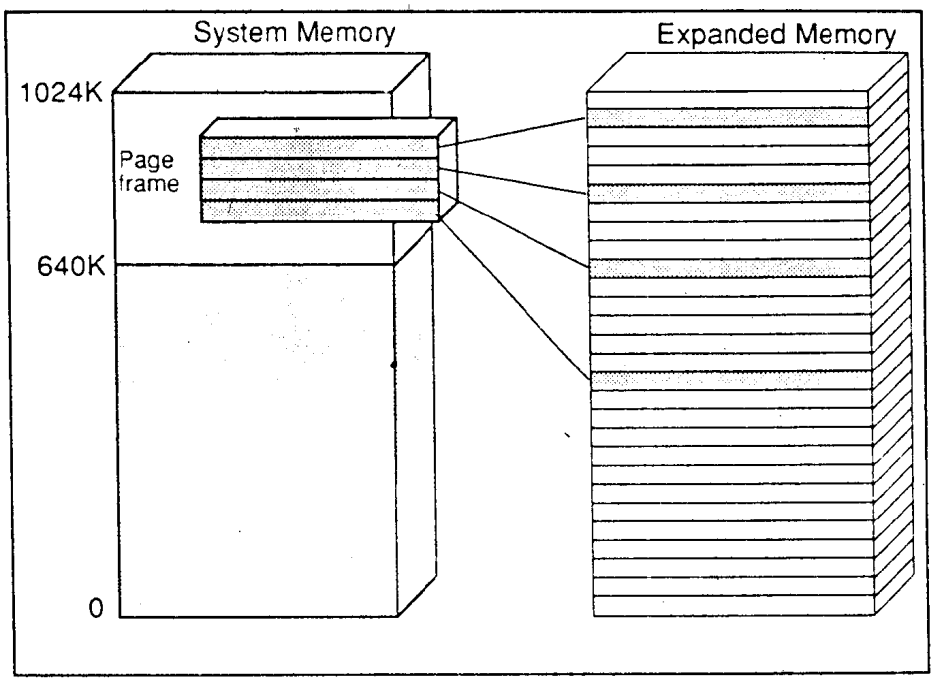


图 8.3　640K 与 1M 间的区域用作扩充内存，但需要一张内存卡和特殊的驱动程序

理器的保护模式下。

扩展内存一般从 1M 开始，地址空间共计可扩展到 80286 处理器下的最大 16M 字节以及 80386 下的 4 千兆字节。

寻址原则(在以下详细讨论)决定了可寻址内存的上限。以 8086 处理器为基础的计算机或以 8088 处理器为基础的计算机不能直接访问扩展内存。

对这种内存类型的访问是由 Windows 或 Windows 应用程序通过一个称作 HIMEM.SYS 的 XMS 驱动程序来进行的。XMS 即 eXtended Memory Speicification 的缩写。

80×86 处理器从 80286 处理器开始，它是在 1982 年由 Intel 引进的。所有这一系列的处理器都可操作于实模式和保护模式下。保护模式提供一种高效的内存处理技术，这一技术可以有效地占用高达 4000 MB 的地址空间。参看关于保护模式的详细描述。

在实模式下，段的初始地址直接存储在段寄存器里。在保护模式下，段寄存器包含对一个表格入口的引用，这个表格入口称为描述符。该引用占用寄存器的高 13 位，代表了一个选择器的主要部分。

8.1.6　选择器

选择器的 0 位和 1 位定义 4 种可能的优先级之一(RPL＝请求优先级)。第 2 位(TI＝表格说明符)说明当前的表类型(参看下面的描述)

8.1.7　描述符

描述符为一个 64 位的数据结构，它包含将要访问的段的 32 位初始地址。描述符也包含

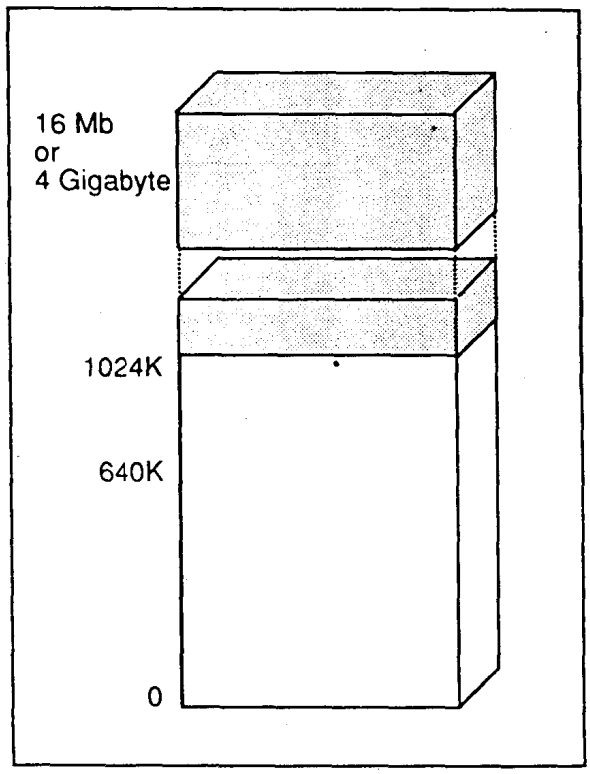


图 8.4　扩展内存从 1M 开始，在 80286 上扩展到 16M，在 80386 上扩展到 4KM

最大段尺寸。

描述符信息允许处理器检测段区越界访问。

附加标志说明访问优先级、段装入信息等。这些描述符可包含于一全局描述符表(GDT)中，该表可接受来自系统内任何部分的访问。描述符也可包含于一局部描述符表(LDT)中，此表含有相关于某一特定任务的所有描述符。以上两个表都在内存中。因此，Windows 系统与所有独立运行的 Windows 应用程序可认为是一个单独的任务。

上图显示的描述符布局既适用于 80286 处理器，也适用于 80386/80486 处理器。在这两种情况下，20 位(可定义 1M 的最大容量)用于段尺寸。在 80386 或 80486 中，区段位使得描述符可访问 4000 MB。若该位被置位，描述符所指定的段尺寸可增大 4096 倍。

只有可引用最大容量为 16M 的基地址的低位部分可寻址 80286 处理器上预定的段。

图 8.7 说明如何将一个来自段寄存器的虚拟地址转化成一个物理地址，以便访问预定的内存内容。

除了专用的内存管理技术外，保护模式也提供了不同的优先级。4 个优先级(从 0 到 3)决定是否可执行确定的指令。

0 级程序以最高优先级运行。所有 Windows 应用程序和所有 Windows DLL 常使用优先级 1。

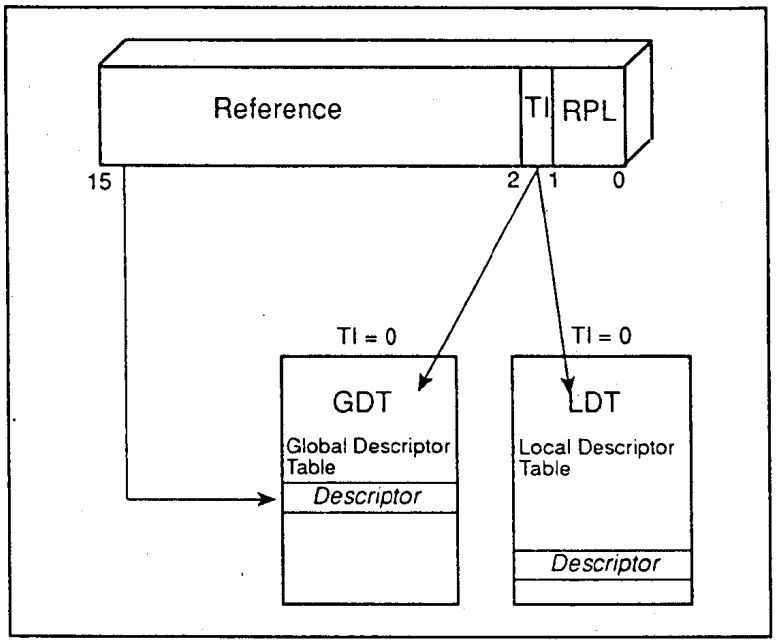


图 8.5　段寄存器包含一个占用寄存器高 13 位的指向称为描述符的表入口的引用

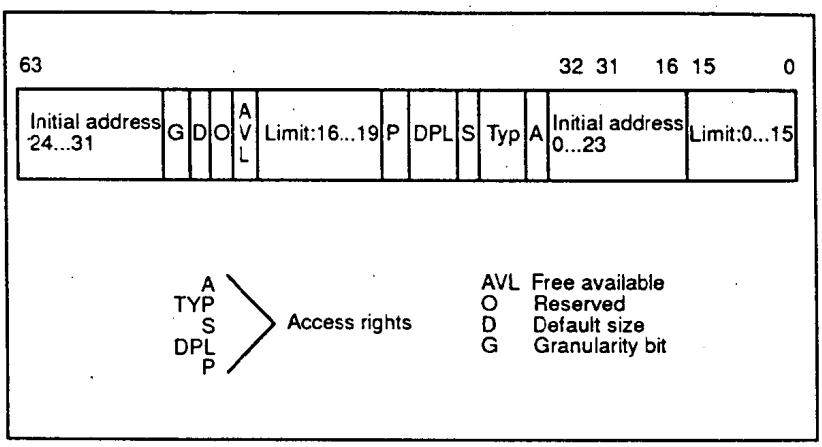


图 8.6

8.1.8　操作模式和内存管理

在实模式下，Windows 3.0 完全在常规内存内运行。在 Windows 装入后，如果至少有 280K 的自由映像页保留在常规内存中的话，则 Windows 自动以 LIM 4.0 页帧工作。

Windows 程序代码和一些附加内存(所有应用程序程序均可访问)的原地址设计为这些物理页面的起始地址，称为 EMS 行；可以在 DOS 提示下运行 WIN 时使用所加的特定开关移位。

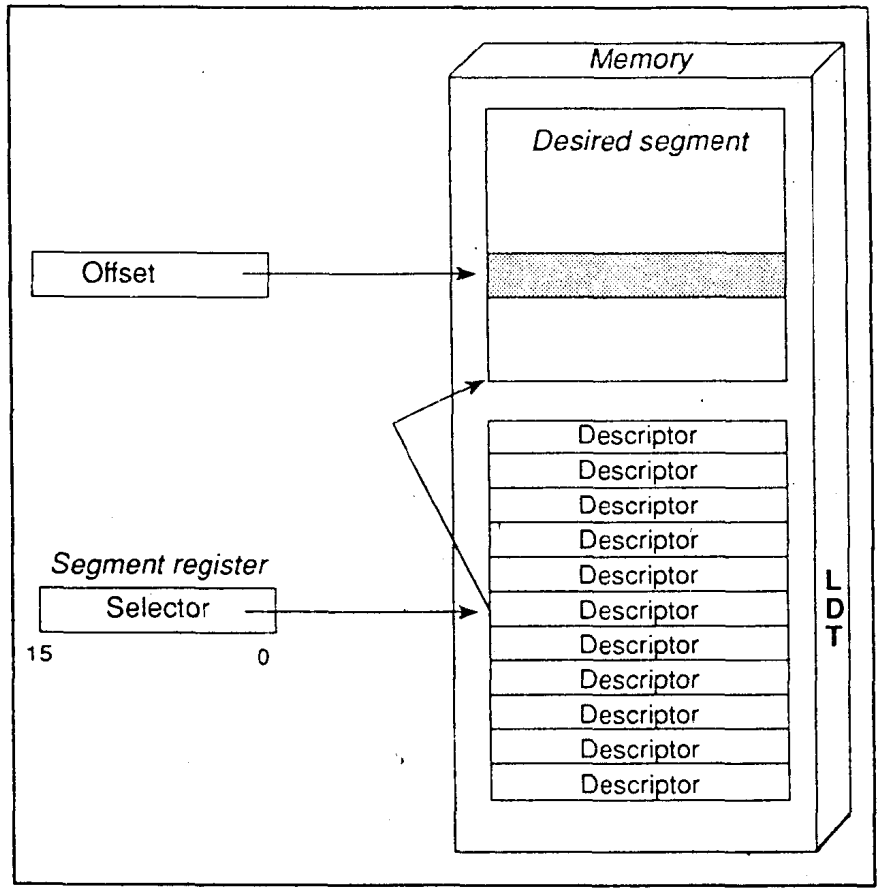


图 8.7　处理器如何将来自段寄存器的虚拟地址转化成物理地址

虽然在实模式下不会直接寻址扩展内存，但仍可以使用扩展内存。程序代码可交换到这个区域内，这样可提高运行速度。

如果另一个程序需要空间，且段设计为 DISCARDABLE，则 Windows 重写内存中的段。若没有扩展内存和 XMS 驱动程序，则在再一次请求内存时，就必须从磁盘上重新装入移出内存的程序代码。

使用 HIMEM.SYS 和扩展内存，程序代码可暂时存于 XMS 区，以后再拷贝回来，这比从磁盘上重新装入要快得多。

在标准模式下，Windows 不访问扩充内存。扩展内存附加到有效的常规内存尾部，它可作为一个邻接的内存块有效地用于 Windows 应用程序。

在 386 增强型模式下，虽然 Windows 不使用扩充内存，但是 DOS 应用程序可以在装配了 PIF 文件的情况下访问这一内存。扩展内存适用于标准模式。此外，虚拟内存管理器实现虚拟内存管理，在本章前面我们早已提到。

8.2 动态存储管理

Windows 程序员通常使用 Windows 提供的两种动态存储资源，它们分别称为全局堆和局部堆。

8.2.1 局部堆

局部堆设置在应用程序的数据段中。由于每一程序有其自身的数据段，所以也有其自身的局部堆。由于 80×86 处理器的设计，局部堆只限制为小容量的内存。

只有程序的数据段(最大尺寸为 64K)对局部堆有效。另外，这一空间还可由全局变量和说明为静态的变量所共享。这是因为 C 编译器在数据段中存放这些全局变量和静态变量。简言之，局部堆只适用于非常小的内存块，并且在应用程序中其使用受到某些限制。

8.2.2 全局堆

全局堆提供解决内存局限性问题的办法。堆区可被设想为一个几乎无限大的内存块，由 Windows 管理这一内存块。

利用 386 增强型模式下的虚拟内存管理能力，全局堆中的某些部分甚至可以驻留在硬盘上。程序可在任一时刻获得来自全局堆的高达 64K 容量的动态内存块。必须调用 WINMEM32. DLL 函数来分配更大的内存块(以后详述)。

全局堆的使用通常有四个步骤。第一步是用 GlobalAlloc 函数分配一内存块，这一步返回一个内存句柄。我们已见过 Windows 使用 LDT 来管理内存的情况。

内存句柄可以形象地描述为对 Windows LDT 的一个引用，当然，为了访问内存块，需要有块地址。地址在第二步中使用 GlobalLock 函数取得。

对于那些继续在 Windows 3.0 实模式下工作的程序员来说，应注意全局堆在这一模式下的使用情况。GlobalLock 通过递增一个锁定计数器，以防止块在内存中移动。

在 Windows 3.1 中，当仅使用标准模式和 386 增强型模式时，处理器总是在保护模式下工作。在这种情况下，知道一个内存块的具体地址是无关紧要的。

由 GlobalLock 返回的地址是一个虚拟地址，处理器的硬件设备将其转化为一个物理地址，编程人员不必操心实际的实现机制。

一旦一个块已由 GlobalAlloc 分配，并且其地址由 GlobalLock 取得以后，该块就能保持锁定状态，且不会干扰实模式下的内存管理。一旦知道了地址，就可在块中填入预定的数据。当块不再需要时，可用 GlobalFree 函数重新释放给 Windows 。

Windows 提供了几个 API 函数以访问全局堆，其中一些如下所示：

函数	描述
GetFreeSpace	返回有效字节数
GlobalAlloc	分配内存
GlobalCompact	压缩全局堆
GlobalDiscard	丢弃内存块
GlobalFlags	返回信息

GlobalFree	自由内存块
GlobalHandle	返回句柄
GlobalLock	返回地址
GlobalReAlloc	改变内存块尺寸
GlboalSize	返回内存尺寸
GlobalUnlock	解开内存块锁

例子：

```
HANDLE hMem;
LPSTR lpStr;

hMem = GlobalAlloc(GMEM_MOVEABLE,1024);
lpStr = GlobalLock(hMem);
lstrcpy(lpStr,"Ready for a vacation?");
...
GlobalFree(hMem);
```

8.3 Windows 开销

Windows 使用句柄来管理其应用程序。除此之外，还需要一定的附加信息。开销量的大小取决于不同的因素。

例如，局部堆和全局堆分配函数 LocalAlloc、GlobalAlloc 和 GlobalAlloc 各自有不同的开销。实际上，并不是按单个字节来分配内存的，LocalAlloc 使用控制信息的 4～6 个字节，GlobalAlloc 使用 24 个字节。

在程序员看来，这意味着内存块应开辟得尽量大，以适合这一开销。比较而言，C 编译器就不产生用于保留局部和静态变量的开销。

不过，应尽可能地动态分配数据，以防止一个程序变得无法使用。

Windows 管理其资源和句柄的技术仍是“保密”的。Windows 将其自身函数分成 GDI 和 USER，它们都是 DLL，有自身的数据段和局部堆。

GDI 函数使用 GDI 堆，而 USER 函数使用 USER 堆。GetDc 函数是一个典型的 GDI 函数的例子，这一函数使用 GDI 堆中的空间。

必须注意的是 USER 和 GDI 之间的一个重要区别。一个应用程序终止后，Windows 释放来自 USER 堆的控制信息；但不能由 Windows 本身来删除 GDI 控制信息。

因此，由 Windows 应用程序来清除 GDI 堆是很重要的。如果应用程序自身不执行这个函数，那么在应用程序结束时，就由 Windows 释放用 GlobalAlloc 和 LocalAlloc 分配的内存。

下面是一些关键的控制信息：

类型	控制域尺寸
Brush	32 字节
Bitmap	28-32 字节
DC	200/DC 和 100/设备
Pen	28 字节

Font	40-44 字节
Region	28-104 字节
Palette	28 字节
Window Class	40-50 字节
Menu	60-70 字节
Menu item	20 字节
GlobalAlloc	24 字节
LocalAlloc	4-6 字节
C variable	0 字节

8.4 存储模式

程序的运行速度极大地受所选择的用于编译此程序的存储模式的影响，因为存储模式决定程序和数据的尺寸。大多数 C 编译器支持小、中、压缩和大存储模式。

存储模式在编译时确定了代码段和数据段的数量。一般来说，程序员所用的代码和数据段越少，应用程序运行得也就越快。

8.4.1 小存储模式

使用小存储模式时，编译器建立一个 64K 代码段和一个 64K 数据段，这通常足够容纳一个常规的 Windows 应用程序了。配套软盘上的应用程序都使用小存储模式编译。

既然所有代码均设置于一个代码段中，所有数据均设置于一个数据段中，所以，数据和代码都可以使用近指针(16 位)来指示地址。

80×86 处理器硬件决定了将地址分为段和偏移量的方法。一个存储单元的确切位置首先由段然后由偏移量决定。对小存储模式，所有数据和代码都归于一个单一的段中。

由于段地址总是相同的，故在向函数传递变量时，只需偏移地址。编译器使用处理器的 DS 段寄存器来实现对偏移量的请求，并且在整个处理过程中管理对地址偏移量的请求。

下列程序使用每种存储模式来编译并且检验所产生的汇编代码。

```
void test( );
int Function( int * ,char * );
void test( )
{
    char cCharacter;
    int iInteger;
    cCharacter = 'C';
    iInteger = 99;
    Function( &iInteger,&cCharacter);
}
int Function( int * pInteger,char * pCharacter)
{
    return( *pInteger + * pCharacter);
}
```

程序声明了两个变量：cCharacter 和 iInteger。这两个变量作为指针传递给 Function 函数。

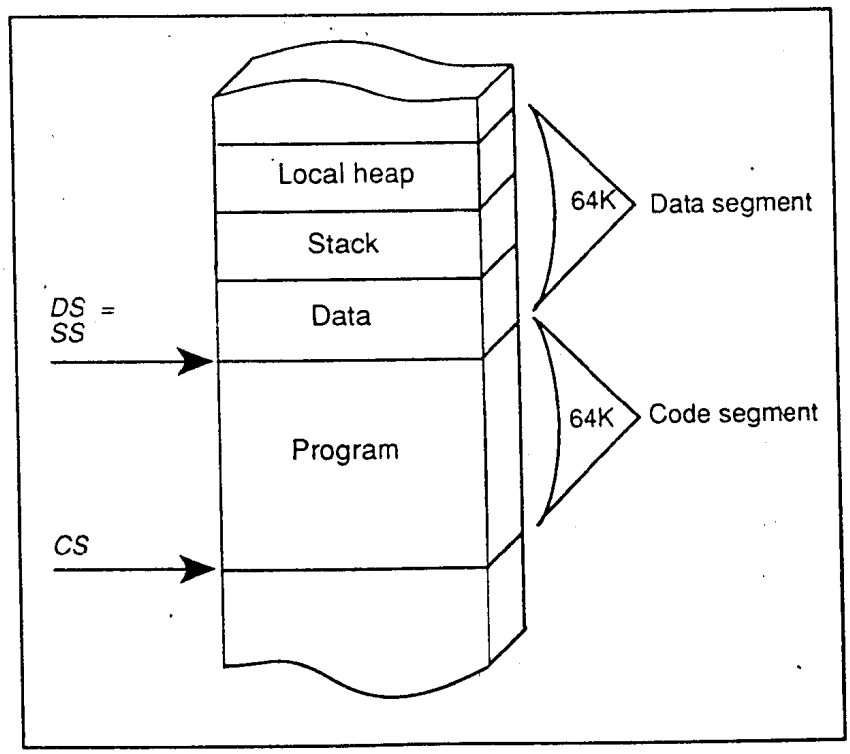


图 8.8　小存储模式

我们发现两个指针在汇编代码中都作为 WORD PTR 列出。

WORD PTR 是一个近指针，只包含偏移地址，并且只占 16 位。远指针需要 32 位，这是因为一个远指针由段和偏移地址两部分组成。同样，Function 函数调用只使用一个 16 位偏移地址，因为在代码段 CS 内发生跳转。

由于小模式只以偏移地址工作，所以可生成运行速度很快的应用程序。

8.4.2　中存储模式

中存储模式生成更大的超过 64K 的程序。这种模式支持多个 64K 代码段，但只支持一个数据段。许多用 C 语言编写的 Windows 应用程序就是用中模式编译的。

与小模式一样，数据仍使用近指针(16 位)指示地址，导致对数据的访问同样快。但是，当必须调用一个函数时，转向函数的指针必须是远地址值(32 位)。

如果观察中模式下的汇编代码，用户将会发现，远程跳转被标注为 FAR PTR_Function。这意味着对于每个函数调用必须改变指令计数器和代码段寄存器的值。

进一步地说，子程序使用 PROC FAR 声明为一个 FAR 过程，该子程序有确定的开销。

另一方面，所有变量和参数用 NEAR 指针指示地址，这是因为中模式只使用单一的数据段。

这一模式当然最适用于包含大量代码但只有少量数据的大型应用程序。

```
;Line 1
;Line 5
PUBLIC _test
_test PROC NEAR
push bp
mov bp,sp
sub sp,4
push di
push si
; cCharacter = -4
; iNumber = -2
; Line 9
mov BYTE PTR[bp-4],90  ;cCharacter
; Line 10
mov WORD PTR [bp-2],99  ;iNumber
; Line 11
lea ax,WORD PTR [bp-2] iNumber
push ax
call Function          <-- Near Call for function
add sp,4
; Line 12
$EX102:
pop si
pop di
mov sp,bp
pop bp
ret

;Line 15
nop
nop
PUBLIC_Function
_Function PROC NEAR
push bp
mov  bp,sp
sub sp,0
push di
push si
;  pZeichen = 6
;  pZahl = 4
;  Line16
mov  bx,WORD PTR[bp+6]  <-- Near Pointer for data
mov al,BYTE PTR [bx]
cbw
mov  bx,WORD PTR[bp+4]  <-- Near Pointer for data
add  ax,WORD PTR [bx]
jmp $EX107
; Line 17
$EX107:
pop si
pop di
mov sp,bp
pop bp
ret
```

图 8.9　编译器和小存储模式

8.4.3　压缩模式

压缩模式与中模式恰好相反。此模式仅提供一个代码段，但可以有多个数据段。由于编

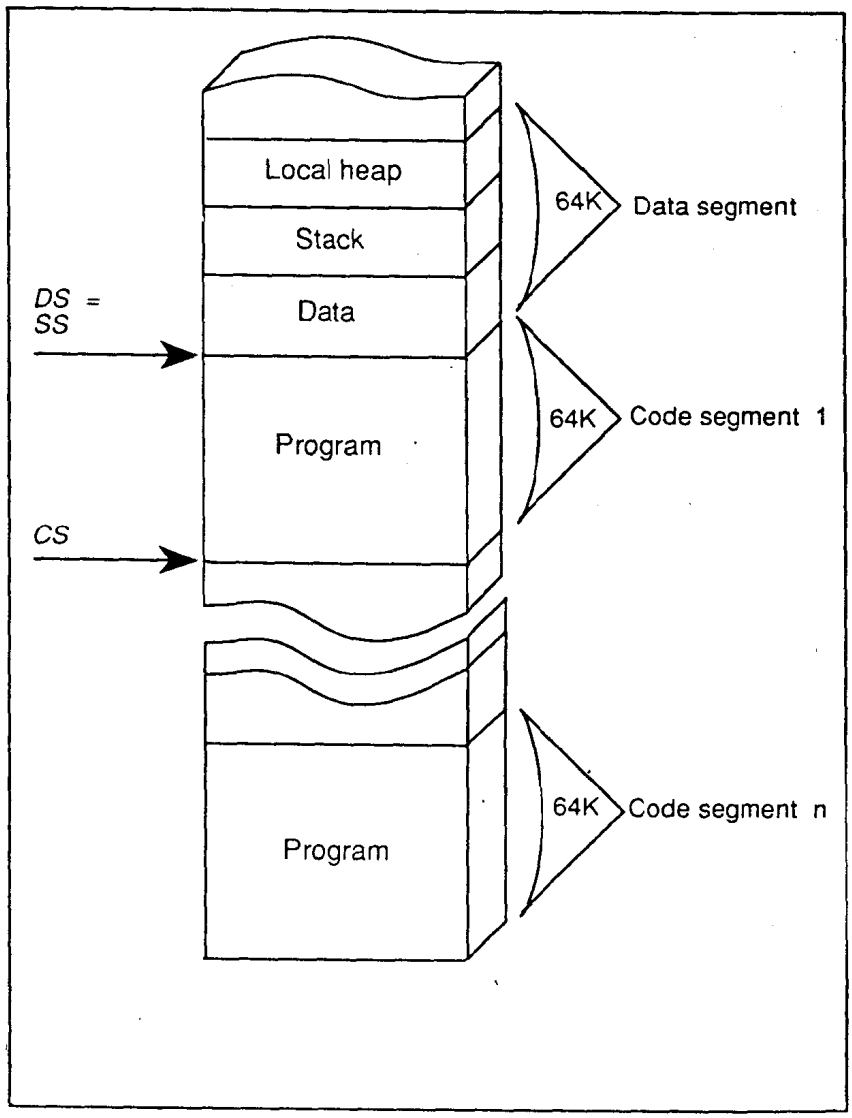


图 8.10　中存储模式

译器只产生一个 64K 的代码段，所以所有函数调用均使用 NEAR 寻址。所有的数据访问当然要使用 FAR 指针。压缩模式在某种程度上减少了大模式下的开销。

压缩模式下的汇编代码显示所有函数调用均为 NEAR，而数据和参数通过 FAR 指针访问，在代码中以 DWORD(双字)表示。由于存在多个数据段，编译器必须为所有参数提供段地址和偏移地址。

8.4.4　大存储模式

如果以上所讨论的模式还不够用的话，就有必要使用速度最慢的一种，即大模式了。

大模式提供多个 64K 数据段和多个 64K 代码段。所有函数调用或对数据的引用都使用 FAR(32 位)指针。

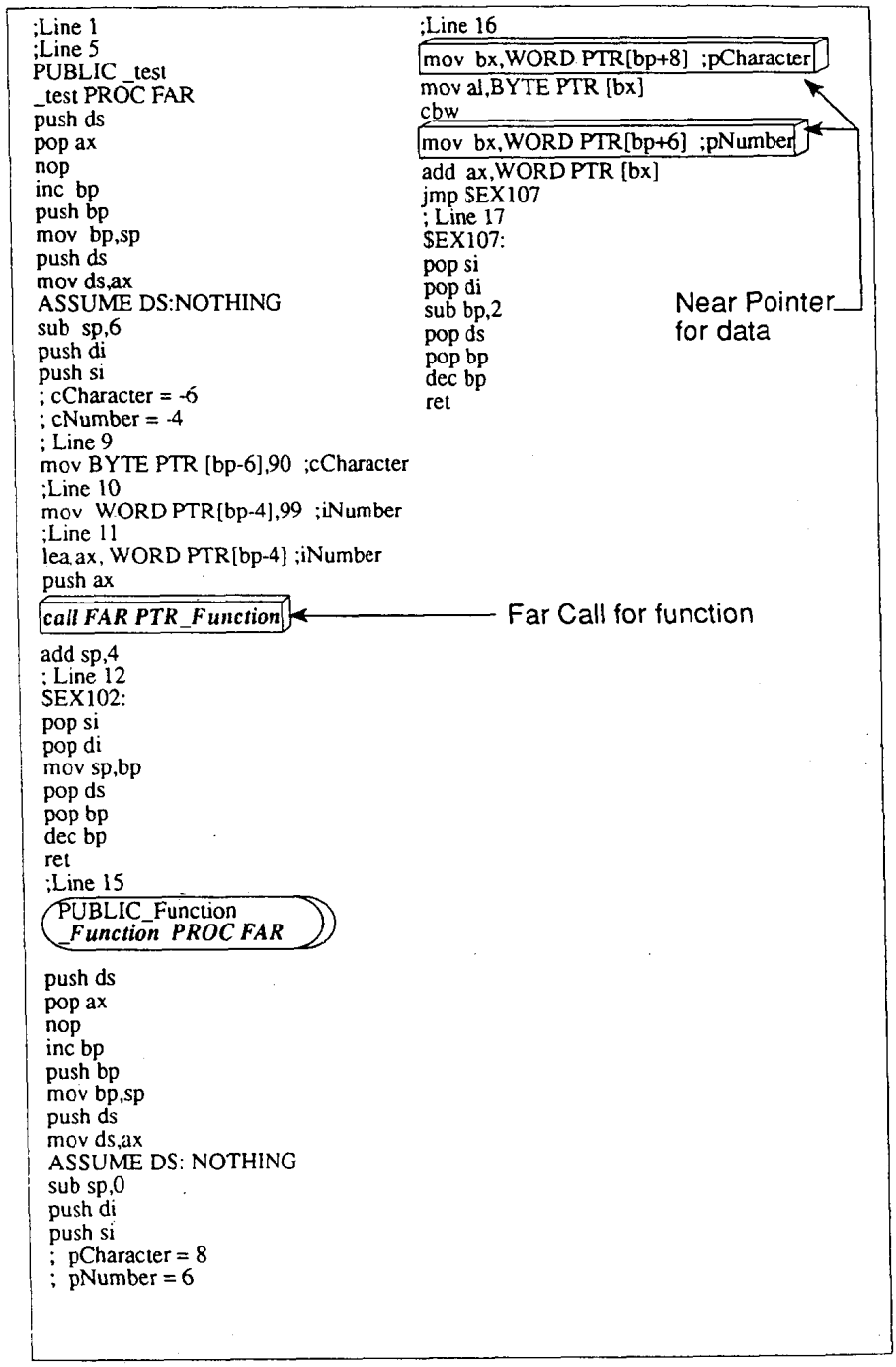


图 8.11　编译器和中存储模式

8.4.5　存储模式与 Windows

在编制一个 Windows 应用程序时，通常不鼓励使用大模式或压缩模式。虽然这两种模

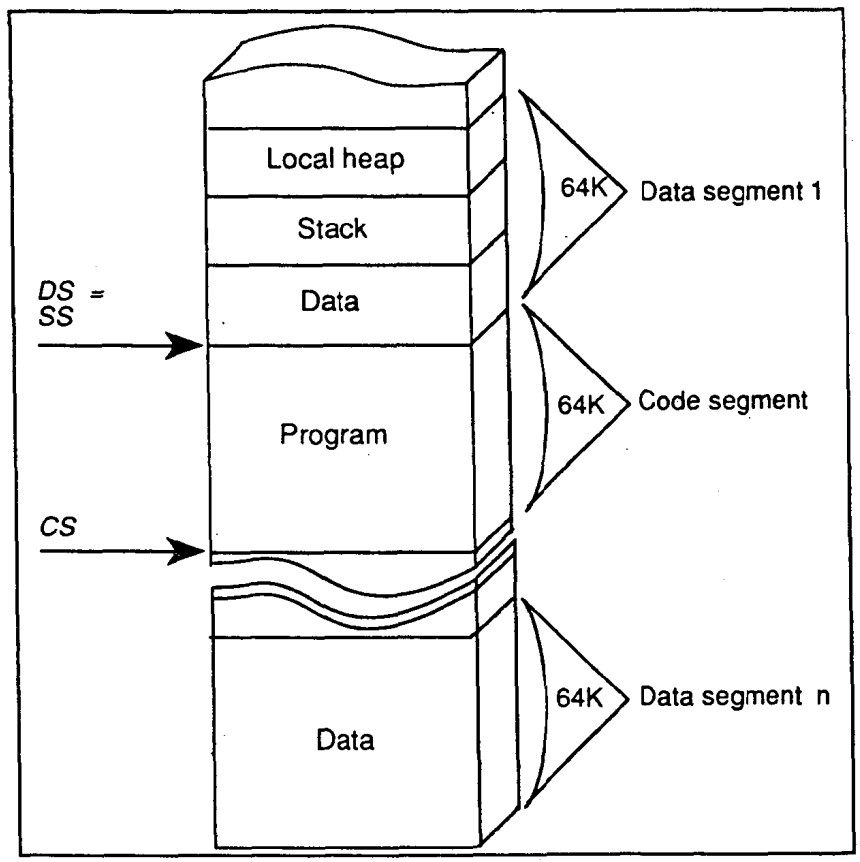


图 8.12　压缩存储模式

式使用的是多个数据段，但是 Windows 在使用这两种模式进行编译时，却只能装入编译程序的一个实例。

分配多个数据段后，Windows 不再在内存中随意移动这些段，因为这些数据段不能作为分开的段加以识别。

Windows 使用处理器硬件和 Windows LDT（局部描述符表）协同工作以便在内存中移动独立的段。

所以，程序必须经常使用处理器的 DS 段寄存器访问静态数据。

使用小模式和中模式时，在仅有一个数据段的程序中，所有静态数据都由 DS 段寄存器和一个偏移量来确定其地址。

DS 中的值一般不随偏移量的改变而改变。但是，一旦有附加数据段，对静态数据的访问就会要求改换 DS 寄存器中的内容。

这样的数据段必须固定于内存中，因为 Windows 提供了将引用重定向到应用程序的一个可移动数据段的机制。

固定段限制了 Windows 内存管理，也就是增强了对多个应用程序实例的限制。

这些限制可在使用小模式或中模式来开发程序时予以避免。如果一个程序需要大量数据，很容易在执行期间由 Windows 来分配所需的内存。

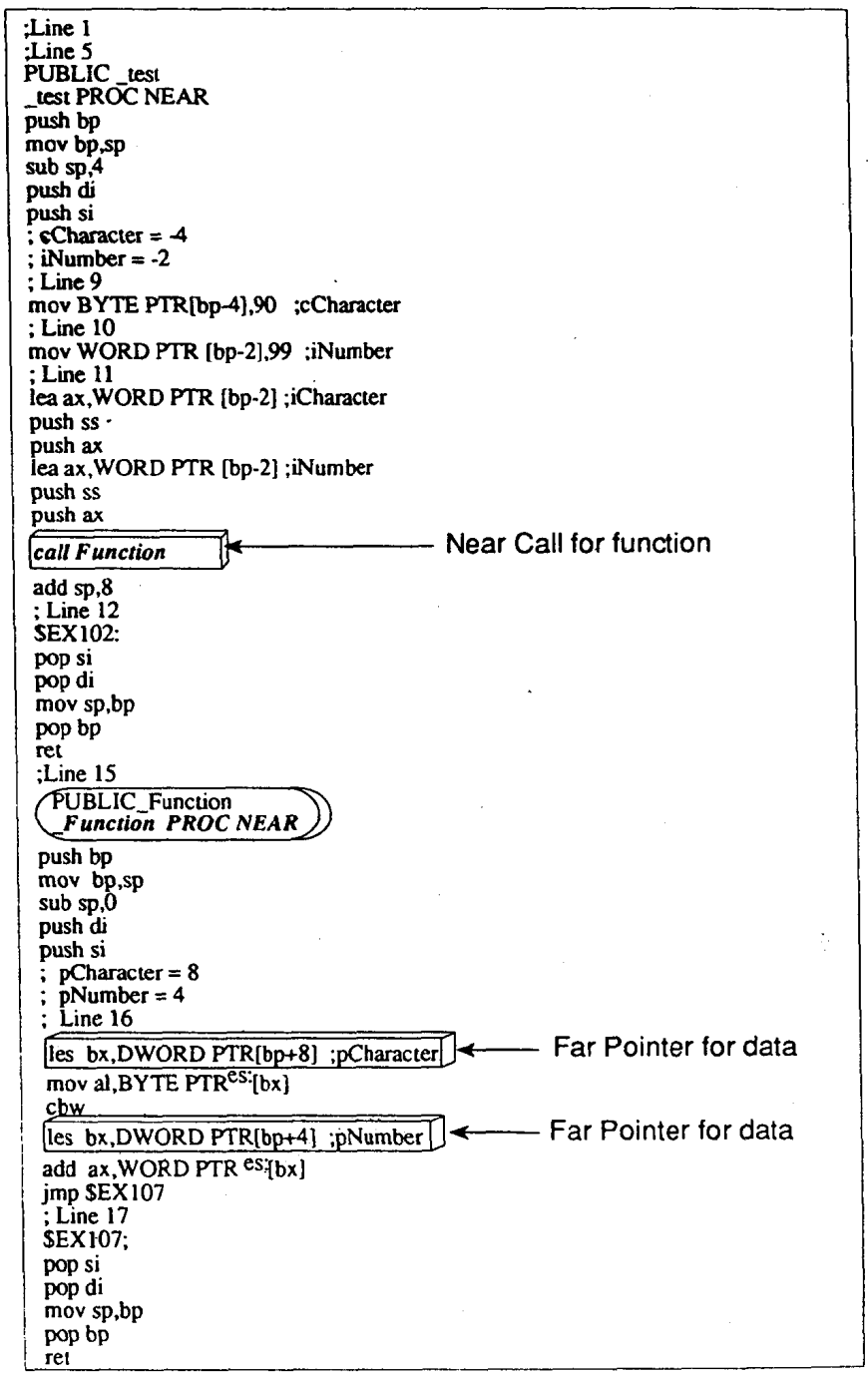


图 8.13 编译器和压缩模式

Windows API 函数 LocalAlloc()和 GlobalAlloc()可用于分配数据。一般来说，这两个函数甚至适用于大型应用程序的内存请求。

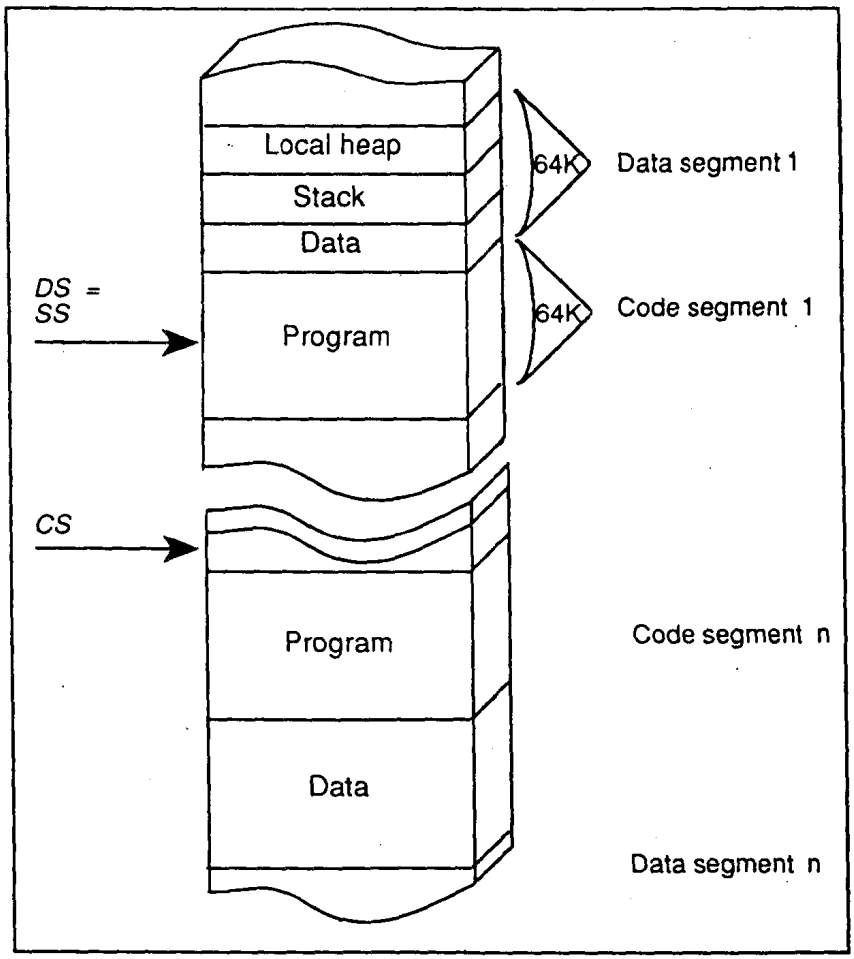


图 8.14 大存储模式

8.5 存在的问题

Windows 下的存储管理有其自身的局限。甚至在有足够大的常规内存和虚拟磁盘内存的情况下，这个管理机制也可能耗尽所提供的内存资源。因此并非所有有效内存都是可使用的。

为获取小块内存而频繁调用函数的程序可能会产生这种情况，因为每个调用都请求一个新的内存句柄。

8.5.1 例子："吃掉"内存句柄

一个给定的应用程序所能使用的有效句柄数部分地依赖于所装入的应用程序数。频繁分配小内存块，一个大型程序能很快达到大约总计 7500 个句柄的内存句柄总量。我们的例子程序就使用这个数量。

程序以一条 WM_TIMER 消息为基础，使用函数分配一个小内存段。产生的内存句柄只

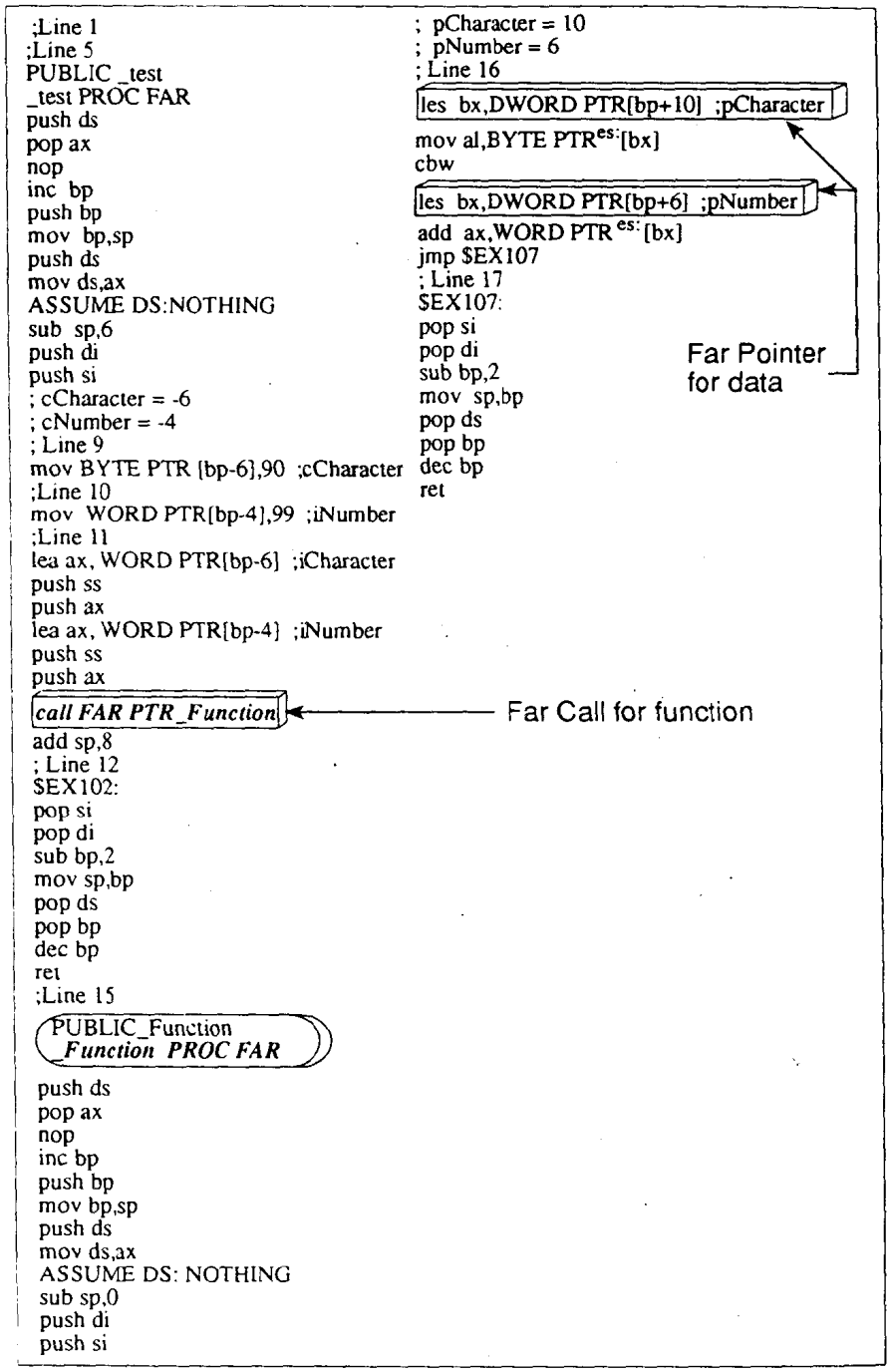


图 8.15　编译器和大存储模式

是被简单地放弃掉，且递增计数器的值。句柄数总是出现在标题条里。

函数在无更多自由内存或内存句柄数已达到极限的情况下返回 NULL。

如果有效内存不够的话，那么 GetFreeSpace 函数能检验内存，返回以字节为单位的自由

内存空间。

如果 Windows 不能使用更多的句柄，那么它就试图显示一个消息框，表明结果不成功。为了对这种情况作更进一步的检查，可以通过消息产生一个静态文本区。

比较实际的有效内存量就能证明句柄数限制了应用程序的运行，即它是引起程序出错的原因。

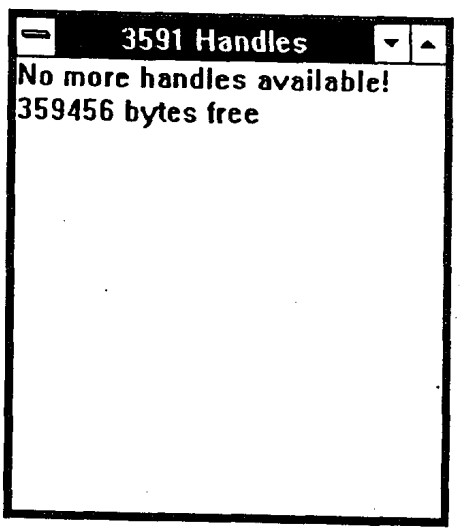


图 8.16　无句柄，有自由内存

新函数	描述
GlobalAlloc	分配全局内存块

EATHANDL. C 源码

```
/* * EATHANDL. C * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays multiple handle access * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "eathandl. h"                                    // Include EATHANDL. H header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain(HANDLE hInstance,                              // Current instance
        HANDLE hPrevInstance,                                     // Previous instance
        LPSTR lpszCmdLine,                                        // Long ptr to string after
                                                                  // program name during execution
        int nCmdShow)                                             // Specifies the application
                                                                  // window's appearance
{
```

```
    MSG msg;                                          // Message variable
    HWND hWnd;                                        // Window handle
    int nError;                                       // Error variable

    strcpy(szAppName,"EatHandl");                     // Application name
    hInst = hInstance;                                // Instance

    if (! hPrevInstance)                              // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
            /* If registration fails */
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;
        }
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow(szAppName,                    // Application name
        "Handle Eater",                               // Window caption
        WS_OVERLAPPEDWINDOW,                          // Overlapped window style
        CW_USEDEFAULT,                                // Default upper-left x pos.
        0,                                            // Upper-left y pos.
        GetSystemMetrics(SM_CXSCREEN)/3,              // Initial x size
        GetSystemMetrics(SM_CYSCREEN)/2,              // Initial y size
        NULL,                                         // No parent window
        NULL,                                         // Window menu used
        hInstance,                                    // Application instance
        NULL);                                        // No creation parameters

    if(hWnd == NULL)
    {
        LoadString(hInstance,IDS_ERR_WINDOW,
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }

    while (! SetTimer(hWnd,1,1,NULL))
        if (IDCANCEL == MessageBox(hWnd,"Too many timers running",
            szAppName,MB_RETRYCANCEL))
        return FALSE;

    ShowWindow( hWnd,nCmdShow );                      // Make window visible
    UpdateWindow( hWnd );                             // Update window
```

```
    while (GetMessage(&msg,NULL,0,0))                    // Message reading
    {
        TranslateMessage(&msg);                          // Message trnaslation
        DispatchMessage(&msg);                           // Send message to Windows
    }
    return (int)msg. wParam;                             // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)
{
    /* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                    // Main window class

    wcClass. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcClass. lpszMenuName = (LPSTR)NULL;
    wcClass. lpszClassName = szAppName;
    wcClass. hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcClass. hInstance = hInstance;
    wcClass. style = CS_VREDRAW | CS_HREDRAW;
    wcClass. lpfnWndProc = StdDlgWndProc;
    wcClass. cbClsExtra = 0 ;
    wcClass. cbWndExtra = 0 ;

    if (! RegisterClass( &wcClass ) )
        return FALSE;
    return TRUE;
}

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,                 // Window handle
        unsigned message,                                // Message type
        WORD wParam,                                     // Message-dependent 16 bit value
        LONG lParam )                                    // Message-dependent 32 bit value
{
    static WORD Counter = 0;                             // Counter for handle
                                                         // to be allocated
    static HWND hWndStatic;                              // Static window handle
    char cText[100];                                     // Text (100 characters)
    int i;

    switch (message)
    {
        case WM_CREATE:                                  // Create static window
            hWndStatic = CreateWindow("STATIC",          // Window class name
                    "",                                  // Window caption
                    WS_CHILD |                           // Child window
```

```
                    WS_VISIBLE,                                    // styles
                    0,0,                                           // Upper-left x and y
                    0,0,                                           // Initial x and y sizes
                    hWnd,                                          // Parent window
                    NULL,                                          // No menu
                    GetWindowWord(hWnd,                            // Get window handle
                    GWW_HINSTANCE),
                    NULL);                                         // Creation parameters
                break;
            case WM_SIZE:                                          // Sizing message
                MoveWindow( hWndStatic,0,0,LOWORD(lParam),         // Move controls
                        HIWORD(lParam),TRUE);
                break;
            case WM_TIMER:                                         // Timer
                for (i=0;i<10;i++)                                 // Increment timer
                {
                    if (GlobalAlloc(GMEM_MOVEABLE,10) == NULL)     // All timers running?
                    {
                        KillTimer(hWnd,1);                         // Kill timer 1
                        wsprintf( cText,"No more handles available! "  // Display message
                            "\n %lu bytes free",
                                GetFreeSpace(0));
                        SetWindowText(hWndStatic,cText);
                        break;
                    }
                    else
                    Counter++;
                }
                wsprintf( cText," %d Handles",Counter);            // List number of open handles
                SetWindowText(hWnd,cText);
                break;
            case WM_COMMAND:
                switch(wParam)
                {
                    default:
                        return (DefWindowProc(hWnd,message,
                            wParam,lParam));
                        break;
                }
                break;
            case WM_CLOSE:                                         //-Close up
                KillTimer(hWnd,1);                                 // Kill timer
                DestroyWindow(hWnd);                               // Destroy window
                PostQuitMessage(0);                                // Exit
                break;
            default:
                return(DefWindowProc(hWnd,message,wParam,lParam));
                break;
        }
        return(0L);
}
```

模块定义文件:EATHANDL. DEF

```
NAME EATHANDL
DESCRIPTION 'Handle Eater'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:EATHANDL. RC

```
#include "eathandl. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件 EATHANDL. H

```
#include <windows. h>
#include <string. h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21

char szString[128];                          /* Variable for loading resource text */
char szAppName[20];                                  /* Window class name */
HANDLE hInst;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:EATHANDL. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = EATHANDL. H
```

```
all: EATHANDL.EXE

EATHANDL.RES: EATHANDL.RC $(RESFILES)

EATHANDL.OBJ: EATHANDL.C $(H)

EATHANDL.EXE: EATHANDL.OBJ EATHANDL.RES EATHANDL.DEF
    link /NOD/A:16 EATHANDL,EATHANDL,nul,libw slibcew,EATHANDL.def
    rc EATHANDL.res EATHANDL.exe
```

运行 NMAKE 实用程序生成 EATHANDL 应用程序：

```
nmake eathandle.mak
```

8.5.2 资源描述文件

资源描述文件 EATHANDL.RC 提供所需的资源数据。我们在后面详细讨论资源。

8.6 ToolHelp DLL

许多内部条件在 Windows 下变得更易访问，这主要归功于提供信息的 API 函数，这些信息包括虚拟内存管理和 Windows 堆的状态等。

从这些信息中，我们将检查处理全局堆和虚拟内存管理器的两个函数。

对于内存紧张的应用程序来说，让人感兴趣的是究意有多少句柄用于全局堆，有多少句柄用于适当地分配适当的资源。

可使用向结构提供值的函数实现这一目的。结构的第一个入口必须初始化为结构的实际尺寸。在完成这一步以前，函数将不会返回任何值。

同样地，函数也将传送关于虚拟内存管理器的信息，这只适用于运行在 386 增强模式下的 Windows。

以下例子演示使用函数进行内存管理的当前状态。一个定时器周期性地调用这一过程，结果显示在一个文本区中。

新函数	描述
GlobalInfo	全局堆信息
MenManInfo	虚拟内存管理器信息

新结构	描述
GLOBALINFO	全局堆信息结构
MEMMANINFO	虚拟内存管理器信息

MEMINFO.C 源码

```
/* * MEMINFO.C * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays memory information * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "meminfo.h"                    // Include MEMINFO.H header file
```

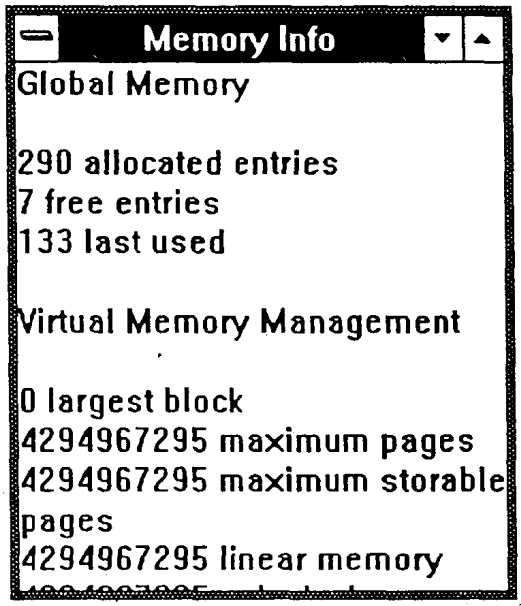


图 8.17　虚拟内存管理器信息

```
#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

/* * WinMain (main function for every Windows application * * * * * * * * */

int PASCAL WinMain(HANDLE hInstance,                          // Current instance
        HANDLE hPrevInstance,                                 // Previous instance
        LPSTR lpszCmdLine,                                    // Long ptr to string after
                                                              // program name during execution
        int nCmdShow)                                         // Specifies the application
                                                              // window's appearance
{
    MSG msg;                                                  // Message variable
    HWND hWnd;                                                // Window handle
    int nError;                                               // Error integer

    strcpy(szAppName,"meminfo");                              // Window class name
    hInst = hInstance;                                        // Instance handle

    if (! hPrevInstance)                                      // Initialize first instance
    {
        /* Register window classes for first instance */
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
            /* Registration fails */
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,MB_ICONEXCLAMATION);
```

```
            return nError;
        }
    }

    /* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow(szAppName,                          // Window class name
            "Memory Info",                                  // Window caption
            WS_OVERLAPPEDWINDOW,                            // Overlapped window style
            CW_USEDEFAULT,                                  // Default upper-left x pos.
            0,                                              // Upper-left y pos.
            GetSystemMetrics(SM_CXSCREEN)/3,                // Initial x size
            GetSystemMetrics(SM_CYSCREEN)/2,                // Initial y size
            NULL,                                           // No parent window
            NULL,                                           // Window menu used
            hInstance,                                      // Application instance
            NULL);                                          // No creation parameters

    if(hWnd == NULL)                                        // If instance handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,               // then display error
            szString,sizeof(szString));
        MessageBox(NULL,
            szString,
            NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    while (! SetTimer(hWnd,1,2000,NULL))                    // Maximum number of timers?
        if (IDCANCEL == MessageBox(hWnd,
            "Too many timers currently running",
            szAppName,MB_RETRYCANCEL))
        return FALSE;

    ShowWindow( hWnd,nCmdShow );                            // Make window visible
    UpdateWindow( hWnd );                                   // Update window as needed

    while (GetMessage(&msg,NULL,0,0))                       // Message reading
    {
        TranslateMessage(&msg);                             // Message translation
        DispatchMessage(&msg);                              // Send message to Windows
    }
    return (int)msg. wParam;                                // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                       // Main window class
```

```
    wcClass.style = CS_VREDRAW | CS_HREDRAW;                    // Horizontal and vertical
                                                                 // redraw of client area
    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );              // Mouse cursor
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);             // Default icon
    wcClass.lpszMenuName = (LPSTR)NULL;                          // No menu
    wcClass.lpszClassName = szAppName;                           // Window class
    wcClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
                                                                 // White background
    wcClass.hInstance = hInstance;                               // Instance
    wcClass.lpfnWndProc = StdDlgWndProc;                        // Window function
    wcClass.cbClsExtra = 0 ;                                     // No extra bytes
    wcClass.cbWndExtra = 0 ;                                     // No extra bytes

    if (! RegisterClass( &wcClass ) )                            // Register window class
        return FALSE;                                    // Return FALSE if registration fails
    return TRUE;                                     // If registration is successful,return TRUE
}

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,                         // Window handle
            unsigned message,                                    // Message type
            WORD wParam,                                         // Message-dependent
                                                                 // 16 bit value
            LONG lParam )                                        // Message-dependent
                                                                 // 32 bit value
{
    static HWND hWndStatic;
    static char cText[1000];
    GLOBALINFO Globalinfo;                                       // Global information
    MEMMANINFO Memmaninfo;                                // Memory management info

    switch (message)                                             // Process messages
    {
        case WM_CREATE:                                          // Create window
            hWndStatic = CreateWindow("STATIC",                  // Window class name
                    "",                                          // Caption
                    WS_CHILD                                     // Child window
                    | WS_VISIBLE,                                // Styles
                    0,                                           // Upper-left x
                    0,                                           // Upper-left y
                    0,                                           // Initial x
                    0,                                           // Initial y
                    hWnd,                                        // Parent window
                    NULL,                                        // Menu used
                    GetWindowWord(hWnd,GWW_HINSTANCE),           // App. instance
                    NULL);                                // No creation parameters
            break;
```

```
/* * List directory and set focus to this window * * * * * * * * * * * */
case WM_SIZE:
    MoveWindow( hWndStatic,0,0,LOWORD(lParam),
            HIWORD(lParam),TRUE);                          // Move controls
    break;

case WM_TIMER:
    Globalinfo.dwSize = sizeof(GLOBALINFO);
    Memmaninfo.dwSize = sizeof(MEMMANINFO);
    GlobalInfo( &Globalinfo);
    MemManInfo( &Memmaninfo);
    wsprintf( cText,"Global Memory \n"                     // List memory allocations
            "\n%u allocated entries "                      // in the window
            "\n%u free entries "
            "\n%u last used \n"
            "\nVirtual Memory Management \n"
            "\n%lu largest block "
            "\n%lu maximum pages "
            "\n%lu maximum storable pages "
            "\n%lu linear memory "
            "\n%lu unlocked pages "
            "\n%lu free pages "
            "\n%lu total number of pages "
            "\n%lu linear free "
            "\n%lu swap file pages "
            "\n%u page size ",
            Globalinfo.wcItems,
            Globalinfo.wcItemsFree,
            Globalinfo.wcItemsLRU,
            Memmaninfo.dwLargestFreeBlock,
            Memmaninfo.dwMaxPagesAvailable,
            Memmaninfo.dwMaxPagesLockable,
            Memmaninfo.dwTotalLinearSpace,
            Memmaninfo.dwTotalUnlockedPages,
            Memmaninfo.dwFreePages,
            Memmaninfo.dwTotalPages,
            Memmaninfo.dwFreeLinearSpace,
            Memmaninfo.dwSwapFilePages,
            Memmaninfo.wPageSize
            );
    SetWindowText(hWndStatic,cText);
    break;

case WM_COMMAND:                                           // Commands
    switch(wParam)
    {
        default:                                           // Send other messages to
                                                           // default window function
            return (DefWindowProc(hWnd,message,
                wParam,lParam));
            break;                                         // End of this message process
```

```
            }
            break;

        case WM_CLOSE:                                          // Close window
            KillTimer(hWnd,1);                                  // Disable timer
            DestroyWindow(hWnd);                                // Destroy window
            PostQuitMessage(0);                                 // Exit
            break;

        default:
            return (DefWindowProc( hWnd,message,wParam,
                    lParam ));
            break;                                              // End of this message process
    }
    return(0L);
}
```

模块定义文件:MEMINFO. DEF

```
NAME MEMINFO
DESCRIPTION 'Memory Info'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:MEMINFO. RC

```
#include "meminfo. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件 MEMINFO. H

```
#include <windows. h>
#include <string. h>
#include <toolhelp. h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21

char szString[128];                                     /* Variable for loading resource text */
```

```
char szAppName[20];                                    /* Window class name */
HANDLE hInst;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:MEMINFO. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = MEMINFO.H
LIBS_EXT = toolhelp.lib

all: MEMINFO.EXE

MEMINFO.RES: MEMINFO.RC $(RESFILES)

MEMINFO.OBJ: MEMINFO.C $(H)

MEMINFO.EXE: MEMINFO.OBJ MEMINFO.RES MEMINFO.DEF
    link /NOD/A:16 MEMINFO,MEMINFO,nul,libw slibcew $(LIBS_EXT),MEMINFO.def
    rc MEMINFO.res MEMINFO.exe
```

第九章　资　源

除了数据段和代码段以外，一个 Windows 应用程序还可定义资源段，这些资源段也同样定位于可执行文件中。本章和接下来的两章介绍 Windows 中的资源。

9.1　资源的类型

Windows 的资源就是那些在应用程序运行期间通常不能改变的数据。与代码段相似的是，资源段仅一次性存在。因此，同一应用程序的若干实例可共享这些资源。

共有九种类型的资源：

- 键盘加速键
- 位图
- 对话框
- 图标
- 串表
- 自定义资源
- 光标
- 字模
- 菜单

这些资源可以用称为“资源描述文件”的文件进行描述。它是一个以 .RC 为扩展名的 ASCII 码文件，它能够引用不同的文件（见下面的图示）。

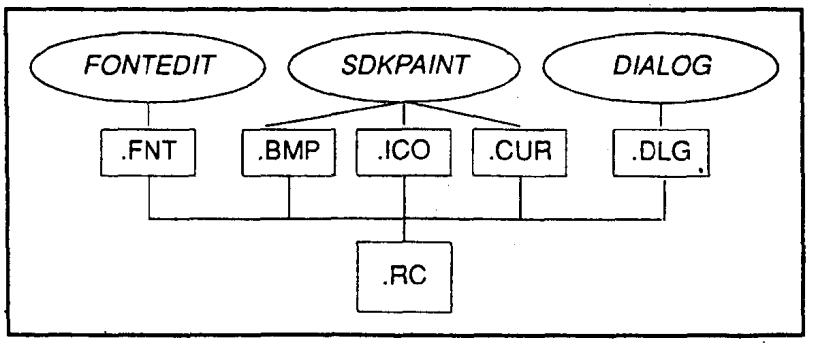


图 9.1　描述资源的资源描述文件

一个专用的编译器（即资源编译器 RC.EXE）将 .RC 文件翻译为二进制格式，将其插入

.EXE 文件中，并将引用项登录进资源表中(应用程序的头区域)。

使用资源编译器的方法有两种：

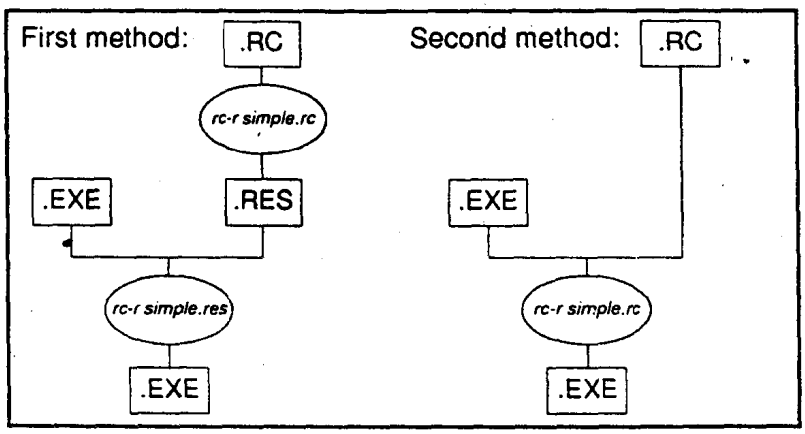


图 9.2　使用资源编译器的方法

第一种方法包括两个分开的程序设计步骤，这种方法是较为适合人意的，因为资源编译器能够通过置-r 开关来检查语法错误，而且如果必要的话，还能终止程序。这种方法将二进制格式(存放在 .RES 文件中)加入到可执行文件之中。

用户也可以将菜单、对话框等置于数据段中。但是，如果用户定义了独自的资源描述文件，那么互换此文件中定义的资源会更容易一些。例如，在一个应用程序中，为将一个英文菜单转换为一个西班牙文菜单，可以简单地用西班牙文的资源描述文件取代英文资源描述文件，并启动资源编译器即可。

用户无需运行 C 语言的编译器或链接器，仅当两个 RC 文件的菜单具有不相同的名字时，这个链接器才是必要的。这是必需的，因为资源编译器不能从资源表中删除资源，资源编译器所使用的预处理器为 RCPP. EXE，它能像 C 的预处理器一样识别和处理 #indude 语句和 #define 语句。

可以将资源划分为以下两组：

(1)单行语句(如光标资源)

(2)多行语句(如菜单资源)

在第一组(即单行语句)中，RC 文件包含对另外一个文件的引用，而不是包含着实际的资源数据。这个被引用的文件可能由另一个应用程序产生，例如实用程序 IMAGEDIT。这些文件的格式通常是二进制的。

当首次调用资源编译器时，这些文件就通过-r 开关置于 .RES 文件中，但是第二组资源(即多行语句资源)可以完全地定义在资源描述文件之中。下面是 .RC 文件中资源的结构(有一个例外情况)。

Name(名字) Type(类型) Data(数据)

在以下部分中将讨论资源，在这以后，用户应清晰地理解这种结构。

对所有单行语句资源，有两个选项可优先于数据规范。这两个选项包括一个加载选项(PRELOAD 或 LONDONCALL)和一个存盘选项(FIXED，MOVEABLE 和 DESCADABLE)。

缺省设置是 LOADONCALL、MOVEABLE 和 DISCARDABLE，但是选项 DISCARDABLE 不能与位图一起使用。

资源必须加载到使用它的源文件中。每个资源(用户定义的资源除外)都有独自的功能，即加载资源和取回一个句柄。

9.2 图 标

图标是一个小的图像，它代表一个最小化的应用程序，而且它还应用于消息框和对话框中。图标也可绘制在客户区中。

可以用来在主窗口中显示图标的方式有三种：在 WNDCLASS 结构中指定适当的显示选择，这样，所有引用同一个资源的窗口就有了相同的图标。被使用的预定义图标有五个。用户可以通过下面的名字来使用这些图标：

- IDI_APPLICATION：缺省图标；空矩形
- IDI_ASTERISK：小星形图标
- IDI_EXCLAMATION：小感叹号图标
- IDI_HAND：句号图标
- IDI_QUESTION：问号图标

在 WNDCLASS 结构中，会驱动用户寻求一种获得图标句柄的方式。为了获得这种句柄，需要使用函数 LoadIcon。这个函数有两个参数，对于预定义的图标，第一个参数必须设置为 NULL。在第二个参数中，要指定上面的五个 ID 值中的一个。

在前面诸章中，所有生成的应用程序都使用了预定义的图标 IDI_APPLICATION。

通常，一个预定义的图标不赋给一个应用程序。相反，用户必须创建一个能够清楚地代表应用程序的图标。这样，用户可以通过此图标识别这一应用程序。

可以使用实用工具 IMAGEDIT. EXE 来创建图标。IMAGEDIT. EXE 是软件开发工具包中的一部分。

创建一个图标后，可以将此图标存入自己的 .ICO 文件中，这个文件包含若干位图形式的图像。尽管这些图标看起来相似，但是它们却分别适用于不同分辨率的屏幕。

当一个应用程序调用图标时，它使用定义在 .RC 文件中的名字，然后由 Windows 决定哪个图标适合于当前的屏幕类型。因此，应用程序无需考虑屏幕类型。

ICO 文件由一条单行语句定义在 .RC 文件中。

名字	类型	数据
HouseIcon	ICON	HOUSE. ICO

关键字 ICON 可用来指定类型，用户可随心所欲地为图标定义任何一个名字，因为它是作为函数 LoadIcon 中的第二个参数而设置的，这个名字以字符串的形式传送。对于用户创建的所有图标，第一个参数表示当前实例的句柄(hInstance)。

```
wcIconcLass. hIcon = LoadIcon (hInstance,"HouseIcon");
```

可以用一个数字 ID 值来识别它。

```
Resource Script File:234 Icon house Ico
Source File:hIcon = LoadIcon (hInstance,MAKEINTRESOURCE (234));
```

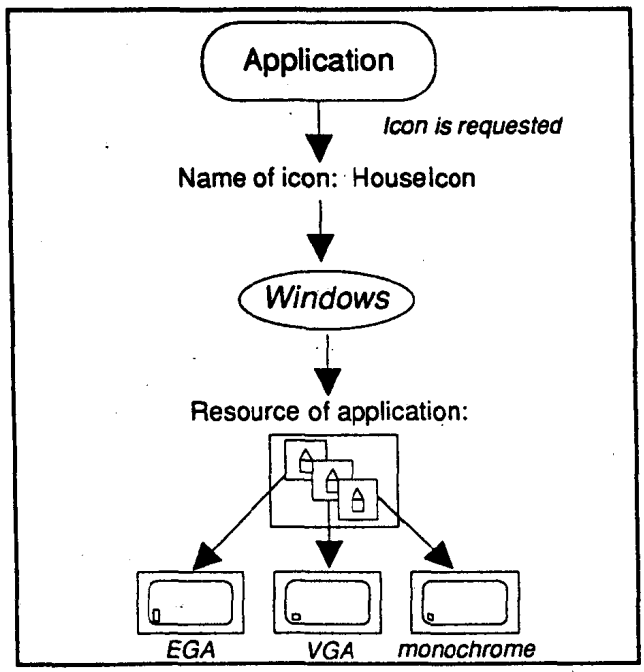


图 9.3　不同图标适合不同分辨率

宏 MAKEINTRESOURCE 将一个整数值转换为一个远程指针。在 Windows. H 中，这个宏还定义了 ID 值或者预定义图标的名字，诸如 IDI_APPLICATION 之类，当需在 .RC 文件中设置许多图标时，用户应使用这种方法，它使得可执行文件较小，而且函数 LoadIcon 的执行速度较快。

在应用程序运行期间，可以使用另外一个预定义在 .RC 文件中的图标（其句柄也可由函数 LoadIcon 获取）取代一个窗口类。在某个类注册后，函数 SetClassWord 能改变结构 WNDCLASS 中的某些域，包括图标的句柄。

如果被指定的图标已在内存中，函数 LoadIcon 仅仅获得这个对象的一个句柄，而不去重新加载它。这也同样适用于一个 Windows 应用程序的不同实例。

9.2.1　访问资源文件中的图标示例

这里给出一个使用资源描述文件和上述图标来访问图标的基本例子。这个例子以及列在其后的源码没有前面章节中出现的那些例子注释得详细，这个例子仅对最重要的行加了注释。有关大多数未注释程序段的细节，可参考前面章节的内容。

新函数	描述
LoadIcon	加载一个图标并取回其句柄

源代码：ICON1. C

```
/* * ICON1. C * * *
```

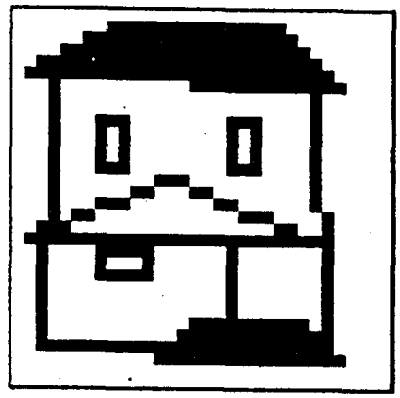


图 9.4

```
/* * * *
```

```
            hInstance,
            NULL);

    ShowWindow( hWnd,cmdShow );                          // Make window visible
    UpdateWindow( hWnd );                                  // Update window

    while (GetMessage(&msg,NULL,0,0))                      // Message reading
    {
        TranslateMessage(&msg);                          // Message translation
        DispatchMessage(&msg);                         // Send message to Windows
    }

    return msg. wParam;                                // Return wParam of last message
}

BOOL Icon1Init( HANDLE hInstance )
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcIcon1Class;

    wcIcon1Class. style = CS_VREDRAW | CS_HREDRAW;
    wcIcon1Class. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcIcon1Class. hIcon = LoadIcon( hInstance,"HouseIcon" );

                                                       // Load HouseIcon resource

    wcIcon1Class. lpszMenuName = NULL;
    wcIcon1Class. lpszClassName = "Icon1";
    wcIcon1Class. hbrBackground = GetStockObject( WHITE_BRUSH );
    wcIcon1Class. hInstance = hInstance;
    wcIcon1Class. lpfnWndProc = Icon1WndProc;
    wcIcon1Class. cbClsExtra = 0;
    wcIcon1Class. cbWndExtra = 0;

    if (! RegisterClass( (LPWNDCLASS) &wcIcon1Class ) )
        return FALSE;
    return TRUE;
}

/* * Icon1WndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL Icon1WndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    switch (message)
```

```
    {
        case WM_DESTROY:                                    // Destroy window
            PostQuitMessage(0);
            break;

        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:ICON1.DEF

```
NAME Icon1

DESCRIPTION 'Icon from resource'

EXETYPE WINDOWS

STUB 'WINSTUB.EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS Icon1WndProc @1
```

资源描述文件:ICON1.RC

```
HouseIcon ICON house.ico
```

MAKE 文件:ICON1.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = house.ico

all: ICON1.EXE

ICON1.RES: ICON1.RC $(RESFILES)

ICON1.OBJ: ICON1.C

ICON1.EXE: ICON1.OBJ ICON1.RES ICON1.DEF
    link /NOD/A:16 ICON1,ICON1,nul,libw slibcew,ICON1.def
```

```
rc ICON1.res ICON1.exe
```

正如前面提到的，使用独立资源的应用程序在编译和链接时需要额外的步骤，这些步骤如下所述：

(1)像先前的例子一样编译源码。在 DOS 提示符下编译源码 ICON1.C 要用到下面命令行：

```
cl -c -Gw -Zp -icon1.c
```

(2)调用加上-r 开关的资源编译器来准备 .RC 文件。下面的命令编译 ICON1.RC 文件。

```
rc -r icon1.rc
```

(3)将源码链接到模块定义文件中，产生一个无资源的可执行文件，下面命令行将 ICON1.OBJ 和 ICON.DEF 链接在一起，生成无资源的 ICON1.EXE 文件：

```
link /align:16 icon1,icon1.exe,libw+slibcew,icon1.def
```

(4)再次调用资源编译器以增添资源引用表，下面命令来完成这一任务：

```
rc icon1.res
```

ICON1.MAK 文件包含了这些在编译和链接期间用来添加离散资源的语句。按照下面的方式运行 NMAKE 以产生 ICON1 应用程序：

```
nmake icon1.mak
```

9.2.2 ICON1.EXE 是如何工作的

这个程序的唯一任务就是当窗口最小化后，显示一个用户图标。

这个图标由 IMAGEDIT.EXE 绘制，它代表一栋房子。在 .RC 文件中，这个图标被赋以名字“House.Icon”，在函数 LoadIcon 中亦指定了这个名字。

定义在 .RC 文件中的图标不能用于动态显示。例如，在应用程序 Clock 中，指针以最小化的图标进行移动。在这种情况下，WNDCLASS 结构将 hIcon 域置为 NULL，，这就告知 Windows，它也应该在最小化窗口时，向窗口过程发送消息 WM_PAINT。

当处理这个消息时，函数 IsIconic 查询窗口在当前是否是作为一个图标进行显示的。如果查询结果为真，方能用图形函数在应用程序中绘制图标。

9.2.3 动态图标访问示例

下面这个应用程序将使用图形函数生成一个最小化的图标。图 9.5 是当用户最小化这个应用程序时，图标的外观。

新函数	描述
isIconic	如果窗口作为一个图标显示则为真

源代码：ICON2.C

```
/* * ICON2.C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Generates minimized icon dynamically from within program * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                    // Include windows.h header file
```

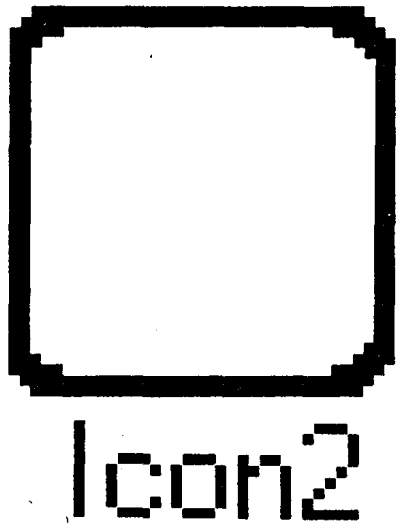


图 9.5　应用程序最小化时的图标外观

```
#if _MSC_VER >= 700                                    // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                         // higher? Disable warning 4028
#endif

long FAR PASCAL Icon2WndProc ( HWND,unsigned,WORD,LONG);
BOOL Icon2Init ( HANDLE );

int PASCAL WinMain ( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                        // Current & previous instance
LPSTR lpszCmdLine;                                     // Long ptr to string after
                                                       // program name during execution
int cmdShow;                                           // Specifies the application
                                                       // window's appearance
{
    MSG msg;                                           // Message variable
    HWND hWnd;                                         // Window handle

    if (! hPrevInstance)                               // Initialize first instance
    {
        if (! Icon2Init( hInstance ))
            return FALSE;
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Icon2",
            "Icon2",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,0,
            CW_USEDEFAULT,0,
            NULL,
```

```
            NULL,
            hInstance,
            NULL);

    ShowWindow( hWnd,cmdShow );                          // Make window visible
    UpdateWindow( hWnd );                                // Update window

    while (GetMessage(&msg,NULL,0,0))                    // Message reading
    {
        TranslateMessage(&msg);                          // Message translation
        DispatchMessage(&msg);                           // Send message to Windows
    }

    return msg. wParam;                                  // Return wParam of last message
}

BOOL Icon2Init( HANDLE hInstance )
{
/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcIcon2Class;

    wcIcon2Class. style = CS_VREDRAW | CS_HREDRAW;
    wcIcon2Class. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcIcon2Class. hIcon = NULL;                          // Note - no icon
    wcIcon2Class. lpszMenuName = NULL;
    wcIcon2Class. lpszClassName = "Icon2";
    wcIcon2Class. hbrBackground = GetStockObject( WHITE_BRUSH );
    wcIcon2Class. hInstance = hInstance;
    wcIcon2Class. lpfnWndProc = Icon2WndProc;
    wcIcon2Class. cbClsExtra = 0;
    wcIcon2Class. cbWndExtra = 0;

    if (! RegisterClass( (LPWNDCLASS) &wcIcon2Class ) )
        return FALSE;
    return TRUE;
}

/* * Icon2WndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL Icon2WndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    PAINTSTRUCT ps;
    HDC hDC;
    HPEN hPen,hPenOld;
```

```
    switch (message)
    {
        case WM_PAINT:                                          // Client area redraw
            hDC = BeginPaint( hWnd,&ps);
            if (IsIconic(hWnd))                                 // Minimized? Draw icon
            {
                hPen = CreatePen(PS_SOLID,3,RGB(0,0,0));
                                                                // Solid pen,black color
                hPenOld = SelectObject(hDC,hPen);
                RoundRect( hDC,ps. rcPaint. left,ps. rcPaint. top,ps. rcPaint. right,
                    ps. rcPaint. bottom,(ps. rcPaint. right - ps. rcPaint. left)/3,
                    (ps. rcPaint. bottom - ps. rcPaint. top)/2);
                                                                // Draw rounded rectangle when
                                                                // minimized icon is needed
                DeleteObject(SelectObject(hDC,hPenOld));
            }
            EndPaint( hWnd,&ps);
            break;
        case WM_DESTROY:                                        // Destroy window
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:ICON2. DEF

```
NAME Icon2

DESCRIPTION 'Dynamic icons'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS Icon2WndProc @1
```

MAKE 文件:ICON2. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo
```

```
.rc.res: ; rc -r $ *.rc

RESFILES =

all: ICON2.EXE

ICON2.OBJ: ICON2.C

ICON2.EXE: ICON2.OBJ ICON2.DEF
    link /NOD/A:16 ICON2,ICON2,nul,libw slibcew,ICON2.def
    rc ICON2.exe
```

9.2.4 ICON2.EXE 是怎样工作的

直至此应用程序执行时，才会生成这个程序的图标。由于在 WNDCLASS 结构中，hIcon 的值被置为 NULL，故而在最小化窗口时，也出现消息 WM_PAINT。

函数 IsIconic 决定了窗口的状态。如果其返回值为 TRUE，那么 ICON2.EXE 将画一个封闭的具有宽度为 3 的字模的矩形图标。

9.3 光 标

光标通常显示鼠标在屏幕上的移动。与图标相似，不同的选择可用来显示不同类型的鼠指针。

除缺省的箭头光标外，用户还可使用其他类型的预定义光标：

IDC_ARROW	缺省箭头光标
IDC_CROSS	十字形光标
IDC_IBEAM	用于文本的"I"字形光标
IDC_ICON	矩形光标
IDC_SIZE	带有 4 个箭头的十字形光标
IDC_SIZENEWS	方向为右上→左下的双箭头光标
IDC_SIZENS	方向为上→下的双箭头光标
IDC_SIZENWSE	方向为左上→右下的双箭头光标
IDC_SIZEWE	方向为左→右的双箭头光标
IDC_UPARROW	垂直箭头光标
IDC_WAIT	沙漏光标

函数 LoadCursor 用于获得预定义光标的句柄。第一个参数设置为 null，第二个参数包含着先前提及的 ID 名字中的一个。

在 WNDCLASS 结构中，正如用于图标句柄的域一样，也存在一个用于光标句柄的域。如果在这个域中指定了光标，那么这一光标适用于所有访问此类的窗口，即使它们是不同的实例。

每当一应用程序开始进行一个持续时间长的操作时(比如，从硬驱中读取数据)，用户应使用沙漏光标。

为了使操作不致于被鼠标的移动破坏，还应保持鼠标器不动。一旦操作结束，旧的光标

恢复，鼠标器亦可释放。

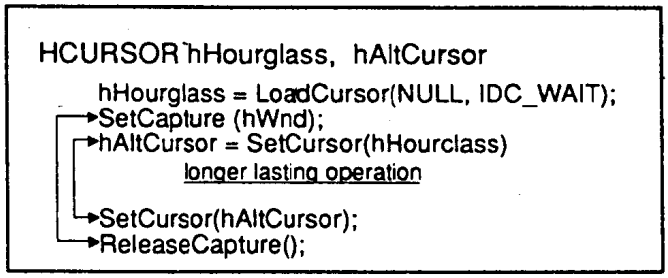


图 9.6　不同的光标

IMAGEDIT 可用来绘制用户自己的光标。在绘制的时候，用户还可设定一个敏感区（热点），以指定光标内哪一点是决定性的点。经此操作后得到的文件以 .CUR 为扩展名。

使用函数 LoadCursor 时，实例句柄传送给第一个参数，而且光标的名字以字符串的方式传递给第二个参数。这个名字亦在 .RC 文件中指定。

```
MouseCursor CURSOR mouse.cur
```

而在源代码中以下述形式出现：

```
wcCursorclass.hCursor = LoadCursor (hInstance,"MouseCursor");
```

由于在结构 WNDCLASS 中设置了 NULL 值而不是调用了函数 LoadCursor，故而当光标进入客户区域时，其形状不发生变化。不幸的是，这对用户而言，起到了反作用（例如，双箭头光标也许会显示在客户区中）。因此，只有当在窗口函数中设置光标时，才应使用这种方法。

举个例子来说，假如用户希望光标在某些事件发生时改变其形状，而在运行期间使用了这种方法，那么用户必须首先使用函数 LoadCursor 重新加载光标，而后用函数 SetCursor 将其设定为当前的光标。

不应在 WNDCLASS 结构中设置一种光标，然后又在消息 WM_MOUSEMOVE 中指定一种不同的光标；否则会引起鼠标指针颤动不定。

光标的设置通常仅适用一客户区。如果将光标定位于 Control 菜单或边界的上方，那么 Windows 将决定光标出现与否，除非使用了函数 SetCapture 使得鼠标保持不动。如果这样做了，那么当脱离客户区时，鼠标指针将不发生变化。

9.3.1　使用自定义光标示例

这个示例不包含一个光标类。相反，当鼠标移动时，会设置不同的鼠标指针，光标是否为当前的取决于是否按下了鼠标按钮。这里使用了三个鼠标指针（两个自定义的和一个预定义的）。其中两个自定义的光标分别命名为 MOUSE.CUR 和 SNMAKE.CUR。

新函数	描述
GetWindowWord	确定当前实例
LoadCursor	获得光标句柄
SetCursor	改变光标形状

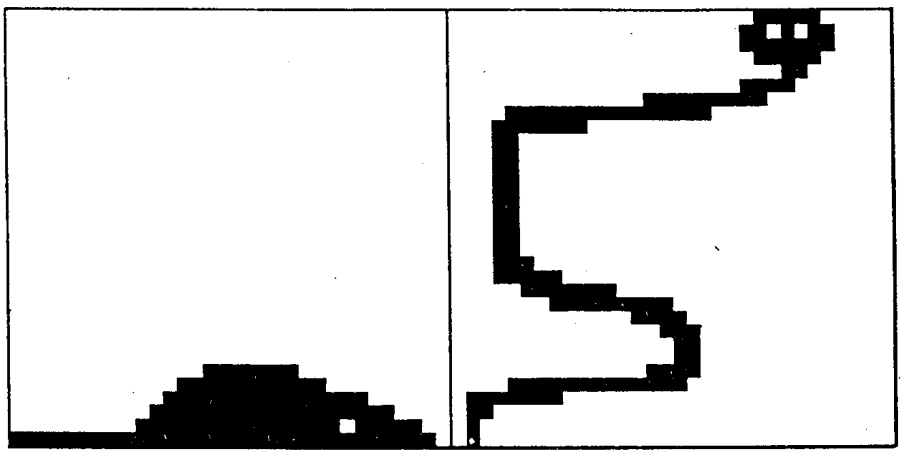


图 9.7　自定义的光标:MOUSE. CUR 和 SNAKE. CUR

源代码:CURSOR. C

```
/* * CURSOR. C * * *
```

```
    hWnd = CreateWindow("Cursor",
            "Cursor",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,0,CW_USEDEFAULT,0,
            NULL,
            NULL,
            hInstance,
            NULL);

    ShowWindow( hWnd,cmdShow );                          // Make window visible
    UpdateWindow( hWnd );                                  // Update window

    while (GetMessage(&msg,NULL,0,0))                      // Message reading
    {
        TranslateMessage(&msg);                          // Message translation
        DispatchMessage(&msg);                         // Send message to Windows
    }

    return (int)msg. wParam;                          // Return wParam of last message
}

BOOL CursorInit( hInstance )
HANDLE hInstance;
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcCursorClass;

    wcCursorClass. hCursor = NULL;
    wcCursorClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcCursorClass. lpszMenuName = NULL;
    wcCursorClass. lpszClassName = "Cursor";
    wcCursorClass. hbrBackground = (HBRUSH)GetStockObject( WHITE_BRUSH );
    wcCursorClass. hInstance = hInstance;
    wcCursorClass. style = CS_VREDRAW | CS_HREDRAW | CS_DBLCLKS;
    wcCursorClass. lpfnWndProc = CursorWndProc;
    wcCursorClass. cbClsExtra = 0 ;
    wcCursorClass. cbWndExtra = 0 ;

    if (! RegisterClass( &wcCursorClass ) )
        return FALSE;

    return TRUE;
}

/* * CursorWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
long FAR PASCAL CursorWndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    static BOOL bCapture = FALSE;
    static HCURSOR hCur;
    static HANDLE hInst;

    switch (message)
    {
        case WM_CREATE:
            hInst = GetWindowWord( hWnd,GWW_HINSTANCE);
            break;
        case WM_LBUTTONDBLCLK:
            if (bCapture)
            {
                ReleaseCapture();
                bCapture = FALSE;
            }
            else
            {
                SetCapture(hWnd);
                bCapture = TRUE;
            }
            break;
        case WM_MOUSEMOVE:                          // Change cursor appearance
                                                    // during mouse movement,
                                                    // depending on which mouse
                                                    // button is pressed
            switch (wParam)
            {
                case MK_LBUTTON:                    // Display mouse cursor
                                                    // if left button is pressed
                                                    // during movement
                    hCur = LoadCursor(hInst,"MouseCursor");
                    break;
                case MK_RBUTTON:                    // Display snake cursor
                                                    // if right button is pressed
                                                    // during movement
                    hCur = LoadCursor(hInst,"SnakeCursor");
                    break;
                default:                            // Load up arrow as default
                    hCur = LoadCursor(NULL,IDC_UPARROW);
                    break;
            }
            SetCursor(hCur);
            break;
        case WM_DESTROY:                            // Destroy window
            PostQuitMessage(0);
```

```
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:CURSOR. DEF

```
NAME Cursor

DESCRIPTION 'Cursor demonstration'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS CursorWndProc @1
```

资源描述文件:COUSOR. RC

```
MouseCursor CURSOR mouse. cur
SnakeCursor CURSOR snake. cur
```

MAKE 文件:CURSOR. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES = snake. cur mouse. cur

all: CURSOR. EXE

CURSOR. RES: CURSOR. RC $ (RESFILES)

CURSOR. OBJ: CURSOR. C

CURSOR. EXE: CURSOR. OBJ CURSOR. RES CURSOR. DEF
    link /NOD/A:16 CURSOR,CURSOR,nul,libw slibcew,CURSOR. def
    rc CURSOR. res CURSOR. exe
```

9.3.2 CURSOR. EXE 是如何工作的

WNDCLASS 结构中的域 wcCursorClass. hCursor 被设置为空。由于在窗口过程中,函数

LoadCursor 需要当前实例的句柄，而这个句柄则由消息 WM_CREATE 和函数 GetWindowWord 来确定，还可以使用参数 lparam 来确定句柄，lParam 中有一个指向包含着 hInstance 的结构的指针。如果按住鼠标左按钮将鼠标移进客户区域，那么光标呈现为一只老鼠。按鼠标的右按钮，鼠标将变成一条蛇。

当鼠标按钮不被按下时，就会出现预定义的包含 IDC_UPARROW 这一 ID 值的光标。

当用户双击鼠标左按钮以使鼠标器保持不动时，当前设置的光标将适用于整个屏幕。如果再次双击鼠标器则将被释放(参阅有关输入的章节中的鼠标示例部分)。

老鼠和蛇形光标由 SDK PAINT. EXE 创建，并作为光标资源写进 . RC 文件中。

9.4 其他单行语句

剩余的两个单行语句用来控制位图和字模。它们不直接定义在资源描述文件中，而是定义在与语句相连接的独立文件中。

9.4.1 位图

位图是 32 个像素宽、32 个像素高的图像，可应用位图在屏幕上画图(比如，系统框即为一个位图)并创建图案刷子。如果用应用程序 SDK PAINT 来创建位图，文件将自动接受扩展名 . BMP。在 . RC 文件中，资源类型为 BITMAP。正象图标和光标一样，位图也有一个独立的加载函数，此函数为 LoadBitMap。

Windows 中预定义的位图可分为两组。在 Windows 的早期版本中，所使用的所有位图的 ID 值均以 OBM_OLD 开头。另一种位图自 3. 0 版的 Windows 就已被使用，其中总计共有 30 多个预定义位图。只有到运行时刻才可频繁地创建位图。

9.4.2 字模

使用工具 FONTEDIT. EXE 可以创建用户自己的字模。用这个编辑器可以在屏幕上创建位图，并且定义字模的属性，结果生成一个具有扩展名 . FNT 的文件。使用下面过程在这个文件之外产生一个字模文件(例如 ROMANFON)。

在一个 . RC 文件中登录进 . FNT 文件，示例如下：

```
5 font Ownfont. fnt
```

这里可以有几种语句，用户必须为链接程序创建一个空白的代码段，在汇编语言中通常这样做。链接器还需要包含下列语句的定义文件：LIBRARY(用于定义任何名字)、DESCRIPTION(用于描述设备的特殊信息)以及具有选项 NONE 的 DATA。

接下来，链接程序便可以开始启动运行，它产生一种类型的 DLL 文件，且具有扩展名 . EXE，用户必须将此扩展名转换为 . FON。

9.5 串 表

串表是本章讨论的第一种多行语句资源。使用串表可保存独立于数据段的字符串。这样，这些字符串能够与不同的外语版本一起使用。

在 .RC 文件中，串表的结构与其他资源的结构是不相同的。既然一个 .RC 文件仅能定义一个串表，串表就没有独自的名字，即 ID 名。

然而个别串(指必须为单行的且由少于 255 个字符组成的串)却可由一个 ID 名来标识。与光标(它具有一个字符串表示其值)不同的是，串表的 ID 值是一个数字。

可以在字符串中指定控制字符 Tab、换行、回车，且指定为八进制常数(\011，\012，\015)。

```
#define ID_ERROR_1 13
#define ID_ERROR_1 14
#define ID_ERROR_1 15

STRINGTABLE
BEGIN
    ID_ERROR_1,"File not found"
    ID_DRROR_2,"Write error"
    ID_DRROR_3,"Not enough memory"
END
```

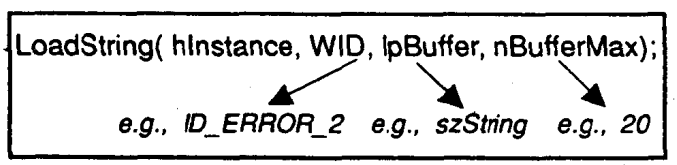


图 9.8

#define 语句是可选的，用户必须将其中一个串加载到一个缓冲区进行显示或者调用函数 LoadString 来处理它们。

第二个参数指定了被选中的串的 ID 值，第三个参数指向用来写入这个串的缓冲区，最后一个参数值确定了应该拷贝进这个缓冲区的字符个数。一个过长的串将被截断，函数的返回值指明了当前拷贝进缓冲区的字符个数。

9.6 菜 单

菜单由许多不同的项组成。这些项能够在一个窗口的标题条上以一行或多行显示。用户可以使用键盘或鼠标来选择一个菜单项，而后 Windows 发送一条消息，指出哪一个项被选中。

9.6.1 资源描述文件中的菜单定义

既然每个菜单必须有一个专用的名字，就可以在资源描述文件中同时定义多个菜单。

下面的菜单定义是一个资源描述文件的一部分(可以在相应盘的目录 WINPRGDE\RESOURCE\FRSTMENU 中，在名字 FRSTIENU 下找到源码以及这个资源描述文件)：

```
FirstMenu MENU
BEGIN
    POPUP "&File"
```

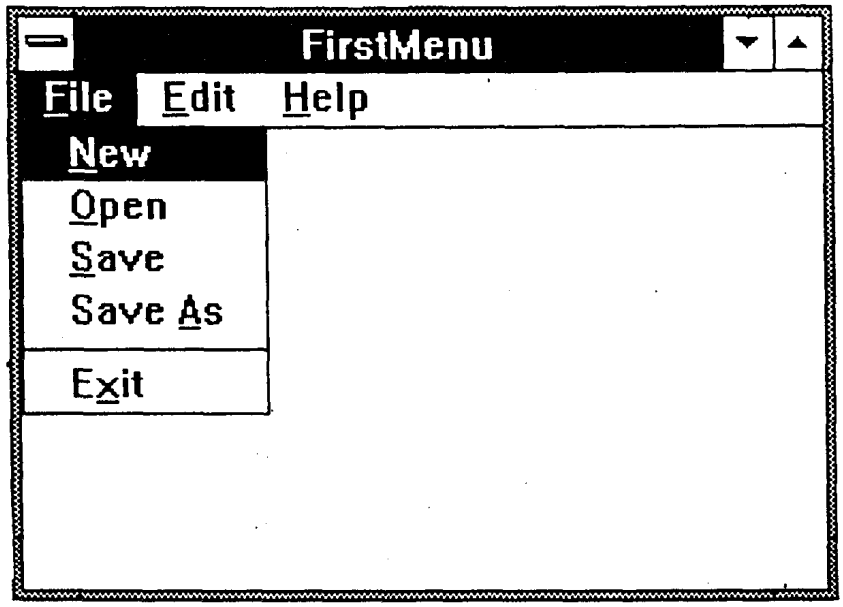


图 9.9

```
    BEGIN
        MENUITEM "&New", IDM_NEW
        MENUITEM "&Open", IDM_OPEN
        MENUITEM "&Save", IDM_SAVE
        MENUITEM "Save &As",IDM_SAVEAS
        MENUITEM "SEPARATOR
        MENUITEM "E&xit", IDM_EXIT
        MENUITEM "A&bout", IDM_ABOUT
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo", IDM_UNDO
        MENUITEM "Cu&t", IDM_CUT,GRAYED
        MENUITEM "&Copy", IDM_COPY,GRAYED
        MENUITEM "&Paste", IDM_PASTE
    END
    MENUITEM"\a&Help" IDM_HELP
END
```

菜单名字起到与源文本文件连结的作用，它位于 MENU 数据类型之前。一个菜单由各种不同的弹出式菜单和菜单命令组成，其中菜单命令通过 BEGIN 和 END 封闭起来。

一条 POPUP 语句定义了一个弹出式菜单，这个菜单每次包含一个弹出式菜单和/或菜单项。当选中了一个弹出式菜单时，Windows 将显示属于它的项列表。

一条 MENUITEM 语句用 ID 定义了一个菜单项。用户可以使用 #define 语句为一个 ID 值赋以一个符号名。然而，这个符号名应该尽可能地特殊，因为它将在窗口函数中使用。

除了显示文本外，菜单项还可以显示一个位图。用户既可以向弹出式菜单中增加选项，又可向菜单项中增加选项。在此示例中，菜单项 Cut 和 Copy 是不可执行的。

由于这两个菜单项包含选项 GRAYED，因此它们是不活跃的。有了这一选项，菜单项就以一种不同于菜单文本颜色的色彩出现。

另一个经常使用的选项是 CHECKED。这一选项在菜单项或弹出式菜单前面设置一个标记(即对号标记)。使用位操作 OR 可以连接若干选项。MENUITEMSEPARATOR 行产生一个不活跃项，此项用作两个其他项的分界线。如果这条语句用作两个其他项的分界线，且如果这条语句用在菜单行中，那么分隔符是垂直的。

某些控制字符可以用在一个文本串内。例如，若在菜单项的一个字符前放置一个"&"，那么在显示时，这个字符将带有下划线。为了选择出现这个字符的菜单项，按下 Alt 键和下划线(_)字符即可。

控制字符"\a"应该用于弹出式菜单或者菜单条中的项，因为它使得此项在对应行的右侧显示。否则，Windows 将自左向右填充菜单条，如果超出了空间，Windows 将分几行来显示它。

9.6.2 与源文本文件链接

在 .RC 文件中定义了菜单后，就需将此菜单与源文本文件链接起来了。主窗口和弹出式窗口可以有菜单，而子窗口却不能有。

将一个菜单与源文件链接起来的方法有两种。一种方法是在结构 WNDCLASS 中用一个串指定菜单的名字，这样，所有引用这个类的窗口将有相同的菜单。

```
wcMenuClass. lpszMenuName = "FirstMenu";
```

另一种方法是在创建一个窗口时给菜单命名。这样做了以后同一个类的窗口就可能有不同的菜单。函数 CreateWindow 要求一个加载后的菜单的句柄。使用函数 LoadMenu 可获得这一句柄。

```
CreateWindow("FirstMenu",
        "FirstMenu",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,0,
        CW_USEDEFAULT,0,
        NULL,
        LoadMenu(hInstance,"FirstMenu")
        hInstance,
        NULL);
```

如果在注册一个窗口类和创建一个窗口时皆指定了一菜单，那么这一特殊的窗口菜单将有优先权。

9.6.3 激活的消息

当用户用键盘或鼠标选中一个活动的菜单项时，将出现一条 WM_COMMAND 消息。因为选中的菜单项的 ID 名由参数 wParam 给出，故而这个消息用于逐个处理菜单项。通常参数 wParam 用于 Switch 语句：

```
case WM_COMMAND:
    switch( wParam )
    {
        case IDM_NEW:
```

```
            /* User-defined data */
            break;
        case IDM_OPEN:
            /* User-defined data */
            break;
        :
        default:
            break;
    }
    break;
```

然而，当选中一个弹出式菜单时，比如选中了 FILE，消息 WM_INITPUP-POP 会被传送给窗口函数。由于弹出式窗口无 ID，故而参数 wParam 包含弹出式菜单的句柄。当期望初始化弹出式菜单的单个菜单项时，也同样处理这一消息，否则这一消息将传送给缺省的窗口过程。

用户不必在以后担心下拉弹出式菜单或消息，因为这已由 Windows 处理过了。

用户还可以将自定义的项加入系统菜单中。一旦从系统框中选中了一个菜单项，Windows 便发送消息 WM_COMMNAD。

所有未经应用程序处理的窗口消息必须传送给函数 DefWindowsProc，否则，该窗口就不能被放大或关闭。

9.6.4 菜单修改

一般地，菜单是动态创建的而不是静态的，这就意味着菜单总是在变化着的。当一个应用程序运行时，能够改变菜单项的状态、增加或删除菜单项等。一个完整的菜单还可进行互换。

可以使用多种不同的函数来改变菜单，下面列出最重要的函数。

改变菜单项状态的函数

CheckMenuItem：设置或清除一个对号标志
EnableMenuItem：激活、禁止、加灰一个菜单项
HiliteMenuItem：设置或清除菜单条中一个项的加亮显示

改变菜单项存在性的函数

AppendMenu 增加一个菜单项
DeleteMenu 删除一个项及其弹出式菜单
InsertMenu 插入一个菜单项
ModifyMenu 修改一个菜单项
RemoveMenu 清除一个项但不删除它

改变整个菜单的函数

CreateMenu 创建一个空菜单
DestroyMenu 删除一个菜单

SetMenu 设置一个新菜单

下面的示例说明了其中大多数函数。还有几个以 Get 开头的函数，它们用来获取当前信息，这些信息可能是一个菜单项的状态或者菜单、弹出式菜单的句柄。

信息函数：

GetMenu	获得一个菜单句柄
GetMenuCheckMarkDimenSions	提供缺省对号标志的尺寸
GetMenuItemCount	提供菜单中项的个数
GetMenuItemID	提供一个项的 ID 名
GetMenuState	返回一个项的状态
GetMenuString	将某项的串拷贝到一缓冲区中
GetSubMenu	获得弹出式菜单的句柄
GetSystemMenu	提供对某控制菜单的访问

除了文本串以外，菜单项还可由位图组成。然而，仅能在运行期间指定这个位图，而不是在 .RC 文件中指定。

用户可以增加、插入一个新项或者修改一现存的项。为了使用这三个功能，用户必须传送标志 MF_BITMAP 和位图的句柄。

9.6.5 浮动的弹出式菜单

这种类型的菜单可以出现在屏幕的任何地方，而不是仅能显示在标题下方。例如，当按下鼠标右按钮时，弹出式菜单能够出现在鼠标指针指向的客户区域的任何方位。

建立一个浮动的弹出式菜单要使用函数 CreatePopupMenu 和 TrackPopupMenu。第一个函数可以创建一个空的弹出式菜单并获取其句柄。然而可以使用函数 AppendMenu 将菜单项填到空菜单中。

在浮动的弹出式菜单在屏幕上显示以前，要将菜单的坐标转换为屏幕坐标。菜单的坐标可以根据消息 WM_RBUTTONDOWN 的参数 lParam 来确定。

从这个菜单中选中某个项后，屏幕将自动清除菜单(见下列样本应用程序)。

9.6.6 定义自己的对号标记

除使用预定义的对号标记以外，用户能够创建位图以用作对号标记，还可以使用一个位图指示一个项是否被选中。

首先，必须绘制位图，例如使用 PAINT.EXE 可进行这一工作。记住自定义的对号标记应与预定义的标记有同样的尺寸。

在 .RC 文件中，位图被指定为单行语句资源。使用 Menus.LoadBitmap 可以加载源码中的每一个位图，然后，用函数 SetMenuItemBitmaps 为状态 MF_CHECKED 分配一个位图，为 MF_UNCHECKED 分配另一个位图。

用户要根据预定义对号标记的尺寸，经常性地调整所绘制的位图的尺寸。可以使用函数

GetMenuCheckMarkDimensions 来确定预定义的对号标志的尺寸，然后用函数 StretchBit 来调整位图的尺寸。如果位图的尺寸稍小于缺省值，那么就不必调整其尺寸了(见下面的应用例程)。

还可创建用户菜单项(比如那些比通常的项大得多的项)。给这些项赋以风格 MF_OWNERDRAW。然而，在这些实例中，由程序员负责绘制菜单项，而不是由 Windows 系统负责。首次绘制菜单时，窗口函数接收消息 WM_MESVREITEM，其参数 lParam 指向一个数据结构。在这个数据结构中，用户可以定义菜单项的尺寸。每当需要重画一个项时，Windows 便发送消息 WM_DRAWITEM。

9.6.7 菜单示例

下面这个示例说明了本节所讨论的许多专题。

新消息	描述
WM_COMMAND	当选中一个活动的菜单项时发送此消息

新函数	描述
AppendMenu	向某菜单增加一项
CheckMenuItem	设置或删除对号标记
ClientToScreen	将用户坐标映射为屏幕坐标
CreatePopupMenu	创建一个空的弹出式菜单
DeleteMenu	删除一个菜单项
DestroyMenu	删除指定的菜单
EnableMenuItem	改变项的活动状态
GetMenu	获得一个菜单句柄
GetMenuCheckMarkDemensions	提供缺省对号标记的尺寸
GetMenuString	向一个串拷贝某个菜单文本
GetMenuSub	获取一个弹出菜单的句柄
InsertMenu	插入一个菜单项
LoadBitmap	提供某个位图的句柄
LoadMenu	提供某个菜单的句柄
LoadString	从串表中加载一个文本
SetMenu	设置一个新的菜单
ModifyMenu	改变现存的菜单项
SetMenuItemBitmaps	将用户定义的位图设置为对号标记
TrackPopupMenu	显示一个浮动的弹出式菜单

源代码：MENU.C

```
/* * MENU.C * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstration of menus and checkmarks. * */
```

```
/* * * *
```

```
BOOL MenuInit( hInstance )                                    // Menu initialization
HANDLE hInstance;
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcMenuClass;

    wcMenuClass. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcMenuClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcMenuClass. lpszMenuName = (LPSTR)"Menu1";
    wcMenuClass. lpszClassName = (LPSTR)"Menu";
    wcMenuClass. hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcMenuClass. hInstance = hInstance;
    wcMenuClass. style = CS_VREDRAW | CS_HREDRAW;
    wcMenuClass. lpfnWndProc = MenuWndProc;
    wcMenuClass. cbClsExtra = 0 ;
    wcMenuClass. cbWndExtra = 0 ;

    if ( ! RegisterClass( &wcMenuClass ) )
        return FALSE;
    return TRUE;
}

/* * MenuWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL MenuWndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    static HMENU hMenu1,hMenu2,hSubMenu;                      // Menu handle variables
    static HMENU hPopup = 0;
    static HANDLE hInst;                                          // Instance handle
    char szNewItem[MAX_STRING_LENGTH];
    char chMenuText[MAX_STRING_LENGTH];
    POINT pt;
    static BOOL bMexico = TRUE;
    static HBITMAP hBitCheck1,hBitNotCheck1,hBitCheck2,hBitNotCheck2;
    static int i = 60;

    switch (message)
    {
        case WM_CREATE:
            hInst = GetWindowWord (hWnd,GWW_HINSTANCE);
            hMenu1 = GetMenu(hWnd);
            hMenu2 = LoadMenu(hInst,"Menu2");
            hSubMenu = GetSubMenu(hMenu1,2);
```

```
            hBitCheck1 = LoadBitmap(hInst,"BitCheck1");
            hBitNotCheck1 = LoadBitmap(hInst,"BitNotCheck1");

            hBitCheck2 = LoadBitmap(hInst,"BitCheck2");
            hBitNotCheck2 = LoadBitmap(hInst,"BitNotCheck2");

// Optional:                {
//                           DWORD dDim;
//                           WORD wHeight,wWidth;
//                           char chText[20];
//
//                           dDim = GetMenuCheckMarkDimensions();
// Displays checkmark        wHeight = HIWORD(dDim);
// dimensions in a           wWidth = LOWORD(dDim);
// message box               wsprintf(chText,"x = %d,y = %d",wWidth,wHeight);
//                           MessageBox(hWnd,chText,"Title",MB_OK);
//                           }

            SetMenuItemBitmaps(hMenu1,ID_AUST,MF_BYCOMMAND ,hBitNotCheck1,
                hBitCheck1);
            SetMenuItemBitmaps(hMenu1,ID_GRMN,MF_BYCOMMAND ,hBitNotCheck2,
                hBitCheck2);
            break;
        case WM_COMMAND:                                    // Get messages from menu bar
            switch(wParam)
            {
                case ID_INSERT:                                              // Insert item
                    LoadString( hInst,IDS_TIG,(LPSTR)szNewItem,MAX_STRING_LENGTH);
                    InsertMenu( hSubMenu,ID_KOAL,MF_BYCOMMAND,IDS_TIG,
                        (LPSTR)szNewItem);
                    EnableMenuItem( hMenu1,ID_INSERT,MF_GRAYED);
                    EnableMenuItem( hMenu1,ID_DELETE,MF_ENABLED);
                    break;
                case ID_DELETE:                                              // Delete item
                    EnableMenuItem( hMenu1,ID_DELETE,MF_GRAYED);
                    EnableMenuItem( hMenu1,ID_INSERT,MF_ENABLED);
                    DeleteMenu( hSubMenu,3,MF_BYPOSITION);
                    break;
                case ID_APPEND:                                              // Append item
                    LoadString( hInst,IDS_LIN,(LPSTR)szNewItem,
                        MAX_STRING_LENGTH);
                    AppendMenu(hSubMenu,MF_ENABLED,IDS_LIN,(LPSTR)szNewItem);
                    break;
                case ID_MODIFY:                                              // Modify item
                    LoadString( hInst,IDS_CHE,(LPSTR)szNewItem,
                        MAX_STRING_LENGTH);
                    ModifyMenu( hMenu1,ID_GIRA,MF_BYCOMMAND,IDS_CHE,
                        (LPSTR)szNewItem);
                    EnableMenuItem( hMenu1,ID_MODIFY,MF_GRAYED);
                    break;
                case ID_POPCRE:                                      // Create Pop-up item
```

```
            hPopup = CreatePopupMenu();
            EnableMenuItem( hMenu1,ID_POPDES,MF_ENABLED);
            EnableMenuItem( hMenu1,ID_POPCRE,MF_GRAYED);
            break;
        case ID_POPDES:                                    // Destroy Pop-up item
            DestroyMenu( hPopup);
            hPopup = 0;
            EnableMenuItem( hMenu1,ID_POPCRE,MF_ENABLED);
            EnableMenuItem( hMenu1,ID_POPDES,MF_GRAYED);
            break;
        case ID_MENU1:                                     // Change Menu item (toggle)
            SetMenu(hWnd,hMenu2);
            break;
        case ID_MENU2:                                     // Change Menu item (toggle)
            SetMenu(hWnd,hMenu1);
            break;
        case ID_GRMN:                                      // Germany item
            CheckMenuItem(hMenu1,ID_AUST,MF_BYCOMMAND |
                MF_UNCHECKED);
            CheckMenuItem(hMenu1,ID_GRMN,MF_BYCOMMAND | MF_CHECKED);
            break;
        case ID_AUST:                                      // Australia item
            CheckMenuItem(hMenu1,ID_AUST,MF_BYCOMMAND | MF_CHECKED);
            CheckMenuItem(hMenu1,ID_GRMN,MF_BYCOMMAND |
                MF_UNCHECKED);
            break;
        case ID_MEXI:                                      // Mexico item
            if (bMexico)
            {
                CheckMenuItem(hMenu1,ID_MEXI,MF_BYCOMMAND |
                    MF_UNCHECKED);
                bMexico = FALSE;
            }
            else
            {
                CheckMenuItem(hMenu1,ID_MEXI,MF_BYCOMMAND |
                    MF_CHECKED);
                bMexico = TRUE;
            }
            break;
        case ID_CATT:                                      // Cat item
        case ID_DACH:                                      // Dachshund item
        case ID_POOD:                                      // Poodle item
        case ID_SETT:                                      // Setter item
        case ID_TERR:                                      // Terrier item
        case ID_GPIG:                                      // Guinea Pig item
        case ID_PRKT:                                      // Parakeet item
            if (hPopup ! = 0)
            {
                GetMenuString(hMenu1,wParam,(LPSTR)chMenuText,
                    MAX_STRING_LENGTH,MF_BYCOMMAND);
```

```
                    AppendMenu(hPopup,MF_ENABLED,i++,(LPSTR)chMenuText);
                }
                break;
            default:
                return (DefWindowProc(hWnd,message,wParam,lParam));
                break;
        }
        break;
    case WM_RBUTTONDOWN:                                    // Right button pressed?
        if (hPopup)
        {
            pt = MAKEPOINT( lParam);
            ClientToScreen (hWnd,(LPPOINT)&pt);
            TrackPopupMenu( hPopup,0,pt.x,pt.y,0,hWnd,NULL);
        }
        break;
    case WM_DESTROY:                                        // Destroy window
        PostQuitMessage(0);
        break;
    default:
        return (DefWindowProc( hWnd,message,wParam,lParam ));
        break;
  }
  return(0L);
}
```

模块定义文件:MENU. DEF

```
NAME Menu

DESCRIPTION 'Menu example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS MenuWndProc @1
```

资源描述文件 MENU. RC

```
/* * MENU. RC * * *
```

```
BitCheck1 BITMAP CHECK1. bmp
BitNotCheck1 BITMAP NOCHECK1. BMP
BitCheck2 BITMAP CHECK2. bmp
BitNotCheck2 BITMAP NOCHECK2. BMP

Menu1 MENU
BEGIN
    POPUP "Change"
    BEGIN
        MENUITEM "Insert", ID_INSERT
        MENUITEM "Delete", ID_DELETE,GRAYED
        MENUITEM "Append", ID_APPEND
        MENUITEM "Modify", ID_MODIFY
        MENUITEM "Create Pop-up", ID_POPCRE
        MENUITEM "Destroy Pop-up", ID_POPDES,GRAYED
        MENUITEM "Change Menus",ID_MENU1,
    END

    POPUP "Countries"
    BEGIN
        MENUITEM "Australia",ID_AUST
        MENUITEM "Germany",ID_GRMN
        MENUITEM "Mexico",ID_MEXI, CHECKED
    END

    POPUP "Pets"
    BEGIN
        POPUP "House Pets"
        BEGIN
            MENUITEM "Cat", ID_CATT
            POPUP "Dog"
            BEGIN
                MENUITEM "Dachshund",ID_DACH
                MENUITEM "Poodle",ID_POOD
                MENUITEM "Setter",ID_SETT
                MENUITEM "Terrier",ID_TERR
            END
            MENUITEM "Guinea Pig",ID_GPIG
            MENUITEM "Parakeet", ID_PRKT
        END

        MENUITEM "Elephant",ID_ELEP
        MENUITEM "Giraffe", ID_GIRA
        MENUITEM "Koala", ID_KOAL
        MENUITEM "Emu", ID_EMU
    END
END

Menu2 MENU
BEGIN
    POPUP "Change"
```

```
    BEGIN
        MENUITEM "Change Menus",ID_MENU2
    END

    POPUP "Plants"
    BEGIN
        MENUITEM "Tree", ID_TREE
        MENUITEM "Flower",ID_BLOO
        MENUITEM "Shrub",ID_SHRB
    END
END

STRINGTABLE
BEGIN
    IDS_TIG, "Tiger"
    IDS_LIN, "Lion"
    IDS_CHE, "Cheetah"
END
```

头文件:MENU. H

```
#define ID_INSERT 10
#define ID_DELETE 11
#define ID_APPEND 12
#define ID_MODIFY 13

#define ID_POPCRE 14
#define ID_POPDES 15
#define ID_MENU1 16
#define ID_MENU2 17

#define ID_AUST 20
#define ID_GRMN 21
#define ID_MEXI 22

#define ID_CATT 30
#define ID_DACH 310
#define ID_POOD 311
#define ID_SETT 212
#define ID_TERR 213
#define ID_GPIG 32
#define ID_PRKT 33

#define ID_ELEP 41
#define ID_GIRA 42
#define ID_KOAL 43
#define ID_EMU 44

#define ID_TREE 51
#define ID_BLOO 52
#define ID_SHRB 53
```

```
#define MAX_STRING_LENGTH 20

#define IDS_TIG 100
#define IDS_LIN 101
#define IDS_CHE 102
```

MAKE 文件：MENU.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = check1.bmp check2.bmp nocheck1.bmp nocheck2.bmp
H = MENU.H

all: MENU.EXE

MENU.RES: MENU.RC $(RESFILES)

MENU.OBJ: MENU.C $(H)

MENU.EXE: MENU.OBJ MENU.RES MENU.DEF
    link /NOD/A:16 MENU,MENU,nul,libw slibcew,MENU.def
    rc MENU.res MENU.exe
```

运行 NMAKE 文件以生成 MENU 应用程序：

```
nmake menu.mak
```

MENU.EXE 如何工作

这个应用程序可生成两种不同的菜单，它们都定义在 .RC 文件中。这两种菜单交替地出现在菜单条中。每个菜单均包含项 MenuChange，它使用函数 SetMenu 来显示其他菜单。所要求的句柄已在消息 WM_CREATE 期间获得。

由于在数据结构 WNDCLASS 中，第一个菜单是作为菜单类登录的，故而函数 GetMenu 可提供其句柄。但是，第二个菜单必须使用函数 LoadMenu 进行加载。

弹出式菜单“Change”中的项 Insert、Delete、Append 和 Modify 都引用了“Pets”这一弹出式菜单。当选中其中“Insert”项后，具有 IDS_TIG 这一 ID 名的字符串将通过函数 LoadString 从串表(此串表亦在 .RC 文件中)加载进一缓冲区中。

其次，这个串被插入弹出式菜单中具有 ID 名 ID_KOAL 的菜单项的前面，这个过程由函数 InsertMenu 来完成。现在，若选中了 Pets 菜单，就会发现其中的项 Tiger，由于这一新项不应出现多次，因此选项 Insert 必须要有 GRAYED 状态。在 .RC 文件中，用 GRAYED 初始化的项 Delete 被设置为可执行的。

如果选中了项 Delete，那么函数 DeleteMenu 将删除弹出式菜单“Pets”中的新项。将标志设置为 MF_BYPOSTION。这个标志使得第二个参数指示为某个弹出式菜单的位置，而不是其 ID 名。

第一个选项所在的位置为 0，它所要求的弹出式菜单的句柄由函数 GetSubMenu 在消息 WM_CREATE 期间确定。最后互换项 Insert 和 Delete 的特征。菜单项 Append 用来从串表中加载进文本串“Icon”，并将其附加在弹出式菜单 Pets 的尾部。这个过程是由函数 AppendMenu 来完成的。这个新的菜单项的状态被设置为 MF_ENABLED。并且指定了这一项的 ID 值，这个值与它在串表中作为一个字符串时的 ID 值相同；但是，并非一定要这样。

函数 ModifyMenu 用标识符 ID_GIRA 来修改菜单项的文本及其 ID 值，直到这一变化发生，ID_GIRA 才是无效的。项 Giraffe 的名字变换为 Cheetah。第三、四、五个参数包含了对新项的说明。

弹出式菜单 Change 的最后两项是 Create Pop－up 和 Destroy Pop－up，它们适用于浮动的弹出式菜单。项 Create Pop－up 可用来创建一个空的弹出式菜单。

当从弹出式子菜单 HousePets 中选择某项时，这一空菜单即被填充。函数 GetMenuString 将由某个 ID 值确定的菜单文本串加载进一个缓冲区中，这个缓冲区可通过函数 AppendMenu 附加在菜单的尾部。

一旦一个空菜单成为可用的，就可以用函数 DestroyMenus 来撤消它，然后弹出式菜单 Change 的句柄即被设置为 0。函数 DestroyMenu 释放掉所占用的内存。无论可时选中 CreatePopup 或 DestroyPopup，都必须将状态 MF_ENABLED 转换为 MF_GRAYED，或者相反（即将 MF_GRAYED 转换为 MF_ENABLED）。

如果当前存在浮动的弹出式菜单，那么当用户按下鼠标右按钮时，它将显示在任何鼠标光标所处的地方。鼠标的位置在消息 WM_RBUTTONDOWN 期间由其参数 lParam 提供，利用宏定义 MAKEPOINT 将鼠标位置置于结构 POINT 的一个变量中。

函数 ClientToScreen 将某项的坐标转换为屏幕坐标，它适用于屏幕的左上角。这对于用来显示弹出式菜单的函数 CheckMenuItem 来说是必需的。

弹出式菜单 Countries 具有三个菜单项。每个项均有一个不同的对号标记。在每次选择之后，选项 Mexico 就使用函数 CheckMenuItem 改变其状态，这个选项在显示时具有预定义的对号标记。

其他两个项互为切换开关 on 和 off，这两个项通常使用自定义的对号标记来达此目的，即都有 MF_CHECKED 状态和 MF_UNCHECKED 状态。

用函数 LoadBitmap 可以在消息 WM_CREATE 期间加载为用作这些对号标记而画的位图，这些位图由 SDK PAINT. EXE 创建，其尺寸为 10×10。在这种情况下，没有必要根据预定义的对号标记来调整位图尺寸。举一个实例来说，过去曾使用的若干行现在却处于注释行中了。用户可以用这些行显示一个具有预定义对话标记的消息框。对于每个菜单项，函数 SetMenuItemBitmaps 为选中的状态传送一个已加载的位图的句柄，为未选中的状态传送另一个。

.RC 文件中包含了四个单行语句，它们分别应用于一个位图、一个串表以及两个菜单。在第一个菜单中，项 Delete 和 DestroyPopup 初始化为状态 GRAYED，而项 Mexico 则有初始化状态 CHECKED。所有项的 ID 值均定义在一个独立的头文件中，此文件将与 .RC 文件和 .C 文件链接起来。

9.7 键盘加速键

加速键用于不打开弹出式菜单而选择其菜单项，这使得应用程序的用户友好性更进了一步。

这些键属于资源描述文件，它们存在表 ACCELERATORS 中，此表也包含了应用程序中对应加速键的菜单项的 ID 值。有 4 种方法定义这些键。

```
AccelMenu ACCELERATORS
BEGIN
    "^I", ID_INSERT
    "A", ID_APPEND
    77, ID_MODIFY
    VK_DELETE, ID_DELETE, SHIFT, ALT, VIRTKEY
END
```

数值 77 是字符“M”的十进制 ASCII 码。必须为所有的虚拟键(例如所有的功能键)写上关键字 VIRTKEY。

```
Menu1 MENU
BEGIN
    POPUP "&Change"
    BEGIN
        MENUITEM "Insert\t CONTROL+I", ID_INSERT
        MENUITEM "Delete\tDEL+SHIFT+ALT", ID_DELETE, GRAYED
        MENUITEM "Append\t A", ID_APPEND
        MENUITEM "Modify\t M", ID_MODIFY
        MENUITEM "PopupCreate", ID_POPCRE
        MENUITEM "PopupDestroy", ID_POPDES, GRAYED
        MENUITEM "MenuChange", ID_MENU1
    END
    ⋮
END
```

这样，Windows 就将这些键识别为加速键。用户必须用函数 LoadAccelerators 来加载这个表，其第二参数是加速键表的名字(例如 AcceMenu)。

还必须在消息循环中加进一个额外的功能：

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

每个由函数 GetMessage 从应用程序消息队列中取来的消息均传送给函数 TranslateAccelerator。借助已加载的加速键表句柄，如果这个函数识别出这个消息是加速键，那么它将把此消息翻译成消息 WM_COMMAND，且返回一个值 TRUE，将消息 WM_COMMAND 传递给其窗口函数。

因此，函数 DispatchMessage 不能运行，加速键的标识符与此同时也是菜单项的 ID 值。它在参数 wParam 中得以传送。

9.8 对话框

对话框就是一个弹出式窗口，应用程序可借助它与用户交换信息。这里使用了不同的控件来显示对话框的内容，每个对话框均由一个窗口及其窗口函数组成。

9.8.1 控件

一个控件就是一个子窗口。这个子窗口具有 Windows 中一个预定义的窗口类及其窗口函数。因此，一个特殊窗口的外观和任务是已经设置好了的。用户亦可以创建控件，使其适用于应用程序中定义的类。这称为用户控件。共有七种预定义的控件类：

BUTTON	创建一个小窗口，用于键入选择(Yes、No 等)
COMBOBOX	创建将 EDIT 和 LISTBOX 联合起来的窗口
EDIT	创建一个用于编辑文本的窗口
LISTBOX	创建一个窗口，在其中用户可以从一个列表中选择一个名字
MDICLIENT	创建一个用户窗口，用以替代在多文档界面(MDI)中的客户区域
SCROLLBAR	创建一个外观和功能皆类似一个滚动条的窗口
STATIC	创建一个包含固定文本或简单图形的窗口

与这些控件类相随的还有风格参数，它们影响控件的外观以及操作的方式。例如，用户可以将按钮控件和风格参数一起使用，以选择四种不同种类的按钮中的一个。

风格参数	描述
BS_PUSHBUTTON	椭圆形按钮
BS_CHECKBOX	矩形按钮
BS_RADIOBUTTON	圆形按钮
BS_GROUPBOX	可对其他按钮分组的矩形按钮

一个下推按钮还可以有风格参数 BS_DEFPUSHBUTTON。为简化程序设计，风格参数 BS_AUTOCHECKBOX 和 BS_AUTORADIOBUTTON 能够用于复选框和单选按钮。

是否自动地检查复选框和单选按钮取决于它们先前的状态，而不必向按钮发送消息 BM_SETCHECK，因为单选按钮仅仅允许“1:n”选择，所以在同一组内曾经设置的单选按钮将被关闭。

除用于对话框外，控件还可出现在正常的窗口之中。每个控件由其 ID 值进行标识，这类似于菜单的 ID 值。

用户可以通过输入来处理相应的控件。然后这个控件就将某种输入信息传送给其父窗口，这是通过向其父窗口的窗口过程发送消息 WM_COMMAND 来进行的。

消息 WM_COMMAND 的参数 wParam 包含着某个控件的 ID 值。参数 lParam 的低位字中包含着相应控件的句柄，其高位字则包含着通报代码。这个代码提供了有关在此控件中正在进行的操作的详细信息。

基于某控件导致的结果，可将通报码划分为以下四组：

类型	示例
按钮通报码	BN_CLICKED
编辑控件通报码	EN_SETFOCUS
列表框通报码	LBN_DBLCLK
组合框通报码	CBN_EDITCHANGE

例如，在一个对话框中，当用户在一个单选按钮上双击鼠标按钮时，这个按钮会向对话框窗口函数发送消息 WM_COMMOND 以及通报码 BN_DOUBLECLICKED。

如果期望某个控件完成确定的任务，父窗口将使用函数 SendMessage 向其子窗口发送一条控制消息。对于这个发送消息的父窗口来说，它必须知道相应控件的句柄，因为这个句柄是消息的第一个参数。

在对话框中，用户可以使用函数 GetDlgItem 确定这一句柄，也可以使用函数 SendDlgMessage，这个函数仅使用控件的 ID 值。

```
hControl = GetDlgItem (hDlg_ID);
SendMessage (hControl,message,wParam,lParam)
```

或者

```
SendDlgItemMessage (hDlg,Control_ID,message,wParam,lParam);
```

有不同的控件消息，它们分别适用于某个确定的类，而且根据这个类的不同，它们分别以不同的字母开头(如 BM_、EM_、LB_、CB_等)。

例如，有一个消息 BM_SETCHECK 被传送给单选按钮或矩形框，以改变当前状态。如果参数具有一个不为 0 的值，那么就设置一个对号标记，否则对号标记就被删除。使用函数 CheckDlgButton 或者函数 CheckRadioButton 可以得到同样的结果。

```
SendDlgItemMessage (hDlg,Control_BM_SETCHECK,TRUE,OL)
```

或者

```
CheckDlgButton (hDlg,Control_ID,TRUE);
```

或者

```
CheckRadioButton (hDlg,FirstC_ID,LastC_ID,Control_ID);
```

函数 CheckRadioButton 仅能够用于一组单选按钮而不可用于单一的按钮，它的第二个参数是一组按钮中第一个按钮的 ID 值，第三个参数包含着一组中最后一个按钮的 ID 值，第四个参数表示即将被设置的按钮。

组内所有其他按钮的对号标记都要删除掉，即便是没有风格 BS_AUTORADIOBUTTON 的单选按钮亦应如此。

图 9.10 展示了使用最为繁频的控件。

9.8.2 创建对话框

创建对话框的方式有两种，一种方法是在 .RC 文件中逐行指定对话框，然而，这此定义一个菜单要复杂得多，因为每一个控制均需要有关它在对话框中定位的信息，另一种方法是使用实用工具 DIALOG.EXE 在屏幕上创建对话框。对话编辑还能够使用户将控制和用户自定义的窗口过程集成在一起。这样一来，用户便可以像缺省控制那样使用它们。

名为 TEST 的菜单定位于 Dialog 这一弹出式菜单里，这个项目可以用来测试一个做好

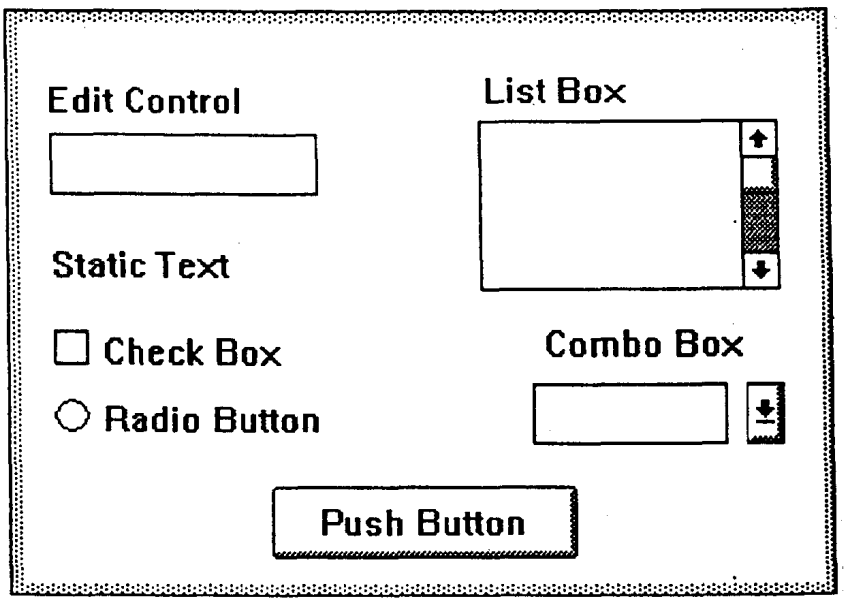


图 9.10　使用最为频繁的控件

的对话框。例如，可以用这个项目来检测是否用风格 BS_ AUTORADIO，BS_AUTOCHECKBOX 和 BS_STATE 定义了所有的单选按钮和复选框。

用户还能确定风格 WM_GROUP 和 WS_TABSTOP 是否被指定给了适当的控制。这些风格参数所提供的信息是关于何时使用 Tab 键和光标键使输入焦点在控制之间移去。通常光标键用于处理属于同一组的控制，而 Tab 键则用来处理属于不同组的控制。例如，一组控制可由环境在一个分组框中的三个单选按钮组成。

在保存了对话框后，对话框编辑器将生成两个文件：一个具有 .DLG 扩展名的 ASCI 码文件。一个扩展名为 .RES 的二进制格式文件，如果使用此编辑器来改变对话框，则需要第二个文件，虽然可以直接改动 .DLG 文件，但这些改变不能自动地为 .RES 文件接受。

首先，在 .RC 文件中需要：

```
#include "filename. dlg"
```

然后启动资源编译器，根据 .RC 文件创建一个新的 .RES 文件。即使这个文件包含其他资源，也可以使用对话框编辑器来读取此文件。除使用这个 #include 语句外，还可将整个 .DLG 文件拷贝到 .RC 文件中 。然而，若拷贝了若干个对话框，就很难保证它们是有组织的。

一个 .DLG 文件大致如下所示：

```
MYBOX DIALOG LOADONCALL MOVEABLE DISCARDABLE 46,19,247,169
CAPTION "Default"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Text Search",ID_SRCH,"button",BS_PUSHBUTTON | WS_GROUP | WS_TABSTOP
        | WS_CHILD,11,45,50,14
    CONTROL "Accept Title",ID_NEW,"button",BS_PUSHBUTTON | WS_TABSTOP
```

```
        | WS_CHILD,3,83,68,14
    CONTROL "New Title",-1,"static,SS_LEFT | WS_GROUP
        | WS_CHILD,81,17,54,8
    CONTROL "",ID_COMB,"combobox",CBS_DROPDOWN | WS_VSCROLL
        | WS_CHILD,81,28,55,85
    CONTROL "Lines",-1,"button",BS_GROUPBOS | WS_GROUP
        | WS_CHILD,153,15,65,61
    CONTROL "Left->Right",ID_LERI,"button",BS_AUTOCHECKBOX | WS_TABSTOP
        | WS_CHILD,155,33,58,12
    CONTROL "Right->Left",ID_RILE,"button",BS_AUTOCHECKBOX
        | WS_CHILD,155,49,59,12
    CONTROL "Icon",-1,"button",BS_GROUPBOX | WS_GROUP
        | WS_CHILD,151,96,64,63
    CONTROL "Stop",ID_ICON,"button",BS_AUTORADIOBUTTON | WS_TABSTOP
        | WS_CHILD,155,109,47,1
    CONTROL "Question",ID_ICON+1,"button",BS_AUTORADIOBUTTON
        | WS_CHILD,155,125,47,12
    CONTROL "Esclamation",ID_ICON+2,"button",BS_AUTORADIOBUTTON
        | WS_CHLID,155,140,58,12
    CONTROL "OK",IDOK,"button",BS_DEFPUSHBUTTON | WS_GROUP | WS_TABOUP
        | WS_CHILD,18,144,36,14
    CONTROL "Cancel",IDCANCEL,"button",BS_PUSHBUTTON | WS_TABSTOP
        | WS_CHILD,75,144,39,14
END
```

其中的对话框“MyBox”将在后面的例子中使用。“MyBox”包括了十三个控件：一个组合框、一个静态域、两个分组框、四个下推按钮、两个复选框、三个单选按钮。每一控件都有独自的 ID 值。这个例子中大多数控件的 ID 值都定义在一个头文件中。由于技术上的原因而不能使用的控件可以有相同的 ID 值(例如-1)。

名为 IDOK 和 IDCANCEL 的 ID 值定义在头文件 WINDOWS.H 中，它们用于下推按钮 CANCEL 和 OK。用户可以使用 Tab 键来启动具有风格 WS_TABSTOP 的控件。

用户还可以通过选择对话编辑器中的 ICON 选项来画自己的图标。正如使用许多其他控件一样，用户也能为这个图标分配文本。在 .RC 文件中，这个文本等同于 ICON 语句中图标的名字。

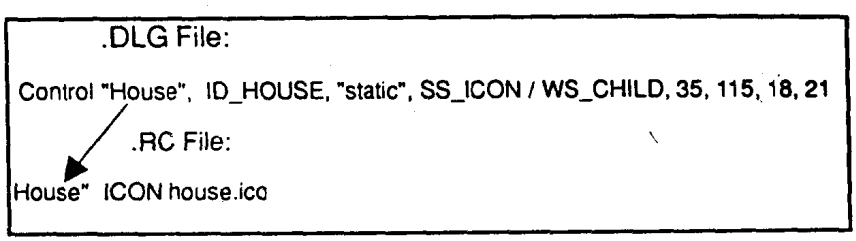


图 9.11

在此例中选择了名字“House”。此静态类的这个控件的 ID 值是 ID_HOUSE，包含这个图

标的文件为 HOUSE. ICO。

9.8.3 对话框类型

有两种类型的对话框，二者之间的差异在于它们对由其所激活的窗口作出响应的方式。

- 非模态对话框
- 模态对话框

当一个窗口过程遇到一个非模态对话框时，这个对话框将与隶属于这个窗口过程的窗口并驾齐驱，而且不以任何方式影响这个窗口，其他的应用程序亦可继续运行。大多数非模态对话框具有风格 WS_POPUP、WS_CAPTION、WS_BORDED 和 WS_SYSTEMMENU。选择 Control 菜单中的 Close 项可以关闭这个对话框。

然而，一个模态对话框则将其父窗口置于备用状态。这样，直至退出此对话框，父窗口才可继续运行。

除了应用程序模态对话框外，还有一个系统模态对话框，这个对话框将终止所有其他 Windows 程序，直至用按钮 CANCEL 或 OK 将其关闭为止。由于系统模态对话框干涉了 Windows 多任务处理，故而用户应尽可能少地使用它。模态对话框具有风格 WS_POPUP 和 WS_DLGFRAME，它们可在对话框周围生成一个框架。

9.8.4 对话框例程

一个对话框例程就是一个回调函数，当一个用于对话函数的消息出现时，Windows 就调用这个函数。对话框例程与正常的窗口函数很类似，例如，必须有 4 个参数传送给这一例程，第一个参数通常是 hDlg，它是一个对话框窗口句柄。

```
BOOL FAR PASCAL DialogProc( hDlg,wMsg,wParam,lParam)
HWND hDlg;
WORD wMsg;
WORD wParam;
DWORD lParam;
```

这个函数必须用 FAR PASCAL 来定义，并且要与定义文件中的窗口函数一起用 EXPORTS 语句指定。

在对话例程内部，仅仅对于这个对话框很重要的消息才用一个开关提示符进行处理。那些在缺省子程序中的不重要的消息不被传送给缺省的对话框例程。但是，对话例程的返回值做到了这一点，如果这个返回值被设置为 FALSE，则 Windows 调用一个缺省的例程；如果返回值为 TRUE，则处理这些消息。

多数对话例程都处理消息 WM_INITDIALOG 和 WM_COMMAND，在对话框出现于屏幕上前不久，消息 WM_INITDIALOG 将被发送。例如，用户能够使用函数 SetFocus 将输入焦点传送给一个确定的控件。

但是，在这种情况下，即使消息本身被处理了，对话例程的返回值也必须为 FALSE。如果用户取得了返回值 TRUE，那么 Windows 就把输入焦点重新设置给第一个具有风格参数 WS_TABSTOP 的控件。

```
BOOL FAR PASCAL DialogProc( hDlg,wMsg,wParam,lParam )
HWND hDlg;
WORD wMsg;
```

```
WORD wParam;
DWORD lParam;
{
    switch( wMsg )
    {
        case WM_INITDIALOG:
            SetFocus( … );
            return( FALSE );
            break;
        case WM_COMMAND:
            switch( wParam )
            {
                case IDOK:
                    EndDialog( hDlg,1 );
                    return( TRUE );
                case IDCANCEL:
                    EndDialog( hDlg,0 );
                    ruturn( TRUE );
                    break;
                default:
            }
        default:
            return( FALSE );
    }
}
```

由于几乎每个对话框都至少有一个 OK 按钮(通常还有一个按钮 CANCEL)。当按下一个下推按钮时,这一消息会出现,按下这些按钮之一时,对话框应从屏幕上消失。

函数 DestroyWindows 使用一个非模态对话框,其主要职责是从屏幕上消除某对话框,它也用于正常的窗口。在一个模态对话框例程中,函数 EndDialog 将被调用,将欲删除的对话框的句柄传送给它的第一个参数。

在其第二个参数中,用户可输入一个值,这个值可作为在对话框例程结束后,调用函数 DialogBox 的返回值。

9.8.5 调用对话框

无论用户是在调用模态对话框,还是非模态对话框,都必须首先使用函数 MakeProcInstance。这是必需的,因为每一对话例程都是一回调函数,函数 MakeProcInstance 保证当启动对话例程时,将当前实例的数据段派上用场。

因此,所有实例可与同一对话例程一起工作。函数 MakeProcInstance 的返回值的数据类型是 FARPROC,并且此返回值作为一个过程实例地址被引用。当不再需要对话框时,用户必须再次释放这个过程实例地址。这种情况类似于使用函数 FreeProcInstance 释放一个设备描述表的句柄。

除了窗口函数以外,所有的回调函数都同时需要调用函数 MakeProcInstance 和 FreeProcInstance。

```
fpProc = makeProcInstance( DialogProc,hInstance );
/* Call dialog box */
```

```
FreeProcInstance( fpProc );
```

使用函数 CreateDialog 可创建一个非模态对话框，它的第二个参数代表对话框的名字；这个名字在创建对话框时给定，第四个参数包含着某一过程实例地址。

因为 Windows 加载了对话框而后返回正常窗口函数，故而这个例程与对话例程可以齐头并进地运行。对话框的消息可以从应用程序的消息队列中取来，这由函数 GetMessage 在消息循环中完成，这就是为什么用户要在消息循环中使用函数 IsDialogMessage 来向窗口提示访问了哪个消息的原因。

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

仅仅当对话框显示在屏幕上时其句柄才不为 NULL，也只有在此刻，才可有发往对话框的消息。

如果取自应用程序消息队列的消息是预期地用于对话框的，那么函数 IsDialogMessage 取值 TRUE，并将这一消息发送给对话框例程。

在这种情况下，用户无需再调用函数 TranslateMessage 和 DispathMessage 了。要激活一个模态对话框，则需要调用函数 DialogBox，这个函数与 CreateDialog 具有相同的参数。

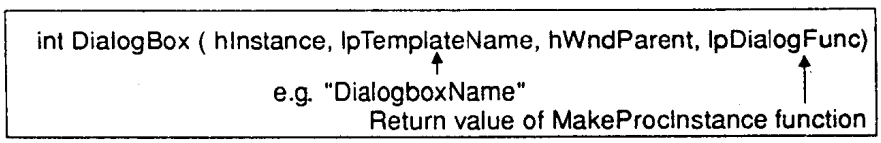


图 9.12

9.8.6 模态对话框示例

在此例子中使用一个对话框，对此对话框及正常的窗口进行一系列变化。

从这三个缺省的图标中，选择其中一个作为窗口的最小化版本，这个新的设置在按下按钮 OK 时生效。使用两个复选框来决定在对话框消失后，在客户区域画一条还是两条对角线。

亦可使用组合框来改变对话框的标题。组合框是由一个编辑控件和一个列表框组成的。通过按下按钮 AcceptTitle，用户可以将编辑控件中的内容转移到标题条上。在此过程中，对话框不应从屏幕上消失。

组合框的列表框中填充着文件名、当前目录的子目录以及在消息 WM_INITDIALOG 作用期间的现存驱动器名。用户还可在组合框的编辑控件中输入一个文本串，这样，当按下按钮 TextSearch 时，这个文本串便作为对列表框中内容进行查找的标准。在一次成功的查找后，第一个找到的串会自动地显示在编辑窗口中。若查找失败，一个消息框会通知用户，没有检索到这个文本串。

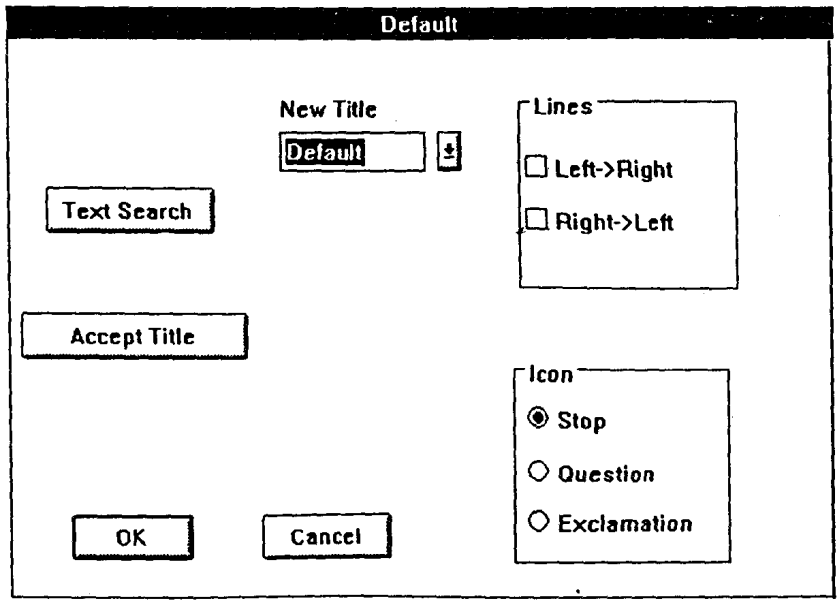


图 9.13

新消息	描述
BM_GETCHECK	判断是否选中了一个按钮
BM_SETCHECK	设置或清除一个复选框或单选按钮的对号标记
CB_DIR	在组合框中插入一文件名列表
CB_SELECTSTRING	选择新串，这个串的一部分在编辑控件中被指定
WM_COMMAND	某个控件向其父窗口发送信息
WM_INITDIALOG	在对话框出现前不久，发送此消息

新函数	描述
DialogBox	创建一个模态对话框
EndDialog	指示某个模态对话框终止
FreProcInstnace	释放程序实例地址
GetDlgItem	从对话框获取一个控件的句柄
GetDlgItemText	向一个缓冲区中拷贝一个控件的文本
GetWindowText	向一个缓冲区中拷贝一个窗口的标题
MakeProcInstance	产生一个程序实例地址
SendDlgItemMessage	在对话框中向一个控件发送消息
SetClassWord	从结构 WNDCLASS 中改变参数
SetFocus	将输入焦点传送给一个窗口
SetWindowText	改变一个窗口标题

源代码:DIALOGBX. C

```
/* * DIALOGBX. C * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays dialog box with options for specifying text and icons * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "dialogbx. h"                                    // Include dialogbx. h header file
#include "windows. h"                                     // Include windows. h header file

#if _MSC_VER >= 700                                       // Running Microsoft C 7. 00 or
#pragma warning (disable:4028)                            // higher? Disable warning 4028
#endif

WORD widIcon = ID_ICON;
BOOL bRILE = FALSE;
BOOL bLERI = FALSE;

BOOL FAR PASCAL DialogProc ( HWND,unsigned,WORD,LONG);
BOOL DialogbxInit ( HANDLE );
long FAR PASCAL DialogbxWndProc( HWND,unsigned,WORD,LONG);


int PASCAL WinMain(hInstance,hPrevInstance,lpszCmdLine,cmdShow)
HANDLE hInstance,hPrevInstance;                           // Current & previous instance
LPSTR lpszCmdLine;                                        // Long ptr to string after
                                                          // program name during execution
int cmdShow;                                              // Specifies the application
                                                          // window's appearance
{
    MSG msg;                                              // Message variable
    HWND hWnd;                                            // Window handle

    if (! hPrevInstance)                                  // Initialize first instance
    {
        if (! DialogbxInit( hInstance ))
            return FALSE;
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Dialogbx",
        "Dialogbx",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,0,
        CW_USEDEFAULT,0,
        NULL,
        NULL,
        hInstance,
        NULL);

    ShowWindow( hWnd,cmdShow );                           // Make window visible
```

```
    UpdateWindow( hWnd );                                    // Update window

    while (GetMessage(&msg,NULL,0,0))                        // Message reading
    {
        TranslateMessage(&msg);                              // Message translation
        DispatchMessage(&msg);                               // Send message to Windows
    }

    return (int)msg. wParam;                                 // Return wParam of last message
}

BOOL DialogbxInit( hInstance )
HANDLE hInstance;

/* * Specify window class * * * * * * * * * * * * * * * * * * * */

{
    WNDCLASS wcDialogbxClass;

    wcDialogbxClass. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcDialogbxClass. hIcon = LoadIcon( NULL,IDI_HAND);
    wcDialogbxClass. lpszMenuName = "Diamenu";
    wcDialogbxClass. lpszClassName = "Dialogbx";
    wcDialogbxClass. hbrBackground = GetStockObject(WHITE_BRUSH );
    wcDialogbxClass. hInstance = hInstance;
    wcDialogbxClass. style = CS_VREDRAW | CS_HREDRAW;
    wcDialogbxClass. lpfnWndProc = DialogbxWndProc;
    wcDialogbxClass. cbClsExtra = 0;
    wcDialogbxClass. cbWndExtra = 0;

    if (! RegisterClass( &wcDialogbxClass ) )
        return FALSE;

    return TRUE;
}

/* * DialogbxWndProc * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL DialogbxWndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    static HANDLE hInst;
    FARPROC lpprocDia;
    HICON hIcon;
    static LPCSTR wStdIcon[3] = {IDI_HAND,IDI_QUESTION,IDI_EXCLAMATION};
    PAINTSTRUCT ps;
```

```
    RECT rect;
    HDC hDC;

    switch (message)
    {
        case WM_CREATE:                                          // Create window
            hInst = GetWindowWord(hWnd,GWW_HINSTANCE);
            break;
        case WM_COMMAND:                                         // Messages from menu bar
            switch(wParam)
            {
                case ID_DIABOX:
                    lpprocDia = MakeProcInstance (DialogProc,hInst);
                    if (DialogBox(hInst,"MyBox",hWnd,lpprocDia))
                    {
                        hIcon = LoadIcon( NULL,
                            wStdIcon[widIcon - ID_ICON]);
                        SetClassWord( hWnd,GCW_HICON,hIcon);
                        InvalidateRect(hWnd,NULL,TRUE);
                    }
                    FreeProcInstance (lpprocDia);
                    break;
                default:
                    break;
            }
            break;
        case WM_PAINT:                                           // Client area redraw
            hDC = BeginPaint(hWnd,&ps);
            GetClientRect(hWnd,&rect);
            if (bLERI)
            {
                MoveTo(hDC,rect. left,rect. top);
                LineTo(hDC,rect. right,rect. bottom);
            }
            if (bRILE)
            {
                MoveTo(hDC,rect. right,rect. top);
                LineTo(hDC,rect. left,rect. bottom);
            }
            EndPaint(hWnd,&ps);
            break;
        case WM_DESTROY:                                         // Destroy window
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}
```

```
BOOL FAR PASCAL DialogProc( hDlg,message,wParam,lParam )
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
    static HWND hRadioB,hComb;                          // Radio button,combo box
    WORD wReturn;
    static char szCombText[MAX_STRING_LENGTH];

    switch (message)
    {
        case WM_INITDIALOG:                             // Dialog box initialization
            hRadioB = GetDlgItem(hDlg,widIcon);
            SendMessage(hRadioB,BM_SETCHECK,TRUE,(LONG)0);
            SendDlgItemMessage(hDlg,ID_RILE,BM_SETCHECK,bRILE,(LONG)0);
            SendDlgItemMessage(hDlg,ID_LERI,BM_SETCHECK,bLERI,(LONG)0);
            GetWindowText(hDlg,szCombText,MAX_STRING_LENGTH);
            hComb = GetDlgItem(hDlg,ID_COMB);
            SetWindowText(hComb,szCombText);
            SetFocus(hComb);
            SendMessage(hComb,CB_DIR,0x10 | 0x4000,(LONG)(LPSTR)"*.*");
            return(FALSE);
            break;
        case WM_COMMAND:                                // Receive messages
            switch (wParam)
            {
                case IDOK:                                          // OK button
                    bRILE = (BOOL)SendDlgItemMessage(hDlg,ID_RILE,BM_GETCHECK,0,0L);
                    bLERI = (BOOL)SendDlgItemMessage(hDlg,ID_LERI,BM_GETCHECK,0,0L);
                    EndDialog (hDlg,1);
                    return(TRUE);
                    break;
                case IDCANCEL:                                      // Cancel button
                    EndDialog (hDlg,0);
                    return(TRUE);
                    break;
                case ID_NEW:                                        // New button
                    GetDlgItemText( hDlg,ID_COMB,szCombText,MAX_STRING_LENGTH);
                    SetWindowText( hDlg,szCombText);
                    return (TRUE);
                    break;
                case ID_SRCH:// Text Search button
                    GetDlgItemText(hDlg,ID_COMB,szCombText,MAX_STRING_LENGTH);
                    wReturn = (WORD)SendMessage(hComb,CB_SELECTSTRING,(WORD)-1,
                        (LONG)(LPSTR)szCombText);
                    if (wReturn == CB_ERR)
                        MessageBox(hDlg,(LPSTR)"No string found",
                            (LPSTR)"Error",MB_OK);
                    return (TRUE);
                    break;
```

```
                case ID_ICON:                                   // Hand/stop icon
                case ID_ICON+1:                                 // Question mark icon
                case ID_ICON+2:                                 // Exclamation point icon
                    widIcon = wParam;
                    return (TRUE);
                    break;
                default:
                    return(FALSE);
            }
        default:
            return(FALSE);
    }
}
```

模块定义文件:DIALOGBX. DEF

```
NAME Dialogbx

DESCRIPTION 'Dialog box demonstration'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS DialogbxWndProc @1
        DialogProc @2
```

资源描述文件:DIALOGBX. RC

```
#include "windows. h"
#include "dialogbx. h"

#include "dialogbx. dlg"

Diamenu MENU
BEGIN
    MENUITEM "Dialog Box", ID_DIABOX
END
```

头文件:DIALOGBX. H

```
#define MAX_STRING_LENGTH 20

#define ID_DIABOX 10

#define ID_SRCH 20
#define ID_ICON 30
```

```
#define ID_NEW 40
#define ID_COMB 50
#define ID_LERI 60
#define ID_RILE 61
```

对话框文件：DIALOGBX. DLG

```
MYBOX DIALOG LOADONCALL MOVEABLE DISCARDABLE 46,19,247,169
CAPTION "Default"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Text Search",ID_SRCH,"button",BS_PUSHBUTTON | WS_GROUP | WS_TABSTOP
        | WS_CHILD,11,45,50,14
    CONTROL "Accept Title",ID_NEW,"button",BS_PUSHBUTTON | WS_TABSTOP
        | WS_CHILD,3,83,68,14
    CONTROL "New Title",-1,"static",SS_LEFT | WS_GROUP
        | WS_CHILD,81,17,54,8
    CONTROL "",ID_COMB,"combobox",CBS_DROPDOWN | WS_VSCROLL
        | WS_CHILD,81,28,55,85
    CONTROL "Lines",-1,"button",BS_GROUPBOX | WS_GROUP
        | WS_CHILD,153,15,65,61
    CONTROL "Left->Right",ID_LERI,"button",BS_AUTOCHECKBOX | WS_TABSTOP
        | WS_CHILD,155,33,58,12
    CONTROL "Right->Left",ID_RILE,"button",BS_AUTOCHECKBOX
        | WS_CHILD,155,49,59,12
    CONTROL "Icon",-1,"button",BS_GROUPBOX | WS_GROUP
        | WS_CHILD,151,96,64,63
    CONTROL "Stop",ID_ICON,"button",BS_AUTORADIOBUTTON | WS_TABSTOP
        | WS_CHILD,155,109,47,12
    CONTROL "Question",ID_ICON+1,"button",BS_AUTORADIOBUTTON
        | WS_CHILD,155,125,47,12
    CONTROL "Exclamation",ID_ICON+2,"button",BS_AUTORADIOBUTTON
        | WS_CHILD,155,140,58,12
    CONTROL "OK",IDOK,"button",BS_DEFPUSHBUTTON | WS_GROUP | WS_TABSTOP
        | WS_CHILD,18,144,36,14
    CONTROL "Cancel",IDCANCEL,"button",BS_PUSHBUTTON | WS_TABSTOP
        | WS_CHILD,75,144,39,14
END
```

MAKE 文件：DIALOGBX. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = dialogbx.dlg
H = DIALOGBX.H

all: DIALOGBX.EXE
```

```
DIALOGBX.RES: DIALOGBX.RC $(RESFILES)

DIALOGBX.OBJ: DIALOGBX.C $(H)

DIALOGBX.EXE: DIALOGBX.OBJ DIALOGBX.RES DIALOGBX.DEF
    link /NOD/A:16 DIALOGBX,DIALOGBX,nul,libw slibcew,DIALOGBX.def
    rc DIALOGBX.res DIALOGBX.exe
```

DIALOGBX.EXE 如何工作

让我们看看这个应用程序做了些什么，然后再研究其根源。

当用户运行应用程序 DIALOGBX 时，将显示一个窗口和单一的菜单标题。在这个菜单标题处按下鼠标按钮，一个对话框将会呈现出来。这个对话框具有一个名字为 Default 的标题条、一系列下推按钮、一个列表框、两个复选框和三个单选按钮。复选框 Left-Right 和 Right-Left 确定了是否此应用程序应在主窗口中画对角线。

列表框 NewTitle 列出了文件名。选中一个文件名并按下按钮 Text Search，如果此文件存在，则不会发生什么事情。如果用户输入一个不存在的文件名，并按下按钮 Text Search，那么将出现一个消息框，它指示一个错误。如果用户选择了一个文件名，之后按下 AcceptTitle，那么对话框的标题就转变为被选中的文件名。

三个单选按钮指定了最小化的图标的外观，STOP、QUESTION 和 EXCLAMATION 是标准的 Windows 图标。按下按钮 OK 将退出，且实现上述那些变化。如果用户最小化了应用程序的窗口，被使用的图标将是用户在对话框中选择的图标。

当菜单 Dialog Box 被选中时，调用程序对话框。这就是在窗口过程中消息 WM_COMMAND 的参数 wParam 指定菜单 ID 值为 ID_DIALOG 的原因。在显示对话框前，用户需要一个指向对话框函数的指针。调用函数 MakeProcInstance 可以获得这个指针。用户选择了 Dialog Box 以后，Windows 会自动启动 DiaglogbxWndProc 窗口过程，仅当对话框由按钮 OK 关闭(如当返回值为 1 时)，用户才应设置一个新的图标，画一行或两行的对角线。

为了为此窗口设置新图标，用户必须首先加载这个图标。函数 LoadIcon 的第二个参数是选中的缺省图标的 ID 值，它可以从一个数组中取得，这个数组包含对话框中可以选择的三个选项。接下来此图标的句柄就传送给了函数 SetClassWord。用户可以使用这个函数，根据注册的类来改变结构 WNDCLASS 中的某些域。由于在这种情况下期望重新设置图标域，故而使用关键字 GCW_HICON 在第二个参数中指定了新图标的句柄。

函数 InvalidRect 将消息 WM_PAINT 插入消息队列中。当处理这个消息时，两个布尔变量的状态均能得以检测。如果它们的状态为 TRUE，那么在客户区中将画一对角线。这条线要么从左上角到右下角，要么从右上角到左下角。函数 GetClientRect 可获取客户区的尺寸。

不管函数 DialogBox 的返回值是什么，指向对话过程的指针都要用函数 FreeProcInstance 释放掉。

对话过程 DialogProc 处理消息 WM_INITDIALOG 和 WM_COMMAND。所有其他消息都根据返回值 FALSE 传送给一个内部缺省过程。

在消息 WM_INITDIALOG 中，单选按钮和复选框被设置成的状态就是用户使用按钮 OK 关闭的最后一个对话框中所设置的状态。参数 wParam 包含值 TRUE 时，消息 BM_SETCHECK 设置其中的一个按钮。函数 SendDlgItemMessage 将这个消息传送给两个复选框。

其他的选项是用于单选按钮的。函数 GetDigItem 获得某个控件的句柄，然后此句柄被传送给函数 SendMessage。

函数 SetFocus 将输入焦点传送给组合框的编辑控件。因此，这个消息必须由其返回值(FALSE)关闭。函数 GetDlgItem 提供此组合框的句柄，对话框标题的内容应立刻出现在编辑控件中。用户可使用函数 GetWindowText 将这个标题写进一个缓冲区，然后将其置于此组合框的编辑控件中。第一个函数需要对话框的句柄，而第二个函数则需要组合框的句柄。

当对话框被显示时，应用程序填充组合框的列表框。向组合框发送消息 CB_DIR 可以填充列表框。CB_DIR 的参数 lParam 包含了 DOS 属性，这个属性可以从函数 DlgDirlist 的描述中获得。任何子目录以及现存的磁盘驱动器应与文件名一道进行显示输出。参数 lParam 是一个字符串指针，它指定了所要查找的文件；在这个例子中，指定通配符如“*.*”等。

在消息 WM_COMMAND 中，引发这个消息的控件的 ID 值传送给参数 lParam。若按下按钮 AcceptTiltle，对话框的标题的内容将发生变化。函数 GetDlgItemText 把组合框的编辑控件中的文本串写进一个缓冲区中。在消息 WM_ INITDIALOG 中所使用的函数 SetWindowText 将此文本串写于对话框的标题条中。

按下按钮 TextSearch，可以在列表框中查寻登录在编辑控件中的文本，并且将找到的字符串显示出来。为此，必须从一个缓冲区读出这个文本，以便于将其用作消息 CB_SELECTSTRING 的参数 lParam。参数 wParam 的值为-1 时表明检索应从头开始。如果搜索过程没有找到与预想的文本相匹配的串，它将返回消息 CB_ERR，同时将出现一个消息框，告知用户检索失败。若检索成功，组合框的窗口过程会自动在其编辑控件中显示第一个找到的串。

三个单选按钮也同样被处理。所按按钮的 ID 值保存在全程变量 WidIcon 中，这样，正常的窗口函数能够设置适当的图标。

两个复选框的当前设置直至用 OK 结束对话框时才确定，然后向两个复选框发送消息 BM_GETCHECK 以提示它们的状态。结果保存在全程变量 bLERI 和 bRILE 中，这两个变量将在消息 WM_PAINT 中加以分析。

对于 IDOK 和 IDCANCEL，都可以调用函数 EndDialog 从屏幕上删除对话框。唯一的差别表现在第二个参数中，这个参数表示调用对话框的返回值。

在 .RC 文件中包含一个由一个菜单项组成的小菜单，以及一个能够使用 #include…语句进行访问的对话框，我们已经讨论过文件 DIALOGBX. DLG 。

由于对话例程是回调函数，故而对话例程必须用 EXPORTS 语句登录在定义文件中。

9.9 消息框

消息框貌似一个简单的对话框，用户无需在 .RC 文件中定义一个消息框，也无需对话过程。相反，通过调用函数 MessageBox 即可创建它。

消息框具有一个标题条和一个静态文本。必要时，Windows 会分多行显示这个文本。若用户不指定标题，那么 Windows 将在标题条中显示缺省标题，即“Error”。

用户可以向一消息框赋予下推按钮，其至可以赋予一个图标。使用 MessageBox 的最后一个参数即可达此目的。例如，用户若使用 MB_YESNOCANCEL，消息框中将包含三个按钮

Yes、No 和 CANCEL。

int MessageBox (hWndParent,lpText,lpCaption,wType)

第一个参数通常为调用这个消息框的窗口的句柄。当消息框从屏幕上消失时，输入焦点会自动传送回这个窗口。

函数 MessageBox 的返回值将提供有关哪个下推按钮被按下的信息，它可接受下列值之一：

IDABORT	按钮 ABORT 被按下
IDCANCEL	按钮 CANCEL 被按下
IDIGNORE	按钮 Ignore 被按下
IDNO	按钮 No 被按下
IDOK	按钮 OK 被按下
IDRETRY	按钮 Retry 被按下
IDYES	按钮 Yes 被按下

第十章　滚动条

数据在窗口的客户区域或其他区域中移动称为滚动。通过滚动，用户可以看到当前不可见的文本和图形。例如，如果文档的边界宽于窗口的显示区域，滚动就使用户能看到整个文档。

滚动条需要滚动窗口的内容。这些滚动条既可以在子窗口中显示为独立的控件，也可在建立窗口时定义为风格参数。如果滚动条定义为风格参数，则可以设置在客户区的右端或底部。这一章我们将讨论滚动条，在第十一章，在介绍了一些控件类型之后，我们还要详细讨论滚动条与其他控件的配合使用。

10.1　带滚动条的设计

滚动条是一种用户友好的工具。用户通过移动姆指图标（即方块按钮，表示文档的位置）或者按滚动箭头来改变客户区中的当前位置。这个动作一旦完成，应用程序必须显示相应的文本或图形。

10.1.1　定义滚动条

CreateWindow 函数中的第三个参数是窗口风格。一个标准窗口通常有多重标志集，例如 WS_ OVERLAPPEDWINDOW 等。参数 WS_ HSCROLL 指定一水平滚动条，参数 WS_ VSCROLL 指定一垂直滚动条。这两个参数由 OR 操作符实现。

```
hWnd = CreateWindow( "Scroll","Scroll Bar Application",
    WS_OVERLAPPEDWIDOW | WSHSCROLL |WS_VSCROLL,WS_USEDEFAULT,
    0,CW_USEDEFAULT,0,NULL,NULL,hInstance,NULL );
```

屏幕驱动程序维护垂直滚动条的宽度和水平滚动条的高度。高度和宽度的值可由参数 SM_CYHSCROLL 和 SM_CXVSCROLL 送给函数 GetSystemMetrics。

10.1.2　滚动范围和滚动条位置

每个滚动条都包含一个滚动范围和一个当前位置。范围由最大和最小值给定。最小值表示的是滚动条的顶边（垂直）或左边（水平），而最大值则指滚动条的底边（垂直）或右边（水平）。

姆指图标（滚动条中的方块）标识着文件的当前位置。程序员还应知道姆指指示的是文档中的正确位置。

姆指可赋与任何能在滚动条中使用的正整数值，该值指出姆指在滚动条中的位置。如

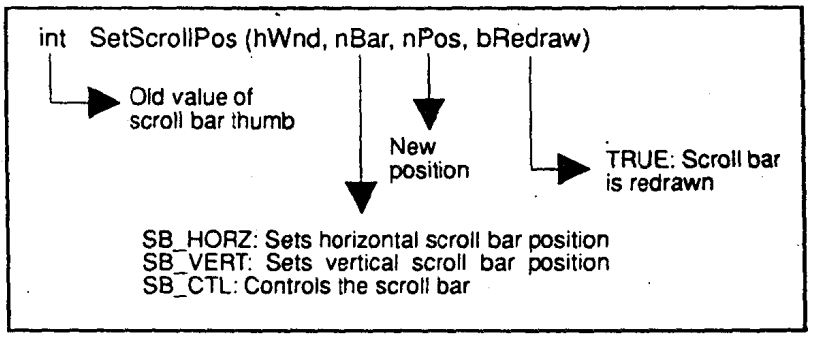


图 10.1 SetScrollPos 函数

果姆指被赋与一个超出范围的值，系统将自动缺省地取最大值或最小值。

函数 SetScrollRange 初始化滚动范围。如果最小值小于最大值，则两边界可取任何正整数。

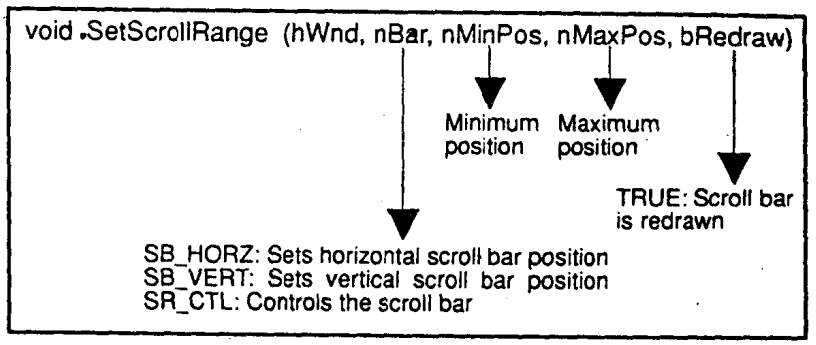


图 10.2 SetScrollRange 函数

滚动范围还可由显示的值确定。例如，如果在 80 行 10 列的客户区中显示一张表，则垂直滚动范围就是 1—80，水平滚动范围是 1—10(或 0—9)。这些值可随时修改。

如果两边界值相同，则滚动条失效(从窗口中消失)。当函数 ShowScrollBar 的第三个参数的值被赋为 FALSE 后可产生上述情况。

该函数还可用于本章一开始提到的滚动条控件设计的第一种方法。

10.1.3 滚动条消息

当用户在滚动条内任意位置按下鼠标按钮或移动姆指时，Windows 都向窗口例程发送 WM_HSCROLL 或 WM_VSCROLL 消息。参数 wParam 指定一个相应的动作。

参数 wParam(除上述六种信息外)还包含三种信息：

SB_BOTTOM	姆指在上(左)边
SB_TOP	姆指在下(右)边
SB_ENDSCROLL	鼠标按钮被释放

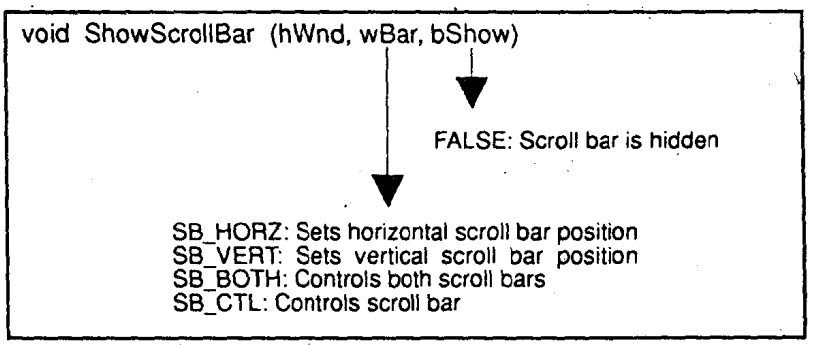


图 10.3 ShowScrollBar 函数

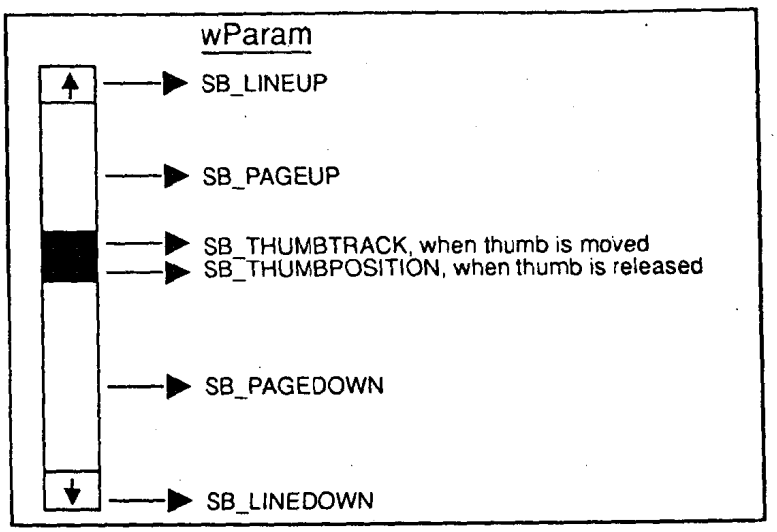


图 10.4 wParam 函数

SB_ENDSCROLL 在用户释放按下的鼠标时被发出。SB_TOP 和 SB_BOTTOM 消息指示姆指位于滚动条的最小值或最大值处(这只用在滚动条控件中,而不是标准滚动条)。

如果还包括键盘对滚动条的支持,则发出 WM_VSCROLL 和 WM_HSCROLL 消息。

SB_THOMBTRACK 和 SB_THUMBPOSITION 消息指示姆指的当前位置,并把这个信息返回到 lParam 的低位字中。这个位置位于滚动条范围定义的两边界之间。

当用户用鼠标移动姆指图标时,wParam 使用 SB_THUMBTRACK 消息;当用户释放鼠标按钮时,wParam 发出 SB_THUMBPOSITION 消息。这对编程可能有些困难。因为应用程序必须比较一定的因子才能选择相应的值,例如:

- 一次显示的数据量
- 数据的复杂性(文本或图形)
- 用户移动姆指的速度

有时,在准备数据显示时,处理 SB_THUMBTRACK 消息会导致问题。然而,姆指图标对文档或图形在窗口中的位置给出了很好的提示。

10.1.4 键盘支持

与滚动条控件不同，标准滚动条提供自动键盘支持。当用户按下特殊的键时，姆指移动。用户必须用 WM_KEYDOWN 消息指定这种移动所需的键。

wParam 参数传送按下的键的虚拟键码。在 WINDOWS.H 中定义的大多数键可用于控制滚动条。

虚拟键码	相应键	移动
VK_LEFT	左箭头键	左移一页或一栏
VK_RIGHT	右箭头键	右移一页或一栏
VK_UP	上箭头键	上移一行
VK_DOWN	下箭头键	下移一行
VK_NEXT	PgDn	上翻一页
VK_PROIR	PgUP	下翻一页
VK_END	END	结尾
VK_HOME	Home	开始位置

用户按下某键时，函数 SendMessage 发出 WM_HSCROLL 消息或 WM_VSCROLL 消息给相应窗口例程的 wParam，这与用鼠标控制滚动条的方法一样。

```
case WM_KEYDOWN:
    switch (wParam)
    {
        case VK_UP:
            SendMessage (hWnd,WM_VSCROLL,SB_LINEUP,OL);
            break;
        case VK_UP:
            SendMessage (hWnd,WM_VSCROLL,SB_LINEDOWN,OL);
            break;
        case VK_UP:
            SendMessage (hWnd,WM_HSCROLL,SB_LINEUP,OL);
            break;
        :
    }
    break;
```

10.1.5 滚动

程序员必须设计姆指的新位置和怎样显示滚动的数据。最简单的方法就是每次发出滚动条消息时都调用函数 InvalidateRect，但这将使整个客户区变成无效。WM_PAINT 消息则试着显示整个文档，Windows 在客户区内只能显示剪裁矩形。这种方法在每次滚动条改变时，都造成客户区的闪烁。这是由于必须删除并重写客户区的内容而造成的。

函数 ScrollWindow 可消除滚动时的闪烁，该函数可以将客户区的某部分移动特定的距离而不删除它，客户区中的其他部分成为无效，因为当时已不需要了。下一个 WM_PAINT 消息可提供范围。ScrollWindow 使用函数 BitBlt 可在客户区域中移动输出的某个预先确定的部分。

```
void ScrollWindow(hWnd,xAmount,yAmount,lpRect,lpClipRect )
```

参数	类型	定义
hWnd	HWND	其客户区滚动的窗口的句柄
xAmount	int	在 x 方向的滚动长度(以驱动器单位)
yAmount	int	在 y 方向的滚动长度(以驱动器单位)
lpRect	LPRECT	指向被滚动的矩形的指针。若为 NULL,整个客户区都滚动
lpClipRect	LPRECT	指向被指派为剪裁矩形的指针。若为 NULL,整个客户区都滚动

当向下(垂直)或向右(水平)滚动时,xAmount 和 yAmount 参数取正值。

既然只是客户区的部分滚动,选择的对象(子窗口)也可改变位置。

ScrollWindow 函数已变成无效,所以,客户区的一部分的坐标可由 PAINTSTRUCT 结构中的 ps 变量送给 WM_PAINT 消息。一旦如此,这个范围即可用于新的输出。

消息 WM_PAINT 可通过 UpdateWindow 函数做到这些,后者由 ScrollWindow 调用。姆指图标的移动和数据滚动大约同时发生,这是由于 UpdateWindow 直接向窗口函数发消息,而不是把消息放在队列中。

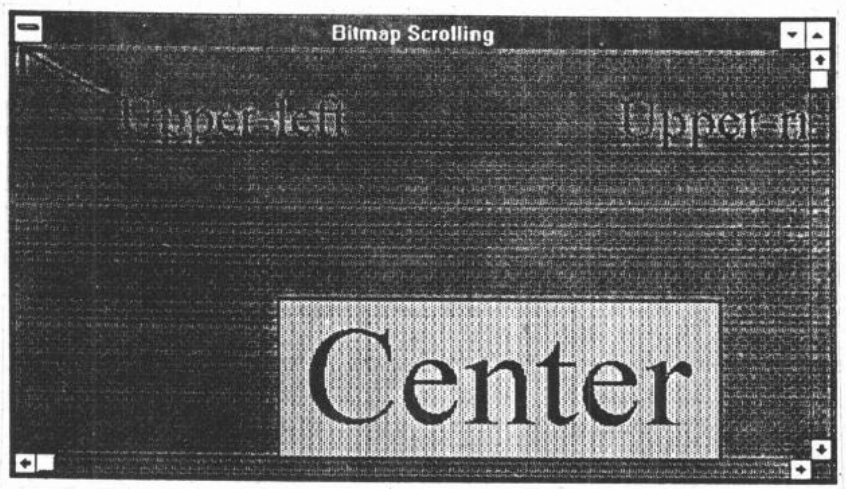


图 10.5

10.2 滚动条示例

这个应用程序使用了垂直和水平滚动条。它显示了本书配套盘中的 ABACUS.BMP 位图,用户可以用滚动条看到整个位图。

新消息	说明
WM_HSCROLL	指示被鼠标按钮按中的水平滚动条
WM_VSCROLL	指示被鼠标按钮按中的垂直滚动条

新函数	说明
GetScrollPos	接收当前姆指位置
SetScrollPos	设置姆指到新位置
SetScrollRange	设置滚动条范围
ScrollWindow	滚动当前客户区的内容

新宏	说明
max	返回两个最大值
min	返回两个最小值

源代码:SCROLL. C

```
/* * SCROLL. C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates a scrolling window using a bitmap * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include "windows. h"                                    // Include windows. h header file

#if _MSC_VER >= 700                                     // Running Microsoft C 7. 00 or
#pragma warning (disable: 4028)                         // higher? Disable warning 4028
#endif

#ifndef max                                                          // Define max
#define max(a,b) ((a) > (b) ? (a) : (b))
#endif                                                               // and

#ifndef min                                                        // min if not yet done
#define min(a,b) ((a) < (b) ? (a) : (b))
#endif

long FAR PASCAL ScrollWndProc (HWND,unsigned,WORD,LONG) ;
/* * WinMain (main window function for all windows applications * * * * * * /
int PASCAL WinMain (hInstance,hPrevInstance,lpszCmdLine,nCmdShow)
    HANDLE hInstance,hPrevInstance ;                    // Current & previous instance
    LPSTR lpszCmdLine ;                                 // Long ptr - string after prg name
    int nCmdShow ;                                      // App. window's appearance
{
HWND hWnd ;                                                          // Window handle
MSG msg ;                                                            // Message handle

/* * Specify window class * * * * * * * * * * * * * * * * * * * * * /
WNDCLASS wndclass ;

    if (! hPrevInstance)
    {
        wndclass. style = CS_HREDRAW | CS_VREDRAW ;
        wndclass. lpfnWndProc = ScrollWndProc;
        wndclass. cbClsExtra = 0 ;
        wndclass. cbWndExtra = 0 ;
```

```
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = "Scroll" ;

        if (! RegisterClass (&wndclass))
            return FALSE ;
    }
/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow ("Scroll",
            "Bitmap Scrolling",
            WS_OVERLAPPEDWINDOW |
            WS_VSCROLL | WS_HSCROLL,
            CW_USEDEFAULT,0,
            CW_USEDEFAULT,0,
            NULL,
            NULL,
            hInstance,
            NULL) ;

    ShowWindow (hWnd,nCmdShow) ;                          // Make window visible
    UpdateWindow (hWnd) ;                                   // Update window

    while (GetMessage (&msg,NULL,0,0))// Message reading
    {
        TranslateMessage (&msg) ;                          // Message translation
        DispatchMessage (&msg) ;                        // Send message to Windows
    }
    return msg.wParam ;
}

/* * ScrollWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL ScrollWndProc (hWnd,message,wParam,lParam)
    HWND hWnd ;
    unsigned message ;
    WORD wParam ;
    LONG lParam ;
{
    HANDLE hInst;
    static short xClient,yClient;
    static short sVertPos,sVertOld,sHorzPos,sHorzOld;
    static BITMAP bm;
    static HBITMAP hBitmap,hBitOld;
    static short sMaxHeight,sMaxWidth;
    HDC hDC,hMemDC;
```

```
PAINTSTRUCT ps;

switch (message)
{
    case WM_CREATE:
        hInst = GetWindowWord(hWnd,GWW_HINSTANCE);
        hBitmap = LoadBitmap(hInst,"Abacus");
        GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bm);
        sMaxHeight = bm.bmHeight ;
        sMaxWidth = bm.bmWidth;

    /* Set scroll bar limits * * * * * * * * * * * * * * * * * * */
        SetScrollRange (hWnd,SB_VERT,0,sMaxHeight,FALSE) ;
        SetScrollPos (hWnd,SB_VERT,sVertPos,TRUE) ;
        SetScrollRange (hWnd,SB_HORZ,0,sMaxWidth,FALSE) ;
        SetScrollPos (hWnd,SB_HORZ,sHorzPos,TRUE) ;
        break ;
    case WM_SIZE:                                          // Client area size
        yClient = HIWORD (lParam) ;
        xClient = LOWORD (lParam) ;
        break ;
    case WM_VSCROLL:                                       // Vertical scroll
        sVertOld = sVertPos;

        switch (wParam)
        {
            case SB_LINEUP:
                sVertPos -= 1 ;
                break ;
            case SB_LINEDOWN:
                sVertPos += 1 ;
                break ;
            case SB_PAGEUP:
                sVertPos -= yClient ;
                break ;
            case SB_PAGEDOWN:
                sVertPos += yClient ;
                break ;
            case SB_TOP:
                sVertPos = 0;
                break;
            case SB_BOTTOM:
                sVertPos = sMaxHeight;
                break;
            case SB_THUMBTRACK:
                sVertPos = LOWORD(lParam);
                break ;
            default:
                break ;
        }
        sVertPos = max (0,min (sVertPos,sMaxHeight)) ;
```

```
    if (sVertPos ! = GetScrollPos (hWnd,SB_VERT))
    {
        ScrollWindow (hWnd,0,sVertOld-sVertPos,NULL,NULL);
        UpdateWindow(hWnd);
        SetScrollPos (hWnd,SB_VERT,sVertPos,TRUE) ;
    }
    break ;
case WM_HSCROLL:                                          // Horizontal scroll
    sHorzOld = sHorzPos;

    switch (wParam)
    {
        case SB_LINEUP:
            sHorzPos -= 1 ;
            break ;
        case SB_LINEDOWN:
            sHorzPos += 1 ;
            break ;
        case SB_PAGEUP:
            sHorzPos -= xClient ;
            break ;
        case SB_PAGEDOWN:
            sHorzPos += xClient ;
            break ;
        case SB_THUMBPOSITION:
            sHorzPos = LOWORD(lParam);
            break ;
        default:
            break;
    }
    sHorzPos = max (0,min (sHorzPos,sMaxWidth)) ;
    if (sHorzPos ! = GetScrollPos (hWnd,SB_HORZ))
    {
        ScrollWindow (hWnd,sHorzOld-sHorzPos,0,NULL,NULL);
        UpdateWindow(hWnd);
        SetScrollPos (hWnd,SB_HORZ,sHorzPos,TRUE) ;
    }
    break ;
case WM_KEYDOWN:                                          // Check mouse
    switch (wParam)
    {
        case VK_UP:
            SendMessage (hWnd,WM_VSCROLL,SB_LINEUP,0L);
            break;
        case VK_DOWN:
            SendMessage (hWnd,WM_VSCROLL,SB_LINEDOWN,0L);
            break;
        case VK_PRIOR:
            SendMessage (hWnd,WM_VSCROLL,SB_PAGEUP,0L);
            break;
        case VK_NEXT:
```

```
                    SendMessage (hWnd,WM_VSCROLL,SB_PAGEDOWN,0L);
                    break;
                case VK_LEFT:
                    SendMessage (hWnd,WM_HSCROLL,SB_LINEUP,0L);
                    break;
                case VK_RIGHT:
                    SendMessage (hWnd,WM_HSCROLL,SB_LINEDOWN,0L);
                    break;
                case VK_HOME:
                    SendMessage (hWnd,WM_VSCROLL,SB_TOP,0L);
                    break;
                case VK_END:
                    SendMessage (hWnd,WM_VSCROLL,SB_BOTTOM,0L);
                    break;
                default:
                    break;
            }
            break;
        case WM_PAINT:                                          // Client area redraw
            hDC = BeginPaint (hWnd,&ps) ;
            hMemDC = CreateCompatibleDC( hDC );
            hBitOld = SelectObject(hMemDC,hBitmap);
            BitBlt(hDC,ps. rcPaint. left,ps. rcPaint. top,
                ps. rcPaint. right,ps. rcPaint. bottom,hMemDC,
                sHorzPos+ps. rcPaint. left,sVertPos+ps. rcPaint. top,SRCCOPY);
            SelectObject(hMemDC,hBitOld);
            DeleteDC(hMemDC);
            EndPaint (hWnd,&ps) ;
            break ;
        case WM_DESTROY:                                        // Destroy window
            PostQuitMessage (0) ;
            break ;
        default:
            return DefWindowProc (hWnd,message,wParam,lParam) ;
    }
    return 0L ;
}
```

模块定义文件:SCROLL. DEF

```
NAME Scroll
DESCRIPTION 'Scroll demonstration with bitmap'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 4096
EXPORTS ScrollWndProc
```

资源描述文件:SCROLL. RC

```
Abacus BITMAP abacus. bmp
```

MAKE 文件:SCROLL. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = abacus.bmp

all: SCROLL.EXE

SCROLL.RES: SCROLL.RC $(RESFILES)

SCROLL.OBJ: SCROLL.C

SCROLL.EXE: SCROLL.OBJ SCROLL.RES SCROLL.DEF
    link /NOD/A:16 SCROLL,SCROLL,nul,libw slibcew,SCROLL.def
    rc SCROLL.res SCROLL.exe
```

运行 NMAKE 得到 SCROLL 应用程序:

```
namke scroll.mak
```

SCROLL.EXE 如何运行

.RC 文件指出位图名称,用户可以在配套盘中找到位图,或用其他位图替换 ABACUS 位图。WM_CREATE 消息把该文件加载到窗口中,函数 GetWindowWord 提供相应的实例句柄。

函数 GetObject 从应用程序所设置的滚动条的最大值中定出位图的高度和宽度。每个滚动条的最小值是 0。姆指出现在滚动条的最左边(水平)和最上边(垂直)。

对于按页滚动,客户区域的大小被传给 lParam 参数的 WM_SIZE 消息。

WM_VSCROLL 消息存放旧姆指位置,因为函数 ScrollWindow 以后还要用到这个信息。如果用户向上滚动客户区内容,姆指的位置值应减少。

因此,当 wParam 参数为 SB_LINEUP 或 SB_PAGEUP 之一时,变量 sVertPos 的内容或者是 1,或者是要移动的客户区的大小。

如果用户向下滚动客户区的内容,姆指的位置应增加,这是对 SB_LINEDOWN 和 SB_DAGEDOWN 消息及 sVertPos 变量做出的反应。

垂直和水平滚动条移动姆指的方式是不同的,而且,开始时,垂直滚动条对释放鼠标按钮不作响应。然而,这里我们不演示这一点。lParam 参数的低位字指出当前姆指的位置,SB_TOP 和 SB_BOTTOM 值加上了键盘支持。

不考虑 wParam 中的信息,变量 sVertPos 的值必须被检验,以确定它是否在滚动区域中。如果值太大或太小,min 和 max 宏将设定最小值和最大值。max 宏返回最大参数,min 宏返回最小参数。

当姆指从最新位置移动时,函数 ScrollWindow 移动客户区。为了确定移动量,要找出新

旧位置的不同，此时应调用 UpdateWindow 函数，使移动和重写几乎同时发生。

用户移动姆指图标后，姆指的新位置必须由应用程序设定。在我们的例子中，用 GetScrollPos 和 sVertOld 比较姆指的新旧位置。

如果用户在水平滚动条上按下鼠标按钮或移动姆指，就产生 WM_HSCROLL 消息。这里移动的规律与上述 WM_VSCROLL 消息中的基本一致，除了水平移动是基于 x 值而不是 y 值之外。WM_HSCROLL 消息首先判断鼠标按钮是否被释放，而不是姆指本身。新位置可在 lParam 中找到。

位图显示的新位置可由新姆指位置和客户区内无效区的左上角计算出来。随后的示例演示了这一点——假设位图向上滚动(即姆指现在在滚动条的下部)。

这个应用程序有多种改进之处。例如，当用户使窗口与位图一样大(或小于位图)时，滚动条应消失。还有，每个姆指可设置不同位置(例如，滚动条定位在下边和右边而不是上边和左边)。

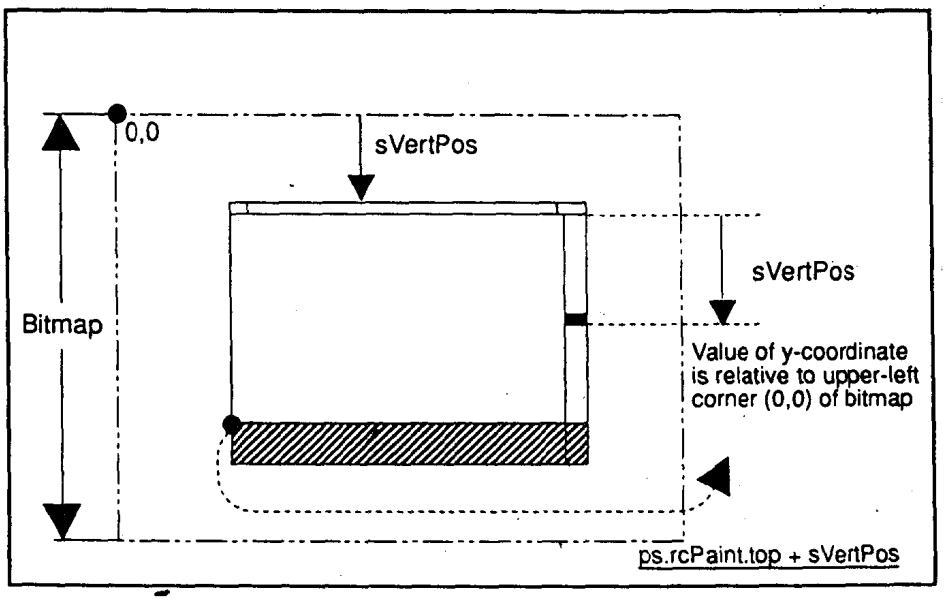


图 10.6 BitBlt 函数

第十一章　Windows 中的控件

用户通过单选按钮、复选框、列表框和组合框等控件与 Windows 通信，这些控件为用户提供了一个友好的界面。用户只需按一下鼠标按钮，就能得到一个值。在面向文本的界面里，所有的值都是通过键盘得到的。这比 GUI 通信方式所费的时间长，而且键盘的出错率较高。

虽然用户可以建立自己的子窗口控件，但也可以利用一些预先定义的窗口类(和窗口过程)来建立子窗口控件。这些控件采用的形式有：按钮、复选框、编辑框、列表框、组合框、文本串和滚动条。例如，假定想在电子表格程序的某角上安装一个标有“Recalculate”的按钮，调用单个 CreateWindow 就能建立这个按钮。用户不必担心鼠标器操作、按钮显示操作或当按下该按钮时的自动闪动操作。这些是由 Windows 内部完成的。用户所要做的只是捕获 WM_COMMAND 消息——这就是当按钮被按下时是怎样通知用户窗口过程的。

在对话框中，子窗口控件最为常用。子窗口控件的位置和尺寸是在示例程序的资源描述文本的对话框模板中定义的。但是，用户也可以使用预先定义的、位于普通窗口客户区表面的子窗口控件。调用一次 CreatWindow 将建立一个子窗口，调用 MoveWindow 可以调整子窗口的位置和尺寸。父窗口过程向子窗口控件发送消息，子窗口控件向父窗口过程返回消息。

在建立普通窗口时，首先定义一个窗口类，并用 RegisterClass 将其注册到 Windows 中，然后用 CreateWindow 命令在该窗口类的基础上建立这个普通窗口。但是，当用预先定义的控件中的某一个时，用户不要为子窗口注册窗口类，窗口类已经存在于 Windows 之内，并且有一个如下的名字：“Button”，“Static”，“Scrollbar”，“edit”，“listbox”或者“combobox”。该名字在 CreateWindow 中简单地用作窗口类参数。CreateWindow 中的窗口风格参数准确地定义了子窗口控件的外形和功能。Windows 包含了处理如下消息的窗口过程，这就是发向基于这些窗口类的子窗口的消息。

直接在用户窗口表面使用子窗口控件要完成某些任务，这些任务的级别低于在对话框中使用子窗口控件所要求的级别。这里，对话框管理器在用户程序和控件自身之间增加一个隔离层。值得指出的是，用户会发现在用户窗口表面建立的子窗口控件没有利用 Tab 键或光标移动键将输入焦点从一个控件移到另一个控件的内部功能。一个子窗口控件能够获得输入焦点，但是一旦获得后，它将不能把输入焦点撤回到父窗口。

11.1　控件类型概述

在这一章中，我们将介绍使用 CreateWindow 函数创建的作为独立子窗口的控件。

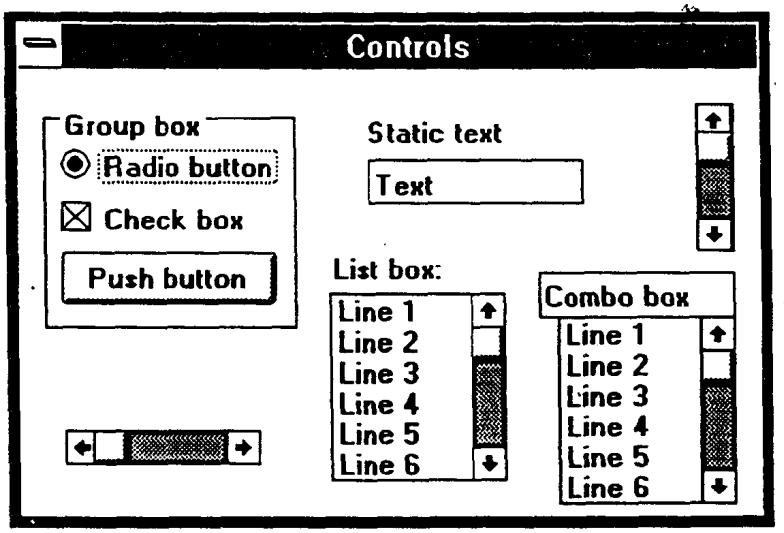


图 11.1 控件类型

用户也许对大多数控件比较熟悉了。在下面的讲述中，对各种类型只做一般的描述，当然要包括各自的独特之处。

· 下推按钮

下推按钮也叫命令按钮，这种类型的控件一般含有文本“OK”或“Cancel”。用户选择这个按钮来初始化某个动作。

· 复选框

对一个复选框按一下鼠标按钮，就选择了相应的选项。还可以同时选择一组中多个选项。

· 单选按钮

也叫选项按钮，这种控件用于各选项都互斥的组中。在任何时候都不可能选择多于一个的选项。

· 列表框

该控件显示一个表，表中的元素都可由用户选择；可进行单项或多重选择。如果该框不能一下将所有内容都显示出来，它就会自动安排一个滚动条。

· 分组框

可以用边界将控件分组。可以进行功能性的分组，也可仅仅为了清晰而分组。当一个分组框放置在一个对话框内时，首先要将边界显示出来，然后是边界中的控件，否则边界会覆盖控件。

• 静态文本

这种文本不能被用户修改，它通常用于解释性的文本或标题。

• 编辑控件

这种控件也叫文本框或编辑框，用户可以进入或操作文本来得到一个选择。

• 组合框

这种控件是编辑控件和列表框的组合，它使用户可以在一个列表中选择一项，或打入所需的文本串。

• 滚动条

当文本量很大时，滚动条变得很有用，它使用户能很快地到达任何期望的位置。水平滚动条向左或向右滚动文本，垂直滚动条则向上或向下滚动文本。

以上所有控件都属于已登录好的 Windows 类，所以，现存的窗口例程决定了它们的表现形式和功能。如果想修改特定控件的某个预定义的功能，例如限制文本项为字母而不是字母数字，则必须使用子类方式。这个功能将在以后详细介绍。除了预定义的控件类之外，还可以创建附加类。这类控件叫作定制控件。

11.2 作为独立窗口的控件

使用控件包含对控件消息的处理。一个控件可看成是一个被消息激活的对象。例如，CB_ADDSTRING 产生一些动作，但当被用户操纵时，还产生自身的消息。该消息给出有关用户活动的信息，然后发送到父窗口。

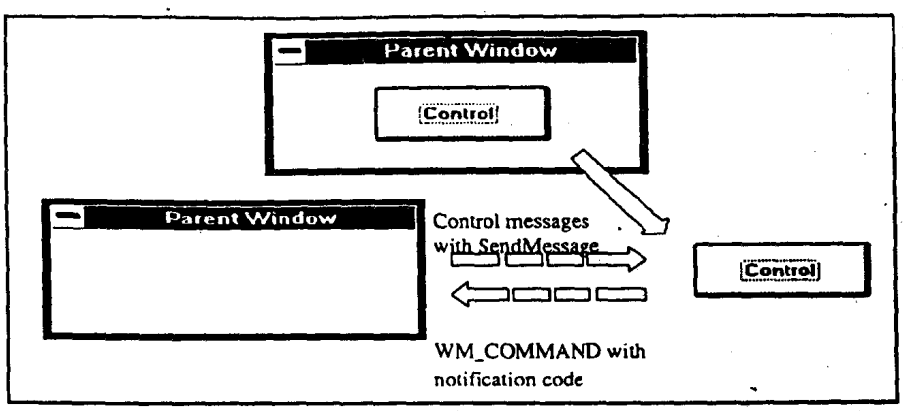


图 11.2 送向和发自控件的消息

11.2.1 发向控件的消息

当控件执行某一特定任务时，父窗口用 SendMessage 或 PostMessage 函数向子窗口发送

一条控件消息。如果控件句柄未知，则可使用 ID 值，即 CreateWindow 函数中倒数第三个参数。这样，最适合的函数是 SendDlgItemMessage，尽管父窗口是主窗口而非对话框。

```
hWndButton = CreateWindow(
    "BUTTON",                                          // Class name
    "20-30 Yrs",                                       // Title
    WS_CHILD | WS_VISIBLE |
    BS_RADIOBUTTON,
    10,10,
    50,60,
    hWndMain,                                          // Handle of parent widow
    ID_OLDER1,
    hInst,
    NULL);
SendMessage (hWndButton,BM_SETSTYLE,
    BS_PUSHBUTTON,TRUE);
```

或

```
SendDlgItemMessage(hWndMain,ID_OLDER,
    BM_SETSTYLE,BS_PUSHBUTTON,TRUE);
```

或

```
hWndPush = GetDlgItem(hDlg,ID_OLDER1);
SendMessage(hWndButton,BM_SETSTYLE,
    BS_PUSHBUTTON,TRUE);
```

以上结构产生一个单选按钮，然后通过传递一个带有新风格参数 BS_PUSHBUTTON 的 BM_SETSTYLE 消息给控件，从而把它变成一个下推按钮。当参数 lParam 带有值 TRUE 时，控件就被重新绘制。

一个控件可接收多种消息。有些是特别针对控件类的，这由消息前缀指定。

控件	前缀	示例
按钮	BM	BM_GETSTATE
省缺下退按钮	DM	DM_GETDEFID
组合框	CB	CB_FINDSTRING
编辑控件	EM	EM_GETSEL
列表框	LB	LB_GETTEXT

11.2.2 来自控件的消息

当用户按下一个鼠标按钮或从列表框中选择一项时，控件对输入产生响应，向父窗口发送 WM_COMMAND 消息。wParam 参数中包含控件 ID，在 lParam 参数的低位字中包含控件句柄，而在高位字中包含通知码。

通知码传达了触发事件的特征。例如，LBN_SELCHANGE 表明了在列表框中选择的变化。类似控件消息，通知码也可按控件类型分组，在前缀后附加字母“N”以替换字母“M”。

11.2.3 通知码

控件	前缀	示例
按钮	BN	BN_DOUBLECLICKED

组合框	CBN	CBN_SELCHANGE
编辑控件	EN	EN_MAXTEXT
列表框	LBN	LBN_EERSPACE

父窗口的窗口函数询问 WM_COMMAND 消息的方式与处理菜单项一样，只不过现在参数 wParam 中包含控件 ID 而不是菜单 ID。

11.2.4 子分类

分配在各个类中的预定义窗口例程是 Windows 的一个方便特性，它可省掉不少编程麻烦。但有时用户也希望能修改这些例程的标准功能。

例如，在形成掩码时，常常必须立即在项上检验输入。EDIT 类在每个字符被键入后就要接收并显示出来。如果用户想禁止某些特殊字符(如数字)的进入，就必须扩展这个例程。

既然标准例程不能直接访问，这时就需编写用户例程，且传给 Windows；在应用程序处理编辑控件时将首先调用它。

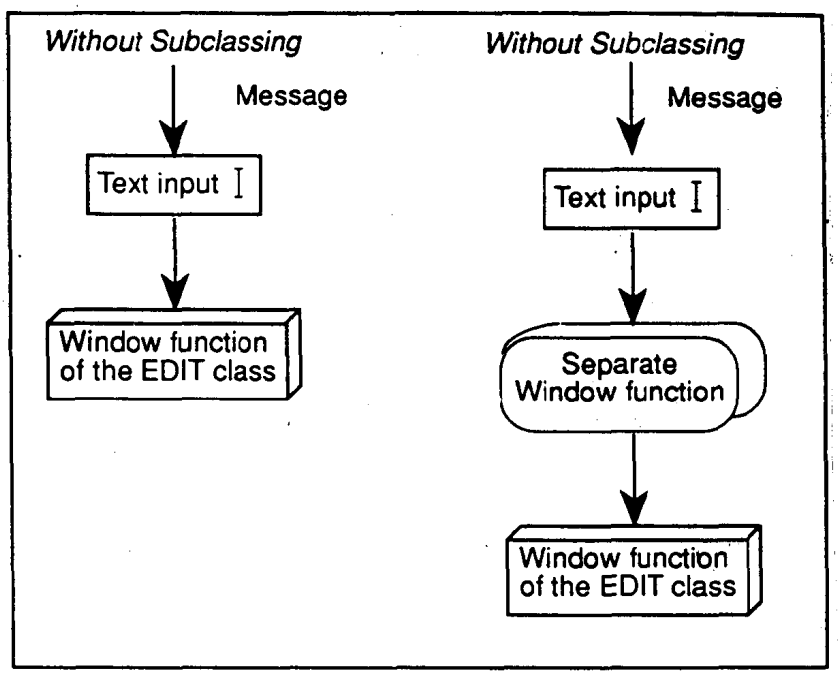


图 11.3

子分类一般只用于某一类中的个别窗口，而非整个类本身。例如，用户可设置 5 个滚动条，而只对其中一个滚动条添加附加特性。这在 Windows 中是可能的，因为每当窗口建立时，都有一个内存块存放该窗口的专用信息。

例如，来自 WNDCLASS 内存块的 IpfnWndProc 指针被复制到这个新区域，且每当有消息必须被传给窗口例程时就引用它。如果一个窗口中应用了子分类，则标准例程的过程实例地址将被用户例程的地址所覆写。

因为这种变化只发生在所使用的窗口的内存块中，因此同类中的其他窗口不受子分类

的影响。

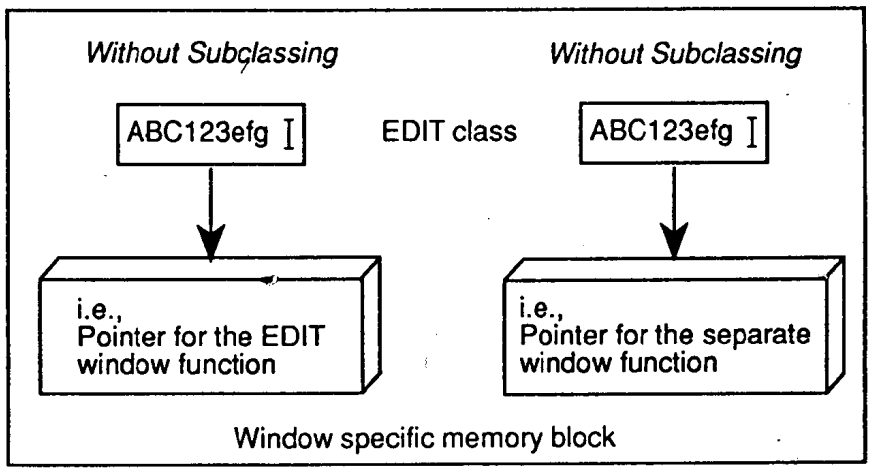


图 11.4

用户例程只处理与用户问题有关的消息，其他的则仍送给标准 EDIT 类窗口函数。这与缺省窗口例程的规则很类似，它被 Windows 程序的缺省分支所调用。

下面是子分类的过程：

(1)建立用户例程的原型。

```
LONG FAR PASCAL EditProc(HWND,WORD,WORD,LONG)
```

(2)用 MakeProcInstance 函数产生用户窗口例程(EditProc)的过程实例地址(lpEditProc)，这是每个回调函数都需要的。

```
FARPROC lpEditProc;
lpEditProc = MakeProcInstance( (FARPROC) EditProc,hInst);
```

(3)为预定义类(如 EDIT)的标准窗口函数提供一个指针(lpEditStandard)。控件所需的用户例程的句柄可用于此。指针应定义为全局的，因为它在主窗口的窗口函数中使用，还在以后的用户例程中使用。

```
FARPROC lpEditStandard;
lpEditStandard=(FARPROC)SetWindowLong(hWndEdit,CWL_WNDPROC,
        (LONG) lpEditProc;
```

(4)现在可以写用户例程本身，所有不在此处处理的消息由 CallWindowProc 函数送到标准窗口例程。除了被用户例程接收的四个参数外，CallWindowProc 的第一个参数是指向缺省窗口函数的指针。

```
LONG FAR RASCAL EditProc(HWND hWnd,WORD Message,WORD wParam,LONG lParam)
{
    switch (Message)
    {
        case WM_CHAR:
            if (('0' <= wParam) & (wParam <= '9')
            {
                MessageBeep(0);
```

```
                break;
            }
            else
                CallWindowProc(lpEditStandard,hWnd,Message,wParam,lParam);
                break;
        default:
            return(CallWindowProc(lpEditStandart,hWnd,Message,wParam,lParam));
    }
    return OL;
}
```

在我们的例子中只能处理字母。当用户输入数字时，就发出蜂鸣声。然而，EDIT 类的所有特性该控件都具有。

(5)最后，在程序结束前，用户例程必须被挪走，标准窗口函数返回给控件，过程实例地址(lpEditProc)也应被释放。

```
SetWindowLong (hWndEdit,GWL_WNDPROC,lpEditStandard);
FreeProcInstance (lpEditProc);
```

(6)既然用户窗口例程是回调函数，而且可被 Windows 激活，所以在定义文件中它必须被输出。

```
EXPORT EditProc @2
```

不论是对列表框、编辑控件还是按钮进行子分类，上述过程都是基本一致的。

11.3 按 钮

与按钮有关的消息如下：

按钮	含义
BS_AUTOCHECKBOX	选中该按钮时自动改变状态
BS_AUTORADIOBUTTON	选中时使组中其他单选按钮失效
BS_AUTO3STATE	选中时自动改变状态的三态复选框
BS_CHECKBOX	标准复选框
BS_DEFPUSHBUTTON	缺省下推按钮，可被 Enter 键激活
BS_GROUPBOX	包围数个单选按钮的边界
BS_LEFTTEXT	单选按钮或复选框中左对齐的文本
BS_OWNERDRAW	应用程序产生的按钮
BS_PUSHBUTTON	下推按钮
BS_RADIOBUTTON	单选按钮
BS_3STATE	三态复选框

三种最常用的按钮(单选按钮、下推按钮和复选框)有许多相似之处。例如，三种类型均可以通过将光标移到控件处、再按一下鼠标按钮激活；这在下推按钮中产生一个动作，在单选按钮或复选框中将一个选项打开或关闭。多数前缀为“BM”的消息可以被送给多种类型的按钮，例如，消息 BM_SETSTYLE 可用于改变一个按钮的风格参数。

下面通过 BTNLOOK(“button look”)程序来介绍按钮窗口类，BTNLOOK 如下所示。它能

建立 10 个子窗口按钮控件，每个控件对应一个按钮风格，共 10 个按钮风格。

```
# BTNLOOK.MAK make file

btnlook.exe : btnlook.obj btnlook.def
     $(WINLINK) btnlook,btnlook,NUL, $(WINLIB),btnlook
     rc -t btnlook.exe

btnlook.obj : btnlook.c
     $(WINCC) btnlook.c

/*---------------------------------------
BTNLOOK.C -- Button Look Program
---------------------------------------*/

#include <windows.h>

struct
{
     long style ;
     char *text ;
}
button[] =
{
     BS_PUSHBUTTON, "PUSHBUTTON",
     BS_DEFPUSHBUTTON, "DEFPUSHBUTTON",
     BS_CHECKBOX, "CHECKBOX",
     BS_AUTOCHECKBOX, "AUTOCHECKBOX",
     BS_RADIOBUTTON, "RADIOBUTTON",
     BS_3STATE, "3STATE",
     BS_AUTO3STATE, "AUTO3STATE",
     BS_GROUPBOX, "GROUPBOX",
     BS_AUTORADIOBUTTON,"AUTORADIO",
     BS_OWNERDRAW, "OWNERDRAW"
} ;

#define NUM (sizeof button / sizeof button[0])

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "BtnLook" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
```

```
        wndclass.cbClsExtra = 0 ;
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName,"Button Look",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
        {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
        }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static char szTop [] = "message wParam lParam",
        szUnd [] = "_____",
        szFormat [] = "%-16s%6X%8X-%04X",
        szBuffer [50] ;
    static HWND hwndButton [NUM] ;
    static RECT rect ;
    static int cxChar,cyChar ;
    HDC hdc ;
    PAINTSTRUCT ps ;
    int i ;
    TEXTMETRIC tm ;

    switch (message)
    {
        case WM_CREATE:
            hdc = GetDC (hwnd) ;
            SelectObject (hdc,GetStockObject (SYSTEM_FIXED_FONT)) ;
            GetTextMetrics (hdc,&tm) ;
            cxChar = tm.tmAveCharWidth ;
```

```
            cyChar = tm.tmHeight + tm.tmExternalLeading ;
            ReleaseDC (hwnd,hdc) ;

            for (i = 0 ; i < NUM ; i++)
                hwndButton [i] = CreateWindow ("button",button[i].text,
                    WS_CHILD | WS_VISIBLE | button[i].style,
                    cxChar,cyChar * (1 + 2 * i),
                    20 * cxChar,7 * cyChar / 4,
                    hwnd,i,
                    ((LPCREATESTRUCT) lParam) -> hInstance,NULL) ;
            return 0 ;
        case WM_SIZE:
            rect.left = 24 * cxChar ;
            rect.top = 2 * cyChar ;
            rect.right = LOWORD (lParam) ;
            rect.bottom = HIWORD (lParam) ;
            return 0 ;
        case WM_PAINT:
            InvalidateRect (hwnd,&rect,TRUE) ;
            hdc = BeginPaint (hwnd,&ps) ;
            SelectObject (hdc,GetStockObject (SYSTEM_FIXED_FONT)) ;
            SetBkMode (hdc,TRANSPARENT) ;
            TextOut (hdc,24 * cxChar,cyChar,szTop,sizeof szTop - 1);
            TextOut (hdc,24 * cxChar,cyChar,szUnd,sizeof szUnd - 1);
            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_COMMAND:
        case WM_DRAWITEM:
            ScrollWindow (hwnd,0,-cyChar,&rect,&rect) ;
            hdc = GetDC (hwnd) ;
            SelectObject (hdc,GetStockObject (SYSTEM_FIXED_FONT)) ;
            TextOut (hdc,24 * cxChar,cyChar * (rect.bottom / cyChar - 1),
                szBuffer,
            wsprintf (szBuffer,szFormat,(LPSTR)
                (message == WM_COMMAND ? "WM_COMMAND" : "WM_DRAWITEM"),
                wParam,HIWORD (lParam),LOWORD (lParam))) ;
            ReleaseDC (hwnd,hdc) ;
            ValidateRect (hwnd,NULL) ;
            return 0 ;
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; BTNLOOK.DEF module definition file

NAME BTNLOOK

DESCRIPTION 'Button Look Program (c) Charles Petzold,1992'
```

```
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

当按动每个按钮时，它就发送 WM_COMMAND 消息给父窗口过程，亦即我们所熟悉的 WndProc。

BTNLOOK 的 WndProc 将该消息的 wParam 参数和 lParam 参数显示在客户区域的右半边，如图 11.5 所示。

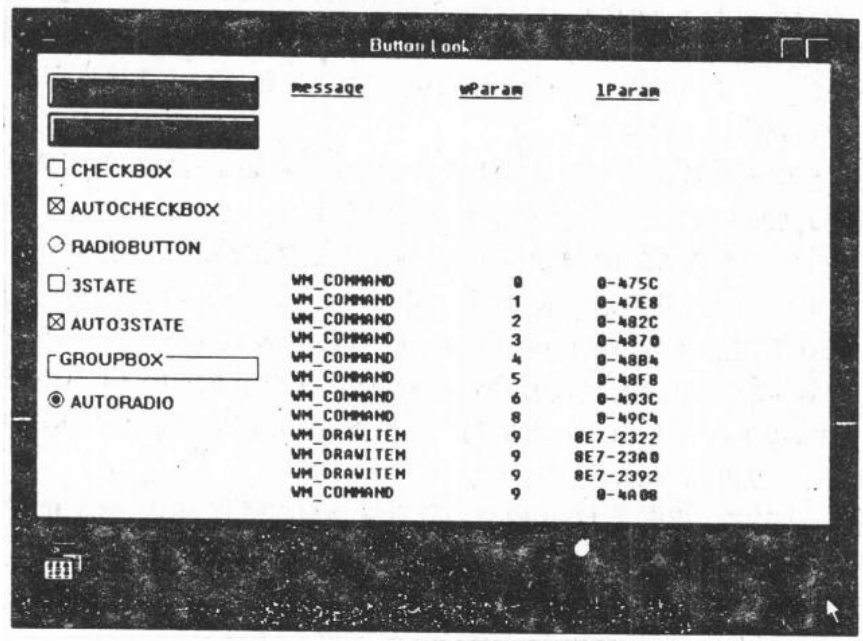


图 11.5　BTNLOOK 的显示

用户在该窗口上看不到 BS_OWNERDRAW 风格的按钮，因为这是一种由程序负责绘制按钮的按钮风格。该按钮表示它需要由包含 lParam 消息参数的 WM_DRAWITEM 消息来绘制，而 lParam 消息参数是一个指向 DRAWITEMSTRUCT 类型结构的指针。

11.3.1　建立子窗口

BTNLOOK 定义了一个叫作按钮(button)的结构，它包含了按钮窗口风格和描述性文本串，它们对应 10 个按钮类型。所有按钮窗口风格的前两个字母都是 BS，它表示“button style”。

10 个按钮子窗口是在 WndProc 中处理 WM_CREATE 消息的过程中用一个 for 循环建立的。CreateWindow 调用使用下面这些参数：

Class name(组名)	"button"
Window text(窗口文本)	button[i]. text

Window style(窗口风格)	WS_CHILD|WS_VISIBLE|button[i].style
x position(x 位置)	cxChar
y position (y 位置)	cyChar * (1 + 2 * i)
Width(宽度)	20 * xChar
Height(高度)	7 * yChar / 4
Parent window(父窗口)	hwnd
Child window ID (子窗口 ID)	i
Instance handle(实例句柄)	((LPCREATESTRUCT)lParam→hInstance
Extra parameters(附加参数)	NULL

类名参数是预定义的名字;窗口风格是 WS_CHILD、WS_VISIBLE 和"按钮(button)"结构中的 10 个按钮风格(BS_PUSHBUTTON、BS_DEFPUSHBUTTON 等)中的一个;窗口文本参数(对普通窗口来说,它是显示在标题行中的文本)与每个按钮相对应地被显示出来。前面曾经简单地使用过识别按钮风格的文本。

x 和 y 这两个位置参数说明子窗口左上角的位置;该子窗口是相对于父窗口客户区的左上角的;宽度和高度参数规定每个子窗口的宽度和高度。

对每个子窗口,它的子窗口 ID 参数应各不相同。在处理来自子窗口的 WM_COMMAND 消息时,ID 帮助用户窗口过程识别子窗口。

CreateWindow 调用的实例句柄(instance handle)参数看起来有点奇怪,但它利用了如下事实:在处理 WM_CREATE 消息的过程中,lParam 实际上是指向 CREATESTRUCT 结构的指针,CREATESTRUCT 有一个 hInstance 成员。所以将 lParam 转换成指向 CREATESTRUCT 结构的远程指针,并取出 hInstance。

有些 Windows 程序使用名为 hInst 的全局变量来让窗口过程访问 WinMain 中能够得到的实例句柄。在 WinMain 中,只需在建立主窗口之前设置:

```
hInst = hInstance;
```

我们曾用 GetWindowWord 来得到实例句柄:

```
GetWindowWord(hwnd,GWW_HINSTANCE);
```

这几种方法都是正确的。

在调用 GreatcWindow 之后,用户不必再为这些子窗口做任何事情,由 Windows 中的按钮窗口过程维护它们并处理所有重画工作(BS_OWNERDRAW 风格的按钮例外,它要求程序绘制它,这些将在后面要讨论)。在程序终止时,如果父窗口已被消除,Windows 就消除这些子窗口。

11.3.2 子窗口向父窗口发信号

当运行 BTNLOOK 时,在客户区域的左边会显示出不同的按钮类型(BS_OWNERDRAW 按钮是不可见的)。前面提到过,当用鼠标器按动一个按钮时,子窗口控件就向其父窗口发送一个 WM_COMMAND 消息。BTNLOOK 捕获 WM_COMMAND 消息并显示 wParam 和 lParam 的值。下面是它们的含义:

wParam	Child window ID (子窗口 ID)
LOWORD(lParam)	Child window handle(子窗口句柄)

HIWORD(lParam)　　Notification code(通知码)

子窗口 ID 是在创建子窗口时传给 CreateWindow 的值。在 BTNLOOK 中,这些 ID 被显示在客户区域中,并用 0 到 9 分别表示 10 个按钮。子窗口句柄是 Windows 从 CreateWindow 返回的值。

通知码是子窗口用来更详细地告诉其父窗口该消息含义的子消息码。按钮通知码的可能值在 WINDOWS.H 中定义如下:

按钮通知码标识符	值
BN_CLICKED	0
BN_PAINT	1
BN_HILITE	2
BN_UNHILITE	3
BN_DISABLE	4
BN_DOUBLECLICKED	5

对废弃的按钮风格来说,从 0 到 5 的通知码被叫做 BS_USERBUTTON。

当用鼠标器按一个按钮时,会看到在该按钮文本的周围有短划线,这说明该按钮有了输入焦点,所有键盘输入都输向子窗口按钮控件,而不是输向主窗口。但是,当该按钮控件有输入焦点时,它忽略所有的键盘输入,但空格键例外,此时空格键与鼠标器有相同的效果。

11.3.3 父窗口向子窗口发信号

虽然 BTNLOOK 没有说明,但是一个父窗口过程也能向子窗口控件发送消息。在 WINDOWS.H 中定义了按钮特有的五条消息。每个消息的开头字母都为"BM",它代表"button message"。在 WINDOWS.H 中,按照 WM_USER 标识符定义了这些消息:

```
#define BM—GETCHECK (WM—USER+0)
#define BM—SETCHECK (WM—USER+1)
#define BM—GETSTATE (WM—USER+2)
#define BM—SETSTATE (WM—USER+3)
#define BM—SETSTYLE (WM—USER+4)
```

对定义它们自己的消息——但预定义过的消息除外——的程序,WM_USER 标识符是可用的。每个窗口类都可以有它自己独立的、独一无二的一组消息。其他预定义的子窗口控件类也可以有它们自己的按照 WM_USER 定义的消息。

BM_GETCHECK 和 BM_SETCHECK 消息被父窗口发送到子窗口控件,以取得并设置复选框和单选按钮的选中标记。BM_GETSTATE 消息和 BM_SETSTATE 消息反映当用鼠标器或者空格键按动一个窗口时窗口的一般状态或者"压入(pushed)"状态。在以下讨论每种按钮时,将会看到这些消息是怎样工作的。BM_SETSTYLE 消息允许用户在按钮建立之后改变按钮的风格。

11.3.4 压入按钮

在 BTNLOOK 中显示的前两个按钮是"压入(push)"按钮。一个按钮是一个矩形,它包含了 CreateWindow 调用中窗口文本参数所指定的文本。该矩形占用了在 CreateWindow 或 MoveWindow 调用中给出的全部高度和宽度,而文本在矩形的中心。

压入按钮控件大多数用来触发一个立即响应的动作，而不保留任何形式的开/关(on/off)指示。两种类型的按钮控件有两种窗口风格，分别叫做 BS_PUSHBUTTON 和 BS_DEFPUSHBUTTON。BS_DEFPUSHBUTTON 中的“DEF”代表“default(缺省)”。当用来设计对话框时，BS_PUSHBUTTON 控件和 BS_DFPUSHBUTTON 控件的作用不同。但是当用作子窗口控件时，两种类型的按钮作用相同，尽管 BS_DEFPUSHBUTTON 的边框要粗些。

当按钮的高度是 SYSTEM_FONT 字符的高度的 7/4 倍时，按钮的外观最好。其中 SYSTEM_FONT 字符由 BTNLOOK 所使用。按钮的宽度至少调节到文本的宽度加上另外两个字符的宽度。

当鼠标器的光标在按钮中时，按下鼠标器按钮就使按钮重新涂上三维阴影，就好像真的被按下了一样。放开鼠标器按钮时，就恢复按钮原来的样子，并向父窗口发送 WM_COMMAND 消息和通知码 BN_CLICKED。与其他按钮类型相似，当按钮有输入焦点时，在文本的周围有短划线，按下和释放空格与按下和释放鼠标器按钮有相同的效果。

通过给窗口发送 BM_SETSTATE 消息，用户可以摸拟按钮闪动。如下操作导致按钮被按下：

```
SendMessage(hwndButton,BM_SETSTATE,1,0L);
```

下面的操作使按钮恢复正常：

```
Send Message(hwnButton,BM_SETSTATE,0,0L);
```

hwndButton 窗口句柄是从 CreateWindow 调用返回的值。

也可以向按钮发送 BM_GETSTATE 消息，子窗口控件返回按钮的当前状态——如果按钮被按下，则为 TRUE；如果按钮是正常状态时则为 FALSE。但是，大多应用程序不要求这一消息。因为按钮不保留任何开(on)/关(off)信息，所以 BM_SETCHECK 消息和 BM_GETCHECK 消息不被用到。

11.3.5 复选框

复选框是一个文本方框，文本通常出现在复选框的左边(在建立按钮时如果包括了 BS_LEFTTEXT 风格，则文本就出现在左边)。复选框通常用于可让用户对选项进行选择的应用程序中。复选框的常用功能如同一个开关：按动复选框一次，就出现×，再按一次则×消失。

复选框最常用的两种风格是 BS_CHECKBOX 和 BS_AUTOCHECKBOX。在使用 BS_CHECKBOX 时，用户必须自己向该控件发送 BM_SETCHECK 消息来设置×标记。wParam 参数置 1 时设置×标记，置 0 则取消×标记。通过向该控件发送 BM_GETCHECK 消息，用户可以得到该复选框的当前状态。在处理来自控件的 WM_COMMAND 消息时，可以用如下指令来触发×标记：

```
SendMessage(LOWORD (lParam),BM_SETCHECK,(WORD)
    ! SendMessage(LOWORD (lParam),BM_GETCHECK,0,0L),0L);
```

注意第二个 SendMessage 调用前面的操作符，其中 lParam 的低字是在 WM_COMMAND 消息中传给用户窗口过程的子窗口句柄。如果以后又想知道按钮的状态，就向它发送另一条 BM_GETCHECK 消息；也可以将当前状态保留在用户窗口过程中的一个静态变量中，或者向它发送 BM_SETCHECK 消息来初始化带×的 BS_CHECKBOX 复选框：

```
SendMessage(hwndButton,BM_SETCHECK,1,0L);
```

对 BS_AUTOCHECKBOX 风格，按钮控件自己触发×标记的开和关，用户窗口过程可忽略 WM_COMMAND 消息。当需要按钮的当前状态时，向该控件发送 BM_GETCHECK 消息：

```
nCheck = (WORD) SendMessage(hwndButton,BM_GETCHECK,0,0L);
```

如果该按钮被选中，则 nCheck 的值是 TRUE(真)或者是非零数；如果按钮未被选中，则 ncheck 的值是 FALSE 或 0。

其余两种复选框风格是 BS_3STATE 和 BS_AUTO3STATE，正如它们的名字所示，这两种风格能显示第 3 种状态——复选框内是灰色——它出现在向控件发送 WM_SETCHECK 消息(wParam=2)的时候。灰色是向用户表明此框不能被选中——它是被禁用的。但是，当按下它时，该复选框继续向其父窗口发送消息。使一个复选框被禁用的较好的方法将在后面叙述。

复选框沿矩形的左边框排列，并集中在调用 CreateWindow 过程中规定的矩形的上线或下线之间。在该矩形内的任何地方按下鼠标都将向其父窗口发送一条 WM_COMMAND 消息。复选框的最小高度是一个字符的高度，最小宽度是文本中的字符数加 2。

11.3.6 单选按钮

单选按钮的形状是个圆圈，而不是方框。除此之外，它非常像复选框。圆圈内的加重圆点表示该单选按钮已被选中。单选按钮有窗口风格 BS_RADIOBUTTON 或 BS_AUTORADIOBUTTON 两种，但后者只用于对话框。

在对话框中，单选按钮组常常用来表示相互排斥的选项(例如，观察 Window Terminal 程序中的对话框，该程序是在从 Setting 菜单中选择 Communication 时出现的)。与复选框不同，单选按钮的工作与开关不一样，当第二次按下单选按钮时，它的状态保持不变。

当用户收到来自单选按钮的 WM_COMMAND 消息时，用户应当向它发送其 wParam 等于 1 的 BM_SETCHECK 消息，以显示它的选中状态：

```
SendMessage(hwndButton,BM_SETCHECK,1,0L);
```

对相同组中的其他所有单选按钮，用户应向它们发送 wParam 等于 0 的 BM_SETCHECK 消息来显示其选断状态：

```
SendMessage(hwndButton,BM_SETCHECK,0,0L);
```

11.3.7 分组框

分组框即风格为 BS_GROUPBOX 的选择框，它是按钮类中的特例，它既不处理鼠标器输入和键盘输入，也不向其父窗口发送消息。分组框是一个矩形框，窗口文本在其顶部显示。分组框常用来包含其他按钮控件。

11.3.8 改变按钮文本

调用 SetWindowText 能够改变按钮(或任何其他窗口)内的文本：

```
SetWindowText(hwnd,lpszString);
```

其中 hwnd 是正在改变的窗口的句柄，lpszString 是指向空终结串的长(或远程)指针。对一般的窗口，这个文本是标题行的文本；对按钮控件，它是随该按钮显示的文本。

也可以得到窗口的当前文本：

```
nLenght = GetWindowText(hwnd,lpszBuffer,nMaxLength);
```

参数 nMaxLength 规定拷贝到 lpszBuffer 指定的缓冲区中的最大字符数。该函数返回拷贝的字符数。首先通过如下调用，用户可以为特定的文本长度准备自己的程序：

```
nLength = GetWindowTextLength(hwnd);
```

11.3.9 可见的和启用的按钮

为了接收鼠标器和键盘输入，子窗口必须是可见的(被显示时)和被启用的。当一个窗口是可见的，但未被启用时，窗口显示为灰色，而不是黑色。

在建立子窗口时，如果没有将 WS_VISIBLE 包含在窗口类内，那么直到调用 ShowWindow 时子窗口才被显示出来：

```
ShowWindow(hwndchild,SW_SHOWNORMAL);
```

如果将 WS_VISIBLE 包含在窗口类内，就不必调用 ShowWindow。但是，调用 ShowWindow 可以将子窗口隐藏起来：

```
ShwoWindow(hwndchild,SW_HIDE);
```

如下调用可以确定子窗口是不是可见的：

```
IsWindowVisible(hwndChild);
```

用户也可以使子窗口被启用或不被启用。缺省情况下，窗口是被启用的。如下调用可使窗口不被启用：

```
EnableWindow(hwndchild,FALSE);
```

对按钮控件，这有使按钮文本串变灰的作用。按钮不再对鼠标器输入和键盘输入作出响应。这是说明按钮选项当前不可用的最好方法。

如下调用能使子窗口再次被启用：

```
EnableWindow(hwndchild,TRUE);
```

用户可以用如下调用确定子窗口是否被启用：

```
IsWindowEnabled(hwndChild);
```

11.3.10 按钮和输入焦点

正如本章前面所提到的，当用鼠标点一下按钮、复选框、单选按钮和自绘按钮时，它们接收输入焦点。这些控件用文本周围的点划线表示它有输入焦点。当子窗口控件得到输入焦点时，其父窗口就失去了它；所有的键盘输入都进入子窗口控件，而不进入父窗口。但是，子窗口控件只对空格键作出响应，此时空格键的作用就如鼠标一样。这种情形导致了一个明显的问题：用户程序失去了对键盘处理的控件。让我们看看对此我们能做些什么。

当 Windows 将输入焦点从一个窗口(例如一个父窗口)转换到另一个窗口(例如一个子窗口)时，它首先给正在失去输入焦点的窗口发送一条 WM_KELLFOCUS 消息，wParam 参数是接受输入焦点的窗口的句柄。然后，Windows 向正在接受输入焦点的窗口发送一条 WM_SETFOCUS 消息，同时 wParam 是正在失去输入焦点的窗口的句柄(在这两种情况下，wParam 可能取值 NULL，它表示没有窗口拥有或正在接收输入焦点)。

通过处理 WM_KELLFOCUS 消息，父窗口能阻止子窗口控件获得输入焦点。假定数组 hwndChild 包含所有子窗口的窗口句柄(它们是在调用 CreateWindow 来建立窗口的过程中

保存到数组中的)。NUM 是子窗口的数目:

```
case WM_KELLFOCUS:
    for(i = 0;i < NUM;i++)
        if(hwndChild[i] == wParam)
        {
            SetFocus(hwnd);
            break;
        }
    return 0;
```

在上面的程序中,当父窗口获知它正在丢失输入焦点而让它的某个子窗口得到输入焦点时,它将调用 SetFocus 来恢复输入焦点给自己所有。

下面是达到相同目的的更简单(但不太自然)的方法:

```
Case WM_KELLFOCUS:
    if (hwnd == GetParent(wParam))
        SetFocus(hwnd);
    return 0;
```

但是,这两种方法都有缺点:它们阻止按钮对空格键作出响应,因为该按钮总是得不到输入焦点。一个更好的方法是能使按钮得到输入焦点,也能让用户用 Tab 键从一个按钮转移到另一个按钮,可以用"窗口分组"技术来实现这种方法。

11.4 控件和颜色

在 BTNLOOK 中稍加些"gotcha",程序就会出错,但可能不会马上明显地表现出来。如下操作可以看到这种情况:运行 BTNLOOK,引出 Windows 中的 Control Panel 程序。选择 Color 图标,引出允许用户改变系统颜色的对话框。选择 Color Palette 并改变窗口背景(Window Backgound)和窗口文本(Window Text)的颜色,同时按下 OK 按钮将新选择的颜色存盘。这时就会看到,BTNLOOK 中的背景和按钮文本呈现出新选择的颜色,但是 BTNLOOK 中客户区域其余部分的背景颜色和文本颜色保持不变——白底黑字。这看起来很别扭。

为什么?很简单。因为按钮的颜色是随控制面板(Control Panel)上设置的系统颜色的改变而改变的。但是,BTNLOOK 客户区域的背景之所以保持白色不变,是由于在窗口类中已指定了白色:

```
wndclass.hbrBackground = GetStockObject(WHITE_BURSH);
```

当 BTNLOOK 写文本到显示器上时,它使用在缺省设备环境中定义的文本颜色和背景颜色。它们通常是黑色或白色,而不管用控制面板设置的系统颜色是哪种颜色。

解决如上所述的问题,要涉及到 Windows 的"系统颜色"。

11.4.1 系统颜色

Windows 支持 21 种系统颜色供各种显示之用。用 GetSysColor 和 SetSysColor 可以得到和设置这些颜色。在 WINDOWS.H 中定义的标识符规定了系统颜色。用 SetSysColor 设置的系统颜色只能改变当前 Windows 使用的颜色而不能保留它。

借助 Windows 中的控制面板程序或通过修改 WIN.INI 文件中的[color]部分,用户可以改变系统颜色。在[colors]部分中,对 21 种系统颜色(与 GetSysColor 和 SetSysColor 中的标识

符不同)使用了关键字,21 种系统颜色的后面跟着红、绿、蓝三种颜色的值,该值的变化范围是 0 到 255。下面的表格说明了对于 GetSysColor,SetSysColor,Win. lin 关键字和控制面板(Control Panel)项,用 WINDOWS. H 标识符是怎样识别这 21 种系统颜色的:

GetSysColor 和 SetSysColor	WIN. INI	Control Panel
COLOR_SCROLLBAR	Scrollbar	Scroll Bars
COLOR_BACKGROUND	Background	Desktop
COLOR_ACTIVECAPTION	ActiveTitle	Active Title Bar
COLOR_INACTIVECAPTION	InactiveTitle	Inactive Title Bar
COLOR_MENU	Menu	Menu Bar
COLOR_WINDOW	Window	Window Background
COLOR_WINDOWFRAME	WindowFrame	Window Frame
COLOR_MENUTEXT	MenuText	Menu Text
COLOR_WINDOWTEXT	WindowText	Window Text
COLOR_CAPTIONTEXT	TitleText	Active Title Bar Text
COLOR_INACTIVECAPTIONTEXT	InactiveTitleText	Inactive Title Bar Text
COLOR_ACTIVEBORDER	ActiveBorder	Active Border
COLOR_INACTIVEBORDER	InactiveBorder	Inactive Border
COLOR_APPWORKSPACE	AppWorkspace	Application Workspace
COLOR_HIGHLIGHT	Highlight	Highlight
COLOR_HIGHLIGHTTEXT	HighlightText	Highlighted Text
COLOR_BTNFACE	ButtonFace	Button Face
COLOR_BTNSHADOW	ButtonShadow	Button Shadow
COLOR_GRAYTEXT	GrayText	Disabled Text
COLOR_BTNTEXT	ButtonText	Button Text
COLOR_BTNHIGHLIGHT	ButtonHighlight	Button Highlight

其中大多数都有自解释性。COLOR_BACKGROUND 是所有窗口后面的工作面区域的颜色,COLOR_WINDOWFRAME 是用于在许多显示的部分之间划线的颜色,例如,在某个菜单和某个客户区之间划线。两种"高亮度"颜色与菜单和列表框中所作的选择有关。五种系统颜色决定了压入按钮中使用的颜色。

这 21 种颜色的缺省值由显示驱动程序提供,Windows 一直使用这些缺省值,除非被 WIN. INI 的[color]修改。

11.4.2 按钮颜色

COLOR_WINDOW 和 COLOR_WINDOWTEXT 被许多窗口用来着色。按扭控件(压入按钮除外)用 COLOR_WINDOW 给按钮背后的背景着色(对分组框,COLOR_WINDOW 只被用来给文本的背景着色)。按钮控件用 COLOR_WINDOWTEXT 给文本、复选框控件中的方框和单选按钮控件中的圆形按钮着色。按钮和分组框的轮廓线用 COLOR_WINDOWFRAME 来定义。

用户可以用两种方法中的任何一种来使主窗口与子窗口控件在颜色的使用上保持一致。第一种方法是对主窗口使用系统颜色。首先,在定义窗口类时用 COLOR_WINDOW 给客户区域的背景着色:

wndclass. hbrBackgroud = COLOR_WINDOW + 1;

当在 wndclass 结构中使用这些标识符时 Windows 要求加 1，这样做的目的是防止其值为 0，而不是为了其他目的。但这会引出另一个问题。当用 TextOut 显示文本时，Windows 使用的是在设备环境中为背景颜色（它将删除文本后的背景）和文本颜色定义的值。其缺省值为白（背景）和黑色（文本），而不管系统颜色和窗口类结构中的 hbrBackground 域为何值。所以，用户必须用 SetTextColor 和 SetBkColor 将文本和文本背景的颜色改变为系统颜色。在获得设备环境的句柄之后可以：

```
SetBkColor(hdc,GetSysColor (COLOR_WINDOW));
SetTextColor(hdc,GetSysColor (COLOR_WINDOWTEXT));
```

这样，客户区域背景、文本背景和文本的颜色都与按钮的颜色一致。这就是第一种方法。

第二种方法是强迫子窗口控件使用所想用的颜色。这种方法要稍复杂一些，它要求处理 WM_CTLCOLOR 消息。

11.4.3 WM_CTLCOLOR 消息

WM_CTLCOLOR 是子窗口即将给它的客户区着色时由预定义的子窗口控件发送给其父窗口过程的消息。父窗口可以借此机会改变子窗口过程将要用来着色的颜色。

当父窗口过程收到一条 WM_CTLCOLOR 消息时，参数 wParam 和 lParam 的含义如下：

wParam	子窗口设备环境的句柄
LOWORD(lParam)	子窗口句柄
HIWORD(lParam)	窗口类型

lParam 的高位字可以取下表中的一个：

HIWORD(lParam)	窗口类型
CTLCOLOR_MSGBOX	Message box（消息框）
CTLCOLOR_EDIT	Edit control（编辑控件）
CTLCOLOR_LISTBOX	List box control（列表框控件）
CTLCOLOR_BTN	Button control（按钮控件）
CTLCOLOR_DLG	Dialog box（对话框）
CTLCOLOR_SCROLLBAR	Scrollbar control（滚动条控件）
CTLCOLOR_STATIC	Static control（静态控件）

此时，感兴趣的是 CTLCOLOR_BTN（来自钮控件的 WM_CTLCOLOR 消息）。当父窗口过程得到这个消息时，子窗口控件已经获得了设备环境。该设备环境句柄保存在 wParam 内。当将控制返回到子窗口时，用该设备环境句柄对 GDI（图形设备界面）的任何调用都将影响到子窗口所进行的着色。

当处理 WM_CTLCOLOR 消息时，必须完成如下三个动作：

- 用 SetTextColor 设置文本颜色。
- 用 SetBkColor 设置背景颜色。
- 将刷子句柄返回到子窗口。

刷子是一个 GDI 对象，它定义像素的一个位映像模式。Windows 借助刷子用颜色来填充区域。用 GetStockObject、CreateSolidBrush、CreateHatchBrush 或 CreatePatternBursh 能获得刷子

的句柄。为了处理 WM_CTLCOLOR 消息，可能会用到 CreateSolidBrush。在结束程序之前，必须明确地删除所建立的所有刷子。删除的最佳时间是在处理 WM_DESTROY 消息期间。

对绝大多数子窗口控件来说，在 SetBkColor 中设置的颜色应该与从 WM_CTLCOLOR 消息返回的刷子的颜色相同。例如，按钮控件用刷子给整个子窗口客户区域的背景着色，文本背景色只用于文本后面的背景，这两种颜色应当相同。为清楚起见，以下将举一个为按钮控件处理 WM_CTLCOLOR 消息的例子来说明。在该按钮控件中，窗口过程设置的是普通的缺省颜色。在初始化过程中(也可能是在处理 WM_CREATE 消息时)，可以设置刷子：

```
hBrush = CreateSolidBrush(GetSysColor(COLOR_WINDOW));
```

hBrush 刷子句柄应存在静态变量中，下面是 WM_CTLCOLOR 处理的情形：

```
case WM_CTLCOLOR:
    if(HIWORD (lParam) == CTLCOLOR_BTN)
    {
        SetBkColor(wParam,GetSysColor(COLOR_WINDOW));
        SetTextColor(wParam,GetSysColor(COLOR_WINDOWTEXT));
        UnrealizeObject(hBrush);
        point.x = point.y = 0
        ClientToScreen(hwnd,&point);
        SetBrushOrg(wParam,point.x,point.y);
        return ((DWORD) hBrush);
    }
    break;
```

注意，wParam 是按钮控件的设备环境句柄。SetBrushOrg 调用以前(包括 SetBrushOrg 调用)的四个语句需要进一步的解释。

如前所述，刷子定义像素的一个位映像模式。当 Windows 用这个刷子给区域着色时，刷子不断地重复水平和垂直运动，直到填满全部区域。刷子的起点——Windows 认定的刷子重复运动的起始位置——在与设备环境有关的客户区域的左上角。

但是，如果用同样的颜色给父窗口的客户区域和子窗口的客户区域着色，则这种模式在子窗口的边缘不能正确合并，这是因为 Windows 对同一刷子使用了不同的起始点。调用 UnrealizeObject 能防止该问题发生。这个函数使得刷子下次被选入设备环境中时(在处理完 WM_CTLCOLOR 消息而返回之后)，Windows 将刷子的起始原点复位。Windows 将要使用的起点是用 SetBrushOrg 设置的那个起始点；该例子中，这个函数将刷子的起点设置在父窗口的屏幕原点上(不要对 GetStockObject 所获得的后援刷子句柄使用 UnrealizeObject，也不必担心这里是否显得有些模糊)。

上例中的刷子是以系统颜色 COLOR_WINDOW 为基础的。如果在运行该程序时系统颜色发生了变化，窗口过程会收到一条 WM_SYSCOLORCHANGE 消息。程序删除该刷子并建立一个新的刷子：

```
case WM_SYSCOLORCHANGE:
    DeleteObject (hBrush);
    hBrush = CreateSolidBrush (GetSysColor (COLOR_WINDOW));
    return 0;
```

最后，当程序结束时，刷子应该被删除：

```
case WM_DESTROY;
    DeleteObject (hBrush);
```

```
            PostQuitMessage (0);
            return 0;
```

上面说明了对于按钮控件如何再生成 WM_CTLCOLOR 消息的缺省处理过程。除非想使刷子以系统颜色为基础，否则，用户不必捕获 WM_SYSCOLORCHANGE 消息。在本章稍后，将再重复讲述 WM_CTLCOLOR 消息，那时我们还将以 COLORS1 程序为例。

11.4.4 自绘按钮

如果想对按钮的全部可见部分实行总的控制，而不想让键盘逻辑所困扰，用户就可以用 BS_OWNERDRAW 来建立一个按钮。这由下面的 OWNERDRAW 程序说明：

```
# OWNERDRW.MAK make file

ownerdrw.exe : ownerdrw.obj ownerdrw.def
     $(WINLINK) ownerdrw,ownerdrw,NUL, $(WINLIB),ownerdrw
     rc -t ownerdrw.exe

ownerdrw.obj : ownerdrw.c
     $(WINCC) ownerdrw.c

/*--------------------------------------------
OWNERDRW.C -- Owner Draw Button Demo Program
--------------------------------------------*/

#include <windows.h>

#define IDC_SMALLER 1
#define IDC_LARGER 2

#define BTN_WIDTH (8 * cxChar)
#define BTN_HEIGHT (4 * cyChar)

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

HANDLE hInst ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName [] = "OwnerDrw" ;
     MSG msg;
     HWND hwnd ;
     WNDCLASS wndclass ;

     hInst = hInstance ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
```

```
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = szAppName ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName,"Onwer Draw Button Demo",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd);

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

void Triangle (HDC hdc,POINT pt[])
{
    SelectObject (hdc,GetStockObject (BLACK_BRUSH)) ;
    Polygon (hdc,pt,3) ;
    SelectObject (hdc,GetStockObject (WHITE_BRUSH)) ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    static HWND hwndSmaller,hwndLarger ;
    static short cxClient,cyClient,cxChar,cyChar ;
    LPDRAWITEMSTRUCT lpdis ;
    POINT pt [3] ;
    RECT rc ;
    short cx,cy ;

    switch (message)
    {
        case WM_CREATE:
            cxChar = LOWORD (GetDialogBaseUnits ()) ;
            cyChar = HIWORD (GetDialogBaseUnits ()) ;
                                                    // Create the owner-draw pushbuttons
```

```
        hwndSmaller = CreateWindow ("button","",
            WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
            0,0,BTN_WIDTH,BTN_HEIGHT,
            hwnd,IDC_SMALLER,hInst,NULL) ;
        hwndLarger = CreateWindow ("button","",
            WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
            0,0,BTN_WIDTH,BTN_HEIGHT,
            hwnd,IDC_LARGER,hInst,NULL) ;
        return 0 ;
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
                                                    // Move the buttons to the new center
        MoveWindow (hwndSmaller,cxClient / 2 - 3 * BTN_WIDTH / 2,
            cyClient / 2 - BTN_HEIGHT / 2,
            BTN_WIDTH,BTN_HEIGHT,TRUE) ;
        MoveWindow (hwndLarger, cxClient / 2 + BTN_WIDTH / 2,
            cyClient / 2 - BTN_HEIGHT / 2,
            BTN_WIDTH,BTN_HEIGHT,TRUE) ;
        return 0 ;
    case WM_COMMAND:
        GetWindowRect (hwnd,&rc) ;
                                                    // Make the window 10% smaller or larger
        switch (wParam)
        {
            case IDC_SMALLER:
                rc.left += cxClient / 20 ;
                rc.right -= cxClient / 20 ;
                rc.top += cyClient / 20 ;
                rc.bottom -= cyClient / 20 ;
                break ;
            case IDC_LARGER:
                rc.left -= cxClient / 20 ;
                rc.right += cxClient / 20 ;
                rc.top -= cyClient / 20 ;
                rc.bottom += cyClient / 20 ;
                break ;
        }
        MoveWindow (hwnd,rc.left,rc.top,rc.right - rc.left,
                rc.bottom - rc.top,TRUE) ;
        return 0 ;
    case WM_DRAWITEM:
        lpdis = (LPDRAWITEMSTRUCT) lParam ;
                                                    // Fill area with white and frame it black
        FillRect (lpdis->hDC,&lpdis->rcItem,
            GetStockObject (WHITE_BRUSH)) ;
        FrameRect (lpdis->hDC,&lpdis->rcItem,
            GetStockObject (BLACK_BRUSH)) ;
                                                    // Draw inward and outward black triangles
        cx = lpdis->rcItem.right - lpdis->rcItem.left ;
        cy = lpdis->rcItem.bottom - lpdis->rcItem.top ;
```

```
switch (lpdis->CtlID)
{
    case IDC_SMALLER:
        pt[0].x = 3 * cx / 8 ; pt[0].y = 1 * cy / 8 ;
        pt[1].x = 5 * cx / 8 ; pt[1].y = 1 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 3 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        pt[0].x = 7 * cx / 8 ; pt[0].y = 3 * cy / 8 ;
        pt[1].x = 7 * cx / 8 ; pt[1].y = 5 * cy / 8 ;
        pt[2].x = 5 * cx / 8 ; pt[2].y = 4 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        pt[0].x = 5 * cx / 8 ; pt[0].y = 7 * cy / 8 ;
        pt[1].x = 3 * cx / 8 ; pt[1].y = 7 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 5 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        pt[0].x = 1 * cx / 8 ; pt[0].y = 5 * cy / 8 ;
        pt[1].x = 1 * cx / 8 ; pt[1].y = 3 * cy / 8 ;
        pt[2].x = 3 * cx / 8 ; pt[2].y = 4 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        break ;
    case IDC_LARGER:
        pt[0].x = 5 * cx / 8 ; pt[0].y = 3 * cy / 8 ;
        pt[1].x = 3 * cx / 8 ; pt[1].y = 3 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 1 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        pt[0].x = 5 * cx / 8 ; pt[0].y = 5 * cy / 8 ;
        pt[1].x = 5 * cx / 8 ; pt[1].y = 3 * cy / 8 ;
        pt[2].x = 7 * cx / 8 ; pt[2].y = 4 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        pt[0].x = 3 * cx / 8 ; pt[0].y = 5 * cy / 8 ;
        pt[1].x = 5 * cx / 8 ; pt[1].y = 5 * cy / 8 ;
        pt[2].x = 4 * cx / 8 ; pt[2].y = 7 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        pt[0].x = 3 * cx / 8 ; pt[0].y = 3 * cy / 8 ;
        pt[1].x = 3 * cx / 8 ; pt[1].y = 5 * cy / 8 ;
        pt[2].x = 1 * cx / 8 ; pt[2].y = 4 * cy / 8 ;
        Triangle (lpdis->hDC,pt) ;
        break ;
}
                                    // Invert the rectangle if the button is selected
if (lpdis->itemState & ODS_SELECTED)
    InvertRect (lpdis->hDC,&lpdis->rcItem) ;
                                    // Draw a focus rectangle if the button has the focus
if (lpdis->itemState & ODS_FOCUS)
{
    lpdis->rcItem.left += cx / 16 ;
    lpdis->rcItem.top += cy / 16 ;
    lpdis->rcItem.right -= cx / 16 ;
    lpdis->rcItem.bottom -= cy / 16 ;
    DrawFocusRect (lpdis->hDC,&lpdis->rcItem) ;
}
```

```
            return 0 ;
        case WM_DESTROY :
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; OWNERDRW.DEF module definition file

NAME OWNERDRW

DESCRIPTION 'Owner Draw Button Demo (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

如图 11.6 所示，该程序在客户区的中央包含了两个按钮。左边的按钮有四个三角形指向按钮的中心。按下该按钮时，窗口的尺寸缩小 10%。右边的按钮有四个向外指的三角形，按下此按钮口时，窗口的尺寸增大 10%。

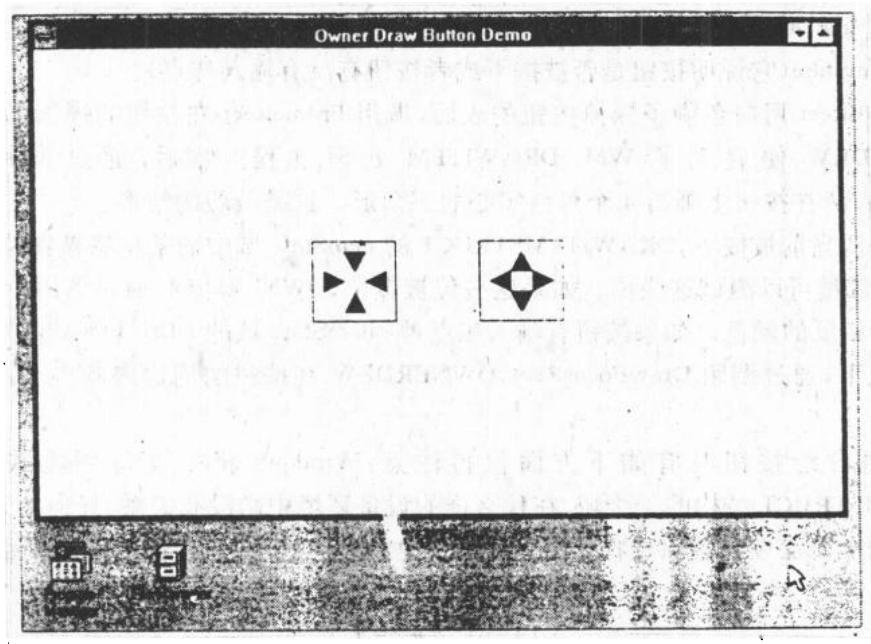


图 11.6　OWNERDRW 显示

使用 BS_OWNERDRAW 按钮风格来画它们自己的按钮的大多数程序通常给按钮使用较小的位图。

在处理 WM_CREATE 消息期间，通过调用 GetDialogBaseUnits 函数，OWNERDRW 得到系统字型的平均宽度和高度。这个函数是 Windows 3.0 版新增的，用起来也很方便。然后，

OWNERDRW 建立两个 BS_OWNERDRW 风格的按钮，按钮的宽度是系统字模的 8 倍，高度是系统字模的 4 倍(在用预定义好的位图绘制按钮时，这些尺寸在 VGA 上建立的按钮是 64 像素×64 像素，知道这些数据非常有用)。这些按钮还没有定位。在处理 WM_SIZE 消息期间，通过调用 MoveWindow 函数，OWNERDRW 将按钮定位在客户区的中心。

按下这些按钮时，它们就会生成 WM_COMMAND 消息，这时，wParam 等于控件窗口的 ID 号，其中控件窗口 ID 已被作为一个参数传送给最初建立它们的 CreateWindow 函数。为了处理 WM_COMMAND 消息，OWNERDRW 调用 GetWindowRect 来将整个窗口(不只是客户区)的位置和尺寸存于 RECT(rectangle)结构中。这个位置与屏幕相关。然后，根据按下的是左边的还右边的按钮，OWNERDRW 调节这个矩形结构的各域值。程序再通过调用 MoveWindow 来重新确定位置和尺寸。这将生成另一条 WM_SIZE 消息，按钮被重新定位在客户区的中央。

如果这是程序所做的全部，按钮已完全能起作用，但是按钮是不可见的。用 BS_OWNERDRAW 风格建立的按钮在需要重新着色的任何时候都会向它的父窗口发送一条 WM_DRAWITEM 消息。这种现象出现在以下几种情况中：按钮是首先建立的、当键扣被按下或者松开时、当按钮得到或者失去输入焦点时、按钮需要重新着色的任何时候。

处理 WM_DRAWITEM 消息期间，lParam 消息参数指向 DRAWITEMSTRUCT 类型结构的指针。OWNERDRW 程序将这个远程指针保存在 lpdis 变量中。这个结构包含了画该按钮时程序所必需的消息(这个结构也可以为自绘列表框和菜单项所使用)。对按钮而言非常重要的结构域有 hDC(对按钮的设备环境)、rcItem(提供按钮尺寸的 RECT 结构)、CtlID(控件窗口 ID)和 itemState(它说明按钮是否被按下或者按钮有没有输入焦点)。

调用 FillRect 用白色刷子抹掉按钮的表面，调用 FrameRect 在按钮的周围画上黑框，由此 OWNERDRW 便启动了 WM_ DRAWITEM 处理过程。然后，通过调用 Polygon，OWNERDRAW 在按钮上画出 4 个黑色实心的三角形。这是一般的情形。

如果按钮当前被按下，DRAWITEMSTRUCT 的 itemState 域中的某位将被置位。用 ODS_SELECTED 常量可以测试这些位。如果这些位被置位，OWNERDRW 通过调用 InvertRect 将按钮转换为相反的颜色。如果按钮有输入焦点，则 itemState 域的 ODS_FOCUS 位将被置位。在这种情况下，通过调用 DrawFocusRect，OWNERDRW 在按钮的周边内画一个点划线式的矩形。

在使用自绘按钮时有如下方面值得注意：Windows 获得设备环境并将其作为 DRAWITEMSTRUCT 结构的一个域。在什么样的设备环境中则没有关系。任何被选进设备环境的 GDI 对象都必须被解除选择。另外，当心不要在定义按钮边界的矩形外面绘制内容。

11.5 静态类

图 11.7 是静态类的典型画面。

与静态类有关的消息如下：

STATIC	含义
SS_BLACKFRAME	矩形框与窗口框同色
SS_BLACKRECT	矩形填充的颜色与窗口框一致

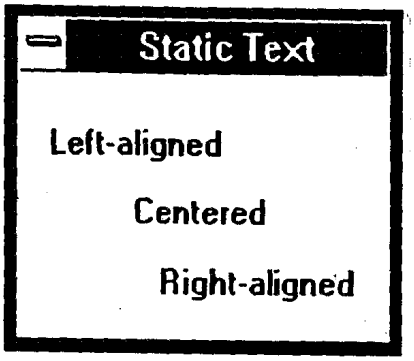


图 11.7

SS_CENTER	产生一个文本居中的简单矩形
SS_GRAYFRAME	矩形框与屏幕背景同色(桌面)
SS_GRAYRECT	矩形填充的颜色与屏幕背景色一致
SS_ICON	产生一个图标
SS_LEFT	产生一个文本左对齐的简单矩形框
SS_LEFTNOWORDWRAP	裁剪掉超出矩形边界的文本
SS_NOPREFIX	下划线(&)不被认为是控制字符
SS_RIGHT	产生文本右对齐的简单矩形
SS_SIMPLE	产生一个文本不能改变的简单矩形
SS_USERITEM	指定一个用户定义项
SS_WHITEFRAME	矩形框与窗口背景同色
SS_WHITERECT	矩形填充的颜色与窗口背景色一致

用户不能改变静态文本输出。除非类型 SS_SIMPLE 被设置,否则它可在应用程序中被覆盖。STATIC 类的窗口主要用作其他控件或标题,例如在编辑字段,它们自己一般是没有标题的。

它还可用于显示信息,告知用户各种条件和事件,例如程序安装的进程等。

在 CreateWindow 函数中将 static 用作窗口类就能建立静态子窗口控制。这些子窗口很少是活跃的子窗口,它既不接受鼠标器或者键盘输入,也不向父窗口发送 WM_COMMAND 消息(当鼠标器在静态子窗口上移过或是按下时,这个子窗口捕获 WM_NCHITTEST 消息并将 HTTRANSPARENT 的值返回给 Windows。这将引起 Windows 向基层窗口(通常是它的父窗口)发送相同的 WM_NCHITTEXT 消息。父窗口常常将该消息传递给 DefWindowProc,在这里,它被转换为客户区鼠标器消息)。

前边六个静态窗口风格只简单地在子窗口的客户区内画一个矩形或边框。"RECT"静态风格(下面左列)填充矩形;三个"FRAME(边框)" 风格(右列)是没有填充的矩形轮廓线:

SS_BLACKRECT	SS_BLACKFRAME
SS_GRAYRECT	SS_GRAYFRAME
SS_WHITERECT	SS_WHITEFRAME

"BLACK(黑)"、"GRAY(灰)"、"WHITE(白)"并不意味着黑、灰和白色,这些颜色是以系

统颜色为基础的，如下所示：

静态控件	系统颜色
BLACK	COLOR_WINDOWFRAME
GRAY	COLOR_BACKGROUND
WHITE	COLOR_WINDOW

大多数显示驱动程序定义 COLOR_WINDOWFRAME 的缺省设置为黑色，定义 COLOR_WINDOW 的缺省设置为白色(当然，用户可以用 Windows 中的控制面板程序改变这些颜色中的任何一种)。在“RECT”和“FRAME”静态风格中使用的颜色不能通过捕获 WM_CTLCOLOR 消息加以改变。对这些风格，CreateWindow 调用的窗口文本区域被忽略。矩形的左上角从 x 位置坐标和 y 位置坐标开始，x、y 位置坐标是相对于父窗口的。

静态类也包括了三种文本风格：SS_LEFT、SS_RIGHT 和 SS_CENTER，它们建立右对齐、左对齐和居中文本。文本在 CreateWindowText 调用的窗口文本参数中给出，且在以后可以用 SetWindowText 改变它。当静态控件的窗口过程显示文本时，它用到 DrawText 函数以及 DT_WORDBREAK、DT_NOCLIP 和 DT_EXPANDTABS 参数。文本在子窗口的矩形内可整字换行。这三种文本风格子窗口的背景一般是 COLOR_WINDOW，并且文本自身是 COLOR_WINDOWTEXT。在截获 WM_CTLCOLOR 消息时，调用 SetTextColor 能改变文本颜色，调用 SetBkColor 且将句柄返回到背景刷子能改变背景颜色。

最后，静态类还包括了窗口风格 SS_ICON 和 SS_USERITEM，但是当它们被用作子窗口控件时没有意义。后面在讨论对话框时还要涉及它们。

11.6 滚动条

滚动条的初始用途是在文本不完全适合于窗口时允许用户滚动文本，而且，它还可用作输入设备，以改变在连续范围内取值的参数。在第十章我们已对滚动条作了初步介绍，由于滚动条的应用较多，这一节再给出一个例子加以说明。

当选中控制面板上的鼠标图标时，可看见滚动条。一个对话框包含两个滚动条，允许用户设置鼠标的移动速度和双击鼠标的时间间隔。

滚动条的典型显示如下：

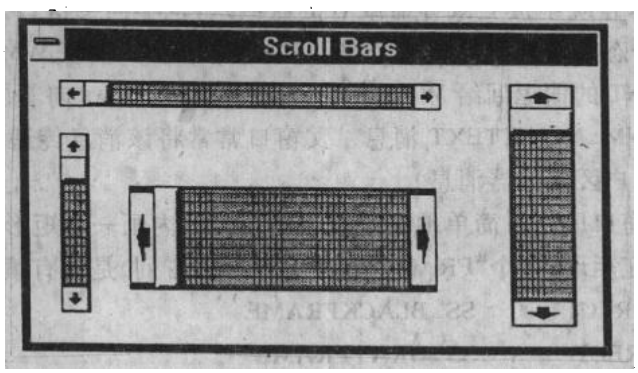


图 11.8

与滚动条有关的消息如下：

SCROLLBAR	含义
SBS_BOTTOMALIGN	SBS_HORZ 为下边界
SBS_HORZ	产生一水平滚动条
SBS_LEFTALIGH	SBS_VERT 以左边界对齐
SBS_RIGHTALIGH	SBS_VERT 以右边界对齐
SBS_SIZEBOX	产生一个大小框
SBS_SIZEBOXBOTTOMRIGHTALIGH	按右下角对齐定位一个大小框
SBS_SIZEBOXTOPLEFTALIGH	按左上角对齐定位一个大小框
SBS_TOPALIGH	SBS_HORZ 为上边界
SBS_VERT	产生一个垂直滚动条

滚动条控件与其他控件不同，因为它发送给父窗口的消息不是 WM_COMMAND 和标识码，而是 WM_HSCROLL 或 WM_VSCROLL 消息。

与窗口边界上的滚动条不同，滚动条控件自动支持键盘；用户可用上、下、左、右键水平或垂直移动“姆指”图标。

11.6.1 带子分类的滚动条例子

下例用三个水平滚动条作为输入设备，允许用户调整主窗口的背景颜色。每个滚动条代表一种基本颜色：红、蓝和绿。

组合每种成分的颜色值就产生了显示器的背景颜色。顶部的滚动条还可用于子分类。

按下 1 或 2 键可分别将焦点移到第二个和第三个滚动条上。

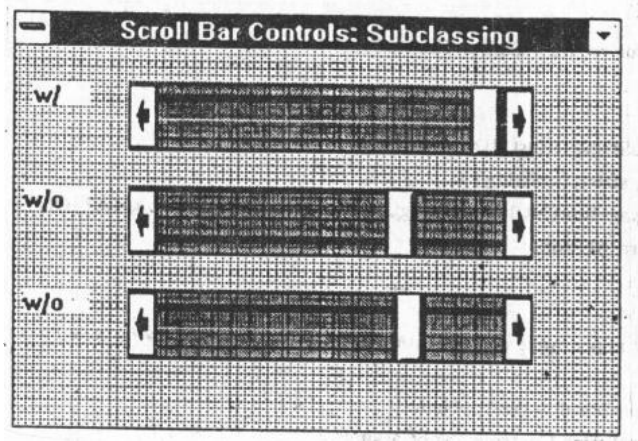


图 11.9 滚动条作为输入设备

新函数	说明
CallWindowProc	调用标准类的窗口例程
SetWindowLong	改变给定窗口的一个属性

源代码:SCROLCTL. C

```
/* * SCROLCTL. C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates scroll bar controls in an application * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "scrolctl. h"                                    // Include SCROLCTL. H header file

#if _MSC_VER >= 700                                       // Running Microsoft C 7. 00 or
#pragma warning (disable:4028)                            // higher? Disable warning 4028
#endif

/* * WinMain (main window function for all Windows applications * * * * * * /

int PASCAL WinMain(HANDLE hInstance,                                  // Current instance
    HANDLE hPrevInstance,                                             // Previous instance
    LPSTR lpszCmdLine,                                                // Long ptr - string after prg.
    int nCmdShow)                                                     // App. window's appearance
{
    MSG msg;                                                          // Message handle
    HWND hWnd;                                                        // Window handle
    int nError;                                                       // Error integer

    strcpy(szAppName,"SCROLCTL");
    hInst = hInstance;
    if(! hPrevInstance)
    {
    /* * Register window class for first instance * * /

        if ((nError = nCwRegisterClasses()) == -1)
        {
                                                                      // If registration fails
            LoadString(hInst,IDS_ERR_CLASS,szString,
                sizeof(szString));
            MessageBox(NULL,szStrihg,NULL,MB_ICONEXCLAMATION);
            return nError;
        }
    }
    /* * Specify appearance of application's main window * * * * * * * * * /

    hWnd = CreateWindow(szAppName,
            "Scroll Bar Controls: Subclassing",
            WS_OVERLAPPED | WS_SYSMENU | WS_BORDER |
            WS_MINIMIZEBOX | WS_CLIPCHILDREN,
            0,0,
            GetSystemMetrics(SM_CXSCREEN)/2,
            GetSystemMetrics(SM_CYSCREEN)/2,
            NULL,NULL,hInstance,NULL);

    if(hWnd == NULL)
    {
```

```
        LoadString(hInst,IDS_ERR_WINDOW,szString,
            sizeof(szString));
        MessageBox(NULL,szString,NULL,MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow(hWnd,nCmdShow);                          // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while(GetMessage(&msg,NULL,0,0))                    // Message reading
    {
        TranslateMessage(&msg);                         // Message translation
        DispatchMessage(&msg);                          // Send message to Windows
    }
    return (int)msg. wParam;                            // Return wParam of last message
}

/* * WndProc * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

LONG FAR PASCAL WndProc(HWND hWnd,WORD Message,WORD wParam,LONG lParam)
{
    static BYTE r,g,b;                                  // Red,Green,Blue elements
    HBRUSH hBrush;                                      // Brush
    BYTE bPos;                                          // Position
    int xScreen,yScreen;
    static FARPROC lpScroll0Proc;                       // Scroll procedure

    switch (Message)
    {
        case WM_CREATE:
            xScreen = GetSystemMetrics(SM_CXSCREEN);    // Draw
            yScreen = GetSystemMetrics(SM_CYSCREEN);    // screen
            hScroll0 = CreateWindow("scrollbar",NULL,
                WS_CHILD | WS_VISIBLE | SBS_HORZ,
                60,yScreen/24,
                xScreen/3,yScreen/12,
                hWnd,NULL,hInst,NULL);
            CreateWindow("static","w/ ",
                WS_CHILD | WS_VISIBLE,
                10,yScreen/24,
                30,15,
                hWnd,NULL,hInst,NULL);
            hScroll1 = CreateWindow("scrollbar",NULL,
                WS_CHILD | WS_VISIBLE | SBS_HORZ,
                60,4 * yScreen/24,
                xScreen/3,yScreen/12,
                hWnd,NULL,hInst,NULL);
            CreateWindow("static","w/o",
                WS_CHILD | WS_VISIBLE.
                5,4 * yScreen/24,
```

```
            35,15,
            hWnd,NULL,hInst,NULL);
        hScroll2 = CreateWindow("scrollbar",NULL,
            WS_CHILD | WS_VISIBLE | SBS_HORZ,
            60,7 * yScreen/24,
            xScreen/3,yScreen/12,
            hWnd,NULL,hInst,NULL);
        CreateWindow("static","w/o",
            WS_CHILD | WS_VISIBLE,
            5,7 * yScreen/24,
            35,15,
            hWnd,NULL,hInst,NULL);
        SetScrollRange( hScroll0,SB_CTL,0,255,TRUE);
        SetScrollRange( hScroll1,SB_CTL,0,255,TRUE);
        SetScrollRange( hScroll2,SB_CTL,0,255,TRUE);
        r = 255;
        g = r;
        b = r;
        SetScrollPos( hScroll0,SB_CTL,255,TRUE);
        SetScrollPos( hScroll1,SB_CTL,255,TRUE);
        SetScrollPos( hScroll2,SB_CTL,255,TRUE);
        lpScroll0Proc = MakeProcInstance((FARPROC)Scroll0Proc,hInst);
        lpScroll0Standard = (FARPROC)SetWindowLong(hScroll0,GWL_WNDPROC,
            (LONG)lpScroll0Proc);
        break;
    case WM_SETFOCUS:
        SetFocus(hScroll0);
        break;
    case WM_HSCROLL:                                    // Horizontal scroll
        SetFocus(HIWORD(lParam));
        if (wParam != SB_THUMBTRACK)
        {
            bPos = (BYTE)GetScrollPos( HIWORD(lParam),SB_CTL);
            switch (wParam)
            {
                case SB_LINEDOWN:
                    bPos += 4;
                    break;
                case SB_LINEUP:
                    bPos -= 4;
                    break;
                case SB_PAGEDOWN:
                    bPos += 20;
                    break;
                case SB_PAGEUP:
                    bPos -= 20;
                    break;
                case SB_THUMBPOSITION:
                    bPos = (BYTE)LOWORD( lParam);
                    break;
            }
```

```
                SetScrollPos( HIWORD(lParam),SB_CTL,bPos,TRUE);

                if ( HIWORD( lParam) == hScroll0)
                    r = bPos;
                if ( HIWORD( lParam) == hScroll1)
                    g = bPos;
                if ( HIWORD( lParam) == hScroll2)
                    b = bPos;

                hBrush = CreateSolidBrush(RGB(r,g,b));
                DeleteObject(GetClassWord(hWnd,GCW_HBRBACKGROUND));
                SetClassWord(hWnd,GCW_HBRBACKGROUND,hBrush);
                InvalidateRect(hWnd,NULL,TRUE);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            SetWindowLong(hScroll0,GWL_WNDPROC,
                (DWORD)lpScroll0Standard);
            FreeProcInstance(lpScroll0Proc);
            break;
        default:
            return DefWindowProc(hWnd,Message,wParam,lParam);
    }
    return 0L;
}

int nCwRegisterClasses(void)
{
    WNDCLASS wndclass;
    memset(&wndclass,0x00,sizeof(WNDCLASS));
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra = 0;
    wndclass.cbWndExtra = 0;
    wndclass.hInstance = hInst;
    wndclass.hIcon = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wndclass.lpszMenuName = (LPSTR)NULL;
    wndclass.lpszClassName = szAppName;
    if(! RegisterClass(&wndclass))
        return -1;

    return(0);
}

LONG FAR PASCAL Scroll0Proc(HWND hWnd,WORD Message,WORD wParam,LONG lParam)
{
    switch (Message)
    {
```

```
        case WM_CHAR:
            switch (wParam)
            {
                case '1':
                    SetFocus(hScroll1);
                    break;
                case '2':
                    SetFocus(hScroll2);
                    break;
                default:
                    CallWindowProc(lpScroll0Standard,hWnd,Message,wParam,lParam);
                    break;
            }
            break;
        default:
            return(CallWindowProc(lpScroll0Standard,hWnd,Message,wParam,lParam));
    }
    return 0L;
}
```

模块定义文件:SCROLCTL. DEF

```
NAME SCROLCTL
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 5110
EXPORTS WndProc @1
        Scroll0Proc @2
```

资源描述文件:SCROLCTL. RC

```
#include "scrolctl. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件:SCROLCTL. H

```
#include <windows. h>
#include <string. h>

#define IDS_ERR_CLASS 1
#define IDS_ERR_WINDOW 2

char szString[128];                    /* Variable for loading resource text */
char szAppName[20];                    /* Window class name */
HWND hInst;
```

```
FARPROC lpScroll0Standard;
HWND hScroll0,hScroll1,hScroll2;

LONG FAR PASCAL WndProc(HWND,WORD,WORD,LONG);
LONG FAR PASCAL Scroll0Proc(HWND,WORD,WORD,LONG);
int nCwRegisterClasses(void);
```

MAKE 文件:SCROLCTL. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = SCROLCTL.H

all: SCROLCTL.EXE

SCROLCTL.RES: SCROLCTL.RC $(RESFILES)

SCROLCTL.OBJ: SCROLCTL.C $(H)

SCROLCTL.EXE: SCROLCTL.OBJ SCROLCTL.RES SCROLCTL.DEF
    link /NOD/A:16 SCROLCTL,SCROLCTL,nul,libw slibcew,SCROLCTL.def
    rc SCROLCTL.res SCROLCTL.exe
```

如下运行 NMAKE 可得 SCROLCTL 应用程序:

```
nmake scrolctl.mak
```

所以,在显示一种新的背景颜色时,滚动条并未着色完毕。风格 WS_CLIPCHILDREN 用于特定的主窗口函数。CreateWindow 发送消息 WM_CREATE 以产生三个滚动条,它们的大小由屏幕大小决定。类似主窗口,用 GetSystemMetrics 函数可获得其大小。每个滚动条的左边是静态编辑控件。这些文本告诉用户支持子分类。

```
hScroll1=CreateWindow("scrollbar",NULL,WS_CHILD|WS_VISIBLE|SBS_HORZ,
        40,4 * yScreen/24,xScreen/3,yScreen/12,hWnd,NULL,
        hInst,NULL);
CreateWindow("static","Without",WS_CHILD|WS_VISIBLE,5,4 * yScreen/24,
        35,15,e hWnd,NULL,hInst,NULL);
```

每种颜色包含红、绿、蓝三种成分,这三个值的范围是 0～255。通过 Windows 宏 RGB 可生成适当的颜色。如果所有值均为 0,则得到黑色。初始时,三个滚动条的"姆指"图标由函数 SetScrollPos 定位在最大值处。

```
SetScrolRange( hScroll0,SB_CTL,0,255,TRUE);
SetXcrollPos( hScroll0,SB_CTL,255,TRUE);
```

当姆指图标移动时,就产生一个新的颜色。在结果消息 WM_HSCROLL 的 wParam 和 lParam 参数中可找到用户滚动要求的细节。先前姆指图标的位置可由 GetScrollPos 得到,并由此计算新位置。

然后，姆指图标的位置由 SetScrollPos 更新。最后，按三个输入位置的综合值得出背景的颜色。检验 if(wParam==SB_THUMBTRACK)在每个消息 WM_HSCROLL 之外进行，它可确保直到姆指图标释放时，颜色才被重置。

名为 ScrolloProc 的用户窗口例程允许用户用键盘从第一个滚动条切换到第二个滚动条。调用函数 SetWindowLog 可把这个新函数拷贝到窗口专用的内存块中，以代替标准窗口例程。

```
lpScroll0Proc = MakeProcInstance((FARPROC) Scroll0Proc,hInst);
lpScroll0Standard = (FARPROC) SetWindowsLong( hScroll0, GWL_WNDPROC,
(LONG)lpScroll0Proc);
```

当两个键(1 或 2)中的一个被按下时，ScrollProc 回调函数处理消息并转换输入焦点，所有其他消息都送给 SCROLLBAR 类的标准函数处理。

```
CallWindowProc(lpScroll0Standard,hWnd,Message,wParam,lParam);
```

在程序最后，用户例程的过程实例地址必须被释放，这由消息 WM_DESTROY 完成。

不像按钮控件(及将要在后面讨论的编辑和列表框控件等)那样，滚动条控件不向父窗口发送 WM_COMMAND 消息，而是像窗口滚动条那样发送 WM_VSCROLL 和 WM_HSCROLL 消息。在处理滚动条消息时，通过 lParam 参数的高位字，用户可以区分开窗口滚动条与滚动条控件：

滚动条类型	HIWORD(lParam)
窗口滚动条	0
滚动条控件	控件的窗口句柄

对窗口滚动条和滚动条控件来说，wParam 参数和 lParam 的低位字的含义相同。

虽然窗口滚动条有固定的宽度，但是 Windows 可利用 CreateWindow 调用中(或者在后面的 MoveWindow 调用中)给出的全矩形尺寸来确定滚动条控件的尺寸。用户可以建立细而长的滚动条控件，也可以建立短而粗的滚动条控件。如果想建立与窗口滚动条尺寸相同的滚动条控件，则用 GetSystemMetrics 可以得到水平滚动条的高度

```
GetSystemMetrics(SM_CYHSCROLL);
```

或者垂直滚动条的宽度：

```
GetSystemMetrics(SM_CXVSCROLL);
```

(滚动条窗口风格识别符 SBS_LEFTALIGN、SBS_RIGHTALIGN、SBS_TOPALIGN 和 SBS_BOTTOMALIGN 以文件的形式来给出滚动条的标准尺寸。但是这些风格只在对话框中对滚动条有效。)

对窗口滚动条使用同样的调用可以建立滚动条控件的范围和位置：

```
SetScrollRange(hwndScroll,SB_CTL,nMin,nMax,bRedraw);
SetScrollPos(hwndScroll,SB_CTL,nPos,bRedraw);
```

其区别在于：窗口滚动条将父窗口的句柄用作第一个参数，而 SB_VERT 或 SB_HORZ 将其用作第二个参数。

滚动条的内部是 COLOR_SCROLLBAR，小方块和箭头的颜色以按钮颜色为基准。如果捕获 WM_CTLCOLOR 消息，可以将刷子由此消息返回并取代该颜色。下一节将尝试这一内容。

11.6.2 COLORS1 程序

为了了解滚动条和静态子窗口的一些用法，也为了深入地了解颜色，需要用到COLORS1 程序，如下所示。COLORS1 将三种滚动条显示在客户区的左半边，并分别标以“Red(红)”、“Green(绿)”和“Blue(蓝)”。当滚动滚动条时，客户区的右半边变为三种元色混合而成的合成色，三种元色的数值显示在三个滚动条下面。

```
# COLORS1.MAK make file

colors1.exe : colors1.obj colors1.def
     $(WINLINK) colors1,colors1,NUL, $(WINLIB),colors1
     rc -t colors1.exe

colors1.obj : colors1.c
     $(WINCC) colors1.c

/*--------------------------------------
COLORS1.C -- Colors Using Scroll Bars
--------------------------------------*/

#include <windows.h>
#include <stdlib.h>

#define min(a,b) (((a) < (b)) ? (a) : (b))
#define max(a,b) (((a) > (b)) ? (a) : (b))

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;
long FAR PASCAL _export ScrollProc (HWND,UINT,UINT,LONG) ;

FARPROC lpfnOldScr[3] ;
HWND hwndScrol[3],hwndLabel[3],hwndValue[3],hwndRect ;
short color[3],nFocus ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "Colors1" ;
     static char *szColorLabel[] = { "Red","Green","Blue" } ;
     FARPROC lpfnScrollProc ;
     HWND hwnd ;
     MSG msg;
     short n ;
     WNDCLASS wndclass ;

     if (hPrevInstance)
          return FALSE ;

     wndclass.style = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc = WndProc ;
     wndclass.cbClsExtra = 0 ;
     wndclass.cbWndExtra = 0 ;
```

```
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = NULL ;
    wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
    wndclass.hbrBackground = CreateSolidBrush (0L) ;
    wndclass.lpszMenuName = NULL ;
    wndclass.lpszClassName = szAppName ;

    RegisterClass (&wndclass) ;

    hwnd = CreateWindow (szAppName,"Color Scroll",
              WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
              CW_USEDEFAULT,CW_USEDEFAULT,
              CW_USEDEFAULT,CW_USEDEFAULT,
              NULL,NULL,hInstance,NULL) ;

    hwndRect = CreateWindow ("static",NULL,
              WS_CHILD | WS_VISIBLE | SS_WHITERECT,
              0,0,0,0,
              hwnd,9,hInstance,NULL) ;

    lpfnScrollProc = MakeProcInstance ((FARPROC) ScrollProc,hInstance) ;

    for (n = 0 ; n < 3 ; n++)
    {
        hwndScrol[n] = CreateWindow ("scrollbar",NULL,
            WS_CHILD | WS_VISIBLE | WS_TABSTOP | SBS_VERT,
            0,0,0,0,
            hwnd,n,hInstance,NULL) ;

        hwndLabel[n] = CreateWindow ("static",szColorLabel[n],
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            0,0,0,0,
            hwnd,n + 3,hInstance,NULL) ;

        hwndValue[n] = CreateWindow ("static","0",
            WS_CHILD | WS_VISIBLE | SS_CENTER,
            0,0,0,0,
            hwnd,n + 6,hInstance,NULL) ;

        lpfnOldScr[n] = (FARPROC) GetWindowLong (hwndScrol[n],GWL_WNDPROC) ;
        SetWindowLong (hwndScrol[n],GWL_WNDPROC,(LONG) lpfnScrollProc) ;

        SetScrollRange (hwndScrol[n],SB_CTL,0,255,FALSE) ;
        SetScrollPos (hwndScrol[n],SB_CTL,0,FALSE) ;
    }

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd);

    while (GetMessage (&msg,NULL,0,0))
    {
```

```
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
                LONG lParam)
{
    static HBRUSH hBrush[3] ;
    char szbuffer[10] ;
    HDC hdc ;
    POINT point ;
    short n,cxClient,cyClient,cyChar ;
    TEXTMETRIC tm ;

    switch (message)
    {
        case WM_CREATE :
            hBrush[0] = CreateSolidBrush (RGB (255,0,0)) ;
            hBrush[1] = CreateSolidBrush (RGB (0,255,0)) ;
            hBrush[2] = CreateSolidBrush (RGB (0,0,255)) ;
            return 0 ;
        case WM_SIZE :
            cxClient = LOWORD (lParam) ;
            cyClient = HIWORD (lParam) ;

            hdc = GetDC (hwnd) ;
            GetTextMetrics (hdc,&tm) ;
            cyChar = tm.tmHeight ;
            ReleaseDC (hwnd,hdc) ;

            MoveWindow (hwndRect,0,0,cxClient / 2,cyClient,TRUE) ;

            for (n = 0 ; n < 3 ; n++)
            {
                MoveWindow (hwndScrol[n],
                    (2 * n + 1) * cxClient / 14,2 * cyChar,
                    cxClient / 14,cyClient - 4 * cyChar,TRUE) ;

                MoveWindow (hwndLabel[n],
                    (4 * n + 1) * cxClient / 28,cyChar / 2,
                    cxClient / 7,cyChar,TRUE) ;

                MoveWindow (hwndValue[n],
                    (4 * n + 1) * cxClient / 28,cyClient - 3 * cyChar / 2,
                    cxClient / 7,cyChar,TRUE) ;
            }
            SetFocus (hwnd) ;
            return 0 ;
        case WM_SETFOCUS:
```

```
        SetFocus (hwndScrol[nFocus]) ;
        return 0 ;
    case WM_VSCROLL :
        n = GetWindowWord (HIWORD (lParam),GWW_ID) ;

        switch (wParam)
        {
            case SB_PAGEDOWN :
                color[n] += 15 ;
                                                        // fall through
            case SB_LINEDOWN :
                color[n] = min (255,color[n] + 1) ;
                break ;
            case SB_PAGEUP :
                color[n] -= 15 ;
                                                        // fall through
            case SB_LINEUP :
                color[n] = max (0,color[n] - 1) ;
                break ;
            case SB_TOP:
                color[n] = 0 ;
                break ;
            case SB_BOTTOM :
                color[n] = 255 ;
                break ;
            case SB_THUMBPOSITION :
            case SB_THUMBTRACK :
                color[n] = LOWORD (lParam) ;
              break ;
            default :
                break ;
        }
        SetScrollPos (hwndScrol[n],SB_CTL,color[n],TRUE) ;
        SetWindowText (hwndValue[n],itoa (color[n],szbuffer,10)) ;

        DeleteObject(
            SetClassWord (hwnd,GCW_HBRBACKGROUND,
                CreateSolidBrush (
                    RGB (color[0],color[1],color[2])))) ;

        InvalidateRect (hwnd,NULL,TRUE) ;
        return 0 ;
    case WM_CTLCOLOR:
        if (HIWORD (lParam) == CTLCOLOR_SCROLLBAR)
        {
            SetBkColor (wParam,GetSysColor (COLOR_CAPTIONTEXT)) ;
            SetTextColor (wParam,GetSysColor (COLOR_WINDOWFRAME)) ;

            n = GetWindowWord (LOWORD (lParam),GWW_ID) ;
            point.x = point.y = 0 ;
            ClientToScreen (hwnd,&point) ;
```

```
                UnrealizeObject (hBrush[n]) ;
                SetBrushOrg (wParam,point.x,point.y) ;
                return ((DWORD) hBrush[n]) ;
            }
            break ;
        case WM_DESTROY:
            DeleteObject (
                SetClassWord (hwnd,GCW_HBRBACKGROUND,
                    GetStockObject (WHITE_BRUSH))) ;
            for (n = 0 ; n < 3 ; DeleteObject (hBrush [n++])) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

long FAR PASCAL _export ScrollProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    short n = GetWindowWord (hwnd,GWW_ID) ;

    switch (message)
    {
        case WM_KEYDOWN:
            if (wParam == VK_TAB)
                SetFocus (hwndScrol[(n +
                    (GetKeyState (VK_SHIFT) < 0 ? 2 : 1)) % 3]) ;
            break ;

        case WM_SETFOCUS:
            nFocus = n ;
            break ;
    }
    return CallWindowProc (lpfnOldScr[n],hwnd,message,wParam,lParam) ;
}

; COLORS1.DEF module definition file

NAME COLORS1

DESCRIPTION 'Colors Using Scroll.Bars (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

COLORS1 利用了 10 个子窗口进行工作，该程序使用了十个子窗口控件：3 个滚动条、6 个静态文本窗口和 1 个静态矩形框。COLORS1 捕获 WM_COLOR 消息来给红、绿、蓝三个滚动条的内部部分着色。用鼠标器或者键盘都能滚动滚动条。在尝试颜色以为用户的 Windows

程序选择漂亮的色彩时，COLOR1 可作为开发工具使用。COLORS1 的单色版本如图 11.10 所示；显然，为了充分利用该程序的颜色处理功能，必须使用彩色显示器。

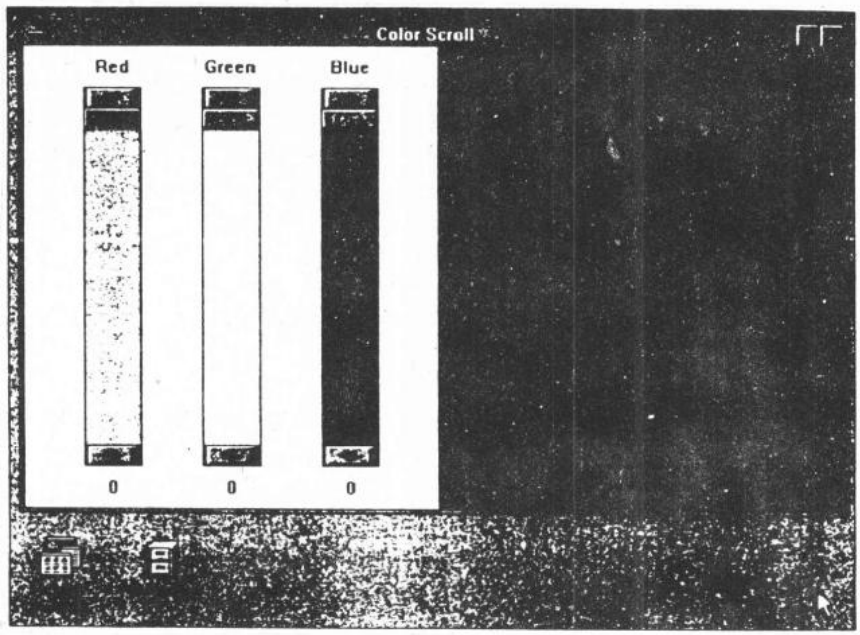


图 11.10　COLORS1 显示的单色版本

COLORS1 不处理 WM_PAINT 消息，而且它获得设备环境句柄只是为了确定字符的高度。COLORS1 中的大多数工作都由其子窗口去完成。

在客户区右半边显示的颜色实际上是窗口的背景颜色。SS_WHITERECT 风格的静态子窗口分块显示在客户区的左半边。三个滚动条是带 SBS_VERT 风格的子窗口控件，它们被定位在 SS_WHITERECT 子窗口的顶部。另外六个 SS_CENTER 风格的静态子窗口提供标签和颜色值。COLORS1 在 WinMain 函数中用 CreateWindow 建立它的普通的重叠式窗口和 10 个子窗口。SS_WHITERECT 和 SS_CENTER 静态窗口使用窗口类“static”，三个滚动条使用窗口类“scrollbar”。

CreateWindow 调用中的 x 位置、y 位置、宽度和高度参数最初置为 0，因为位置和大小都取决于客户区的尺寸，而它目前还未确定。COLORS1 的窗口过程在收到 WM_SIZE 消息时，它就用 MoveWindow 给 10 个子窗口重新确定大小，所以，只要重新规定了 COLORS1 窗口的大小，滚动条的尺寸就会按比例变化。

在 WndProc 窗口过程接收到 WM_VSCROLL 消息时，lParam 参数的高位字就是子窗口的句柄。用 GetWindowWord 能得到窗口的 ID 号：

```
n = GetWindowWord(HIWORD (lParam),GWW_ID);
```

对这三个滚动条，已将其 ID 号置为 0、1 和 2，所以 WndProc 能告诉是哪个滚动条在产生消息。

因为子窗口在建立时其句柄被保留在数组内，所以 WndProc 通过 SetScrollPos 调用可处理滚动条消息并给适当的滚动条设置新值：

SetScrollPos(hwndScrol[n],SB_CTL,color[n],TRUE);

WndProc 也改变滚动椅的底部的子窗口的文本：

SetWindowText(hwndValue[n],itoa(color[n],szbuffer,10));

11.6.3 自动键盘接口

滚动条控件也能处理键盘输入，但只能是在有输入焦点时。下面说明怎样将键盘光标键转变成滚动条消息：

光标键	滚动条消息的 wParam 值
Home	SB_TOP
End	SB_BOTTOM
Page Up	SB_PAGEUP
Page Down	SB_PAGEDOWN
Left or Up	SB_LINEUP
Right or Down	SB_LINEDOWN

实际上，SB_TOP 和 3B_BOTTOM 滚动条消息只能用键盘产生，在用鼠标器按动滚动条时如果想使该滚动条获得输入焦点，必须将 WS_TABSTOP 标识符包含到 CreateWindow 调用的窗口类参数内。当滚动条有了输入焦点时，在该滚动条的小方框内显示有一个闪动的灰色块。

为了给滚动条提供全面的键盘接口，还需要做另外一些工作。首先，WndProc 窗口过程必须使滚动条有输入焦点，它是通过处理 WM_SETFOCUS 消息来完成这一点的，该 WM_SETFOCUS 消息是当滚动条获得输入焦点时其父窗口接收到的。WindProc 给一个滚动条设置输入焦点：

SetFocus(hwndScrol[nFocus]);

但是，还需要一些借助键盘尤其是 Tab 键来从一个滚动条转换到另一个滚动条的方法。这比较困难，因为一旦某一个滚动条有了输入焦点，它就处理所有的键盘输入，但该滚动条只关心光标键，对 Tab 键不作响应。走出这一困境的方法是“窗口分组”技术。使用该技术可给 COLORS1 增加一种借助 Tab 键从一个滚动条跳到另一个滚动条的手段。

11.6.4 窗口分组

滚动条控件的窗口过程在 Windows 内的某处，但是，将 GWL_WNDPROC 标识符作为参数来调用 GetWindowLong，可以得到这个窗口过程的地址。另外，调用 SetWindowLong 可以给该滚动条设置一个新的窗口过程。这个技术叫作“窗口分组”，它很有用。它能让用户使用现存窗口过程，也能让用户在自己的程序中处理一些消息，并能让用户将其他所有消息传递给旧的窗口过程。

在 COLORS1 中处理初级滚动条消息的窗口过程叫做 ScrollProc，它在 COLORS1.C 清单中靠后排列。因为 ScrollProc 是 COLORS1 中的函数，Colors1 又被 Windows 调用，所以，它必须定义为 FARPASCAL _export。

首先，为确保 ScrollProc 访问恰当的数据段，COLORS1 必须用 MakeProcInstance 来给这个函数取得一个远程地址。

```
lpfnScrollProc = MakeProcInstance((FARPROC)
    ScrollProc,hInstance);
```

对三个滚动条中的每一个，COLORS1 用 GetWindowLong 来获得并保存现存滚动条窗口过程的地址：

```
lpfnOldScr[n] = (FARPROC) GetWindowLong
    (hwndScrol [n],GWL_WNDPROC);
```

其次，程序建立新的滚动条窗口过程：

```
SetWindowLong(hwndScrol [n],GWL_WNDPROC,
    (LONG) lpfnScrollProc);
```

现在，函数 ScrollProc 得到了 Windows 发送到 COLORS1 中三个滚动条（当然不是其他程序中的滚动条）的滚动条窗口过程的全部消息。ScrollProc 窗口过程在接收到 Tab 或者 Shift-Tab 键时，它就将输入焦点改变到下一个（或上一个）滚动条。

11.6.5 给背景着色

当 COLORS1 定义它的窗口类时，它为它的客户区背景定义一个实心的黑色刷子：

```
Wndclass. hbrBackgroud = CreateSolidBrush(0L);
```

在改变 COLORS1 的滚动条的设置时，程序必须建立一个新的刷子，并将该新刷子句柄放入窗口类结构中。如同用 GetWindowLong 和 SetWindowLong 能得到并设置滚动条窗口过程一样，用 GetClassWord 和 SetClassWord 能得到并设置这个刷子的句柄。

建立新刷子并将其句柄插入窗口类结构中，然后删除旧刷子：

```
DeleteObject(SetClassWord (hwnd,GCW_HBRBACKGROUND,
    CreateSolidBrush (RGB (color[0],color[1],color[2]))));
```

下一次 Windows 重新给窗口的背景着色时，Windows 将使用这个新刷子。为了强迫 Windows 抹掉背景，就使整个客户区无效：

```
InvalidateRect(hwnd,NULL,TRUE);
```

TRUE（非零）值作为第三个参数表示希望在重新着色之前删去背景。

InvalidateRect 使 Windows 在窗口过程的消息队列中排进一条 WM_PAINT 消息。由于 WM_PAINT 消息的级别较低，所以，如果还在用鼠标器或光标键移动滚动条的话，这条消息将不会被立即处理。如果想在颜色改变之后使该窗口立即变成最新的（当前的），就在 InvalidateRect 之后增加下面语句：

```
UpdateWindow(hwnd);
```

但这使得键盘和鼠标处理变慢。

COLORS1 中的 WndProc 函数不处理 WM_PAINT 消息，而是将其传递给 DefWindowProc。Windows 对 WM_PAINT 消息的缺省处理只涉及到调用 BeginPaint 和 EndPaint 使窗口生效。因为在 InvalidateRect 调用中已指定背景要被抹掉，所以 BeginPaint 调用使 Windows 发出一条 WM_ERASEBKGND(erase background)消息，WndProc 也将忽略这条消息。Windows 用在窗口类中指定的刷子将用户区域的背景抹去，这样就处理了这条消息。

一般地，Windows 用窗口类刷子将整个用户区抹去，这样就抹去了 10 个子窗口。但是，Windows 必须给所有子窗口发送 WM_PAINT 消息，以使它们能自己重新着色——这很麻

烦。避免这一问题的方法是，在用 CreateWindow 首先建立父窗口时，使用 WS_CLIPCHILDREN 风格值，这种风格能防止父窗口对它的子窗口着色。我们可以在 CreateWindow 中将 WM_CLIPCHILDREN 风格去掉，以看看 COLORS1 是怎样工作的。我们将看到很大的差别。

像所有的 GDI 对象一样，程序利用 CreateSolidBrush 建立的刷子在该程序结束时不会被 Windows 自动删去。我们习惯于在建立新的刷子之前删除旧刷子，但是当程序将结束时，窗口类中的最后一个刷子也应该被放弃。所以，在处理 WM_DESTROY 消息的过程中，再一次调用 DeleteObject：

```
DeleteObject (SetClassWord (hwnd,GCW_HBRBACKGROUND
    GetStockObject (WHITE_BRUSH)));
```

11.6.6 处理多个实例

一般情况下，当用户装入程序的多个实例时，Windows 就重复使用同一个窗口类。只有在前面的实例为 NULL(空)时，窗口类才被注册：

```
if(! hPrevInstance)
{
    wndclass.style = CS_HREDRAW|CS_VERDRAW;
    [more programm lines]
```

但是 COLORS1 不能做这一工作，因为背景色是在窗口类中规定的。如果 COLORS1 的所有实例都使用同一个窗口类，那么每个实例都将使用(或改变)同一背景色。完全避免上述问题的办法是只允许 COLORS1 的一个实例运行：

```
if(hPrevInstance)
    return FALSE;
```

11.6.7 COLORS1 作为图标

当使 COLORS1 成为一个图标时，该图标的整个表面——而不只是右半边——都呈现父窗口的背景颜色，COLORS1 也似乎没有任何独立的图标逻辑。

用户也将注意到在窗口类中 COLORS1 规定了一个 NULL 图标：

```
wndclass.hIcon = NULL;
```

这表明 COLORS1 对给它的图标着色会作出响应。整个图标显示为背景色，这是因为，当一个程序变成图标时 Windows 隐藏了子窗口，这样，着色的背景就被完全暴露出来了。

11.7 编辑类

编辑控件是一种很重要的控件，几乎在所有的应用程序中都能找到它。它能使用户键入文本。大多数字处理器都有这种控件，但在每次加载文件或存储新生成的图形时都要输入文件名。在编辑控件中，用户可以键入并修改文本。函数 SetWindowText 也能用以在控件中设置文本。

图 11.11 是编辑控件的典型画面。

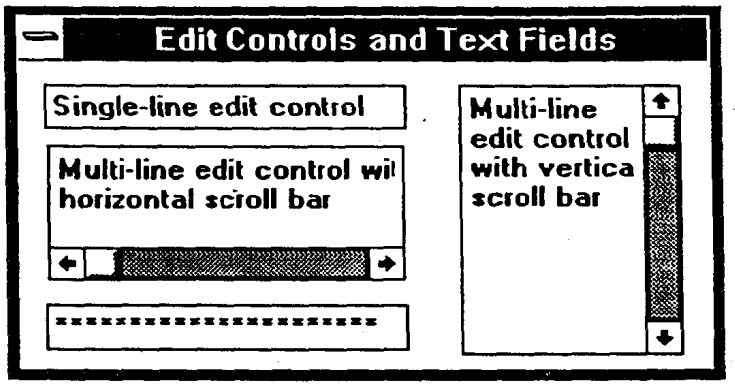


图 11.11

与编辑控件有关的消息如下：

EDIT	含义
ES_AUTOHSCROLL	如果文本长于编辑控件则自动水平滚动
ES_AUTOVSCROLL	如果文本长于编辑控件则自动上下滚动
ES_CENTER	在多行编辑控件中文本居中
ES_LEFT	文本左对齐
ES_LOWERCASE	将输入字母转换成小写
ES_MULTILINE	标识一个多行编辑控件
ES_NOHIDESEL	当焦点移动时，选择不像通常那样移动
ES_OEMCONVERT	将输入文本用 OEM 字符集转换或转换成 OEM 字符
ES_PASSWORD	对所有键入字符显示为“*”
ES_RIGHT	文本右对齐
ES_UPPERCASE	将输入字符转换成大写

只要编辑控件接收到焦点，在它的最左端就会出现一个加亮显示的垂直线，该点标识着当前输入位置。EDIT 类窗口例程功能较强，它支持键盘输入和鼠标输入。

例如，光标可通过按鼠标左按钮进行定位，文字可通过双击鼠标按钮进行选择。

一个缺省的编辑控件包含一行，如果类型 ES_AUTOHSCROLL 被设置，且文本多于矩形框所能容纳的范围，则文本会自动移动而使输入位置始终可见。

用户还可用箭头键水平移动文本。对较大的文本输入，使用参数 ES_MULTILINE 比较方便；当与 ES_AUTOVSCROLL 联合使用时，该类型允许在文本多于可见输入区时，多行输入的文本可以上下滚动。

若用户给编辑控件配备滚动条，则还可在文本中更方便地定位。像通常窗口一样，这可以由标志 WS_VSCROLL 和 WS_HSCROLL 实现。

在某些方面，编辑类是最简单的预定义的窗口类；在另一些方面，它又是最复杂的。在用类名“edit”建立子窗口时，也就在 CreateWindow 调用中的 x 位置、y 位置、宽度和高度这些参数的基础上定义了一个矩形。当子窗口控件有输入焦点时，可以打印文本、移动光标、用鼠标器或者 Shift 键与一个光标键来选取部分文本、按 Ctrl-X 或 Shift-Del 来删除并将所选文本拷

贝到剪贴板、按 Ctrl-V 或 Shift-Ins 键插入剪贴板上的文本。

编辑控件的最简单的应用之一是对单行输入区域。例如，Windows PIF Editor 程序在它的主窗口中以这种方式使用编辑控件。但是编辑控件并不限于单行。例如，Windows Notepad 程序就使用了多行编辑控件。Notepad 程序的文件出奇地小——小于 32KB。编辑的大多数操作根本不在 Notepad 中，而在 Windows 内的编辑控件逻辑中。

11.7.1 POPPAD1 程序

为了使读者对编辑控件的功能有一个认识，下面写一个叫做 POPPAD1 的“Notepad clone”程序。POPPAD1 在下面说明。

```
# POPPAD1.MAK make file

poppad1.exe : poppad1.obj poppad1.def
     $(WINLINK) poppad1,poppad1,NUL, $(WINLIB),poppad1.def
     rc -t poppad1.exe

poppad1.obj : poppad1.c
     $(WINCC) poppad1.c

/*-------------------------------------------------------
POPPAD1.C -- Popup Editor using child window edit box
-------------------------------------------------------*/

#include <windows.h>

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG);

char szAppName[] = "PopPad1" ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     HWND hwnd ;
     MSG msg;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = NULL ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
```

```
    }

    hwnd = CreateWindow (szAppName,szAppName,
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        GetSystemMetrics (SM_CXSCREEN) / 2,
        GetSystemMetrics (SM_CYSCREEN) / 2,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd);

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    static HWND hwndEdit ;

    switch (message)
    {
        case WM_CREATE:
            hwndEdit = CreateWindow ("edit",NULL,
                WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
                WS_BORDER | ES_LEFT | ES_MULTILINE |
                ES_AUTOHSCROLL | ES_AUTOVSCROLL,
                0,0,0,0,
                hwnd,1,
                ((LPCREATESTRUCT) lParam) -> hInstance,NULL) ;
            return 0 ;
        case WM_SETFOCUS:
            SetFocus (hwndEdit) ;
            return 0 ;
        case WM_SIZE:
            MoveWindow (hwndEdit,0,0,LOWORD (lParam),
                HIWORD (lParam),TRUE) ;
            return 0 ;
        case WM_COMMAND :
            if (wParam == 1 && HIWORD (lParam) == EN_ERRSPACE)
                MessageBox (hwnd,"Edit control out of space.",
                    szAppName,MB_OK | MB_ICONSTOP) ;
            return 0 ;
        case WM_DESTROY :
            PostQuitMessage (0) ;
            return 0 ;
```

```
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; POPPAD1. DEF module definition file

NAME POPPAD1

DESCRIPTION 'Popup Editor Version 1 (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

POPPAD1 是一个多行编辑器(只是还没有文件输入/输出),其源代码要少于 100 行。如上所述,POPPAD1 本身并没有太多功能,预定义的编辑控件完成较多的操作。以这种形式,该程序能使用户弄清楚没有其他程序帮助时编辑控件能做些什么。

11.7.2 编辑类风格

如前面所提到的,在 CreateWindow 调用中将"edit"作为窗口类建立了一个编辑控件,窗口风格是 WS_CHILD 加上几个选项。如同在静态子窗口控件中一样,编辑控件中的文本可以按左对齐、右对齐或居中。欲指定这种格式,可使用窗口风格 ES_LEFT、ES_RIGHT 和 ES_CENTER。

缺省操作时,编辑控件是单行的。用 ES_MULTILINE 窗口风格可以建立多行编辑控件。对一个单行编辑控件,一般用户只可以在编辑控件矩形的末端输入正文。要建立一个自动水平滚动的编辑控件,可以采用风格 ES_AUTOHSCROLL。对一个多行编辑控件,文本会整字回行,除非使用 ES_AUTOHSCROLL 风格,这种情况下必须按 Enter 键来开始新的一行。用户也可以用风格 ES_AUTOHSCROLL 尝试一下。可以用风格 ES_AUTOVSCROLL 来将垂直滚动条包括进多行编辑控件中。

当在多行编辑控件中包括进这些滚动风格后,用户也许还想给编辑控件增加滚动条,这样做可以达到如下目的:对非子窗口,使用同一窗口风格标识符 WS_HSCROLL 和 WS_VSCROLL。

缺省操作时,编辑控件没有边界,用风格 WS_BORDER 可以增加之。

在编辑控件中选择文本时,Windows 将选择的文本显示成反白图像。但在该编辑控件失去输入焦点时,被选择的文本不再以高亮度显示。如果在编辑控制没有输入焦点的情况下想使被选择的文本以高亮度显示,采用风格 ES_NOHIDESEL 即可。

在 POPAD1 建立它的编辑控件时,在 CreateWindow 调用中按如下形式给出风格:

```
WS_CHILD|WS_VISIBLE|WS_HSCROLL|
    WS_VSCROLL|WS_BORDER|ES_LEFT|
    ES_MULTLINE|ES_AUTOHSCROLL|ES_AUTOVSCROLL
```

在 POPPAD1 中,编辑控件的大小是后面当 WndProc 接收到 WM_SIZE 消息时通过调用 MoveWindow 来定义的。编辑控件的尺寸被简单地设置为主窗口的尺寸:

```
MoveWindow (hwndEdit,0,0,LOWORD (lParam),
            HIWORD (lParam),TRUE);
```

对于单行编辑控件，其高度必须可容纳一个字符。如果编辑控件有边界(大多数都有)，就使用一个字符高度的 1.5 倍(包括外部引线)。

11.7.3 编辑控件通知

编辑控件给父窗口过程发送 WM_COMMAND 消息。对按钮控件来说，wParam 和 lParam 变量的含义是相同的：

参数	描述
wParam	子窗口 ID
LOWORD(lParam)	子窗口句柄
HIWORD(lParam)	通知码

下面是通知代码：

EN_SETFOCUS	编辑控件已经获得输入焦点
EN_KILLFOCUS	编辑控件已经失去了输入焦点
EN_CHANGE	编辑控件的内容将改变
EN_UPDATE	编辑控件的内容已经改变
EN_ERRSPACE	编辑控件运行已超出空间
EN_MAXTEXT	编辑控件在插入时运行超出空间
EN_HSCROLL	编辑控件的水平滚动条已被按动
EN_VSCROLL	编辑控件的垂直滚动条已被按动

POPPAD1 只捕获 EN_ERRSPACE 通知码并显示一个消息框。

编辑控件将文本保存在父窗口过程的局部堆中，编辑控件的内容限制在 32KB 之内。用户会注意到，POPPAD1 只为它在模块定义文件中的局部堆保留了 1KB 的空间。如果编辑控件需要更多的空间，Windows 将会扩大程序的局部堆。

11.7.4 使用编辑控件

如果在用户主窗口的表面使用几个单行编辑控件，则需要将窗口分类以把输入焦点从一个控件移到另一个控件，像 COLORS1 所做的那样。这种移动是通过截获 Tab 键和 Shift—Tab 键来完成的(窗口分类的另一个例子在本章后面的 HEAD 程序中说明)。怎样操作 Enter 键呢？它可以像 Tab 键那样使用，也可以用作给程序的信号，表示所有编辑区都准备好了。

如果想将文本插入编辑区，用 SetWindowText 可以做到。将文本从编辑控件中取出涉及到 GetWindowTextLength 和 GetWindowText。我们将在 POPPAD1 程序的修订版本中看到这些操作的实例。

11.7.5 给编辑控件发送消息

因为用 SendMessage 发送给编辑控件的消息很多，并且其中的几个还将在后面的 POPPAD1 的修订版本中用到，所以这里不涉及所有的用 SendMessage 发送给编辑控件的消息，只概要地说明一下。

这些消息允许用户剪取、拷贝或者清除当前选择　用户用鼠标器或者 Shift 键加光标控件键来选择文本并进行上面的操作，这样，在编辑控件中被选择的文本将以高亮度显示：

```
SendMessage (hwndEdit,WM_CUT,0,0L);
SendMessage (hwndEdit,WM_COPY,0,0L);
SendMessage (hwndEdit,WM_CLEAR,0,0L);
```

WM_CUT 将当前选择从编辑控件中移走，并将其发送到剪贴板；WM_COPY 将选择的文本拷贝到剪贴板上并保持编辑控件中的内容完好无损；WM_CLEAR 将选择的内容从编辑控件中删除，但不向剪贴板传送。

也可以将剪贴板上的文本插入到编辑控件中的光标位置。

```
SendMessage(hwndEdit,WM_PASTE,0,0L);
```

如下操作可以得到当前选择的起始光标位置和末尾光标位置：

```
lSelect = SendMessage (hwndEdit,EM_GETSEL,0,0L);
```

lSelect 的低位字含有起始位置，高位字含有末尾位置加上 1。

用如下方法选择文本：

```
SendMessaeg (hwndEdit,EM_SETSEL,0;
    MAKELONG (wBegin,wEnd));
```

也可以用另外的文本置换当前的选择：

```
SendMessage (hwndEdit,EM_REPLACESEL,0,
    (LONG) lpszString);
```

对多行编辑控件，可以获得行数：

```
nCount = SendMessage(hwndEdit,EM_GETLINECOUNT,0,0L);
```

对任何特定的行，可以得到离编辑缓冲区文本的开头的偏移量：

```
nOffset = SendMessage (hwndEdit,EM_LINEINDEX,wLine,0L);
```

行数从 0 开始计算，因为 wLine 为－1 时返回包含光标的行的偏移量。由下面的操作可以得到行的长度：

```
nOffSet = SendMessage (hwndEdit,EM_LINELENGTH,wLine,0L);
```

下面的操作可以将行自身拷入缓冲区：

```
nLength = SendMessage (hwndEdit,EM_GETLINE,wLine,lpszBuffer);
```

11.8　列表框

列表框允许用户从列表中选出一个输入；可使用鼠标按钮进行选择或用上下箭头键来选择。当前选择被加亮显示。如果入口项多于可供显示的行，则列表框的内容自动上下滚动。通常，双击鼠标按钮即可选择一个串，同时还启动一个动作。例如，将字符移入缓冲区中。

图 11.12 是一个典型的列表框。

与列表框有关的消息如下：

LISTBOX	含义
LBS_EXTENDEDSEL	多重入口可被同时选中
LBS_HASSTRINGS	指定一个其入口由字符串组成的自绘列表框

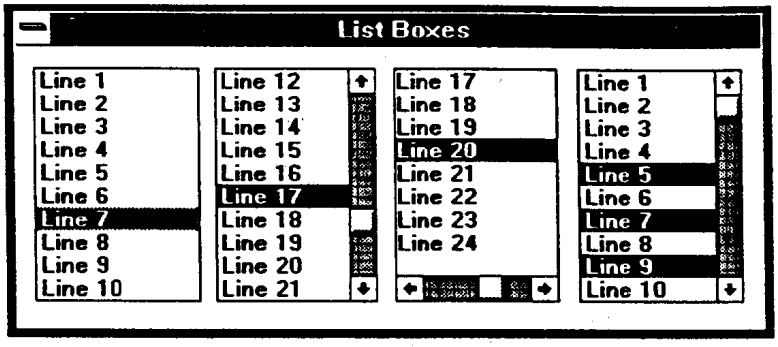


图 11.12

LBS_MULTICOLOMN	指定一个多列的列表框
LBS_MULTIPLESEL	每次按鼠标按钮或双击鼠标按钮时切换选择与放弃状态
LBS_NOINTEGRALHEIGHT	列表框大小在创建时与指定的正好相符
LBS_NOREDRAW	当改变时,入口显示不更新
LBS_NOTIFY	当有输入时,父窗口收到一条消息
LBS_OWNERDRAWFIXED	由程序员显示列表框的内容
LBS_OWNERDRAWVARIABLE	除入口高度可变外,其余同上
LBS_SORT	入口按字母排序
LBS_STANDARD	LBS_SORT 和 LBS_NOTIFY 的组合
LBS_USETABSTOPS	识别制表位
LBS_WANTKEYBOARDINPUT	当列表框有输入焦点时,每按一次键就发送一条消息

列表框是文本串的集合,这些文本串是在一个矩形中可以滚动显示的栏状列表。程序通过向列表框窗口过程发送消息,可以在列表中增加或者去掉串。当列表框中的某项被选择时,列表框控件就向其父窗口发送 WM_COMMAND 消息,父窗口也就可确定选择的是哪一项。

从 File 菜单中选择 Open 即可调出对话框,列表框在这种对话框中最为常用。列表框能显示当前目录中的文件,也能显示其他子目录和磁盘驱动器。列表框也用在 WRITE 程序中来选择字模。一个列表框可以是单项选择,也可以是多项选择,后者允许用户从列表框中选择多个项。当列表框有输入焦点时,列表框中项的周围显示有点划线,光标并不说明被选择的项在列表框中。被选择的项用高亮度显示,而且是反白显示。

在单项选择的列表框中,用户按 Spacebar(空格)键就可以选择光标所在位置的项。箭头键移动光标和当前选择提示,并能滚动列表框的内容。Page Up 和 Page Down 键也能滚动列表框,但它是移动的是光标而不是选择提示。按一个字母键能将光标和以此字母开头的选项移到第一项(或者第二项)。也可以用鼠标器在要选择的项上点一下或者两下来选择之。

在多项选择列表框中,空格键可切换光标所在位置的项的选择状态(如果该项已被选择,就取消选择)。如同在单项选择列表框中一样,箭头键解除前面选择过的项,并移动光标

和选择。但是,Ctrl 键和箭头键能在移动光标的同时而不移动选择,Shift 键和箭头键能展开一个选项。

在多项选择列表框中,用鼠标器按动一次或两次就能解除以前所有的选择,而选择被按动的项。但是,如果在用鼠标器按动某一项的同时也按下 Shift 键,这样只能切换该项的选择状态,而不改变任何其他项的选择状态。

11.8.1 列表框风格

用 CreateWindow 建立列表框子窗口时,要将"listbox"作为窗口类,将 WS_CHILD 作为窗口风格。但是,这个缺省列表框风格不向父窗口发送 WM_COMMAND 消息,其含义是程序必须向列表框询问其中的项的选择状态(借助发送给列表框控件的消息)。所以,列表框控件通常都包括列表框风格标识符 LBS_NOTIFY,它允许父窗口接收来自列表框的 WM_COMMAND 消息。如果希望列表框控件对列表框中的项进行排序,可以使用另一种常用风格 LBS_SORT。

缺省操作时,列表框是单项选择的。多项选择的列表框相当少。如果想建立一个多项选择列表框,可使用风格 LBS_MULTIPLESEL。

通常,当给滚动框列表增加新项时,列表框会修改它自身。将风格 LBS_NOREDRAW 包含进去就能防止这种现象。但是用户也许不想使用这种风格,这时可使用 WM_SETREDRAW 消息来暂时防止列表框控件的重新着色;WM_SETREDRAW 消息在稍后讲述。

缺省操作时,列表框窗口过程只显示列表项,它的周围没有任何边界。用窗口风格标识符 WS_BORDER 可加上边界。另外,用窗口风格标识符 WS_VSCROLL 可增加垂直滚动条,以便用鼠标器滚动列表项。

WINDOWS.H 定义了一个列表框风格,叫作 LBS_STANDARD,它包含了最常用的风格,其定义如下:

```
(LBS_NOTIFY|LBS_SORT|WS_VSCROLL|WS_BORDER)
```

也可以采用 WS_SIZEBOX 和 WS_CAPTION 标识符,但是,这两个标识符允许用户重新定义列表框的大小,也允许在列表框父窗口的用户区周围移动列表框。

列表框的宽度应该能容纳最长串的宽度加上滚动条的宽度。用 GetSystemMetrics (SM_CXVSCROLL)能取得垂直滚动条的宽度。用一个字符的高度乘以想要在视口中显示的项数就能计算出列表框的高度。

11.8.2 将串放入列表框

建立列表框之后的下一步是将文本串放入其中,其方法是:调用 SendMessage 给列表框窗口过程发送消息。文本串通常以从 0 开始计数的下标数引用,其中 0 对应最上面的页。在下面的例子中,hwndList 是子窗口列表框控件的句柄,wIndex 是下标值。

在用 SendMessage 传递文本串的情况下,lParam 参数是指向空终结串的远程指针。为了避免编译错误,将这个远程指针转换成 LPSTR(long pointer to string),再转换成 LONG。

在大多例子中,当窗口过程超出可用内存空间而存储列表框的内容时,SendMessage 能返回 LB_ERRSPACE(定义为-2)。如果因其他原因出错,SendMessage 就返回 LB_ERR(-1)。如

果操作成功，SendMessage 返回 LB_OKAY(0)。对非零值可测试 SendMessage 来判明这两种错误。列表框为列表框内容分配总内存(在程序数据段之外)。

如果采用 LBS_SORT 风格(或者在列表框中按想要呈现的顺序排列字符串)，那么填充列表框的最简单的方法是借助 LB_ADDSTRING 消息：

```
SendMessage(hwndList,LB_ADDSTRING,0,(LONG)(LPSTR)szString);
```

如果没有采用 LBS_SORT，就可以用 LB_INSERTSTRING 指定一个下标值来将字符串插入到列表框中：

```
SendMessage(hwndList,LB_INSERTSTRING,wIndex,(LING)(LPSTR)szString);
```

例如，假如 wIndex 等于 4，szString 就变成下标值为 4 的串——从顶头开始算起的第五个串，因为是从 0 开始计数的。该点后面的所有串都向下推。下标值为－1 时就将串增加在最后。可对风格为 LBS_SORT 的列表框使用 LB_INSERTSTRING，但这个列表框的内容不能重新排序(也可以用 LB_DIR 消息将串插入到列表框中)。

用 LB_DELETESTRING 消息指定下标值，借此可以从列表框中删除串：

```
SendMessage(hwndList,LB_DELETESTRING,wIndex,0L);
```

用 LB_RESETCONTENT 可清除列表框的内容：

```
SendMessage(hwndList,LB_RESETCONTENT,0,0L);
```

当在列表框中增加或删除串时，列表框窗户过程就更新显示。如果有许多串需要增加或删除，那么用户也许希望临时阻止这一动作，其方法是关掉控件的重画标志：

```
SendMessage(hwndList,WM_SETREDRAW,FALSE,0L);
```

在完成插入或删除之后，可打开重画标志：

```
SendMessage(hwndList,WM_SETREDRAW,TRUE,0L);
```

用 LBS_NOREDRAW 风格建立的列表框开始时其重画标志是关闭的。

11.8.3 选择和提取条目

SendMessage 完成了下面所描叙的任务之后，通常要返回一个值。如果出错，这个值将被赋给 LB_ERR(定义为－1)。注意，从 SendMessage 返回的值通常是有符号 LONG 型整数，而不是无符号整数(WORD)，所以需要做一些转换。

在列表框中放入一些条目之后，可以弄清目录中有多少项：

```
nCount = (WORD)SendMessage(hwndList,LB_GETCOUNT,0,0L);
```

某些其他调用对单项选择列表框和多项选择列表框是不同的，下面我们先看看单项选择列表框的情形。

通常，应让用户能在列表框中选择条目。如果想高亮度显示一个缺省选择，则可执行：

```
SendMessage(hwndList,LB_SETCURSEL,nIndex,0L);
```

在该调用中给 wParam 置－1 就可取消所选项。

也可以根据项的第一个字母来挑选项：

```
nIndex = (WORD)SendMessage(hwndList,LB_SELECTSTRING,wIndex,
    (LONG) (LPSTR) szSearchString);
```

在 SendMessage 调用中将 wIndex 作为 wParam 参数时，wIndex 是下标。可根据它搜索其开头字母与 szSearchString 相匹配的项。wIndex 的值等于－1 时从开头开始搜索，SendMessage

返回被选项的下标。如果没有开头字母与 szSearchString 相匹配时，就返回 LB_ERR。

当得到来自列表框的 WM_COMMAND 消息时（或在任何其他时候），用 LB_GETCURSEL 能确定当前选择项的下标：

```
nIndex = (WORD)SendMessage(hwndList,LB_GETCURSEL,0,0L);
```

如果没有项被选中时，从调用中返回的 nIndex 值就是 LB_ERR。

以下操作可确定列表框中串的长度：

```
nLength = (WORD)SendMessage(hwndList,LB_GETTEXTLEN,nIndex,0L);
```

欲将某项拷入文本缓冲区：

```
nLength = (WORD)SendMessage(hwndList,LB_GETTEXT,nIndex,
    (LONG)(LPSTR) szBuffer);
```

在这两种情况下，从调用中返回的 nLength 值是字符串的长度。对以 NULL 终结的串的长度来说，szBuffer 数组必须足够大。用户也许想用 LB_GETTEXTLEN 先分配一些局部内存来存放串。

对于多选择的列表框，就不能够使用 LB_SETCURSEL，LB_GETCURSEL 或者是 LB_SELECTSTRING。但可改用 LB_SETSEL 来设置某特定项的选择状态，而不影响有可能被选择的其他项：

```
SendMessage(hwndList,LB_SETSEL,wParam,(LONG)wIndex);
```

wParam 参数不为零时，选择并以高亮度显示某项；等于 0 时即取消选择。lParam 等于－1 时，选择所有项或取消所有被选择的项。也可以如下确定某特定项的选择状态：

```
wSelect = (WORD) SendMessage(hwndList,LB_GETSEL,wIndex,0L);
```

其中，如果由 wIndex 指定的项被选择时，wSelect 被置为非 0；未被选择时置为 0。

11.8.4 接收来自列表框的消息

当用户用鼠标器按动列表框时，列表框就接受输入焦点。下面的操作可使父窗口将输入焦点转交列表框控件：

```
SetFocus (hwndList);
```

当列表框有输入焦点时，光标移动键、字母键和空格键都可用来在该列表框中选择某项。

列表框控件向它的父窗口发送 WM_COMMAND 消息。对按钮和编辑控件来说，wParam 和 lParam 变量的含义是相同的：

wParam	子窗口 ID
LOWORD(lParam)	子窗口句柄
HIWORD(lParam)	通知码

通知码及其值如下：

LBN_ERRSPACE	－2
LBN_SELCHANGE	1
LBN_DBLCLK	2
LBN_SELCANCEL	3
LBN_SETFOCUS	4

LBN_KILLFOCUS　　5

只有在列表框窗口风格包括 LBS_NOTIFY 时，列表框控件才向父窗口发送 LBN_SELCHANGE 和 LBN_DBLCLK 码。

LBN_ERRSPACE 码表明列表框已运行超出空间。LBN_SELCHANGE 码表明当前选择已被改变；这些消息出现在下列情况下：用户在列表框中移动高亮度显示项时、用空格键切换选择状态时或者用鼠标器按动某项时。LBN_DBLCLK 码说明某项已被鼠标器两次按动（对 LBN_SELCHANGE 和 LBN_DBLCLK，通知码的值取决于鼠标器按动的次数）。

根据应用的需要，也许要使用 LBN_SELCHANGE，也许要使用 LBN_DBLCLK，也许二者都要使用。程序会接收到许多 LBN_SELCHANGE 消息，但 LBN_DBLCLK 消息只有当用户两次按动鼠标器时才会出现。如果程序使用两次按动，那么用户需要提供一个复制 LBN_DBLCLK 的键盘接口。

11.8.5 文件列表

LB_DIR 是最有效的列表框消息。它用文件目录表填充列表框，且可选择性地包括子目录和有效的磁盘驱动器。

```
SemdMessage(hwndList,LB_DIR,wAttr,(LONG)(LPSTR)lpszFileSpec);
```

使用文件属性码

wAttr 参数是文件属性代码。在进行 MS－DOS 功能调用时该最末有效的字节是普通的文件属性代码：

wAttr	属性
0×0000	普通文件
0×0001	只读文件
0×0002	隐含文件
0×0004	系统文件
0×0010	子目录文件
0×0020	有归档位集的文件

高位字节提供一些对所要求的项的附加控制：

wAttr	Option（选择）
0×4000	包括驱动器的字母
0×8000	互斥搜索

当 LB_DIR 消息的 wAttr 值是 0×0000 时，列表框列出普通文件、只读文件和带归档位集的文件；这与利用 MS_DOS 功能调用来寻找文件的方法一致。当 LB_DIR 的值是 0×0010 时，列表框除列出上述文件之外，还列出子目录，这相当于用 Directory 命令或 Windows 的 File Manager（文件管理器）所显示的内容。当 LB_DIR 的值是 0×4010 时，它除包括其值为 0×0010 时所显示的内容之外，还包括了所有有效的驱动器。对很多 Windows 程序来说，这是对话框中的列表，而对话框是通过在程序的 File 菜单中选择 Open 来调用的。要列出所有的文件、子目录和驱动器，将 wAttr 的值置为 0×4037 即可。

将 wAttr 的最高位置位就可列出带标志的文件，而不包括普通文件。例如，对 Windows

的文件备份程序，也许用户只想列出自最后一次备份后修改过的文件，这样的文件有归档位集，所以 wAttr 的值为 0×8020。wAttr 的值为 0×8010 时只列出子目录，为 0×C000 时只列出有效的磁盘驱动器，为 0×C010 时列出子目录和驱动器，但不列出文件。

文件排序

lParam 参数是指向文件说明串如“*.*”的远程指针，这个文件说明串不影响列表框中的子目录。

用户也许希望给列有文件清单的列表框使用 LBS_SORT 消息。列表框首先列出符合文件说明的文件，再(有选择地)用下面的形式列出磁盘驱动器：

[—A—]

也可有选择地列出子目录名。第一个列出的子目录为如下形式：

[..]

这个“双点”子目录条目允许用户向根目录返回一级(在根目录中列出文件名时这个“双点”条目不会出现)。最后，具体的子目录名采用下面的形式：

[SUBDIR]

如果不用 LBS_SORT，文件名和子目录名就被混在一起，而且驱动器字母出现在列表框的底部。

11.9 COMBOBOX 类

组合框是编辑控件和列表框的组合。此类首先在 Windows 3.0 中实现，且有三种变体。当风格 CBS_DROPDOWN 被设置时，控件首先表现为一个包含当前选择的矩形，并具有编辑控件的特性。

组合框的典型画面如图 11.13 所示。

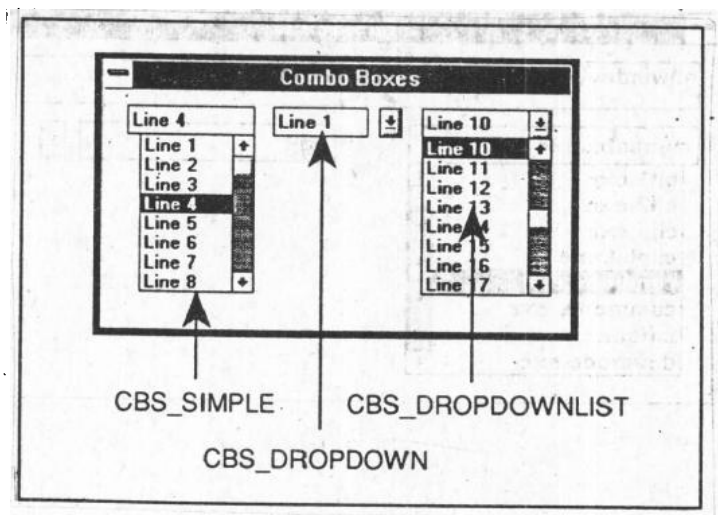


图 11.13

与组合框有关的消息如下：

COMBOBOX	含义
CBX_AUTOHSCROLL	文本如果长于编辑控件则自动滚动
CBS_DROPDOWN	列表框不显示，直到用户按箭头图标
CBS_DROPDOWNLIST	编辑控件作为静态域实现
CBS_HASSTRINGS	自绘组和合框的风格
CBS_OEMCONVERT	文本转换成或来自于 OEM 字符集
CBS_OWNERDRAWFIXED	程序员负责绘制列表框的内容
CBS_OWNERDRAWVARIABLE	除入口高度可变外，其余同上
CBS_SIMPLE	列表框始终显示
CBS_SORT	入口项自动排序

如果用户还想做其他选择，可把箭头图标移到选择域的右边，且按下鼠标按钮打开列表框。列表框显示了其他选择。再按一下箭头可去掉列表框。

在列表框中选择一个入口项会自动地把该项送到编辑控件中，并去掉列表框。用户还可在编辑控件中输入一个串，它在列表框中被加亮显示。

风格参数 CBS_SIMPLE 产生一个其列表框始终显示的组合框，因此没有打开列表框的箭头图标。带风格参数 CBS_DROPDOWNLIST 的组合框很像 CBS_DROPDOWN 组合框。

然而，随着用户的使用，区别会变得明显起来。最上面的区域不是编辑控件，而是静态域。用户可以从列表框中选择一个输入，而无需键入一个字符串。选中的输入作为静态文本，且不能修改。

组合框的例子

组合框通常用来选择和显示驱动器、路径和文件。下面的应用程序包括两个组合框。驱动器可通过其他信息得到，当前路径显示在静态域中。

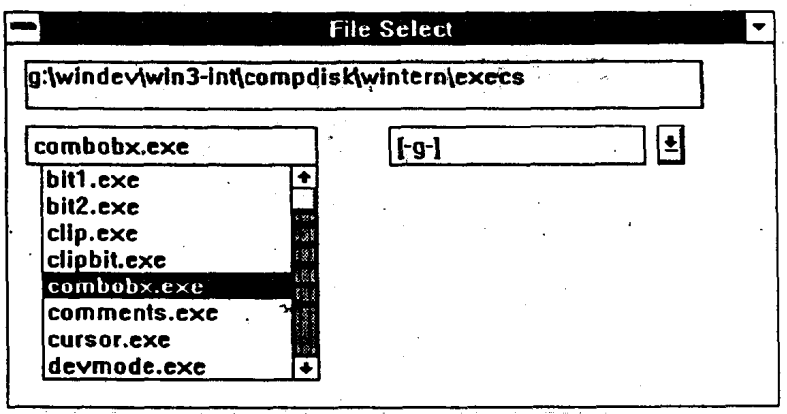


图 11.14

新函数	说明
DlgDirListComboBox	组合框的内容被更新

DlgDirSelectComboDox	给出选中的组合框入口项

新消息	说明
CB_DIR	用文件名填充的组合框
CB_RESETCONTENT	清除所有入口项,释放自由内存

源代码:COMBOBX.C

```
/* * COMBOBX.C * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates combo box implementation in Windows * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "combobx.h"                          // Include COMBOBX.H header file

#if _MSC_VER >= 700                           // Running Microsoft C 7.00 or
    #pragma warning (disable:4028)            // higher? Disable warning 4028
#endif

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,
    LPSTR lpszCmdLine,int nCmdShow)
{
    MSG msg;                                  // Message handle
    HWND hWnd;                                // Window handle
    int nError;                               // Error integer

    strcpy(szAppName,"Combobx");              // Application name
    hInst = hInstance;                        // Instance handle

    if (! hPrevInstance)
    {                                         // Register first instance

        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                              // If registration fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;                    // display error
        }
    }

    /* * Specify appearance of application's main window * * * * * * * * */

    hWnd = CreateWindow(szAppName,
        "File Select",
        WS_OVERLAPPED | WS_SYSMENU |
        WS_MINIMIZEBOX | WS_BORDER,
        CW_USEDEFAULT,0,
        2 * GetSystemMetrics(SM_CXSCREEN)/3,
```

```
        GetSystemMetrics(SM_CYSCREEN)/2,
        NULL,NULL,hInstance,NULL);

    if(hWnd == NULL)                                    // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;                          // display error
    }
    ShowWindow( hWnd,nCmdShow );                        // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while (GetMessage(&msg,NULL,0,0))                   // Message reading
    {
        TranslateMessage(&msg);                         // Message translation
        DispatchMessage(&msg);                          // Send message to Windows
    }
    return (int)msg. wParam;
}

int nCwRegisterClasses(HANDLE hInstance)
{
/* * Specify window class * * *
```

```
static HWND hWndStatic,hWndCombo,hWndList;
RECT rect;
static char szPathSpec[150];
BOOL bCurrentDir;
int i;

switch (message)
{
    case WM_CREATE:
        GetClientRect(hWnd,&rect);                          // Get client rectangle
        hWndStatic = CreateWindow("static",                 // Static window class name
            "",                                             // Window caption (none)
            WS_CHILD |                                      // Child window
            WS_VISIBLE |                                    // Visible window
            WS_BORDER,                                      // Include border
            10,10,                                          // Upper-left x and y pos.
            rect.right-50,30,                               // Initial x and y sizes
            hWnd,                                           // Window handle
            IDC_STATIC,                                     // Menu type used
            hInst,                                          // Application instance
            NULL);                                          // No creation parameters

        hWndCombo = CreateWindow("combobox",                // Combo box window
            NULL,WS_CHILD | WS_VISIBLE |
            WS_VSCROLL | CBS_SIMPLE | CBS_SORT,
            10,50,rect.right/2-50,rect.bottom-60,
            hWnd,IDC_COMBO,hInst,NULL);

        SetWindowText(hWndStatic,"Select the current directory");
        SendMessage(hWndCombo,CB_RESETCONTENT,0,0L);
        SendMessage(hWndCombo,CB_DIR,(WORD)(0x0|0x10),(LONG)(LPSTR)"*.*");
        hWndList = CreateWindow("combobox",NULL,
            WS_CHILD | WS_VISIBLE | CBS_DROPDOWN | CBS_SORT,
            rect.right/2,50,
            rect.right/2-50,rect.bottom-70,
            hWnd,IDC_LIST,hInst,NULL);
        SendMessage(hWndList,CB_RESETCONTENT,0,0L);
        SendMessage(hWndList,CB_DIR,(WORD)(0x4000),(LONG)(LPSTR)"");
        break;
    case WM_COMMAND:
        switch(wParam)
        {
            case IDC_COMBO:
                if ( HIWORD(lParam) == CBN_DBLCLK )
                {
                    DlgDirSelectComboBox(hWnd,szPathSpec,IDC_COMBO);
                    bCurrentDir = TRUE;
                    for (i=1; i<9;i++)
                    {
                        if (szPathSpec[i] == '\0')
                            break;
```

```
                    if (szPathSpec[i] == '.')
                    {
                        bCurrentDir = FALSE;
                        break;
                    }
                }
                if (szPathSpec[0] == '.')
                    bCurrentDir = TRUE;

                if (bCurrentDir == TRUE)
                {
                    strcat(szPathSpec," * . * ");
                    DlgDirListComboBox(hWnd,szPathSpec,IDC_COMBO,
                        IDC_STATIC,0x0 |0x10);
                }
            }
            break;
        case IDC_LIST:
            if ( HIWORD(lParam) == CBN_SELCHANGE )
            {
                DlgDirSelectComboBox(hWnd,szPathSpec,IDC_LIST);
                strcat(szPathSpec," * . * ");
                DlgDirListComboBox(hWnd,szPathSpec,IDC_COMBO,IDC_STATIC,
                    0x0 |0x10);
                SetWindowText(hWndCombo,"");
                bCurrentDir = TRUE;
            }
            break;
        default:
            return (DefWindowProc(hWnd,message,wParam,lParam));
            break;
    }
    break;
  case WM_DESTROY:
    PostQuitMessage(0);
    break;
  default:
    return (DefWindowProc( hWnd,message,wParam,lParam ));
    break;
  }
  return(0L);
}
```

模块定义文件:COMBOBX. DEF

```
NAME Combobx
DESCRIPTION 'File selection using combo boxes'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
```

```
HEAPSIZE 4096
STACKSIZE 5012

EXPORTS WndProc @1
```

资源描述文件:COMBOBX.RC

```
#include "combobx.h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件:COMBOBX.H

```
#include <windows.h>
#include <string.h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define IDC_COMBO 100
#define IDC_STATIC 110
#define IDC_LIST 120

char szString[128];                          /* Variable for loading resource text */
char szAppName[20];                          /* Window class name */
HANDLE hInst;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL WndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:COMBOBX.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $*.rc

RESFILES =
H = COMBOBX.H

all: COMBOBX.EXE

COMBOBX.RES: COMBOBX.RC $(RESFILES)

COMBOBX.OBJ: COMBOBX.C $(H)

COMBOBX.EXE: COMBOBX.OBJ COMBOBX.RES COMBOBX.DEF
    link /NOD/A:16 COMBOBX,COMBOBX,nul,libw slibcew,COMBOBX.def
```

```
rc COMBOBX. res COMBOBX. exe
```

如下运行 NMAKE 就得到 COMBOBX 应用程序：

```
nmake combobx. mak
```

主窗口大小随屏幕大小改变。三个子窗口、一个静态编辑控件和两个组合框的大小则依赖于父窗口的大小。所以不必再计算窗口的总大小。WS_BORDER 设置标志 WS_THICKFRAME，且删除 WS_MAXIMIZEBOX 标志。

在上例中，组合框填满当前数据是通过发送 CB_DIR 消息和调用 DlgDirListComboBox 来完成的，因为控件是在一个窗口而非对话框中，函数传递的是主窗口句柄。

这两种方法都包括用 DOS DIR 命令后的字符串来选择要显示的文件。也可以包括字符 * 和?。如果要显示所有文件，则该串为“ * . * ”。

除了串之外，还要知道文件、路径、驱动器或其组合是否应被列出。下面的值都是很有用的，也可将它们 OR(或)起来。

值	含义
0×0000	不带属性的文件
0×0001	只读文件
0×0002	隐藏文件
0×0004	系统文件
0×0010	路径
0×0020	归档文件
0×4000	驱动器

请看下面的例子：

```
SendMessage(hWndList,CB_RESETCONTENT,0,OL);
SendMessage(hWndList,CB_DIR,(WORD)(0x4000),(LONG)(LPSTR)"");

DlgDirListComboBox(hWnd,szPathSpec,IDC_COMBO,IDC_STATIC,0x0,| 0x10);
```

在第一个组合框中，双击鼠标按钮可改变路径。而在第二个组合框中，单击鼠标按钮可改变驱动器。因此，为了标识码 CBN_DBLCLK 和 CBN_SELCHANGE，要检查 lPrarm 参数的高位字。

函数 DlgDirSelectComboBox 可用来确定哪个组合框入口项被选中。对于 ID 为 IDC_COMBO 的组合框，第二个参数必须检查，以确定是文件还是目录被选中。

后一种情况应导致组合框和静态域输出的变化。如果入口项中包括一个句号，则可认为是文件名，除非句号在字符串的前面。

然而，如果一个新的驱动器被选中，则函数 DlgDirSelectComboBox 的第二个参数更新第一个组合框。

```
DlgDirSelectComboBox(hWnd,szPathSpec,IDC_LIST);
strcat(szPathSpec," * . * ");
DlgDirListComboBox(hWnd,szPathSpec,IDC_COMBO,IDC_STATIC,0x0 |0x10);
```

在对控件类的讨论中，还应提一下 MDICLIENT 类，它用于 MDI(多媒体接口)编程。

第十二章　通用对话框

12.1　概　述

12.1.1　COMMDLG.DLL

Windows 3.1 版包含的命名为 COMMDLG.DLL 的动态链接库(DLL)可以调用很多现在应用程序中遇到的通用对话框。这些通用对话框极大地简化了 Windows 应用程序的 C 语言编程。

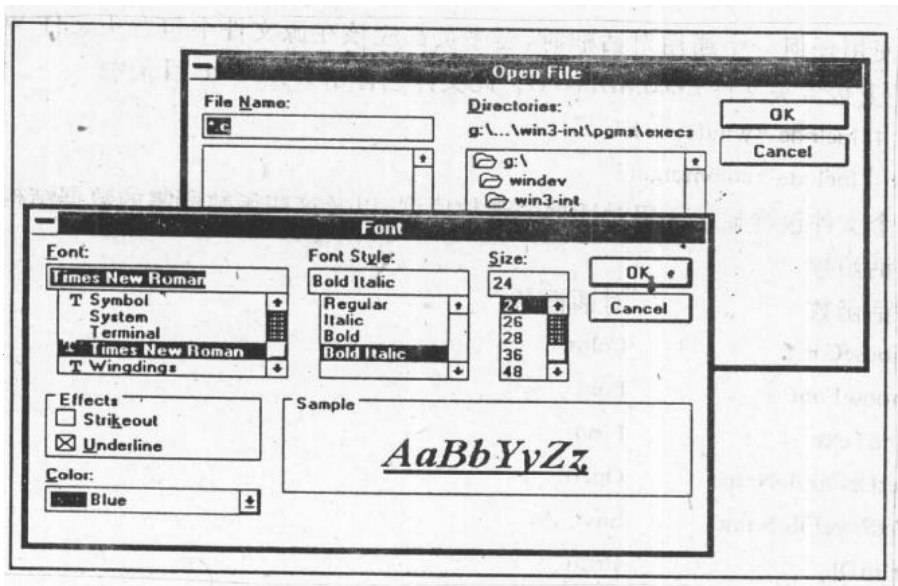


图 12.1　通用对话框示例

动态链接库根据程序员要求的结构和功能提供如 DLG 文件一样的资源。程序员不是用对话框编辑器和与写操作有关的对话过程来产生对话框，而是简单地利用已存在的对象，并根据需要进行修改。这种标准化也给用户带来了方便。用户只要熟悉了这些对话框，便可以轻松地学会使用其应用程序。

第九章已对对话框作了说明，本章在第九章的基础上介绍 Windows 以及应用程序中经常使用的一些对话框。

12.1.2 标准 DLG 文件

SDK 提供的 5 个通用对话框可以实现下列任务：

- 颜色控制
- 打开和保存文件
- 寻找和替换文本
- 字模控制
- 打印机控制

本章向读者介绍这些对话框的细节。虽然他们的外观和目的各自不同，但其实现机制和编程代码是相似的。

12.1.3 COMMDLG.LIB 输入库

如果用户想在程序中使用某个通用对话框，就必须在链接此应用程序时加上输入库 COMMDLG.LIB，链接程序需要这个库来调用对话函数。这样，链接过程中用户的程序代码被加长，从而包含动态链接库 COMMDLG.DLL 中所需的函数。

12.1.4 头文件 COMMDLG.H

在使用任何一个通用对话框时，程序员都应该在源文件中包含头文件 WINDOWS.H。之后，还要包含头文件 COMMDLG.H。此文件也存于 INCLUDE 目录中。

- #inclnde "windows.h"
- #include "commdlg.h"

这个文件包含显示通用对话框的函数原型，以及这些函数所需的数据结构。下面列出了通用对话函数：

对话函数	对话框名
ChooseColor	Color
ChooseFont	Font
FindText	Find
GetOpenFileName	Open
GetSaveFileName	Save As
PrintDlg	Print
ReplaceText	Replace

12.1.5 对话函数

有一部分对话框使用 OK 和 Cancel 按钮，分别向调用函数返回 TRUE 或 FALSE。这种过程很像用 EndDialog 函数传值给 DialogBox 函数以消去一个对话框。

程序员需要用“hook”过程来响应通用对话框控件传来的消息(例如，在组合框中选择了下箭头键)；这与普通的对话框例程很相似。

因为 hook 过程是带返回值的函数，所以它的过程实例地址必须在使用 MakeProcInstance 函数时被首先确定下来。

12.1.6 函数 CommDlgExtendedError

函数 CommDlgExtendeError 分配了调用或处理一个对话框时发生的错误。它返回一个特殊的值来指明问题的性质。

例如，返回 CDERR_ININIALIZIATION 意味着错误发生在初始化阶段，原因大约是内存不够。

12.2 打开和保存文件

12.2.1 文件 FILEOPEN.DLG

文件 FILEOPEN.DLG 包含了一个对话框的详细说明，允许用户指定在当前窗口要打开的文件名或把当前窗口的内容保存到哪个文件中去。

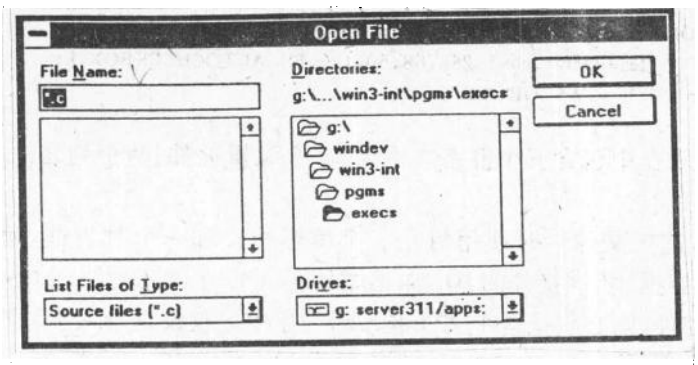


图 12.2

Open 和 Save As 对话框十分相似。它们具有相同的操作控制，仅仅是标题条和提示用户的静态文本区有所不同。

Open 框的提示为“List Files of Type:”，Save As 框的提示为“Save Files as Type:”。这两种对话框的主要区别是对话函数。Saving 函数返回一个文件名，但是并不打开这个文件。

左上方的文件名组合框中的文本域显示了缺省文件名。这个入口可在任何时刻被它下面的列表框中选中的另一个文件名代替。

文件列表框中显示的文件类型可由用户在“List Files of Type”框中指定。第二个组合框指定被选文件所在的驱动器，它上面的列表框指定被选文件的目录。

静态文本域初始时是空的。当对话框被调用时，文本域中填入当前目录或输入的目录。

下面显示了在 FILEOPEN.DLG 文件中对话框是怎样定义的：

```
FILEOPENORD DIALOG LOADONCALL MOVEABLE DISCARDABLE 36,24,264,134
CAPTION "Open"
STYLE WS_CAPTION |WS_SYSMENU | WS_POPUP|DS_MODALFRAME
FONT 8,"Helv"
BEGIN
    LTEXT "File &Name:",stc3,6,6,76,10
```

```
    CONTROL "",edt1,"edit",ES_LEFT | ES_AUTOHSCROLL | WS_BORDER | WSTABSTOP |
        WS_CHILD | ES_OEMCONVERT | ES_LOWERCASE,6,16,90,12
    CONTROL "",lst1,"listbox",LBS_SORT | LBS_HASSTRINGS |LBS_NOTIFY |
        LBS_DISABLENOCSROLL | WS_VSCROLL | WS_CHILD | WS_BORDER |WS_TABSTOP |
        LBS_OWNERDRAWFIXED,12,28,84,68
    LTEXT"&Directories:" -1,110,6,92,9
    LTEXT "",stc1,110,16,92,9,SS_NOPREFIX
    CONTROL "",lst2,"listbox",
        LBS_SORT|LBS_HASSTRINGS|LSB_NOTIFY|LBS_DISABLENOSCROLL|WS_VSCROLL|
        WS_CHILD | WS_BORDER | WS_TABSTOP | LBS_OWNERDRAWFIXED,110,28,92,68
    LTEXT "List Files of &Type:",STC2,6,102,90,9
    CONTROL "",cmb1,"combobox",CBS_DROPDOWNLIST |CBS_AUTOHSCROLL |
        WS_BORDER| WS_VSCROLL |WS_TABSTOP |WS_CHILD,6,112,90,36
    LTEXT "Dri&ves:" STC4, 110,,102,92,9
    CONTROL "",cmb2,"combobox",CBS_SORT|CBS_HASSTRINGS|CBS_OWNERDRAWFIXED|
        CBS_DROPDOWNLIST|WS_CHILD|CBS_AUTOHSCROLL|WS_BORDER|WS_VSCROLL|
        WS_TABSTOP,110,112,92,68
    DEFPUSHBUTTON "OK",IDOK,208,6,50,14,WS_GROUP
    PUSHBUTTON "Cancel",IDCANCEL,208,24,50,14,WS_GROUP
    PUSHBUTTON "&Help",pshHelp,208,46,50,14,WS_GROUP
    CHECKBOX "&Read Only",chx1,208,68,50,122,BS_AUTOCHECKBOX |
        WS_TABSTOP | WS_GROUP
END
```

根据定义，对话框中包含五个静态文本域、一个编辑控件、两个列表框、两个组合框、三个按钮和一个复选框。

更深入一步，选择文件名时使用的是一个编辑控件和一个列表框，而不是一个 CBS_SIMPLE 风格的组合框。不同控件的 ID 值，比如 stc4，可以在伴随文件 DLGS. H 中找到。如果直接涉及到某个控件，则必须把这个文件嵌入到程序中。仅仅在加入了一个 hook 过程时这种特殊情况才会发生。

这个文件还定义了 FILEOPERORD 对话框。理论上可以使用传统的函数如 DialogBox 或 CreateDialg 来显示对话框，但这里实际上使用的是上面提到的这种特殊对话函数。

使用标准定义并不意味着打开或保存文件必须使用同一格式。程序中可以去掉某些按钮或修改某些控件的标题。

这就是使用 OPENFILENAME 结构的目的，此结构在赋与特定值后传递给函数 GetOpenFileName 或 GetSaveFileName。如果调用成功，此结构中包含有被选文件的文件名和完整的路径信息。

OPENFILENAME 数据结构如下：

域名	数据类型	描述
lSructSize	DWORD	结构的大小(以字节为单位)
hwndOwner	HWND	空值或有效窗口的句柄
hInstance	HANDLE	仅在设置了标志 OFN_ENABLETEMPLATE 时使用
lpstrFilter	LPSTR	组合框"List File of Type"的数据结构
nFilterIndex	DWORD	初始化数据结构的索引
lpstrCustomFilter	LPSTR	自定义数据类型

nMaxCustFilter	DWORD	lpstrCustomFilter 指示的缓冲区尺寸
LpstrFile	LPSTR	编辑或控件的初始化入口
nMaxFile	DWORD	lpstrFile 指示的缓冲区的尺寸
lpstrInitialDir	LPSTR	初始化目录
lpstrTitle	LPSTR	对话框标题
Flags	DWORD	各个指示符的组合
nFileOffset	WORD	返回值:lpstrFile 所指的缓冲区,文件名的偏移量
nMaxFileTile	DWORD	lpstrFileTitle 指示的缓冲区的尺寸
lpstrFileTitle	LPSTR	返回值:被选中的文件名
nFileExtension	WORD	返回值:文件扩展名的偏移地址
lpstrDefExt	LPSTR	如果没有给定文件扩展名,将自动附加到文件上的扩展名
lCustData	DWORD	将由参数 lParam 传送的应用程序定义的数据
lpTemplateName	LPSTR	未使用通用对话框时,对话框的名字

初看此结构会感到有点困惑,因此下面讲解一下此结构中常用的域。

为了使列表框中的文件容易被选择,常常只显示特定类型的文件,例如所有扩展名为 txt 的文件。用户可以通过在"List Files of Type"组合框中选择别的文件名限制来改变选择规范。

"文件名限制"的个数和表现可以由指向一个串对链的 lpstrFilter 域来决定。

这些串对必须有特定的格式。每对中的第一个串描述在组合框中用户所见的"文件名限制"。第二个串指定了相应的"文件名限制"的模式。图 12.3 显示了一些可能会见到的串对。链的最后一个串必须以两个空字符 NULL 结尾。

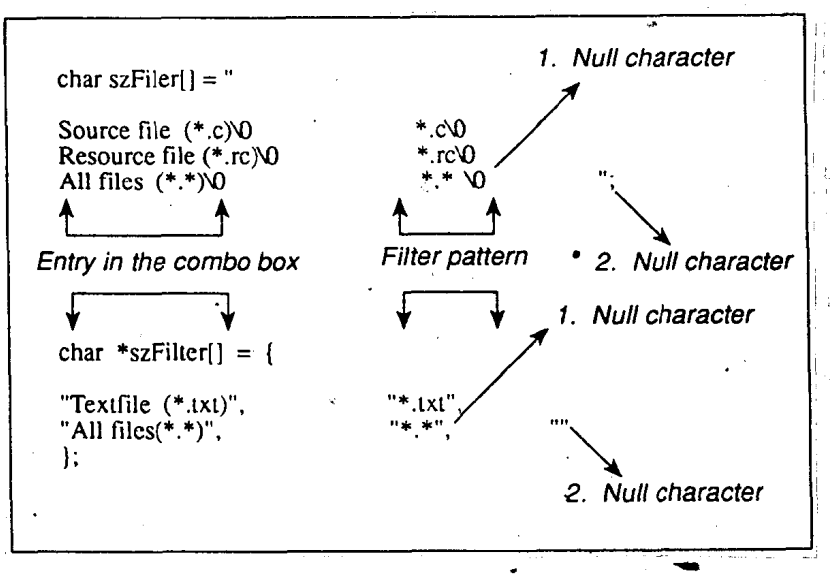


图 12.3 结构 OPENFILENAME 中的文件名限制

nFilterIndex 域指示左下方组合框中的文件名限制，同时还决定了它上面的列表框中显示哪些文件。链中的第一对串的索引号为 1。当用户从屏幕上消去一个对话框时，系统把当前被选的文件索引号赋值到 nFilterIndex 域中。

如果想让文件名限制的文本域预先装入某个文件名，则这个文件名必须被当作字符串传给 lpstrFile 域。成功地调用了 GetOpenFileName 或 GetSaveFileName 函数之后，缓冲区中将包含被选文件的文件名和完整的路径。

nMaxFile 域指定了缓冲区的长度。缓冲区的内容可以用来打开一个文件。与此相反，lpstrFileTitle 域指向的缓冲区仅仅包含文件名。

如果初始时想显示非当前路径，就必须使用 lpstrInitialDir 域。这个在文件 FILEOPEN. DLG 中被定义的域指定了对话框的标题为“Open”，所以当使用对话框保存文件时，必须修改这个标题。新标题写在 lpstrTitle 域中。

为了使程序对用户更为友好，可用 lpstrDefExt 域指示一个当未指定文件扩展名时自动加到文件名后的文件扩展名。例如，Microsoft Word 在这种方式的文件名后加上扩展名 .doc。

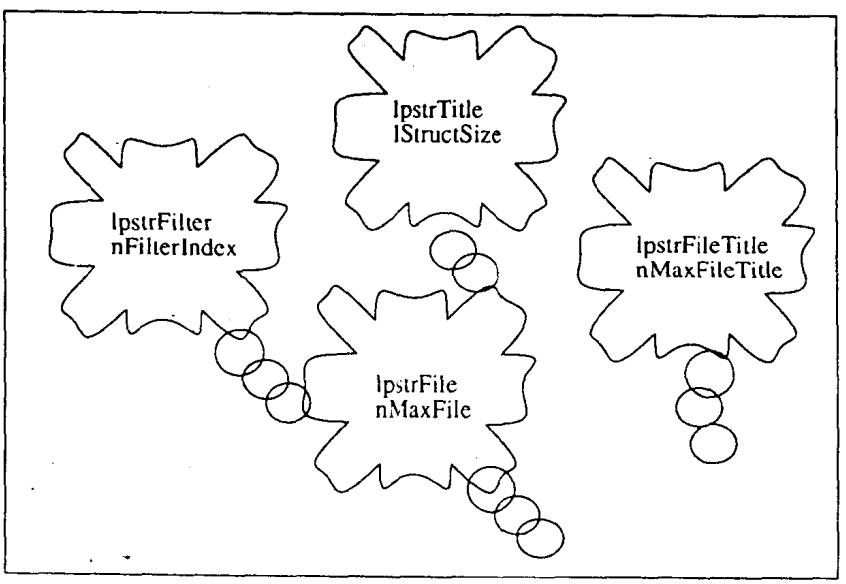


图 12.4　结构 OPENFILENAME 中最重要的域

lpTemplateName 域和 lpfnHook 域的用法将在后面讨论。

12.2.2　标志域(Flags)

除了上图所列的域之外，标志域通常也被赋以特定的值。这样便可以指定显示一个“帮助”按钮，或指定取消“Read Only”复选框。可以把个别的值用位操作 OR 组合到一起。

位	意义
OFN_ ALLOWMULTISELECT	文件名列表框中允许多个选择
OFN_CREATEPROMPT	若文件不存在时，用户将被询问是否建立新文件
OFN_ENABLEHOOK	lpfnHook 域指定的 Hook 过程将被屏蔽掉
OFN_ENABLETEMPLATE	lpTemplateName 域指定的对话框将被屏蔽掉

OFN_ENABLETEMPLATEHANDLE	hInstance 定义的数据段包含一个预先调用的对话框
OFN_FILEMUSTEXIST	用户只能指定已存在的文件，否则将自动显示一个消息框
ONF_HIDEREADONLY	复选框“Read_Only”不被显示
OFN_NOCHANGEDIR	在调用对话框之前，目录将被设置到状态条中
OFN_OVERWRITEPROMPT	保存文件时，若此文件已存在，将显示一条信息
OFN_READONLY	当对话框被调用时，“Read_Only”复选框被选择
OFN_SHOWHELP	显示 Help 按钮

若要设置标志 OFN_ ALLOWMULTYSELECT，应该从 FILEOPEN. DLG 文件中选用 MULTYFILEOPENORD 对话框而不是 FILEOPENORD 对话框。虽然这两种方法看起来很相似，但前者在两种列表框中都使用了 LBS_EXTENDEDSEL 风格。

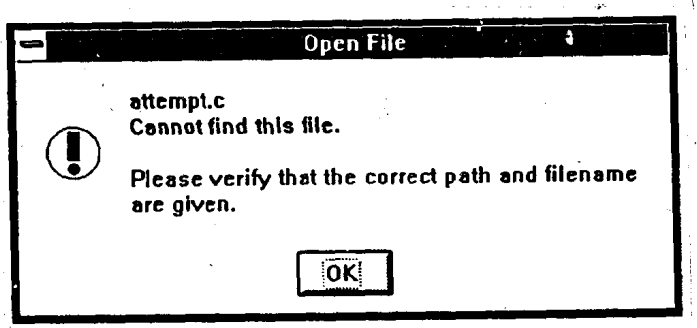


图 12.5

12.2.3 GetOpenFileName 和 GetSaveFileName

初始化后的 OpenFileName 结构变量的地址被传给函数 GetOpenFileName 和 GetSaveFileName。如果一个对话框调用成功，则返回 TRUE，被选中的文件随后被打开以备读写。

```
OPENFILENAME ofn;
                                                    // Initialize structure with appropriate values
ofn. lStructSize = sizeof(OPENFILENAME);
                                                    // Call dialog box for opening a file
if (GetOpenFileName(&ofn))
{
                                                    // Perform corresponding file operatins
    hFile = _lopen(ofn,lpstrFile,OF_READWRITE);
    :
}
```

12.2.4 简单的字处理器

这个例子会使读者想起 Windows 提供的应用程序 NOTEPAD. EXE。本例可对文本文件进行调用、修改、保存或用一新文件名进行保存。

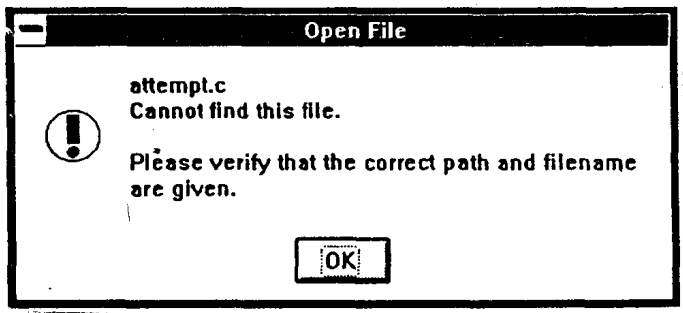


图 12.6

新函数	描述
GetOpenFileName	为打开文件而显示一个对话框
GetSaveFileName	为保存文件而显示一个对话框

新结构	描述
OPENFILENAME	上面几个函数用到的结构

源代码:OPENSAVE. C

```
/* * OPENSAVE.C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates standard open & save dialog boxes * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "opensave.h"                                    // Include OPENSAVE.H header file

#if _MSC_VER >= 700                                     // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                          // higher? Disable warning 4028
#endif

/* * WinMain (main function for every Windows application * * * * * * * * * */

int PASCAL WinMain(HANDLE hInstance,                                  // Current instance
        HANDLE hPrevInstance,                                         // Previous instance
        LPSTR lpszCmdLine,                                            // Ptr to string after prg name
        int nCmdShow)                                                 // App. window's appearance
{
    MSG msg;                                                          // Message handle
    HWND hWnd;                                                        // Window handle
    int nError;                                                       // Error integer

    strcpy(szAppName,"Opensave");                                     // Application name
    hInst = hInstance;                                                // Instance handle

    if (! hPrevInstance)                                              // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
```

```
        {
            LoadString(hInstance,IDS_ERR_CLASS,                    // If initialization fails
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;
        }
    }
/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow(szAppName,                                 // Window class name
        "Standard Dialog Boxes",                                   // Window caption
        WS_OVERLAPPEDWINDOW,                                       // Overlapped window style
        CW_USEDEFAULT,0,                                           // Upper-left x,y pos.
        GetSystemMetrics(SM_CXSCREEN)/2,                           // Initial x size
        GetSystemMetrics(SM_CYSCREEN)/2,                           // Initial y size
        NULL,                                                      // No parent window
        NULL,                                                      // Window menu used
        hInstance,                                                 // Application instance
        NULL);                                                     // No creation parameters

    if(hWnd == NULL)                                               // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,                      // Display error
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );                                   // Make window visible
    UpdateWindow( hWnd );                                          // Update window

    while (GetMessage(&msg,NULL,0,0))                              // Message reading
    {
        TranslateMessage(&msg);                                    // Message translation
        DispatchMessage(&msg);                                     // Send message to Windows
    }
    return (int)msg.wParam;                                        // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                              // Main window class

    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );                // Mouse cursor
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);               // Default icon
    wcClass.lpszMenuName = (LPSTR)"DlgMenu";                       // Menu name
    wcClass.lpszClassName = szAppName;                             // Window class
```

```
    wcClass.hbrBackground =
        (HBRUSH)GetStockObject(WHITE_BRUSH);          // White background
    wcClass.hInstance = hInstance;                    // Instance
    wcClass.style = CS_VREDRAW | CS_HREDRAW;          // Horizontal and vertical redraw
    wcClass.lpfnWndProc = StdDlgWndProc;              // Window function
    wcClass.cbClsExtra = 0 ;                          // No extra bytes
    wcClass.cbWndExtra = 0 ;                          // No extra bytes

    if (! RegisterClass( &wcClass ) )                 // Register window class
        return FALSE;                                 // Return FALSE if registration fails
                                                      // If registration is successful,
    return TRUE;                                      // Return TRUE
}

/* * 'Open' dialog box * * * * * * * * * * * * * * * * * * * * */

void PASCAL Open(HWND hEdit)
{
    int hFile,Len;                                    // File handle and length
    HANDLE hMem;                                      // Memory handle
    LPSTR lpBuff;                                     // Long pointer to buffer

    ofn.lpstrTitle = "Open File";                     // Dialog box title
    ofn.Flags = ofn.Flags | OFN_PATHMUSTEXIST |// File-path must exist flags
        OFN_FILEMUSTEXIST;

    if (GetOpenFileName(&ofn))                        // Open··· selected? Then:
    {
        SetWindowText(GetParent(hEdit),ofn.lpstrFile);// Window title=file name
        hFile = _lopen(ofn.lpstrFile,OF_READWRITE);   // Open an existing file
        hMem = GlobalAlloc(GMEM_MOVEABLE,65535);      // Allocate global heap memory
        lpBuff = GlobalLock(hMem);                    // Get pointer to global memory block
        Len = _lread(hFile,lpBuff,(WORD)65535);       // Read file data
        * (lpBuff+Len) = 0;                           // Set buffer pointer position to zero
        _lclose(hFile);                               // Close file
        SetWindowText(hEdit,lpBuff);                  // Window title=buffer contents
        EnableMenuItem(GetMenu(GetParent(hEdit)),     // Enable menu
            IDM_SAVE,                                 // Save··· item
            MF_ENABLED);                              // Enabled status
        GlobalUnlock(hMem);                           // Unlock memory set by GlobalLock()
        GlobalFree(hMem);                             // Free global memory block
    }
    return;
}

/* * 'Save' dialog box * * * * * * * * * * * * * * * * * * * * */

void PASCAL Save(HWND hEdit)
{
    int hFile,Len;                                    // File handle and length
    HANDLE hMem;                                      // Memory handle
```

```
    LPSTR lpBuff;                                   // Long pointer to buffer
    OFSTRUCT ofs;                                   // Open-file structure

    ofn.lpstrTitle = "Save File";                   // Dialog box title
    ofn.Flags = ofn.Flags |                         // Enable
        OFN_OVERWRITEPROMPT;                        // Overwrite prompt flag

    if (GetSaveFileName(&ofn))                      // Save··· item selected? Then:
    {
        hFile = OpenFile(ofn.lpstrFile,&ofs,OF_CREATE | OF_WRITE);   // Open file
        hMem = GlobalAlloc(GMEM_MOVEABLE,65535);    // Allocate global heap memory
        lpBuff = GlobalLock(hMem);                  // Get pointer to global memory block
        Len = GetWindowTextLength(hEdit);           // File length=window text length
        GetWindowText(hEdit,lpBuff,Len);            // Get window text from buffer
        _lwrite(hFile,lpBuff,Len);                  // Write file
        _lclose(hFile);                             // Close file
        SetWindowText(GetParent(hEdit),ofn.lpstrFile);   // Window title=file name
        GlobalUnlock(hMem);                         // Unlock memory set by GlobalLock()
        GlobalFree(hMem);                           // Free global memory block
    }
    return;
}

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,
            unsigned message,
            WORD wParam,
            LONG lParam )
{
    static HWND hEdit;

    switch (message)
    {
        case WM_CREATE:                             // Create window
            hEdit = CreateWindow("Edit",            // Window class (Edit)
                    NULL,                           // Window name
                    WS_CHILD |                      // Child window
                    WS_VISIBLE |                    // Initially visible window
                    WS_HSCROLL |                    // Horizontal scroll bar
                    WS_VSCROLL |                    // Vertical scroll bar
                    WS_BORDER |                     // Border on window
                    ES_LEFT |                       // Align text flush left
                    ES_MULTILINE,                   // Multiple-line edit control
                    0,0,                            // Default x and y pos.
                    0,0,                            // Default width and height
                    hWnd,                           // Parent window
                    50,                             // Menu or child window i.d.
                    hInst,                          // Instance
```

```
                NULL);                                          // LP parameter (none)

            ofn.lStructSize = sizeof(OPENFILENAME);
            ofn.hwndOwner = hWnd;
            ofn.lpstrFilter = szFilterSel;
            ofn.lpstrCustomFilter = NULL;
            ofn.nMaxCustFilter = 0;
            ofn.nFilterIndex = 1L;
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = MAXIMUMNAME;
            ofn.lpstrInitialDir = NULL;
            ofn.lpstrFileTitle = szFileTitle;
            ofn.nMaxFileTitle = MAXIMUMNAME;
            ofn.lpstrDefExt = "c";
            ofn.Flags = OFN_HIDEREADONLY;
            break;
        case WM_SIZE:                                           // Client area size
            MoveWindow(hEdit,                                   // Edit window
                    0,                                          // New x position
                    0,                                          // New y position
                    LOWORD(lParam),                             // New window width
                    HIWORD(lParam),                             // New window height
                    TRUE);                                      // Do repaint window
            break;
        case WM_COMMAND:                                        // Commands
            switch(wParam)
            {
                case IDM_OPEN:                                  // Open selected
                    Open(hEdit);                                // Open file
                    break;
                case IDM_SAVE:                                  // Save selected
                    Save(hEdit);                                // Save file
                    break;
                default:                                        // Send other messages
                    return (DefWindowProc(hWnd,message,wParam,lParam));
                    break;
            }
            break;
        case WM_SETFOCUS:                                       // Set window focus
            SetFocus(hEdit);
            break;
        case WM_DESTROY:                                        // Exit
            PostQuitMessage(0);
            break;
        default:                                                // Send other messages
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                              // End of this message process
    }
    return(0L);
}
```

模块定义文件:OPENSAVE. DEF

```
NAME Opensave
DESCRIPTION 'Dialog boxes for Opening and Saving'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:OPENSAVE. RC

```
#include "opensave. h"
#include "windows. h"

DlgMenu MENU
BEGIN
    POPUP "&Dialog Boxes"
    BEGIN
        MENUITEM "&Open…", IDM_OPEN
        MENUITEM "&Save…", IDM_SAVE, GRAYED
    END
END

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件:OPENSAVE. H

```
#include <windows. h>
#include <string. h>
#include <commdlg. h>

#define IDM_OPEN 10
#define IDM_SAVE 11
#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define MAXIMUMNAME 50

char szString[128];                          /* Variable for loading resource text */
char szAppName[20];                          /* Window class name */
HANDLE hInst;
OPENFILENAME ofn;
char szFilterSel[] = "Source files (*.c)\0*.c\0Resource files (*.rc)\0*.rc\0
```

```
                    All files ( * . * )\0 * . * \0";
char szFileName[MAXIMUMNAME];
char szFileTitle[MAXIMUMNAME];

void PASCAL Open(HWND);
void PASCAL Save(HWND);
int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:OPENSAVE. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS = -AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: rc -r $ * .rc

RESFILES =
H = OPENSAVE. H
LIBS_EXT = commdlg. lib

all: OPENSAVE. EXE

OPENSAVE. RES: OPENSAVE. RC $ (RESFILES)

OPENSAVE. OBJ: OPENSAVE. C $ (H)

OPENSAVE. EXE: OPENSAVE. OBJ OPENSAVE. RES OPENSAVE. DEF
    link /NOD/A:16 OPENSAVE,OPENSAVE,nul,libw slibcew $ (LIBS_EXT),OPENSAVE. def
    rc OPENSAVE. res OPENSAVE. exe
```

像下面这样运行 NMAKE 便可产生应用程序 OpenSave:

```
nmake opensave. mak
```

EDIT 类的一个子窗口取代了主窗口。这个窗口是在处理 WM_CREATE 消息时创建的,它和父窗口有同样的大小。函数 MoveWindow 通过消息 WM_SIZE 修改窗口的大小。

```
case WM_SIZE:
    MoveWindow(hEdit,0,0,LOWORD(lParam),
            HIWORD(lParam),TRUE;
    break;
```

每当主窗口接到键盘输入时,就将焦点传给其子窗口,由子窗口负责接收文本。

```
case WM_SETFOCUS:
    SetFocus (hEdit);
    break;
```

OPENFILENAME 结构变量 ofn 被定义为全局变量,于是结构中那些应用于 Open 和 Save 对话框的域可以用 WM_CREATE 消息赋值。只有对话框的标志和标题是在调用 box 之前由各个 Open 和 Save 子程序独立设置的。

```
ofn. lStructSize = sizeof(OPENFILENAME);
ofn. hwndOwner = hWnd;
ofn. lpstrFilter = szFilterSel;
```

```
ofn.nMaxCustFilter = 0;
ofn.nFilterIndex = IL;
ofn.lpstrFile = szFileName;
ofn.nMaxFile = MAXIMUMNAME;
ofn.lpstrInitialDir = NULL;
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = MAXIMUMNAME;
ofn.lpstrDefExt = "c";
ofn.Flags
```

每个被读写的文件分配有一块 64K 的内存作为缓冲区，用完后内存块立刻被释放。

```
hMem = GlobalAlloc(GMEM_MOVEABLE,65535);
lpBuff = GlobalLock(hMem);

GlobalUnlock(hMem);
GlobalFree(hMem);
```

12.3 查找和替换文本

12.3.1 文件 FINDTEXT.DLG

文件 FINDTEXT.DLG 定义了两个对话框，一个用来查找串，另一个用来替换串。

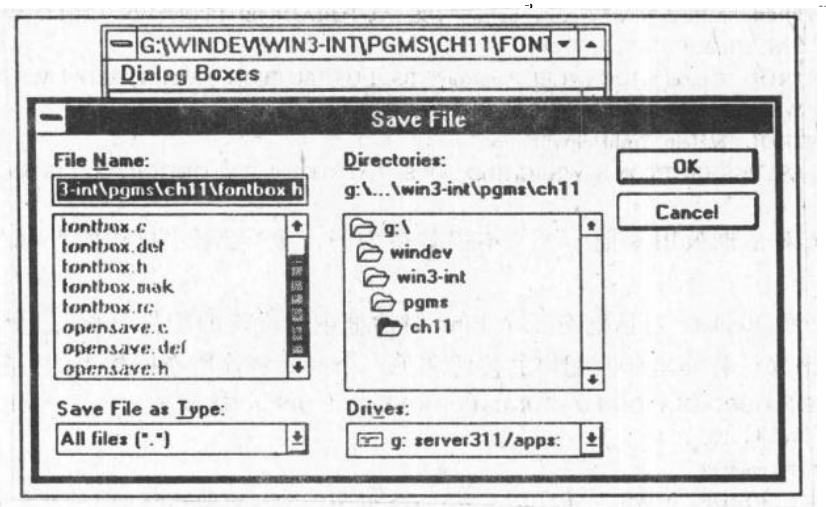


图 12.7

这两种对话框与通用对话框有两个区别。它们的用法与非模态对话框(modeless)相同，即它们不影响激活它们的应用程序，于是用户便可以在这两种对话框中分支前向和后向。

第二个不同点从程序设计的角度来看特别有趣：这两种对话框向函数 FindText 或 ReplaceText 发送消息。

Find 对话框总有一个让用户输入要查找的文本串的编辑控件和两个决定开始查找还是取消查找的单选按钮。

另外，用户通常可以指定查找的方向(从当前位置)，以及查找时是否区分大小写。

同时，复选框“Match Whole Word only”可以指定是否为全词匹配。如果此框被设置了，则在查找“Start”时将不会找到“Starting”。和复选框一样，两个控制查找方向的单选按钮可以用设置标志位的方式屏蔽掉。

最后，如果 Help 按钮被显示了，它将出现在 Cancel 按钮的下面。

```
FINDDLGORD DIALOG LOADONCALL MOVEABLE DISCARDABLE
        36,24,264,134
CAPTION "Fild"
STYLE WS_BORDER | WS_CAPTION | DS_MODALFRAME | WS_POPUP | WS_SYSMENU
FONT 8,"Helv"
BEGIN
    CONTROL "Fi&nd What:",-1,"static",SS_LEFT | WS_CHILD,4,8,42,8
    CONTROL "",edtl,"edit",ES_LEFT|ES_AUTOHSCROLL | WS_BORDER | WS_GROUP |
        WS_TABSOP | ES_CHILD | WS_BORDER,47,7,128,12
    CONTROL "Match &Whole Word Only",chx1,"button,BS_AUTOCHECKBOX|WS_TABSTOP
        |WS_CHILD|WS_GROUP,4,26,100,12
    CONTROL "Match &Case",chx2,"button,BS_AUTOCHECKBOX | WS_TABSTOP | WS_CHILD,
        4,42,64,12
    CONTROL "Direction",grp1,"button",BS_GROUPBOX | WS_CHILD,107,26,68,28
    CONTROL "&Up",rad1,"button",BS_AUTORADIOBUTTON | WS_TABSTOP | WS_CHILD |
        WS_GROUP,111,38,20,12
    CONTROL "&Down",rad1,"button",BS_AUTORADIOBUTTON | WS_TABSTOP |WS_CHILD,
        138,38,30,12
    CONTROL "&Find Next",rad2,"button",BS_AUTORADIOBUTTON|WS_CHILD|WS_TABSTOP|
        WS_GROUP,182,5,50,14
    CONTROL "Cancel",IDCANCEL,"button",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP |
        WS_GROUP,182,23,50,14
    CONTROL "&Help",pshHelp,"button",
        BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_GROUP,182,45,50,14
END
```

Replace 对话框被用来把一个文本串替换为另一个，与刚才描述的 Find 对话框非常相似。

除此之外，Replace 对话框有三个 Find 对话框中所没有的控件：一个用于输入新串的编辑控件、一个确认替换当前匹配串的按钮和另一个确认替换所有匹配串的按钮。

```
REPLACEDLGORD DIALOG LOADONCALL MOVEABLE DISCARDABLE
        36,44,230,94
CAPTION "Replace"
STYLE WS_BORDER | WS_CAPTION | WS_SYSMENU | DS_MODALFRAME | WS_POPUP
FONT 8,"Helv"
BEGIN
    CONTROL "Fi&nd What:",-1,"static",SS_LEFT | WS_CHILD,4,9,48,8
    CONTROL "",edtl,"edit",ES_LEFT|ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP |
        WS_CHILD | WS_GROUP,54,7,114,12
    CONTROL "Re&place With:",-1,"static",SS_LEFT | WS_CHILD,4,26,48,8
    CONTROL "",edt2,"edit",ES_LEFT | ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP |
        WS_CHILD | WS_GROUP,54,24,114,12
    CONTROL "Match &Whole Word Only",chx1,"button,BS_AUTOCHECKBOX | WS_TABSTOP |
        WS_CHILD | WS_GROUP,5,46,104,12
    CONTROL "Match &Case",chx2,"button",BS_AUTOCHECKBOX | WS_TABSTOP | WS_CHILD,
```

```
        5,62,59,12
    CONTROL "&Find Next",IDOK,"button",BS_DEFPUSHBUTTON | WS_CHILD | WS_TABSTOP |
        WS_GROUP,174,4,50,14
    CONTROL "&Replace",psh1,"button",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP |
        WS_GROUP,174,21,50,14
    CONTROL "Replace &All",psh2,"button",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP |
        WS_GROUP,174,38,50,14
    CONTROL "Cancel",IDCANCEL,"button",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP |
        WS_GROUP,174,55,50,14
    CONTROL "&Help",pshHelp,"button",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP |
        WS_GROUP,174,75,50,14
END
```

12.3.2 数据结构 FindReplace

同一个数据结构 FindReplace 被用于两个对话框。这个结构中只有两个域与替换串有关。

域名	数据类型	描述
lStructSize	DWORD	本结构的大小(以字节为单位)
hwindOwner	HWND	有效的窗口句柄
hInstance	HANDLE	仅在标志 FR_ENABLETEMPLATE 被设置时使用
Flags	DWORD	各个指示符的组合
lpstrFindWhat	LPSTR	被查找串和返回值
lpstrReplaceWith	LPSTR	替换串和返回值
wFindWhatLen	WORD	被查找串的长度
wReplaceWithLen	WORD	替换串的长度
lCustData	DWORD	应用程序定义的、将由参数 lParam 传送给 hook 过程的数据
lpTemplateName	LPSTR	在没有显示标准对话框时,对话框的名字

结构中的某些域在 OpenFileName 结构中也有。但是,FindReplace 结构必须根据 hwndOwner 域中的有效窗口句柄来赋值。对于 Replace 对话框,下面这些域应该被写成最小值。

```
FINDREPLACE fr;
char szFindText[128];

memset(&fr,0x00,sizeof(FINDREPLACE));
fr.lStructSize = sizeof(FINDREPLACE);
fr.hwndOwner = hWnd;
fr.wFindWhatLen = sizeof(szFindText);
fr.lpstrFindWhat = szFindText;
FindText((&fr);
```

如需要,可以用一个串来预置 szFindText 域,这个串将出现在编辑控件中。在任何情况下,都需要缓冲区来决定用户输入的是什么。如果调用了 Replace 对话框,还需要使用另外那两个域。

```
fr.lpstrReplaceWith = szReplText;
```

```
fr. wReplaceWithLen = szeof(szReplText);
ReplaceText(&fr);
```

12.3.3 标志域

几个标志域可用来在调用对话框时设置复选框和单选按钮的状态。

用来设置复选框和单选按钮的状态的标志域如下：

位	含义
FR_DOWN	单选按钮 Down 已被初始化
FR_MATCHCASE	查找时区分大小写的复选框已被初始化
FR_WHOLEWORD	全词匹配查找的复选框已被初始化
FR_NOUPDOWN	选择查找方向的单选按钮被屏蔽
FR_NOWHOLEWORD	查找时要求全词匹配的复选框被屏蔽
FR_SHOWHELP	显示一个 Help 按钮
FR_ENABLEHOOK	域 lpfnHook 指定的 hook 过程被启用
FR_ENABLETEMPLATE	域 lpTemplateName 所指示的对话框被启用
FR_ENABLETEMPLATEHANDLE	hInstance 定义的数据段中包含一个预调入的对话框

12.3.4 对话框发送消息

用户使用 Find 和 Replace 对话框的权限不仅仅是利用 FindReplace 变量的地址来调用函数 FindText 或 ReplaceText。程序将予以响应的消息必须被登录，其原因与非模态对话框的类型有关。

为了说明这是怎么实现的，先假定用户已经从菜单中选择了 Find 命令，于是相应的对话框被显示出来。现在键入了想到查找的文本串，并按下了 FindNext 按钮。

对话框一直保持在屏幕上，直到用户按下了 Cancel 按钮时才被消去。这样做的原因是，用户可能想输入另一个串或出现其他别的情况。当用户指示开始查找时，控制权将转移给通用窗口函数。

在大多数情况下，这意味着每当 Find 和 Replace 按钮被按下时，将发送一个消息给结构 FindReplace 的 hwndOwner 域所指定的窗口。

因为通用对话框有一个程序员不能直接调用的对话函数，所以查找或替换文本串的程序代码必须被安排在主窗口的窗口函数中。

12.3.5 函数 RegisterWindowMessage

从对话框发送给主窗口函数的消息不是永久定义的，而是在两个通用对话框的对话函数中，通过函数 RegisterWindowMessage 登录的。这个函数定义了一个新的消息，并确保此消息对系统来说是唯一的。

反之，我们假设两个应用程序可以用同一个值(比如说 WM_USER+4)来定义消息，但相应的处理程序却完全不同。那么，发送消息(WM_USER+4)给所有运行程序将产生意料不到的效果，因为不同的应用程序将用不同的方式解释它。

```
SendMessage (NULL,WM_USER+4,0,0L);
```

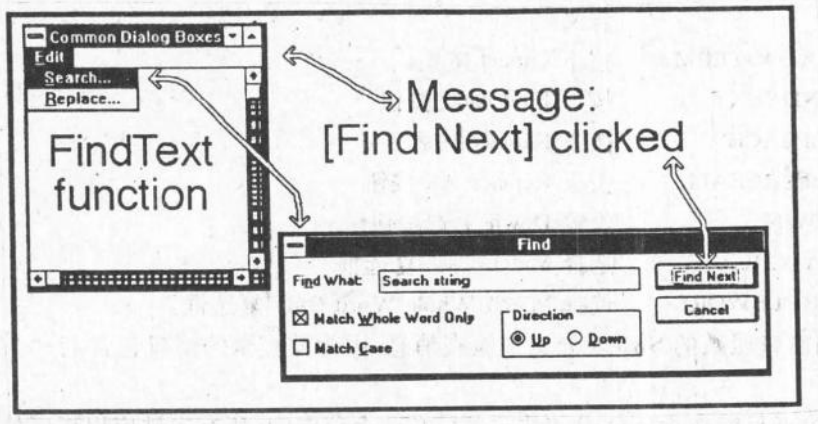


图 12.8

函数 RegisterWindowMessge 是用于两段应用程序通信的典型函数。两段应用程序都必须把自己定义的消息串传给函数 RegisterWindowMessage 。

返回的消息是一个可用作通常消息的从 0×C000 到 0×FFFF 的值。这条消息被登录且在 Windows 对话期间能保持持久性。

Find 和 Replace 对话框的消息名为“Commdlg_FindReplace”，在文件 COMMDLG. H 中被定义为常量 FindMsgString：

```
#define FindMsgString "Commdlg_FindReplace"
```

对函数 RegisterWindowMessage 的调用应该被安排在标准对话例程中，于是我们还必须将它包含到其 hwndOwner 窗口函数中。实现方法可以用消息 WM_CREATE。从对话框中常可以接收到消息。

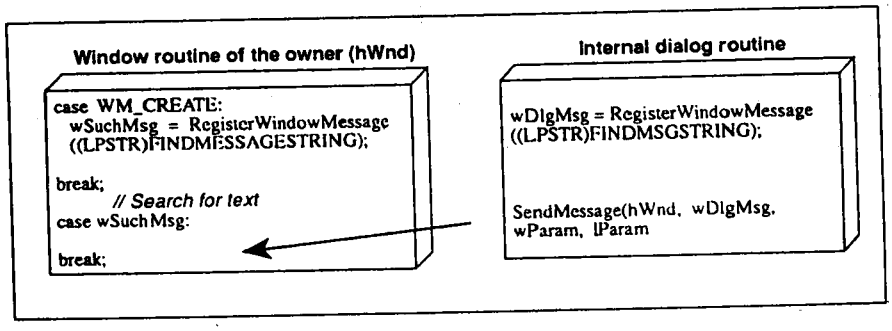


图 12.9

当对话框首次出现在屏幕上时，确认开始查找或开始替换的按钮还无法使用。只有当用户在第一个编辑控件中键入了字符时，它们才可以使用。

随后的按钮操作把先前定义好的消息从对话函数传递给主窗口函数。对话函数带有一个指向 FindReplace 结构的参数 lParam，它使用标志来指示哪个按钮被按下了。单选按钮和

复选框的状态也是如此被改变的。在初始化单选按钮和复选框时，部分标志是相同的。

位	含义
FR_DIALOGTERM	按下 Cancel 按钮
FR_FINDNEXT	按下 Find Next 按钮
FR_REPLACE	按下 Replace 按钮
FR_REPLACEALL	按下 Replace All 按钮
FR_DOWN	设置 Down 单选按钮
FR_MATCHCASE	选择 Match Case 复选框
FR_WHOLEWORD	选择 Match Whole Word Only 复选框

不能在窗口函数的 Switch 分支中检查消息，因为被登录的消息包含着一个变量而不是常量。

于是，检查只能在 default 分支中进行。在这个分支中，所有未被应用程序处理的消息都被送给缺省窗口例程。

```
default:
    if ( message == wFindMsg)
    {
        lpFind = (LPFINDREPLACE) lParam;
        dwFlags = lpFind->Flags;
        fDown = (dwFlags & FR_DOWN ? TRUE : FALSE);
        fCase = (dwFlags & FR_MATCHCASE ? TRUE : PALSE);
        fWord = (dwFlags & FR_WHOLEWORD ? TRUE : FALSE);
        if (dwFlags & FR_FINDNEXT)                          // Find text
        if (dwFlags & FR_REPLACE)                  // Replace one occurrence of text
        if (dwFlags & FR_REPLACEALL)               // Repiace all occurrences of text
    }
    return (DefWindowProc( hWnd,message,wParam,lParam ));
    break;
}
```

12.4 查找和替换对话框

应用程序有两个子菜单来调用 Find 和 Replace 对话框。与打开和保存文件的程序不同，不是重复主窗口的编辑控件中的串查找或串替换，而是在应用程序中把按钮状态和执行状态显示出来。

新函数	描述
FindText	显示一个文本查找对话框
ReplaceText	显示一个文本替换对话框
RegisterWindowMessage	登录一条定制的消息，使之对系统唯一

新结构	描述
FINDREPLACE	上述函数所需的结构

源代码：SRCHTEXT. C

```
/* * SRCHTEXT. C * * * * * * * * * * * * * * * * * * * * * * * * */
```

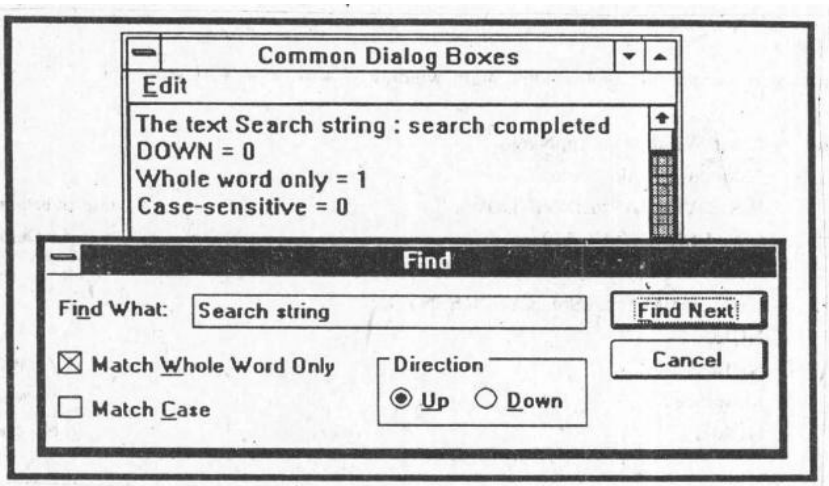


图 12.10

```
/* * Searches for and replaces text * */
/* * * *
```

```
        }
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow(szAppName,                          // Window class
            "Standard Dialog Boxes",                        // Window name
            WS_OVERLAPPEDWINDOW,                    // Use overlapped window style
            CW_USEDEFAULT,0,                                // Default x and y pos.
            GetSystemMetrics(SM_CXSCREEN)/3,                // Initial width
            GetSystemMetrics(SM_CYSCREEN)/2,                // Initial height
            NULL,                                           // Parent window
            NULL,                                           // Window menu used
            hInstance,                                      // Application instance
            NULL);                                          // No creation parameters

    if(hWnd == NULL)                                // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,               // Display error
            szString,sizeof(szString));
        MessageBox(NULL,szString,
            NULL,MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );                    // Make window visible
    UpdateWindow( hWnd );                           // Update window

    while (GetMessage(&msg,NULL,0,0))               // Message reading
    {
        TranslateMessage(&msg);                     // Message translation
        DispatchMessage(&msg);                  // Send message to Windows
    }
    return (int)msg.wParam;                 // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                               // Main window class

    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );     // Mouse cursor
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);    // Default icon
    wcClass.lpszMenuName = (LPSTR)"DlgMenu";            // Menu name
    wcClass.lpszClassName = szAppName;                  // Window class
    wcClass.hbrBackground =
            (HBRUSH)GetStockObject(WHITE_BRUSH);        // White background
    wcClass.hInstance = hInstance;                      // Instance
    wcClass.style = CS_VREDRAW | CS_HREDRAW;        // Horizontal and vertical redraw
    wcClass.lpfnWndProc = StdDlgWndProc;                // Window function
```

```
    wcClass.cbClsExtra = 0 ;                                    // No extra bytes
    wcClass.cbWndExtra = 0 ;                                    // No extra bytes

    if (! RegisterClass( &wcClass ) )                           // Register window class
        return FALSE;                                           // Return FALSE if registration fails
    return TRUE;                                                // If registration is successful,
}                                                               // Return TRUE

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,                        // Window handle
            unsigned message,                                   // Message type
            WORD wParam,                                        // Message dependent 16 bit value
            LONG lParam )                                       // Message dependent 32 bit value
{
    static HWND hEdit;                                          // Window handle
    LPFINDREPLACE lpSrch;                                       // Srch pointer
    static char szSrchText[TEXTLEN],                            // Search string
            szRplcText[TEXTLEN];                                // Replace string
            char szDOutput[3 * TEXTLEN]= "The text ";           // Starting string
            char szBool[TEXTLEN];                               // Boolean text
    BOOL fDown,fWord,fCase;
    DWORD dwFlags;

    switch (message)
    {
        case WM_CREATE:                                         // Create window
            hEdit = CreateWindow("Edit",                        // Window class (Edit)
                    NULL,                                       // Window name
                    WS_CHILD |                                  // Child window
                    WS_VISIBLE |                                // Initially visible window
                    WS_HSCROLL |                                // Horizontal scroll bar
                    WS_VSCROLL |                                // Vertical scroll bar
                    WS_BORDER |                                 // Border on window
                    ES_LEFT |                                   // Align text flush left
                    ES_MULTILINE,                               // Multiple-line edit control
                    0,0,                                        // Default x and y pos.
                    0,0,                                        // Default width and height
                    hWnd,                                       // Parent window
                    100,                                        // Menu or child window i. d.
                    hInst,                                      // Instance
                    NULL);                                      // LP parameter (none)

            wSrchMsg = RegisterWindowMessage((LPSTR)FINDMSGSTRING);
            memset(&fr,0x00,sizeof(FINDREPLACE));
            fr.lStructSize = sizeof(FINDREPLACE);               // Structure size
            fr.hwndOwner = hWnd;                                // Window owner
            fr.wFindWhatLen = TEXTLEN;                          // Text length
            fr.lpstrFindWhat = szSrchText;                      // Search text
```

```
        fr.lpstrReplaceWith = szRplcText;                    // Replace text
        fr.wReplaceWithLen = TEXTLEN;                        // Text length
        break;
    case WM_SIZE:                                            // Client area size
        MoveWindow(hEdit,                                    // Edit window
                0,0,                                         // New x & y positions
                LOWORD(lParam),                              // New window width
                HIWORD(lParam),                              // New window height
                TRUE);                                       // Do repaint window
        break;
    case WM_COMMAND:                                         // Commands
        switch(wParam)
        {
            case IDM_SRCH:                                   // Search selected
                fr.Flags = FR_DOWN;                          // Direction flag DOWN
                FindText(&fr);
                break;
            case IDM_RPLC:                                   // Replace selected
                fr.Flags = FR_WHOLEWORD |                    // Whole word flag
                        FR_MATCHCASE;                        // Match case flag
                ReplaceText(&fr);
                break;
            default:                                         // Send other messages
                return (DefWindowProc(hWnd,message,wParam,lParam));
                break;
        }
        break;
    case WM_SETFOCUS:                                        // Set window focus
        SetFocus(hEdit);
        break;
    case WM_DESTROY:                                         // Exit
        PostQuitMessage(0);
        break;
    default:                                                 // Send other messages
        if ( message == wSrchMsg)
        {
            lpSrch = (LPFINDREPLACE) lParam;
            dwFlags = lpSrch->Flags;
            fDown = (dwFlags & FR_DOWN ? TRUE : FALSE);      // Up or down?
            fCase = (dwFlags & FR_MATCHCASE ? TRUE : FALSE); // Match case?
            fWord = (dwFlags & FR_WHOLEWORD ? TRUE : FALSE); // Whole word?
            strcat(szDOutput,szSrchText);

            if (dwFlags & FR_FINDNEXT)                       // End Search
                strcat(szDOutput," : search completed\r\n");
            if (dwFlags & FR_REPLACE)                        // End Replace
            {
                strcat(szDOutput," was replaced with ");
                strcat(szDOutput,szRplcText);
                strcat(szDOutput,"\r\n");
            }
```

```
                if (dwFlags & FR_REPLACEALL)                    // End Replace All
                {
                    strcat(szDOutput," was completely replaced with ");
                    strcat(szDOutput,szRplcText);
                    strcat(szDOutput,"\r\n");
                }
                if ((dwFlags & FR_REPLACEALL) |
                    (dwFlags & FR_REPLACE) |
                    (dwFlags & FR_FINDNEXT))
                {                                               // Display option status
                    wsprintf(szBool,"DOWN = %d\r\n"
                        "Whole word only = %d\r\n"
                        "Case-sensitive = %d",
                        fDown,
                        fWord,
                        fCase);
                    strcat(szDOutput,szBool);
                    SetWindowText(hEdit,szDOutput);
                }
            }
            return (DefWindowProc( hWnd,message,wParam,
                    lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:SRCHTEXT. DEF

```
NAME SrchText
DESCRIPTION 'Dialog boxes for text search and replacement'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:SRCHTEXT. RC

```
#include "srchtext. h"
#include "windows. h"

DlgMenu MENU
BEGIN
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Search…", IDM_SRCH
```

```
        MENUITEM "&Replace…", IDM_RPLC
    END
END

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件:SRCHTEXT. H

```
#include <windows. h>
#include <string. h>
#include <commdlg. h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define TEXTLEN 90
#define IDM_SRCH 50
#define IDM_RPLC 51

char szString[128];                                    /* Variable for loading resource text */
char szAppName[20];                                    /* Window class name */
HANDLE hInst;
FINDREPLACE fr;
WORD wSrchMsg;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:SRCHTEXT. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS=-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ *. rc

RESFILES =
H = SRCHTEXT. H
LIBS_EXT = commdlg. lib

all: SRCHTEXT. EXE

SRCHTEXT. RES: SRCHTEXT. RC $ (RESFILES)

SRCHTEXT. OBJ: SRCHTEXT. C $ (H)

SRCHTEXT. EXE: SRCHTEXT. OBJ SRCHTEXT. RES SRCHTEXT. DEF
    link /NOD/A:16 SRCHTEXT,SRCHTEXT,nul,libw slibcew $ (LIBS_EXT),SRCHTEXT. def
    rc SRCHTEXT. res SRCHTEXT. exe
```

像下面这样运行 nmake 可产生应用程序 SRCHTEXT：

```
nmake srchtext. mak
```

在链接程序时必须使用输入库 COMMDLG. LIB，因为它包含着动态链接库 COMMDLG. DLL 的交叉索引。同时还要在程序中包含文件 COMMDLG. H，因为它定义了 FindReplace 结构和它的两个相关函数原型。

虽然函数 FindText 和 ReplaceText 返回一个标识被调用的对话框的句柄，但是在程序中并不需要这个句柄，因此，返回值没有被接收。

12.5 字模选择

12.5.1 文件 FONT. DLG

为了使字处理应用程序的用户可以选择字模，Windows 3.1 包含有像通用对话框一样的字模选择窗口。

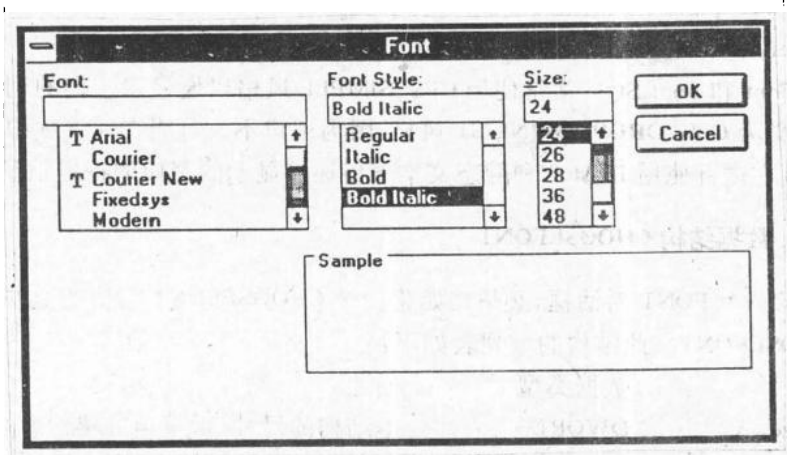


图 12.11

上图说明对话框是经常被使用的，虽然其外观可以像标准的 Open 对话框一样任意修改。字模选择对话框定义在文件 FONT. DLG 中。定义中还包含了一些上图中看不见的附加控件。改变标志域，便可得到另外两个按钮，一个用来提供帮助，另一个用来增加新字模。另外，还有一个标志位被用来清除对话框中的所有按钮和其他元素。

```
FORMATDLGORD31 DIALOG 13,54,264,147
STYLE DS_MODALFRAME|WS_POPUP |WS_CAPTION| WS_SYSMEU
CAPTION "Font"
FONT 8,"Helv"
BEGIN
    LTEXT "&Font:",stc1,6,3,40,8
    COMBOBOX cmbl,6,12,94,54,CSS_SIMPLE|CBS_AUTOHSCROLL|CBS_SORT|WS_VSCROLL
        |WS_TABSTOP|CBS_HASSTRINGS|CBS_OWNERORAWFIXED
    LTEXT "Font St&ylee:",stc2,108,3,44,8
    COMBOBOX cmb2,108,12,64,54,CBS_SIMPLE | WS_VSCROL | WS_TABSTOP |
```

```
    LTEXT "&Size:",stc3,179,3,30,8
    COMBOBOX cmb3,179,12,32,54,CBS_SIMPLE | WS_VSCROLL | WS_TABSTOP |
        CBS_HASSTRINGS | CBS_OWNERDRAWFIXED | CBS_SORT
    GROUPBOX "Effects",grp1,6,72,84,61,WS_GROUP
    CONTROL "Stri&keout",chx1,"Button,BS_AUTOCHECKBOX |
        WS_TABSTOP,10,82,49,10
    CONTROL "&Underline",chx1,"Button,BS_AUTOCHECKBOX,10,94,51,10
    LTEXT "&Color:",stc4,10,106,30,8
    COMBOBOX cmb4,10,116,75,50,CBS_DROPDOWNLIST | CBS_OWNERDRAWFIXED |
        CBS_AUTOHSCROLL|CBS_HASSTRINGS|WS_BORDER|WS_VSCROLL|WS_TABSTOP
    DEFPUSHBUTTON "OK",IDOK,218,6,40,14,WS_GROUP
    PUSHBUTTON "Cancel",IDCANCEL,218,40,40,14,WS_GROUP
    PUSHBUTTON "&Help",pshHelp,218,57,40,14,WS_GROUP
    GROUPBOX "Sample",grp2,98,72,160,49,WS_GROUP
    CTEXT "",stc6,98,124,160,20,SS_NOPREFIX | NOT WS_GROUP
    CTEXT "AaBbYyZz",stc5,104,81,149,37,SS_NOPREFIX | NOT WS_VISIBLE
END
```

对话框的名字 FORMATDLGORD31 是一个 ID 数(而不是一个字符串),它定义在文件 COMMONDLG.H 中:

```
#define FORMATDLORD31 1543
```

组合框 Font 和 Font Style 常常使用 CBS_SIMPLE 风格以便显示它们的列表。但组合框 Color 却被定义为 CBS_DROPDOWNLIST 风格,因为用户不大可能在此选项中选一个新值。

Windows 在内部使用 ID stc6 和静态文本来传递被显示的字模的特定信息。

12.5.2 数据结构 CHOOSEFONT

为了显示一个 FONT 对话框,必须初始化一个 CHOOSEFONT 结构变量并把它的地址传给函数 CHOOSEFONT。此结构的域列表如下:

域名	数据类型	描述
lStructSize	DWORD	本结构的尺寸(以字节为单位)
hwndOwner	HWND	有效窗口的句柄或 NULL
hDC	HDC	如果设置了标志 CF_PRINTERFONTS 或 CF_BOTH,则此值为其字模被列出的打印机设备环境句柄
lpLogFont	LPLOGFONT	返回值:字模的形状
iPointSize	int	返回值:以 1/10 点为单位的被选字模之尺寸
Flags	DWORD	各个指示符的组合
rgbColors	DWORD	被选中的文本颜色
lCustData	DWORD	由参数 lParam 传给 hook 过程的应用程序定义的数据
lpTemplateName	LPSTR	未使用通用对话框时对话框的名字
hInstance	HANDLE	仅在标志 CF_ENABLETEMPLATE 被设置时有用
lpszStyle	LPSTR	若设置了标志 CF_USESTYLE,则为"风格"数据

		的缓冲区
nFontType	WORD	被选字模类型
nSizeMin	int	最小的可用点尺寸
nSizeMax	int	最大的可用点尺寸

并非所有的域都要在对话框中列出。填充了下面四个域后便可成功地调用此对话框：

```
chf. lStructSize = sizeof (CHOSEFONT);
chf. lpLogFont = &lf;
chf. FLags = CF_SCREENFONTS;
chf. nFontType = SCREEN_FONTTYPE;
ChooseFont(&chf);
```

数据类型 LogFont 的变量 lf 包含了用户选中后并用 OK 按钮确认了的新字模的信息。

当指定了字模类型和 CF_SCREENFONTS 特征之后，所有的屏幕可用字模(除了打印机字模)就都列于组合框中了。

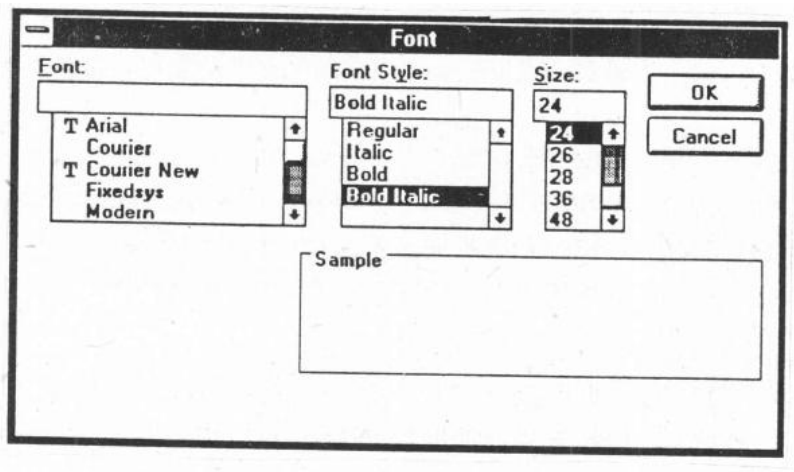


图 12.12

如果在机器上调试一下上面那四行程序，就会发现这个对话框(见上图)还不能作为最后版本。

比如，用户应能改变文本颜色，或显示刺穿字模或下划线。此时必须把 CF_SCREENFONTS 组合到特征 CF_EFFECTS 上。

```
chf. Flags = CF_SCREENFONTS|CF_EFFECTS;
```

更进一步，当对话框被首次调用时，必须在组合框中预置几个特定的入口，以便用户可以立刻看到 Sample 框中的结果。这需要两个更深入的步骤。必须加上 CF_INITTOLOGSTRUCTFONT 标志，并像下面几行这样给变量 lf 的域赋值：

```
if. lfHeight = 32
if. lfItalic = 1;
if. lfUnderline = 1;
strcpy(lf. lfFaceName,"Modern");
```

设置的入口根据它们的设置而显示在对话框上，组合框“Underline”被屏蔽掉了(参看图示)，在 Smaple 框中显示了被选字模的样板。这种字模的尺寸被内部转换为 24 点阵。因此 iPointSize 域的值为 240，因为它的单位为 1/10 点。

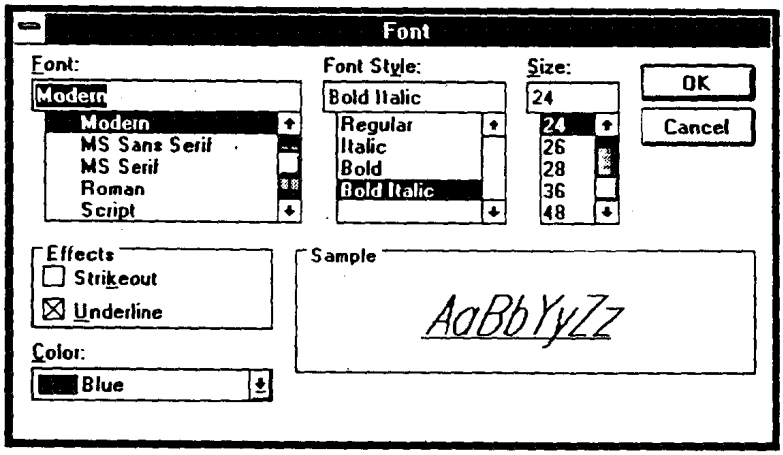


图 12.13　设置字模的 Font 对话框

因为 rgbColors 域还未设置，它的值为 0，所以样板文本的颜色为黑色。下面的命令将置颜色为红色。

```
chf. rgbColors = RGB(255,0,0);
```

通过设置标志 CF_LIMITSIZE 可以使 CHOOSEFONT 结构的 nSizeMin 和 nSizeMax 域有效。这两个域分别定义了在“Size”组合框中显示的最小和最大字模尺寸。

```
chf. Flags = CF_SCREENFONTS | CF_EFFECTS | CF_ITITTOLOGFONTSTRUCT CF_ITMITSIZE
chf. nSizeMin = 24;
chf. nSizeMax = 48;
```

为了列出某种打印机的可用字模类型，必须把打印机的 DC 句柄传递给 hDC 域，这可以用函数 CreateDC 来实现。此外，还要把常量 PRINTER_FONTTYPE 赋给 nFontType 域并设置标志 CF_PRINTERFONTS。

如果屏幕上并没有出现对话框，而是出现了一个消息窗口，提示还未设置字体，其原因可能是 hDC 的句柄不对或根本没有。当标志 CF_PRINTERFONTS 被改为 CF_BOTH 时，可用的屏幕字模都被显示出来。在 Sample 框下还将显示出一则通知：打印机将使用最新被选择的字模。

对于非无极字模类型，在相应的文件 FON 中没有实现斜体和下划线风格，而是在需要的时候使用图形设备界面(GDI)来模拟。通过设置标志 CF_NOSIMULATIONS 可以看到这些。

下表描述了所有标志的值：

位	含义
OF_APPLY	Apply 按钮被显示
CF_ANSIONLY	只显示 Windows 字符使用的字模
CF_BOTH	显示打印机和屏幕字模(CF_SCREENFONTS 和 CF_PRINTERFONTS 的组合)
CF_TTONLY	只显示 TrueType 字模
CF_EFFECTS	显示改变文本颜色和选择下划线的域

CF_ENABLEHOOK	lpfnHook 域指示的 hook 过程被允许
CF_ENABLETEMPLATE	lpTemplateName 域指示的对话框被允许
CF_ENABLETEMPLATEHANDLE	hInstance 定义的数据段包含一个预先调入的对话框
CF_FIXTEDPITCHONLY	只显示固定间距字模(不显示比例字模)
CF_FORCEFONTEXIST	当用户选中不存在的字模时显示错误信息
CF_INITTOLOGFONTSTRUCT	lpLogFont 域指示的 LOGFONT 结构,用于初始化
CF_LIMITSIZE	nSizeMin 域和 nSizeMax 域中可选的字模尺寸的范围
CF_NOSIMULATIONS	不允许模拟的 GDI 字模
CF_NOVECTORFONTS	不显示向量字模
CF_PRINTERFONTS	只显示 hDC 域指示的打印字模
CF_SCALABLEONLY	只显示无极字模(向量字模、打印机无极字模、TrueType 和其他无极字模)
CF_SCREENFONTS	只显示屏幕可用字模
CF_SHOWHELP	显示 Help 按钮
CF_USESTYLE	lpszStyle 域(例如 bold)中的字模风格被允许
CF_WYSIWYG	显示既可用于屏幕又可用于打印机的字模

12.5.3 字模选择示例

下面的程序用 Font 对话框中选择的字模显示一小段文本。初始时字模被选为 Modern。

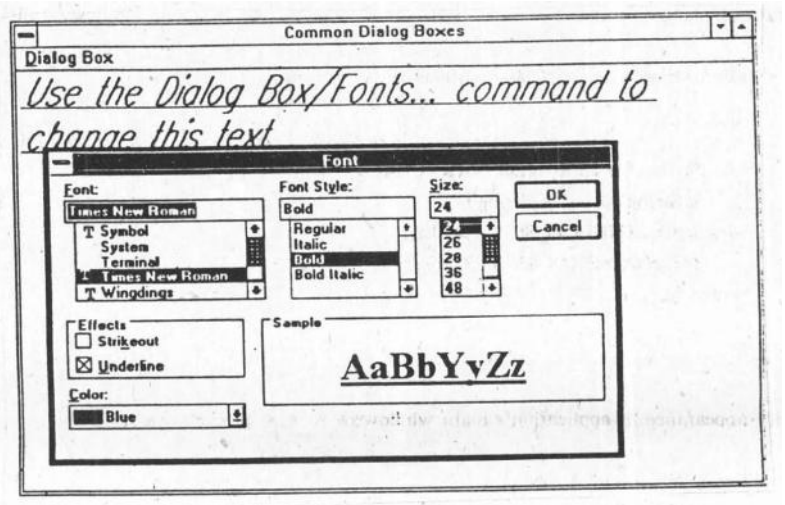


图 12.14

新函数	描述
ChooseFont	显示对划框以选择字模

新结构	描述
CHOOSEFONT	以上函数所需的结构

源代码:FONTBOX.C

```
/* * FONTBOX.C * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates access to default font dialog box * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "fontbox.h"                                    // Include FONTBOX.H header file

#if _MSC_VER >= 700                                     // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                          // higher? Disable warning 4028
#endif

/* * WinMain: Main window function for all Windows applications * * * * * * /

int PASCAL WinMain(HANDLE hInstance,                                  // Current instance
        HANDLE hPrevInstance,                                         // Previous instance
        LPSTR lpszCmdLine,                                  // Long ptr - string after prg
        int nCmdShow)                                       // App. window's appearance
{
    MSG msg;                                                          // Message handle
    HWND hWnd;                                                        // Window handle
    int nError;                                                       // Error integer

    strcpy(szAppName,"FontBox");                                      // Application name
    hInst = hInstance;                                                // Instance handle

    if (! hPrevInstance)                                    // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                                                      // If initialization fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;
        }
    }

/* * Specify appearance of application's main window * * * * * * * * * /

    hWnd = CreateWindow(szAppName,                          // Window class name
        "Standard Dialog Boxes",                            // Window caption
        WS_OVERLAPPEDWINDOW,                    // Overlapped window style
        CW_USEDEFAULT,0,                                    // Upper-left x,y pos.
        CW_USEDEFAULT,0,                                    // Initial x,y sizes
        NULL,                                               // No parent window
        NULL,                                               // Window menu used
        hInstance,                                          // Application instance
        NULL);                                  // No creation parameters
```

```
    if(hWnd == NULL)                                    // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,                    // Display error
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );                        // Make window visible
    UpdateWindow( hWnd );                               // Update window

    while (GetMessage(&msg,NULL,0,0))                   // Message reading
    {
        TranslateMessage(&msg);                         // Message translation
        DispatchMessage(&msg);                          // Send message to Windows
    }
    return (int)msg.wParam;                             // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)
{

/* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                   // Main window class

    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );     // Mouse cursor
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);    // Default icon
    wcClass.lpszMenuName = (LPSTR)"DlgMenu";            // Menu name
    wcClass.lpszClassName = szAppName;                  // Window class
    wcClass.hbrBackground =
        (HBRUSH)GetStockObject(WHITE_BRUSH);            // White background
    wcClass.hInstance = hInstance;                      // Instance
    wcClass.style = CS_VREDRAW | CS_HREDRAW;            // Horz.-vert. client area redraw
    wcClass.lpfnWndProc = StdDlgWndProc;                // Window function
    wcClass.cbClsExtra = 0 ;                            // No extra bytes
    wcClass.cbWndExtra = 0 ;                            // No extra bytes

    if (! RegisterClass( &wcClass ) )                   // Register window class
        return FALSE;                                   // Return FALSE if registration fails
    return TRUE;                                        // If registration is successful,return TRUE
}

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,
        unsigned message,
        WORD wParam,
        LONG lParam )
```

```
{
    static CHOOSEFONT chf;                                          // Font selection
    static LOGFONT lf;                                              // Logical font
    static HFONT hFont;
    HFONT hFontOld;                                                 // Old font
    HDC hDC;                                                        // Device context handle
    PAINTSTRUCT ps;                                                 // Paint structure
    char szText[] = "Use the Dialog Box/Fonts··· command to change this text";
    RECT rc;                                                        // Rectangle

    switch (message)                                                // Process messages
    {
        case WM_CREATE:                                             //.Create window
            memset(&lf,0x00,sizeof(LOGFONT));                       // Assign default font,
            lf.lfHeight = -32;                                      // height,
            lf.lfWeight = FW_BOLD;                                  // styles (bold,
            lf.lfItalic = 1;                                        // italic and
            lf.lfUnderline = 1;                                     // underline)
            lf.lfCharSet = OEM_CHARSET;                             // Assign character set
            strcpy(lf.lfFaceName,"Modern");                         // and typeface (Modern)
            hFont = CreateFontIndirect( &lf);                       // Font creation

            chf.lStructSize = sizeof (CHOOSEFONT);                  // Structure size
            chf.hwndOwner = hWnd;                                   // Window owner
            chf.hDC = NULL;                                         // Handle device context = zero
            chf.lpLogFont = &lf;                                    // Set font flags
            chf.Flags = CF_SCREENFONTS | CF_EFFECTS |
            CF_INITTOLOGFONTSTRUCT | CF_LIMITSIZE;
            chf.rgbColors = RGB(0,0,255);                           // Set color to blue
            chf.lCustData = 0L;
            chf.lpfnHook = NULL;
            chf.lpTemplateName = (LPSTR) NULL;
            chf.hInstance = (HANDLE) NULL;
            chf.lpszStyle = (LPSTR) NULL;
            chf.nFontType = SCREEN_FONTTYPE;
            chf.nSizeMin = 12;                                      // Minimum font size
            chf.nSizeMax = 48;                                      // Maximum font size
            break;
        case WM_COMMAND:                                            // Commands
            switch(wParam)
            {
                case IDM_FONT:                                      // Fonts··· selected
                    if (ChooseFont(&chf))
                    {
                        DeleteObject(hFont);
                        hFont = CreateFontIndirect( &lf);
                        InvalidateRect(hWnd,NULL,TRUE);
                    }
                    break;
                default:
                    return (DefWindowProc(hWnd,message,
```

```
                                wParam,lParam));
                        break;
                }
                break;
        case WM_DESTROY:                                                    // Exit
                PostQuitMessage(0);
                break;
        case WM_PAINT:                                                      // Paint window
                hDC = BeginPaint(hWnd,&ps);
                hFontOld = SelectObject(hDC,hFont);
                SetTextColor(hDC,chf.rgbColors);
                GetClientRect(hWnd,&rc);
                DrawText(hDC,szText,-1,&rc,DT_WORDBREAK);
                SelectObject(hDC,hFontOld);
                EndPaint(hWnd,&ps);
                break;
        default:
                return(DefWindowProc(hWnd,message,wParam,lParam));
                break;
    }
    return(0L);
}
```

模块定义文件:FONTBOX.DEF

```
NAME FontBox
DESCRIPTION 'Font selection through a dialog box'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:FONTBOX.RC

```
#include "fontbox.h"
#include "windows.h"

DlgMenu MENU
BEGIN
    POPUP "&Dialog Box"
    BEGIN
        MENUITEM "&Fonts…", IDM_FONT
    END
END

STRINGTABLE
```

```
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件:FONTBOX.H

```
#include <windows.h>
#include <string.h>
#include <commdlg.h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define IDM_FONT 12

char szString[128];                                    /* Variable for loading resource text */
char szAppName[20];                                    /* Window class name */
HANDLE hInst;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:FONTBOX.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = FONTBOX.H
LIBS_EXT = commdlg.lib

all: FONTBOX.EXE

FONTBOX.RES: FONTBOX.RC $(RESFILES)

FONTBOX.OBJ: FONTBOX.C $(H)

FONTBOX.EXE: FONTBOX.OBJ FONTBOX.RES FONTBOX.DEF
    link /NOD/A:16 FONTBOX,FONTBOX,nul,libw slibcew $(LIBS_EXT),FONTBOX.def
    rc FONTBOX.res FONTBOX.exe
```

像下面这样运行 MAKE 便可产生应用程序 FONTBOX:

```
nmake fontbox.mak
```

用来初始化对话框的 LogFont 结构完全像上面所描述的那样进行控制。为了提供使用 ChooseFont 的域,只需很少的修改。函数 SetTextColor 把 Font 对话框的颜色改为新的字符颜色。

在头文件 FONTBOX.H 中必须包含定义结构 CHOOSEFONT 的文件 COMMDLG.H。

第十三章　GDI 简介

当我们第一次向窗口的客户区域里写东西时，就在使用图形设备界面(GDI)函数了。在第三章，我们已初步介绍了 Windows 中绘图工具的使用，并给出了一些图形输出函数，现在要对这一主题作更为形式化的概括。

13.1　GDI 基础

Windows 中的图形基本上是由 GDI.EXE 中的函数处理的(尽管一些绘图函数实际上在 USER.EXE 文件中有入口点)。GDI.EXE 模块调用各种驱动程序文件中的例程，如处理视频显示屏幕的 .DRV 以及其他控制打印机或绘图仪的 .DRV 驱动程序文件。视频驱动程序访问视频显示器硬件。不同的视频显示适配器和打印机需要不同的驱动程序文件。

GDI 系统的构造使 Windows 可以确定驱动程序能处理什么和不能处理什么。例如，如果视频硬件包括一个可以画椭圆的图形协处理器，则 GDI 可以利用它；否则，GDI 模块必须自己计算椭圆上的点，并将点传给驱动程序。

由于大量不同的显示设备可以连接到 PC 兼容机上，所以 GDI 的一个主要目标是在输出设备(如视频显示、打印机和绘图仪)上支持与设备无关的图形。Windows 程序必须能毫无困难地在 Windows 所支持的图形输出设备上运行。为了达到这个目的，GDI 提供了能使程序与不同输出设备的特征无关的功能。从这种角度上看，它与那些与设备无关的图形程序设计语言相像。但 Windows 还具有对像素点级操作的强有力的支持。

所有图形输出设备划分为两大类：光栅设备和向量设备。大部分 PC 输出设备是光栅设备，它们以点的形式来表示图像。这一类中包括视频显示适配器、点阵打印机和激光打印机。向量设备用线条来画图像，通常只限于绘图仪。

虽然大多数视频显示适配器和计算机是光栅设备，但大多数图形界面语言却仅基于向量设备，这就意味着使用这些图形语言的程序是处于一个与硬件分离的抽象级上。输出设备用像素点来表示图形，但是程序不是以像素点为基础与界面交互。程序员在顺理成章地将 Windows GDI 用作高级向量绘图系统的同时，又可以把它作为相对低级的像素点操作工具来使用。

在这一方面，Windows GDI 与其他图形界面语言相比，就像 C 与其他程序设计语言相比一样。C 由于在不同操作系统和不同环境之间的高度可移植性而声名卓著；同时，C 还允许程序员执行低级的系统功能，这在其他高级语言中是不可能的。正像 C 有时候能认为是一种“高级汇编语言”一样，我们也可以认为 GDI 是图形设备硬件的高级界面。

读者已经知道，在缺省情况下，Windows 使用一种基于像素点的坐标系统。大多数其他

图形语言使用一种"虚拟的"坐标系统，其水平和垂直轴范围为从 0 到 32767。有些图形语言不允许使用像素点坐标，但 Windows GDI 允许在程序中使用两种中的任一种。程序可以使用一种虚拟的坐标系统，使程序与硬件分离。也可以使用设备坐标系统，使程序与硬件紧密相关。

有些程序员认为，一旦开始以像素点为基础编程，就抛弃了与设备无关的原则。我们已经看到，这种想法并不完全正确。诀窍在于以一种与设备无关的方式使用像素点。这就要求，图形界面语言为程序提供确定设备硬件特性的功能，并做适当的调整。例如，我们已经频繁地使用标识系统字模字符的像素点大小来给屏幕上的文本留空。这种方法使得程序能适应于具有不同分辨率、文本尺寸及纵横比的不同显示适配器。在文中，读者还会看到确定显示器大小的其他方法。

Windows 既可以在单色显示器上运行，也可以在彩色显示器上运行。如果愿意，程序员可以在编写程序时不考虑颜色问题。如果在程序中使用了颜色，而程序又在单色显示适配器上运行，Windows 将使用灰色阴影来代表每种颜色。当然，程序员也可以在程序中确定在特定显示设备上可以使用多少种颜色，从而最大限度地利用硬件的特性。

当然，与 C 程序在其他机器上运行时会出现移植性问题一样，在 Windows 程序中，也不可避免地要涉及与设备相关的特性。这是不完全与硬件分离的代价之一。

读者还会意识到 Winows GDI 的局限性，GDI 现在还不能做用户想让图形界面做到的所有事情。虽然用户可以在显示器上移动图形对象，但总的来说，GDI 是一个静态显示系统，并不真正支持动画。GDI 不直接支持三维表示和物体的旋转。例如，在画椭圆时，椭圆轴必须平行于垂直和水平坐标轴。有些图形语言使用浮点数作虚拟坐标，但 Windows 为了性能上的考虑，还是使用 16 位符号整数作坐标。

13.2 设备环境

当我们想在图形输出设备（如屏幕或打印机）上画图时，必须先获取设备环境（DC）的句柄。在向程序提供这个句柄的同时，Windows 许可该程序使用该设备。然后，程序员可以将这个句柄作为 GDI 函数的参数，来标识用以绘图的设备。

设备环境包含许多当前"属性"，它们确定 GDI 函数在设备上如何工作。这些属性允许 GDI 函数的参数只需包含开始坐标、大小以及 Windows 在设备上显示对象所必需的其他数据，其他一概可以不要。例如，在调用 TextOut 时，只需在函数中指定设备环境句柄、开始坐标、文本和文本长度，无需指定字模、文本颜色、文本后面背景色和字符间距，因为这些属性都是设备环境的组成部分。如果想改变某个属性，可以调用改变设备环境中某属性的函数。其后的 TextOut 调用将使用改变了的属性。

13.2.1 获取设备环境的句柄

Windows 提供了几种获取设备环境句柄的方法。如果在处理一条消息时获取了设备环境句柄，应该在退出窗口函数之前释放它（或删除它）。一旦释放了句柄，它就不再有效。

获取设备环境句柄并释放它的最常见的方法涉及到，在处理 WM_PAINT 消息时，使用 BeginPaint 和 EndPaint 调用：

```
hdc = BeginPaint(hwnd,&ps);
    [其他程序行]
EndPaint(hwnd,&ps);
```

变量ps是类型为PAINTSTRUCT的结构。该结构的hdc字段是BeginPaint返回的设备环境句柄。PAINTSTRUCT结构又包含一个名为rcPaint的RECT(矩形)结构,rcPaint定义一个包围窗口客户区域的合法范围的矩形。使用从BeginPaint获得的设备环境句柄,只能在这个区域内绘图。BeginPaint调用确认了该区域的有效性。

Windows程序还可以在处理非WM_PAINT消息时获取设备环境句柄:

```
hdc = GetDC(hwnd);
    [其他程序行]
ReleaseDC(hwnd,hdc);
```

这个设备环境适用于其句柄为hwnd的窗口的客户区域。这些调用和BeginPaint与EndPaint的组合之间的基本区别是,用从GetDC返回的句柄可以在整个客户区域上绘图。当然,GetDC和ReteaseDC都不确认客户区域中可能的无效区域。

Windows程序还可以获取适用于整个窗口(而不仅限于窗口的客户区域)的设备环境句柄:

```
hdc = GetWindowDC(hwnd);
    [其他程序行]
  ReleaseDC(hwnd,hdc);
```

这个设备环境除了客户区域之外,还包括窗口的标题条、菜单、滚动条和框架。GetWindowDC函数很少使用。如果想用它试试,则必须捕获WM_NCPAINT("非客户绘制")消息,它阻止Windows在窗口的非客户区域上绘图。

BeginPaint、GetDC和GetWindowDC获得的设备环境都与某个特定窗口相关。通过调用CreateDC,可以获得整个显示的设备环境:

```
hdc = CreateDC(lpszDriver,lpszDevice,lpszOutput,lpData);
    [其他程序行]
DeleteDC(hdc);
```

这个函数可获取一个设备环境句柄,允许我们在窗口的客户区域之外写内容:

```
hdc = CreateDC("DISPLAY",NULL,NULL);
```

在窗口之外写内容一般是不合适的,但对于一些不同寻常的应用程序来说,这样做又很方便,程序员也可通过在调用GetDC时使用一个NULL参数,来获取整个屏幕的设备环境句柄;不过这一事实没有载入文档。

在后面,我们将使用CreateDC函数来获取一个打印机设备环境的句柄:

```
hdcPrinter = CreateDC("IBMGRX","IBM Graphics","LPT1:",NULL);
```

当然,我们将不会在程序中包含具体的打印机名。程序可以从WIN.INI中获取一个"信息环境"的句柄。其参数与CreateDC函数相同。

```
hdcInfo = CreatIC(lpszDriver,lpszDevice,lpszOutput,lpData);
    [其他程序行]
DeleteDC(hdcInfo);
```

程序员不能用这个信息环境句柄往设备上写内容。

我们可获取一个内存设备环境，以对一些位图进行操作，内存设备环境总是创建为与已存在的设备环境兼容：

```
hdcMem = CreateCompatibleDC(hdc);
    [其他程序行]
Delete(hdcMem),
```

在开始获取内存设备环境时，它所代表的显示表面只含有一个像素点。

我们还会用到“元文件”。元文件是一些 GDI 调用的集合，以二进制形式编码。通过获取一个元文件设备环境，即可创建元文件：

```
hdcMeta = CreateMetaFile(lpszFilename)
    [其他程序行]
hmf = CloseMetaFile(hdcMeta);
```

在元文件设备环境有效期间，任意用 hdcMeta 所做的 GDI 调用都变成元文件的一部分。在调用 CloseMetaFile 之后，设备环境句柄变为无效。函数返回一个指向元文件(hmf)的句柄。

13.2.2 获取设备环境信息

设备环境通常是指物理显示设备，如视频显示器和打印机。通常，程序需要获取有关该设备的信息，包括显示器的大小(单位为像素点或物理尺度)和颜色、容量。程序员可以通过调用 GetDeviceCaps(“获取设备能力”)函数来获取这些信息：

```
nValue = GetDeviceCaps(hdc,nIndex);
```

其中，nIndex 参数是 WINDOWS.H 中定义的 28 个标识符之一。例如，nIndex 为 HORZRES 时将导致 GetDeviceCaps 返回设备的宽度(以像素点为单位)；为 VERTRES 时将导致 GetDeviceCaps 返回设备的高度(单位为像素点)。如果 hdc 是屏幕设备环境的句柄，则这个信息与调用 GetSystemMetrics 获得信息的相同。如果 hdc 是打印机设备环境的句柄，则 GetDeviceCaps 返回打印机显示区域的高度和宽度，它们也以像素点为单位。

还可以用 GetDeviceCaps 来确定设备处理不同类型的能力。这对于视频显示器并不是很重要，但对于打印机却非常重要。例如，大多数绘图仪不能画位图图像，GetDeviceCaps 就可以做到这一点。

13.2.3 DEVCAPS1 程序

下面的 DEVCAPS1 程序显示了所有可以从 GetDeviceCaps 函数中获得的视频显示器或选中的打印机的信息。如果用户用 Windows 的控制面板程序改变当前的打印机，DEVCAPS1 就更新打印机信息。

```
# DEVCAPS1.MAK make file

devcaps1.exe : devcaps1.obj devcaps.obj devcaps1.def devcaps1.res
    $(WINLINK) devcaps1 devcaps,devcaps1,NUL,$(WINLIB),devcaps1
    rc -t devcaps1.res

devcaps1.obj : devcaps1.c devcaps1.h
    $(WINCC) devcaps1.c
```

```
devcaps.obj : devcaps.c
     $(WINCC) devcaps.c

devcaps1.res : devcaps1.rc devcaps1.h
     $(WINRC) devcaps1.rc

/*------------------------------------------------------
DEVCAPS1.C -- Displays Device Capability Information
------------------------------------------------------*/

#include <windows.h>
#include <string.h>
#include "devcaps1.h"

void DoBasicInfo (HDC,HDC,short,short) ;                    // in DEVCAPS.C
void DoOtherInfo (HDC,HDC,short,short) ;
void DoBitCodedCaps (HDC,HDC,short,short,short) ;

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
     LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "DevCaps" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = LoadIcon (NULL,IDI_APPLICATION) ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = szAppName ;
          wndclass.lpszClassName = szAppName ;

          ! RegisterClass (&wndclass) ;
     }

     hwnd = CreateWindow (szAppName,"Device Capabilities",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,CW_USEDEFAULT,
          CW_USEDEFAULT,CW_USEDEFAULT,
          NULL,NULL,hInstance,NULL) ;
```

```
    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam ;
}

HDC GetPrinterIC ()
{
    char szPrinter [64] ;
    char * szDevice, * szDriver, * szOutput ;

    GetProfileString ("windows","device","",szPrinter,64) ;

    if (NULL ! = (szDevice = strtok (szPrinter,"," )) &&
        NULL ! = (szDriver = strtok (NULL, ",")) &&
        NULL ! = (szOutput = strtok (NULL, ",")))

    return CreateIC (szDriver,szDevice,szOutput,NULL) ;

    return NULL ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static short cxChar,cyChar,nCurrentDevice = IDM_SCREEN,
        nCurrentInfo = IDM_BASIC ;
    HDC hdc,hdcInfo ;
    HMENU hMenu ;
    PAINTSTRUCT ps ;
    TEXTMETRIC tm ;

    switch (message)
    {
        case WM_CREATE:
            hdc = GetDC (hwnd) ;
            SelectObject (hdc,GetStockObject (SYSTEM_FIXED_FONT)) ;

            GetTextMetrics (hdc,&tm) ;
            cxChar = tm. tmAveCharWidth ;
            cyChar = tm. tmHeight + tm. tmExternalLeading ;

            ReleaseDC (hwnd,hdc) ;
            return 0 ;
        case WM_COMMAND:
            hMenu = GetMenu (hwnd) ;
```

```
        switch (wParam)
        {
            case IDM_SCREEN:
            case IDM_PRINTER:
                CheckMenuItem (hMenu,nCurrentDevice,MF_UNCHECKED) ;
                nCurrentDevice = wParam ;
                CheckMenuItem (hMenu,nCurrentDevice,MF_CHECKED) ;
                InvalidateRect (hwnd,NULL,TRUE) ;
                return 0 ;
            case IDM_BASIC:
            case IDM_OTHER:
            case IDM_CURVE:
            case IDM_LINE:
            case IDM_POLY:
            case IDM_TEXT:
                CheckMenuItem (hMenu,nCurrentInfo,MF_UNCHECKED) ;
                nCurrentInfo = wParam ;
                CheckMenuItem (hMenu,nCurrentInfo,MF_CHECKED) ;
                InvalidateRect (hwnd,NULL,TRUE) ;
                return 0 ;
        }
        break ;
    case WM_DEVMODECHANGE:
        InvalidateRect (hwnd,NULL,TRUE) ;
        return 0 ;
    case WM_PAINT:
        hdc = BeginPaint (hwnd,&ps) ;
        SelectObject (hdc,GetStockObject (SYSTEM_FIXED_FONT)) ;

        if (nCurrentDevice == IDM_SCREEN)
            hdcInfo = CreateIC ("DISPLAY",NULL,NULL,NULL) ;
        else
            hdcInfo = GetPrinterIC () ;

        if (hdcInfo)
        {
            switch (nCurrentInfo)
            {
                case IDM_BASIC:
                    DoBasicInfo (hdc,hdcInfo,cxChar,cyChar) ;
                    break ;
                case IDM_OTHER:
                    DoOtherInfo (hdc,hdcInfo,cxChar,cyChar) ;
                    break ;
                case IDM_CURVE:
                case IDM_LINE:
                case IDM_POLY:
                case IDM_TEXT:
                    DoBitCodedCaps (hdc,hdcInfo,cxChar,cyChar,
                        nCurrentInfo - IDM_CURVE) ;
```

```
                    break ;
                }
                DeleteDC (hdcInfo) ;
            }

            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

/*------------------------------------------------------
DEVCAPS.C -- Display routines for DEVCAPS1 and DEVCAPS2
------------------------------------------------------*/

#include <windows.h>
#include <string.h>
#include <stdio.h>

typedef struct
{
    short nMask ;
    char *szMask ;
    char *szDesc ;
}
BITS ;

void DoBasicInfo (HDC hdc,HDC hdcInfo,short cxChar,short cyChar)
{
    static struct
    {
        short nIndex ;
        char *szDesc ;
    }
    info[] =
    {
        HORZSIZE, "HORZSIZE Width in millimeters:",
        VERTSIZE, "VERTSIZE Height in millimeters:",
        HORZRES, "HORZRES Width in pixels:",
        VERTRES, "VERTRES Height in raster lines:",
        BITSPIXEL, "BITSPIXEL Color bits per pixel:",
        PLANES, "PLANES Number of color planes:",
        NUMBRUSHES, "NUMBRUSHES Number of device brushes:",
        NUMPENS, "NUMPENS Number of device pens:",
        NUMMARKERS, "NUMMARKERS Number of device markers:",
        NUMFONTS, "NUMFONTS Number of device fonts:",
        NUMCOLORS, "NUMCOLORS Number of device colors:",
        PDEVICESIZE, "PDEVICESIZE Size of device structure:",
```

```
        ASPECTX, "ASPECTX Relative width of pixel:",
        ASPECTY, "ASPECTY Relative height of pixel:",
        ASPECTXY, "ASPECTXY Relative diagonal of pixel:",
        LOGPIXELSX, "LOGPIXELSX Horizontal dots per inch:",
        LOGPIXELSY, "LOGPIXELSY Vertical dots per inch:",
        SIZEPALETTE, "SIZEPALETTE Number of palette entries:",
        NUMRESERVED, "NUMRESERVED Reserved palette entries:",
        COLORRES, "COLORRES Actual color resolution:"
    } ;
    char szBuffer [80] ;
    short i ;

    for (i = 0 ; i < sizeof info / sizeof info [0] ; i++)
        TextOut (hdc,cxChar,(i + 1) * cyChar,szBuffer,
            sprintf (szBuffer,"%-40s%8d",info[i].szDesc,
            GetDeviceCaps (hdcInfo,info[i].nIndex))) ;
}

void DoOtherInfo (HDC hdc,HDC hdcInfo,short cxChar,short cyChar)
{
    static BITS clip [] =
    {
        CP_RECTANGLE, "CP_RECTANGLE", "Can Clip To Rectangle:"
    } ;

    static BITS raster [] =
    {
        RC_BITBLT, "RC_BITBLT", "Capable of simple BitBlt:",
        RC_BANDING, "RC_BANDING", "Requires banding support:",
        RC_SCALING, "RC_SCALING", "Requires scaling support:",
        RC_BITMAP64, "RC_BITMAP64", "Supports bitmaps >64K:",
        RC_GDI20_OUTPUT,"RC_GDI20_OUTPUT","Has 2.0 output calls:",
        RC_DI_BITMAP, "RC_DI_BITMAP", "Supports DIB to memory:",
        RC_PALETTE, "RC_PALETTE", "Supports a palette:",
        RC_DIBTODEV, "RC_DIBTODEV", "Supports bitmap conversion:",
        RC_BIGFONT, "RC_BIGFONT", "Supports fonts >64K:",
        RC_STRETCHBLT, "RC_STRETCHBLT", "Supports StretchBlt:",
        RC_FLOODFILL, "RC_FLOODFILL", "Supports FloodFill:",
        RC_STRETCHDIB, "RC_STRETCHDIB", "Supports StretchDIBits:"
    } ;

    static char * szTech [] = { "DT_PLOTTER (Vector plotter)",
        "DT_RASDISPLAY (Raster display)",
        "DT_RASPRINTER (Raster printer)",
        "DT_RASCAMERA (Raster camera)",
        "DT_CHARSTREAM (Character-stream,PLP)",
        "DT_METAFILE (Metafile,VDM)",
        "DT_DISPFILE (Display-file)" } ;
    char szBuffer [80] ;
    short i ;
```

```
    TextOut (hdc,cxChar,cyChar,szBuffer,
        sprintf (szBuffer,"%-24s%04XH",
            "DRIVERVERSION:",GetDeviceCaps (hdcInfo,DRIVERVERSION))) ;

    TextOut (hdc,cxChar,2 * cyChar,szBuffer,
        sprintf (szBuffer,"%-24s%-40s",
            "TECHNOLOGY:",szTech [GetDeviceCaps (hdcInfo,TECHNOLOGY)])) ;

    TextOut (hdc,cxChar,4 * cyChar,szBuffer,
        sprintf (szBuffer,"CLIPCAPS (Clipping capabilities)")) ;

    for (i = 0 ; i < sizeof clip / sizeof clip [0] ; i++)
        TextOut (hdc,9 * cxChar,(i + 6) * cyChar,szBuffer,
            sprintf (szBuffer,"%-16s%-28s %3s",
            clip[i].szMask,clip[i].szDesc,
            GetDeviceCaps(hdcInfo,CLIPCAPS) & clip[i].nMask ?
                "Yes" : "No")) ;

    TextOut (hdc,cxChar,8 * cyChar,szBuffer,
        sprintf (szBuffer,"RASTERCAPS (Raster capabilities)")) ;

    for (i = 0 ; i < sizeof raster / sizeof raster [0] ; i++)
        TextOut (hdc,9 * cxChar,(i + 10) * cyChar,szBuffer,
            sprintf (szBuffer,"%-16s%-28s %3s",
                raster[i].szMask,raster[i].szDesc,
                GetDeviceCaps (hdcInfo,RASTERCAPS) & raster[i].nMask ?
                "Yes" : "No")) ;
}

void DoBitCodedCaps (HDC hdc,HDC hdcInfo,short cxChar,short cyChar,
            short nType)
{
    static BITS curves [] =
    {
        CC_CIRCLES, "CC_CIRCLES", "circles:",
        CC_PIE, "CC_PIE", "pie wedges:",
        CC_CHORD, "CC_CHORD", "chord arcs:",
        CC_ELLIPSES, "CC_ELLIPSES", "ellipses:",
        CC_WIDE, "CC_WIDE", "wide borders:",
        CC_STYLED, "CC_STYLED", "styled borders:",
        CC_WIDESTYLED,"CC_WIDESTYLED","wide and styled borders:",
        CC_INTERIORS, "CC_INTERIORS", "interiors:"
    } ;

    static BITS lines [] =
    {
        LC_POLYLINE, "LC_POLYLINE", "polyline:",
        LC_MARKER, "LC_MARKER", "markers:",
        LC_POLYMARKER,"LC_POLYMARKER","polymarkers",
        LC_WIDE, "LC_WIDE", "wide lines:",
        LC_STYLED, "LC_STYLED", "styled lines:",
```

```
    LC_WIDESTYLED,"LC_WIDESTYLED","wide and styled lines:",
    LC_INTERIORS, "LC_INTERIORS", "interiors:"
} ;

static BITS poly [] =
{
    PC_POLYGON, "PC_POLYGON", "alternate fill polygon:",
    PC_RECTANGLE, "PC_RECTANGLE", "rectangle:",
    PC_WINDPOLYGON,"PC_WINDPOLYGON","winding number fill polygon:",
    PC_SCANLINE, "PC_SCANLINE", "scanlines:",
    PC_WIDE, "PC_WIDE", "wide borders:",
    PC_STYLED, "PC_STYLED", "styled borders:",
    PC_WIDESTYLED,"PC_WIDESTYLED","wide and styled borders:",
    PC_INTERIORS, "PC_INTERIORS", "interiors:"
} ;

static BITS text [] =
{
    TC_OP_CHARACTER,"TC_OP_CHARACTER","character output precision:",
    TC_OP_STROKE, "TC_OP_STROKE", "stroke output precision:",
    TC_CP_STROKE, "TC_CP_STROKE", "stroke clip precision:",
    TC_CR_90, "TC_CP_90", "90 degree character rotation:",
    TC_CR_ANY, "TC_CR_ANY", "any character rotation:",
    TC_SF_X_YINDEP, "TC_SF_X_YINDEP","scaling independent of X and Y:",
    TC_SA_DOUBLE, "TC_SA_DOUBLE", "doubled character for scaling:",
    TC_SA_INTEGER, "TC_SA_INTEGER", "integer multiples for scaling:",
    TC_SA_CONTIN, "TC_SA_CONTIN", "any multiples for exact scaling:",
    TC_EA_DOUBLE, "TC_EA_DOUBLE", "double weight characters:",
    TC_IA_ABLE, "TC_IA_ABLE", "italicizing:",
    TC_UA_ABLE, "TC_UA_ABLE", "underlining:",
    TC_SO_ABLE, "TC_SO_ABLE", "strikeouts:",
    TC_RA_ABLE, "TC_RA_ABLE", "raster fonts:",
    TC_VA_ABLE, "TC_VA_ABLE", "vector fonts:"
} ;

static struct
{
    short nIndex ;
    char * szTitle ;
    BITS ( * pbits) [] ;
    short nSize ;
}

bitinfo [] =
{
    CURVECAPS, "CURVCAPS (Curve Capabilities)",
        (BITS ( * )[]) curves,sizeof curves / sizeof curves [0],
    LINECAPS, "LINECAPS (Line Capabilities)",
        (BITS ( * )[]) lines,sizeof lines / sizeof lines [0],
    POLYGONALCAPS,"POLYGONALCAPS (Polygonal Capabilities)",
        (BITS ( * )[]) poly,sizeof poly / sizeof poly [0],
```

```
        TEXTCAPS, "TEXTCAPS (Text Capabilities)",
            (BITS ( * )[]) text,sizeof text / sizeof text [0]
    } ;

    static char szBuffer [80] ;
    BITS ( * pbits) [] = bitinfo [nType]. pbits ;
    short nDevCaps = GetDeviceCaps (hdcInfo,bitinfo [nType]. nIndex) ;
    short i ;

    TextOut (hdc,cxChar,cyChar,bitinfo [nType]. szTitle,
            strlen (bitinfo [nType]. szTitle)) ;

    for (i = 0 ; i < bitinfo [nType]. nSize ; i++)
        TextOut (hdc,cxChar,(i + 3) * cyChar,szBuffer,
            sprintf (szBuffer,"%-16s %s %-32s %3s",
            ( * pbits)[i]. szMask,"Can do",( * pbits)[i]. szDesc,
            nDevCaps & ( * pbits)[i]. nMask ? "Yes" : "No")) ;
}

; DEVCAPS1. DEF module definition file

NAME DEVCAPS1

DESCRIPTION 'Displays Device Capability Info (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

DEVCAPS1 的 Device 菜单允许用户选择屏幕或打印机。由于 DEVCAPS1 只需要获取一个信息环境句柄(用 CreateIC 函数),而不是获取设备环境句柄。获取视频设备的信息句柄很简单:

```
HIC = CreateIC("DISPLAY",NULL,NULL,NULL);
```

13.2.4 设备的大小

Windows 程序能从 GetDeviceCaps 获得的关于视频设备的最重要的信息是显示器的大小(两种单位:mm 和像素点)和显示器像素点的纵横比。这些尺寸可以帮助缩放图像。为了使读者对这些数值有一个了解,下面给出一个表,其中列出了四种常用 IBM 视频适配器的 GetDeviceCaps 信息:彩色/图形适配器(LGA)、增强型图形适配器(EGA)、视频图形阵列(VGA)和 8514/A。

HORZSIZE 和 VERTSIZE 值是显示区域的宽度和高度(以 mm 为单位)。当然,Windows 驱动程序并不知道用户连接到视频适配器上的显示器的真实尺寸。这些数值是根据适配器的标准显示尺寸给出的。

HORES 和 VERTRES 值是显示区域的宽度和高度(以像素点为单位)。对于一个视频显示设备环境,它们与从 GetsystemMetrice 返回的值相同。

GetDeviceCaps Index	CGA	EGA	VGA	8541/A
HORZSIZE(宽 mm)	240	240	208	280
VERTSIZE(高 mm)	180	175	156	210
HORZRES(像素宽)	640	640	640	1024
VERTRES(像素高)	200	350	480	768
ASPECTX(水平)	5	38	36	10
ASPECTY(垂值)	12	48	36	10
ASPECTXY(对角)	13	61	51	14
LOGPIXELSX(水平像素/英寸)	96	96	96	120
LOGPIXELSY(垂值像素/英寸)	48	72	96	120

ASPECTX、ASPECTY 和 ASPECTXY 的值分别是每个像素点的相对宽、高和对角线长度。ASPECTXT 与 ASPECTX 和 ASPECTY 的平方和的平方根相等。

LOGPIXELSX 和 LOGPIXELSY 的值分别是每个水平和垂直"逻辑英寸"的像素点数。一逻辑寸并不等于 1 英寸(25.4mm),只要用 HORZSIZE、VERTSIZE、HORZRES 和 VERTRES 的值进行简单的运行就可以知道。LOGPIXELSX 和 LOGPIXELS 的含义需要解释一下。读者可能已经注意到,WRITE 程序和其他一些 Windows 程序显示的标尺并不正确,如果用户测量 VGA 监视器上显示的标尺,就可以发现,它的 1 寸实际上比 1.5 寸还要多。这些程序的标尺使用的正是 LOGPIXELSX 和 AOGPIXELSY 的值,如果 WRITE 使用真实的物理尺度,则普通的 10 点或 12 点文本将会小到几乎难以辨认。这些逻辑尺度的实际效果是放大了显示,使得显示的文本能有一个恰当的尺寸。它只影响视频显示;对于打印机,GetDeviceCaps 返回的所有量都是一致的。

13.2.5 获取颜色信息

在讨论位图时,我们注意到视频显示适配器的内存对于颜色有两种组织方法。在一些视频适配器中,内存是以许多颜色平面来组织的。在每个平面内,一位对应于一个像素点,代表一个特定的颜色(红、绿、蓝之一)。其他视频适配器只有一个颜色平面,其中,许多邻接位代表每个像素点的颜色。

GetDeviceCaps 使程序员可以确定视频适配器的内存组织及它能表示的颜色数目。这个调用返回颜色平面的数目:

```
nPlanes = GetDeviceCaps(hdc,PLANES);
```

下面的调用返回每个像素点的颜色位数;

```
nBitsPixel = GetDeviceCaps(hdc,BITSPIXEL)
```

大多数具有彩色的显示设备使用多颜色平面或者每像素点有多个颜色位,但不能同时兼有二者;换句话说,这两个调用必有一个返回 1。视频适配器能够表示的颜色数可以用如下公式计算出来:

```
nColors = 1 << (nPlanes * nBitsPixel);
```

这个值与用 NUMCOLORS 参数得到的颜色值可能一样,也可能不一样。NUMCOLRS 获取如下:

```
nColors = GetDevicecCps(hdc,NUMCOLORS);
```

对于大部分绘图仪，这两个值都不同。对于绘图仪，PLANES 和 BITPIXEL 的值都等于1，但 NUMCOLORS 的值反映绘图仪拥有的彩色画笔数。对于单色设备，GetDeviceCaps 对于 NUMCOLORS 参数返回 2。

这两个值对于在 Windows 3.0 下可以加载颜色调色板的视频适配器（如 IBM 8514/A 适配器）也可以是不同的。8514/A 有一个平面，且每个像素点有 8 位。这意味着可能有 256 种颜色。GetDeviceCaps 以 NUMCOLORS 为参数返回的颜色值由 Windows 保留（对于 8514/A 返回 20）。剩下的 236 种颜色可以由 Windows 程序用调色板管理器设置。

GetDeviceCaps 所返回的颜色数是设备能显示的纯颜色数。Windows 可以使用振动（包括组合不同颜色像素点的像素模式）来代替纯颜色之外的颜色。

颜色通常用 3 字节的无符号长整数来表示，三个字节分别表示红、蓝、绿的密度，通常调用 GetNearestColor 也许可以确定与某颜色最接近的纯色。

```
rgbPureColor = GetNearestColor(hdc,rgbColor);
```

13.2.6 保存设备环境

在整个第四部分，读者会遇到许多改变设备环境属性的函数。通常，在程序员调用 GetDC 或 BeginPaint 时，Windows 用缺省值创建一个新设备环境，程序员对属性所做的一切变化在设备环境用 ReleaseDC 或 EndPaint 调用释放后，都会丢失。如果程序需要使用非缺省的设备环境属性，则程序员必须在每次获取设备环境句柄时初始化设备环境：

```
WM_PAINT:
    hdc = BeginPaint (hwnd,&ps);
    [初始化设备环境属性]
    [绘出窗口客户区]
    EndPaint (hwnd,&ps);
    return 0;
```

虽然在通常情况下这种方法已经很令人满意了，但程序员还可能想要在释放设备环境之后保存在程序中对设备环境属性所做的改变，以便在下一次调用 GetDC 或 BeginPaint 时它们仍能起作用。只需在登录窗口类时将 CS_OWNDC 标志包含为窗口类的一部分，即可达到这一目的：

```
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
```

现在，基于这个窗口类所创建的每个窗口都将拥有自己的设备环境，它一直存在，直到窗口被删除。如果使用了 CS_OWNDC 风格，就只需初始化设备环境一次，也许是在处理 WM_CREATE 消息期间完成这一操作：

```
WM_CREATE:
    hdc = GetDC (hwnd);
    [初始化设备环境属性]
    ReleaseDC (hwnd,hdc);
```

这些属性在被改变之前一直有效。

CS_OWNDC 风格只影响用 GetDC 和 BeginPaint 获得的设备环境，不影响其他函数（如 GetWindowsDC）获得的设备环境。CS_OWNDC 风格并不是没有代价的：Windows 需要为每个

用这种风格创建的窗口花费大约 800 个字节来保存设备环境。即使使用了 CS_OWNDC，程序员仍然应该在退出窗口函数之前释放设备环境。

也可以使用 CS_CLASSDC 风格：

```
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_CLASSDC;
```

这种风格导致每个窗口类拥有自己的设备环境，环境由所有在该类上创建的窗口共享。现在，可以在 WinMain 中在创建第一个基于该窗口类的窗口之后初始化设备环境属性：

```
if (! hPrevInstance)
{
    hdc = GetDC (hwnd);
    [初始化设备环境属性]
    ReleaseDC (hwnd,hdc);
}
```

一般来说，CS_CLASSDC 类型的设备环境比 CS_OWNDC 类型更难于使用，因为对设备环境属性所做的任意改变都影响所有基于同一窗口类的实例中的所有窗口。这可能会产生奇怪的效果，特别是在使用基于窗口大小的定制映像方式(下面就要讨论这一问题)时更是如此。

有些情况下，程序员可能想改变某些设备环境属性，用改变后的属性进行绘图，然后还原到原来的设备环境。要简化这一过程，可以通过如下调用来保存设备环境的状态：

```
nSavedID = SaveDC(hdc);
```

现在，可以改变一些属性，在想要返回为调用 SaveDC 前存在的设备环境时，调用：

```
RestoreDC(hdc,nSaveID);
```

在调用 SaveDC 前可调用 RestoreDC 数次。若想恢复到最后一次调用 SaveDC 前存在的设备环境，可调用：

```
RestoreDC(hdc,-1);
```

13.3 映像模式

Windows 的 GDI(Graphic Device Interface)在把显式图像送至物理设备之前，先在逻辑环境中建立好图像。

映像模式规定了逻辑环境中的字符单位与物理点阵的对应关系。因此，设置好映像模式，便决定了程序中的坐标系是怎样对应到输出设备上的。

另外有四个和映像模式密切相关的参数，它和映像模式一样是设备环境的属性。

参数	由哪个函数改变
窗口原点	SetWindowOrg
窗口范围	SetWindowExt
视口原点	SetViewPortOrg
视口范围	SetViewportExt

窗口参数 Window Origin 和 Window Extents 设置了逻辑坐标系的原点和范围。两个视口参数 Viewport Origin 和 Viewport Extents 设置了系统坐标系的原点和范围。

映像模式还决定了 x 轴与 y 轴的方向。原点(0,0)通常在屏幕的左上角。越靠右侧的点其 x 值越大,越靠近底部其 y 值越大。

几乎所有使用逻辑坐标系的函数都把设备环境句柄作为第一个参数,因此,它们是依赖于当前映像模式的。

例如,GetTextMetrics 计算出基于当前所设映像模式的当前字模的尺寸。与此相对应,像 WM_SIZE 和 WM_MOUSEMOVE 这样的消息总是使用以点为单位的设备坐标。Windows 使用两个公式来把逻辑坐标转化为设备坐标。

$$VpX = (WinX - WinOrgX) * \frac{VpExtX}{WinExtX} + VpOrgX$$

$$VpY = (WinY - WinOrgY) * \frac{VpExtY}{WinExtY} + VpOrgY$$

图 13.1

逻辑点的坐标为 WinX 和 WinY,它们将被转化为设备坐标 VpX 和 VpY。另外八个参数(WinOrgX,WinOrgY,WinExtX,WinExtY,VpOrgX,VpOrgY,VpExtX 和 VpExtY)代表着窗口和视口坐标的原点和单位。

这些公式也可以反过来计算(从设备坐标转化为逻辑坐标)。

13.3.1 映像模式的类型

有八种不同的映像模式:

- MM_TEXT
- MM_LOMETRIC
- MM_LOENGLISH
- MM_ISOTROPIC
- MM_TWIPS
- MM_HIMETRIC
- MM_HIENGLISH
- MM_ANISOTROPIC

使用 SetMapMode 可以设置任何一种映像模式。一旦设置了某种模式,它将保持到下一次设置或解除了设备环境(device context)为止。映像模式可以被分为三组。

与设备相关的模式

此模式即 MM_TEXT,它是设备环境的缺省模式。因为这种模式下随着 y 值增大,坐标点由上而下,这与我们书写时的习惯相同,所以常被用作简单的文本输出。

因为这种模式的单位是点,所以函数调用 Ellipse(hDC,20,20,80,80)将生成一个圆或椭圆,这完全依赖于设备。如果像素是方的,那么将画出一个圆来。

在这种模式下,窗口和视口的坐标可以改变,坐标的单位则被固定地置为(1,1)。比如,可以把系统坐标系的原点移到显示区域的中心。

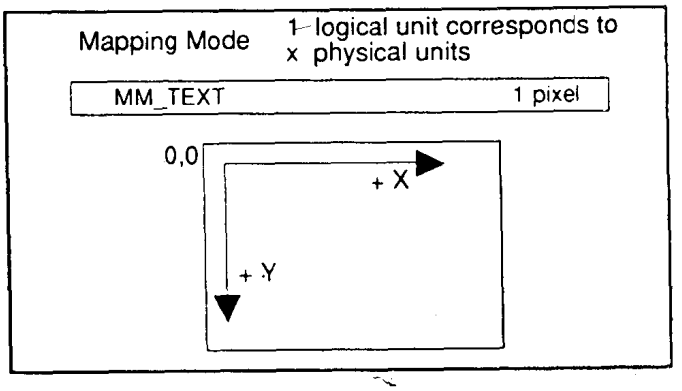


图 13.2

为了清楚起见，最好只改变原点的一个坐标值。

度量模式

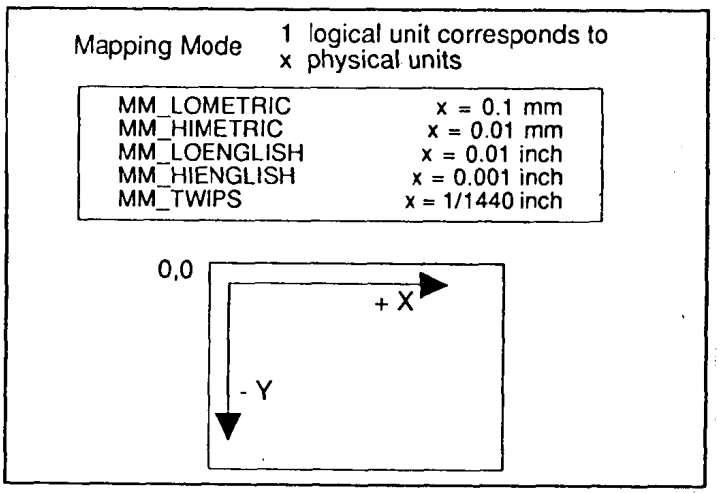


图 13.3

"Twip"是"twentieth of a point"的缩写。它是打印机使用的单位。当然也可以用 mm 或 inch 作单位。

例如，设置了映像模式 MM_LOMETRIC 之后调用 Ellipse(hDC,20,20,80,80)，结果总是一个直径为 60×0.1mm 的圆。这与屏幕设备无关。

度量模式在系统坐标系的定义上也与 MM_TEXT 不同。其原点仍在左上方，但越向下的点其 y 值越小。

只需要改变窗口或视口的坐标原点，便可轻易得到有四个像限的笛卡尔坐标系。但是改变窗口的坐标原点更麻烦一些，因为客户区域的坐标必须用函数 DPtoLP 转化为逻辑坐标。

例如，消息 WM_SIZE 决定了两个值 xClient 和 yClient，从而确定了客户区域的尺寸。

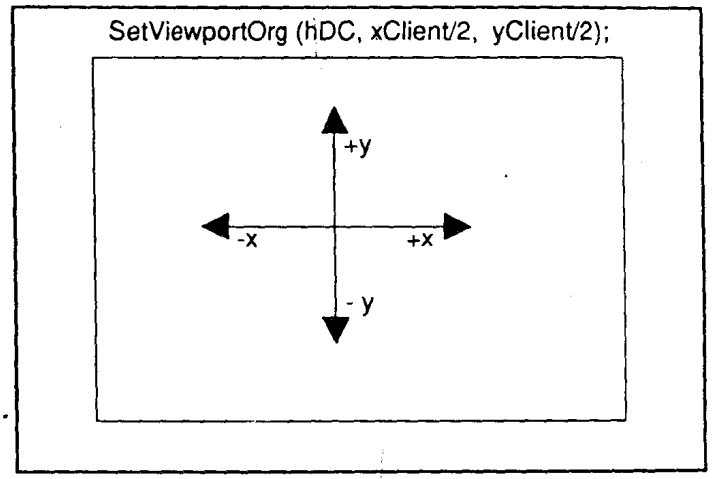


图 13.4

在这种模式下，坐标系的单位是预置好的，且不准更改。

定制模式

映像模式 MM_ISOTROPIC 和 MM_ANISOTROPIC 被称作定制模式，因为不仅可以改变窗口和视口的原点，还可以改变它们的坐标单位。于是在我们自己的坐标系里可以用任意的尺寸，然后再映射到客户区域的坐标系中。

在 MM_ISOTROPIC 模式下，逻辑坐标系与物理坐标系的 x:y 比率值相同，因此，可能有一部分视口不能显示图像。术语 isotropic 的意思就是在各个方向上都一样。

与此相反，MM_ANISOTROPIC 模式尽可能地利用视口空间，尽管可能会造成图像变形。通常，视口的坐标点对应到了整个客户区域上。函数 GetClientRect 可以求出客户区域的当前尺寸。

```
GetClientRect( hDC,&rect );
SetViewportOrg( hDC,rect. left,rect. top );
SetViewportExt( hDC,rect. right,rect. bottom );
```

在定制模式下，除了窗口的坐标单位外，还要考虑坐标轴的方向。坐标原点可以在四个角的任何一个上。窗口的宽度是水平单位，高度是垂直单位。

通常定义窗口和视口的坐标原点与单位，可以把整个画面缩小后再画到客户区域。

例如，在 WM_ANISOPROPIC 模式下可以指明在第几行第几列输出文本，而不必考虑字符的尺寸。为此，必须改变窗口和视口的坐标单位。

```
SetMapMode( hDC,MM_ANISOTROPIC );
SetWindowExt ( hDC,1,1 );
SetViewPortExt( hDC,xChar,yChar );

TextOut( hDC,4,5,"new text",10 );
```

变量 xChar 和 yChar 代表字符的高度和平均宽度。在函数 TextOut 的帮助下，便可在第五行上从第四列开始显示文本串。

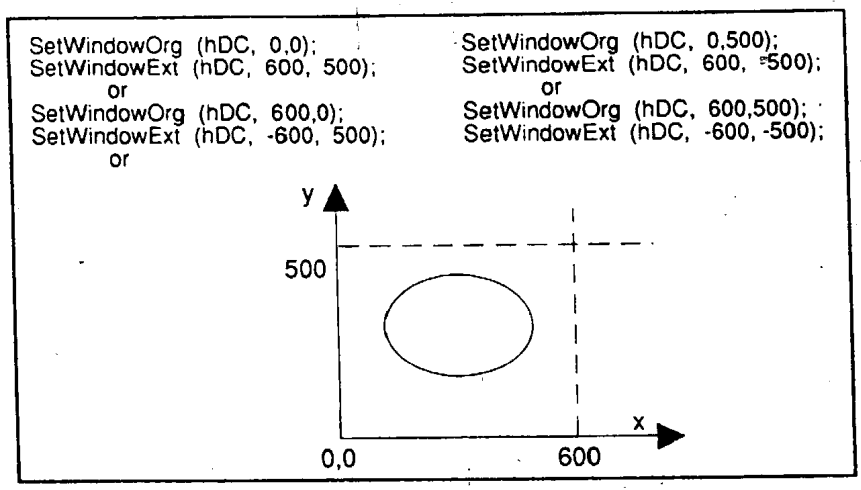


图 13.5

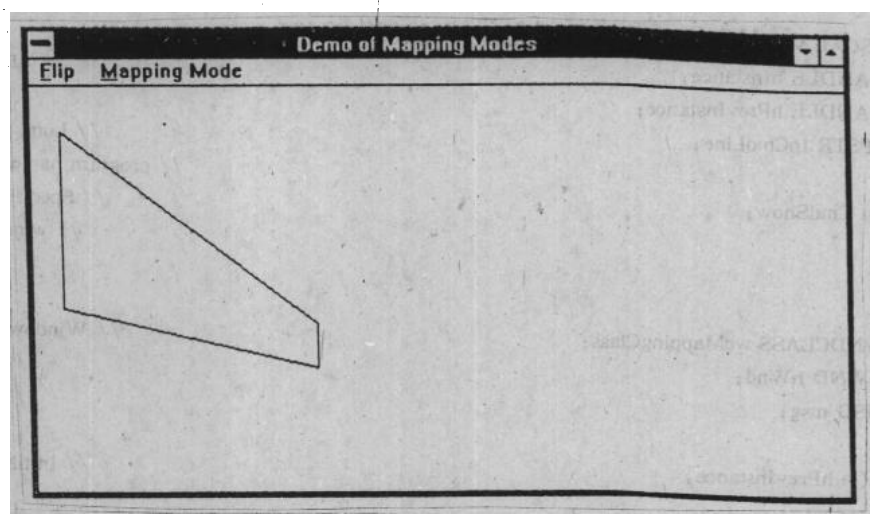


图 13.6

13.3.2 定制模式的例子

这个应用程序的任务是映像一个平行于 x 轴和 y 轴的图形。应用程序使用两种坐标原点和单位都不同的定制映像模式。另外，应用程序还可以在度量模式下或在依赖于设备的模式下显示图形。

新函数	描述
SetMapMode	设置映像模式
SetViewportExt	设置视口范围
SetViewportOrg	设置窗口的原点
SetWindowExt	设置窗口的范围

SetWindowOrg　　设置窗口的原点

源代码:MAPPING. C

```
/* * MAPPING. C * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates isotropic and anisotropic mapping modes * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows. h"                                    // Include windows. h header file
#include "mapping. h"                                    // Include mapping. h header file

#if _MSC_VER >= 700                                      // Running Microsoft C 7. 00 or
#pragma warning (disable:4028)                           // higher? Disable warning 4028
#endif

int PASCAL WinMain(HANDLE,HANDLE,LPSTR,int);
long FAR PASCAL MappingWndProc(HANDLE,WORD, WORD, LONG);

/* * WinMain * * * * * * * * * * * * * * * * * * * * * * * * * */

int PASCAL WinMain(hInstance,hPrevInstance,lpCmdLine,CmdShow)
    HANDLE hInstance;                                    // Current instance
    HANDLE hPrevInstance;                                // Previous instance
    LPSTR lpCmdLine;                                     // Long ptr to string after
                                                         // program name during execution
    int CmdShow;                                         // Specifies the application
                                                         // window's appearance
{

    WNDCLASS wcMappingClass;                             // Window class declaration
    HWND hWnd;                                           // Window handle
    MSG msg;                                             // Message variable

    if(! hPrevInstance)                                  // Initialize first instance
    {

/* * Specify window class * * * * * * * * * * * * * * * * * * */

        wcMappingClass. style = CS_HREDRAW | CS_VREDRAW;
        wcMappingClass. lpfnWndProc = MappingWndProc;
        wcMappingClass. cbClsExtra = 0;
        wcMappingClass. cbWndExtra = 0;
        wcMappingClass. hInstance = hInstance;
        wcMappingClass. hIcon = LoadIcon (NULL,IDI_APPLICATION);
        wcMappingClass. hCursor = LoadCursor(NULL,IDC_ARROW);
        wcMappingClass. hbrBackground = GetStockObject(WHITE_BRUSH);
        wcMappingClass. lpszMenuName = "Mapping";
        wcMappingClass. lpszClassName = "Mapping";

        if(! RegisterClass(&wcMappingClass))
            return FALSE;
```

```
    }

/* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Mapping",
        "Demo of Mapping Modes",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,0,
        CW_USEDEFAULT,0,
        NULL,
        NULL,
        hInstance,
        NULL);

    ShowWindow(hWnd,CmdShow);                          // Make window visible
    UpdateWindow(hWnd);                                  // Update window

    while(GetMessage(&msg,NULL,0,0)) {                   // Message reading
        TranslateMessage(&msg);                        // Message translation
        DispatchMessage (&msg);                      // Send message to Windows
    }
    return msg. wParam;                            // Return wParam of last message
}

/* * MappingWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL MappingWndProc(hWnd,message,wParam,lParam)
    HANDLE hWnd;
    WORD message;
    WORD wParam;
    LONG lParam;
{
    HDC hDC;
    PAINTSTRUCT ps;

    static HMENU hMenu;
    static WORD wFigurOld,wMMOld;
    static int iMapping;
    RECT rect;

    static int sWinOrgX,sWinOrgY,sWinExtX,sWinExtY;

    struct
    {
        int WinOrgX;
        int WinOrgY;
        int WinExtX;
        int WinExtY;
    } WinPoints [4] =.{ 0,200, 200,-200,
```

```
    200,200,-200,-200,
    0, 0, 200,200,
    200, 0,-200,200};

int map [4] = {MM_TEXT,MM_ISOTROPIC,
    MM_ANISOTROPIC,MM_LOMETRIC};

POINT FourSider [5] = {{ 20,150},
    { 20, 30},
    {190,160},
    {190,190},
    { 20,150}};

switch (message)
{
    case WM_CREATE:                                          // Draw window
        hMenu = GetMenu(hWnd);
        iMapping = MM_TEXT;
        wMMOld = ID_DEF;
        wFigurOld = ID_ORIGIN;

        sWinOrgX = 0;
        sWinOrgY = 200;
        sWinExtX = 200;
        sWinExtY = -200;
        break;
    case WM_COMMAND:
        switch(wParam)                                       // Respond to menu items
        {
            case ID_ORIGIN:
            case ID_RIGHT:
            case ID_DOWN:
            case ID_LEFT:
                sWinOrgX = WinPoints [wParam - ID_ORIGIN].WinOrgX;
                sWinOrgY = WinPoints [wParam - ID_ORIGIN].WinOrgY;
                sWinExtX = WinPoints [wParam - ID_ORIGIN].WinExtX;
                sWinExtY = WinPoints [wParam - ID_ORIGIN].WinExtY;

                CheckMenuItem(hMenu,wFigurOld,MF_UNCHECKED);
                CheckMenuItem(hMenu,wParam,MF_CHECKED);
                wFigurOld = wParam;
                InvalidateRect(hWnd,NULL,TRUE);
                break;
            case ID_DEF:                                     // Default mode
            case ID_ISO:                                     // Isotropic mode
            case ID_ANI:                                     // Anisotropic mode
            case ID_LOM:                                     // Lometric mode
                iMapping = map [wParam - ID_DEF];
                CheckMenuItem(hMenu,wMMOld,MF_UNCHECKED);
                CheckMenuItem(hMenu,wParam,MF_CHECKED);
                InvalidateRect(hWnd,NULL,TRUE);
```

```
                    wMMOld = wParam;
                    break;
                default:
                    break;
            }
            break;
        case WM_PAINT:                                          // Client area redraw
            hDC = BeginPaint(hWnd,&ps);
            SetMapMode(hDC,iMapping);
            GetClientRect (hWnd,&rect);
            if ((iMapping == MM_ISOTROPIC) || (iMapping == MM_ANISOTROPIC))
            {
                SetWindowOrg(hDC,sWinOrgX,sWinOrgY);
                SetWindowExt(hDC,sWinExtX,sWinExtY);
                SetViewportOrg(hDC,rect. left,rect. top);
                SetViewportExt(hDC,rect. right,rect. bottom);
            }
            if (iMapping == MM_LOMETRIC)
                SetViewportOrg (hDC,0,rect. bottom);
            Polygon(hDC,FourSider,4);
            EndPaint(hWnd,&ps);
            break;
        default:
            return(DefWindowProc(hWnd,message,wParam,lParam));
            break;
    }
    return(0L);
}
```

模块定义文件:MAPPING. DEF

```
NAME Mapmode

DESCRIPTION 'Example of mapping modes'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 1024
STACKSIZE 4096

EXPORTS MappingWndProc @1
```

资源描述文件:MAPPING. RC

```
#include "windows. h"
#include "mapping. h"
```

```
Mapping MENU
BEGIN
    POPUP "&Flip"
    BEGIN
        MENUITEM "&Original", ID_ORIGIN, CHECKED
        MENUITEM "&Right", ID_RIGHT
        MENUITEM "&Down", ID_DOWN
        MENUITEM "Down &and Right", ID_LEFT
    END

    POPUP "&Mapping Mode"
    BEGIN
        MENUITEM "&Default", ID_DEF, CHECKED
        MENUITEM "&Isotropic", ID_ISO
        MENUITEM "&Anisotropic", ID_ANI
        MENUITEM "&Lometric", ID_LOM
    END
END
```

头文件:MAPPING. H

```
#define ID_ORIGIN 20
#define ID_RIGHT 21
#define ID_DOWN 22
#define ID_LEFT 23

#define ID_DEF 10
#define ID_ISO 11
#define ID_ANI 12
#define ID_LOM 13
```

MAKE 文件:MAPPING. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = MAPPING.H

all: MAPPING.EXE

MAPPING.RES: MAPPING.RC $(RESFILES)

MAPPING.OBJ: MAPPING.C $(H)

MAPPING.EXE: MAPPING.OBJ MAPPING.RES MAPPING.DEF
    link /NOD/A:16 MAPPING,MAPPING,nul,libw slibcew,MAPPING.def
    rc MAPPING.res MAPPING.exe
```

像下面这样运行 NMAKE 便可产生应用程序 MAPPING：

```
nmake mapping. mak
```

应用程序 MAPPING. EXE 是怎样工作的

可以从一个弹出式菜单中设置不同的映像模式。用于 Flipping 的弹出式菜单有些不同，因为它只能使用定制模式(Anisotropic 模式或 Isotropic 模式)。

在得到消息 WM_CREATE 后，将用不同的值进行初始化。刚开始运行时，当前映像模式是 MM_TEXT，矩形图形未经反射便显示出来。

基于此应用程序的结构，在处理消息 WM_PAINT 时，所有链接于此映像模式的函数引用都将被调用。

这是因为在处理消息 WM_PAINT 时已经得到了设备环境的句柄，所以在处理其他消息时，只对一些变量进行赋值。

在此消息被处理时，弹出式菜单 FLIP 的四个菜单项可以被组合。正确的窗口坐标原点和单位可以从用户自定义的结构数组中取得。

在头文件 MAPPING. H 中按从下往上的顺序定义了四个菜单项的 ID 值。在新的菜单项中必须设置检查标志，而在旧的菜单项中应清除检查标志。为了显示新的设置情况，函数 InvalidateRect 将产生一条消息 WM_PAINT。

如果从弹出式菜单 MappintMode 中选中了一项，所作的处理与选中菜单 Flip 中的某项会差不多，正确的映像模式将从映像模式数组中得到，数组的下标号是由菜单的 ID 值决定的。应用程序必须修改检查标志并且产生消息 WM_PAINT。

在处理消息 WM_PAINT 时，窗口和视口的在 MM_ISOTROPIC 或 MM_ANISOTROPIC 模式下的坐标原点和单位都将被设置，函数 GetClientRect 将计算出视口的有关值。

图 13.7 描述了逻辑坐标系。

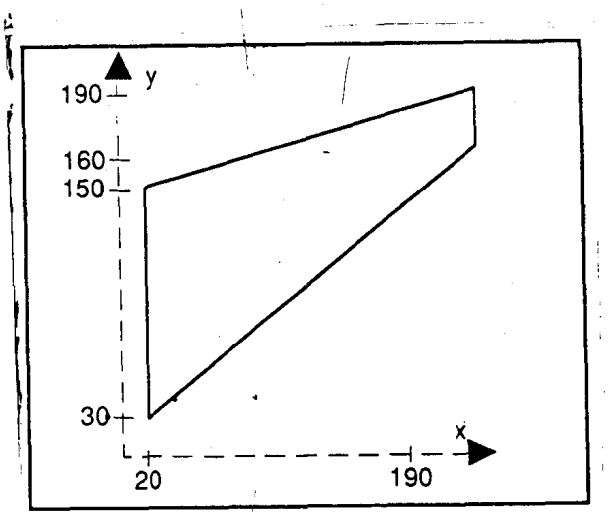


图 13.7 逻辑坐标系

在缺省模式 MM_TEXT 下，这个矩形以底边为轴经反射后显示出来。这是通过增大 y

轴而获得的。如果选中了映像模式MM_LOMETRIC，还必须改变视口的原点。否则，这个矩形将定位于客户区中，因为度量模式的y轴方向不同。

改变了原点后，便可显示笛卡尔坐标系的第一像限，矩形就定位于这个区域中。

第十四章　绘　图

第三章和第十三章分别介绍了绘图工具的使用方法和 GDI 的基本概念，本章主要以大量例子说明如何在 Windows 环境中绘制各种基本图形。

14.1　画　点

用 GDI 的 SetPixel 函数，可以画一个特定颜色的像素点：

```
rgbActualColor = SetPixel(hdc,x,y,rgbColor);
```

rgbColor 参数是无符号长整数(32 位)，它的低 3 位字节分别代表红、蓝、绿的密度。可以用 RGB 宏来构造这个颜色值：

```
rgbColor = RGB(byRed,byGreen,byBlue);
```

虽然 x 和 y 是逻辑坐标，但 SetPixel 只对一个物理像素点着色，不管使用哪种映像方式。由于 SetPixel 只画一个像素点，因此使用振动颜色(各种组合不同纯色像素点的颜色)是毫无意义的。基于这个理由，Windows 将 rgbColor 参数转换成纯色，然后返回这个颜色。

SetPixel 几乎不在 Windows 程序中使用，但这并不妨碍我们使用它。用 GetPixel 函数即可获得特定像素点的颜色。

```
rgbColor = GetPixel(hdc,x,y);
```

14.2　画　线

会画点之后，下一步便是画线。Windows 可画直线和椭圆形的线，椭圆形的线是指一条曲线，它是椭圆周线的一部分。四个画线的函数是 LineTo(直线)、PolyLine(一组相连的线)、PolyPolyLine(多组相连的线)和 Arc(椭圆形的线)。设备环境的五种属性影响用这些函数画线的外观，这些属性是：当前画笔位置(只影响 LineTo)、画笔、背景方式(对于非固态笔)、背景色(对于 OPAQUE 背景方式)和绘图方式。

LineTo 函数是上述几个不包含画对象全部尺寸的函数之一。它从设备环境中定义的当前画笔位置开始，一直画到函数中所指定的逻辑点为止，但不包含该逻辑点。在缺省设备环境里，当前画笔位置最初设置在逻辑点(0,0)。如果在调用 LineTo 函数之前不先设定当前画笔位置(或者改变窗口或视口原点)，则它从客户区域的左上角开始画线。

要在逻辑点(xStart,yStart)到(xEnd,yEnd)之间画一条直线，应该先用 MoveTo 函数将当前画笔位置设置到点(xStart,yStart)：

```
MoveTo(hdc,xStart,yStart);
```

MoveTo 不画任何对象，它只是改变当前画笔位置。然后就可以用 LineTo 来画线：

```
LineTo(hdc,xEnd,yEnd);
```

这条语句从当前画笔位置开始画一条直线到点(xEnd,yEnd)(不包含点 xEnd,yEnd)。在调用 LineTo 之后，当前画笔位置改变为(xEnd,yEnd)。

通常，LineTo 是唯一使用当前画笔位置的 Windows 函数，而只有 MoveTo 和 LineTo 才能改变当前画笔位置。SetTextAlign 所设置的一个选项也允许 TextOut 使用当前画笔位置。调用 GetCurrentPosition 即可获取当前画笔位置：

```
dwPoint = GetCurrentPosition(hdc);
```

dwPoint 的返回值是一个无符号长整数(四字节)，低位字包含 x 坐标，高位字包含 y 坐标。可以用 LOWORD 和 HIWORD 宏来取出这两个坐标，或者可以用 MAKEPOINT 宏将 dwPoint 的值转换成一个 POINT 结构：

```
point = MAKEPOINT(dwPoint);
```

下面这段代码在窗口的客户区域中画一个网格，从左上角开始，每条线间隔为 1 寸。假定 hwnd 是窗口的句柄，hdc 是设备环境句柄，rect 是 RECT 类型的结构，x 和 y 为短整数。

```
SetMapMode (hdc,MM_LOENGLISH);
GetClintRect (hwnd,&rect);
DPtolP (hdc,(LPPOINT) &rect,2);

for (x = 0 ; x < rect. bottom);
{
    MoveTo (hdc,x,rect. bottom);
    LineTo (hdc,x,rect. bottom);
}
for (y = 0 ; y > rect. bottom ; y -= 100)
{
    MoveTo (hdc,0,y);
    LineTo (hdc,rect. right,y);
```

客户区域的大小保存在 RECT 类型的结构 rect 中，并用 DPtoLP 转换成逻辑点。在经过 DPtoLP 的转换后，rect. right 是客户区域的宽度(单位为 0. 01 寸)，rect. bottom 为客户区域高度的负值。注意，在第二个循环中，y 是递减的，因为在 MM_LOENGLISH 映像方式中，随显示往下移动时，y 值将减小。

也许在画一条直线时居然不得不使用两个函数显得有些可笑，但在希望画一组相连的直线时，当前画笔位置属性又会显得很有用。例如，程序员可能想定义一个包含 5 个点(10 个值)的数组，来画一个矩形的边缘框

```
POINT pt [5] = {100,100,200,100,200,200,
    100,200,100,100};
```

注意，最后一个点与第一个点相同。现在，只需用 MoveTo 移到第一点，并对后面的点使用 LineTo：

```
MoveTo (hdc,pt[0],x,pt[0]. y);
for (i= 1; i < 5; i++)
    LineTo (hdc,pt[i]. x,pt[i]. y;
```

由于 LineTo 从当前位置画到(但不包含)在 LineTo 函数中给出的点，所以这段代码没对

任何坐标画两次。虽然在显示器上多输出几次不存在问题，但是在绘图仪上或者在其他绘图方式（下面即会讲到）下；这样就不太好。

当要将点数组连接成线时，使用 PolyLine 函数要简单得多。下面这条语句画出与上面一段代码相同的矩形：

```
PolyLine(hdc,&pt,5);
```

最后一个参数是点的数目。我们还可以用(sizeof(pt)/sizeof(POINT)) 来表示这个值。PolyLine 与一个 MoveTo 函数后面加几个 LineTo 函数的效果相同。但是，PolyLine 既不使用也不改变当前位置。

Arc 函数稍微要复杂一些。下面是通常的语法：

```
Arc(hdc,xLeft,yTop,xRight,yBottom,
    xStart,yStart,xEnd,yEnd);
```

Arc 函数在一个内接于矩形的椭圆周线上画曲线，矩形的左上角为(xLeft,yTop)，右下角为(xRight,yBottom)。弧线从椭圆与(xStart,yStart)和椭圆中心点的连线的交点开始，弧线沿反时针方向，在椭圆周线上前进，结束于椭圆与点(xEnd,yEnd)和椭圆中心点连线的交点。读者也许会觉得要明白这些有些困难，不必担心；在讲过矩形和椭圆之后，我们还会更详细地讨论 Arc 函数。

14.2.1 使用后援画笔

在调用 LineTo，PolyLine，PolyPolyLine 和 Arc 时，Windows 使用设备环境中当前选择的“画笔”来画线。画笔确定线的颜色、宽度和风格（风格可以是实线、点画线或虚线）。缺省设备环境中画笔为 BLACK_PEN。这种画笔不管映像方式是什么都使用像素点宽的连续黑线画线。BLACK_PEN 是 Windows 提供的三种后援画笔之一。其他两种是 WHITE_PEN 和 NULL_PEN。NULL_PEN 是不画对象的画笔。也可以创建自制的画笔。

在 Windows 程序中，程序员用句柄来引用画笔。WINDOWD.H 包含一个类型定义 HPEN，它是画笔的句柄。程序员可以用这个类型定义来定义一个变量 hPen：

```
HPEN hpen;
```

调用 GetStockOject 即可获得一个后援画笔的句柄。例如，假设程序员想使用名为 WHITE_PEN 的后援画笔，可用如下语句获取画笔的句柄：

```
hPen = GetStockObject(WHITE_PEN);
```

现在，必须使画笔成为设备环境中当前被选中的画笔，这需要调用 SelectObject；

```
SelectObject(hdc,hPen);
```

在这个调用之后，用 LineTo、PolyLine 和 Arc 所画的线将使用 WHITE_PEN，直到又选择了另一种画笔进入设备环境或者释放了设备环境为止。

也可以不显式定义 hPen 变量，而直接将 GetStockObject 和 SelectObjct 合并到一个语句里：

```
SelectObject(hdc,GetStockObject(WHITE_PEN));
```

如果想返回到使用 BLACK_PEN 的状态，可以在一个语句里获取这种后援画笔的句柄，并将它选进设备环境中：

```
SelectObject(hdc,GetStockObject(BLACK_PEN));
```

SelectObject 的返回值是在调用它之前选进设备环境中的画笔句柄。如果正处于缺省设备环境，并调用

```
hPen = SelectObject(hdc,GetStockObject(WHITE_PEN));
```

则设备环境中的当前画笔将为 WHITE_PEN，变量 hPen 将会是 BLACK_PEN 的句柄。以后，可通过调用

```
SelectObject(hdc,hPen);
```

重新将 BLACK_PEN 选择到设备环境中。

14.2.2 画笔的创建、选择和删除

尽管使用定义为后援对象的画笔很方便，但限于只有连续的黑画笔、连续的白画笔或者完全没有画笔。如果希望画笔丰富一些，就必须创建自己的画笔。下面是一般的过程。使用函数 CreatePen 或 CreatePenInDirect 创建一个“逻辑画笔”，这仅仅是对画笔的描述。这些函数返回逻辑画笔的句柄。然后，调用 SelectObject 将画笔选择到设备环境中。现在，就可以用新的画笔画线了。在任何时候，都只能有一种画笔选择到设备环境中。在释放设备环境（或者在选择了另一种画笔到设备环境中）之后，就可以调用 DeleteObject 以删除所创建的逻辑画笔。在删除后，该画笔的句柄就不再有效。

逻辑画笔是一种“GDI 对象”。程序员可创建和使用画笔，但画笔并不属于程序。画笔实际上属于 GDI 模块。画笔是程序员能够创建的六种 GDI 对象之一，其他五种是刷子、位图、区域、字模和调色板。

这就带来了一些非常重要的问题。通常，在一个程序终结时，Windows 将彻底地清除之。但一个很大的例外是对于 GDI 对象，在程序终结后，Windows 并不自动删除程序所创建的 GDI 对象，程序自己应该负责删除这些 GDI 对象。

在使用 GDI 对象（如画笔）时应把握以下三点：

- 删除自己创建的所有 GDI 对象。
- 当 GDI 对象选进一个有效的设备环境中时，不要删除它。
- 不要删除后援对象。

它们并不是令人信服的，但有时的确管用。下面我们将通过例子来看一看这些规则的要领。

通常，CreatePen 函数的语法形如：

```
hPen = CreatePen(nPenStyle,nWidth,rgbColor);
```

其中，nPenStyle 参数确定画笔是实线、点画线还是虚线。该参数可以是 WINDOWS. H 中定义的以下标识符：PS_SOLID，PS_DASH，PS_DOT，PS_DASHDOT，PS_DASHDOTDOT，PS_NULL，PS_INSIDEFRAME。图 14.1 显示了每种风格所产生的线的类型。

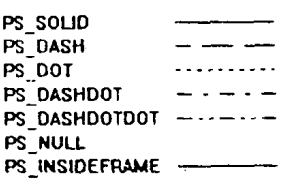


图 14.1 七种线和风格（又称“线型”）

对于 PS_SOLID、PS_NULL 和 PS_INSIDEFRAME 风格，nWidth 参数是画笔的宽度，它使用逻辑单位。例如，如果映像方式为 MM_LOENGLISH，则 nWidth 为 10 的画笔其宽度为 0.1 寸。在用 PS_SOLID 或 PS_NULL 画笔画线时，画笔宽度将向线两边延伸 0.05 寸。PS_INSIDEFRAME 风格则稍有不同。对于 MM_ANISOTROPIC 映像方式，Windows 使用 x 轴上的逻辑单位来确定线的物理宽度。如果 nWidth 为 0，则指示 Windows 用一物理单元（一个像素点）作为笔宽。后援画笔的宽度为一个像素点。如果指定一个点画线式或虚线式画笔风格，而宽度又大于 1，则 Windows 将使用实画笔来代替。

CreatePen 的 rgbColor 参数是一个无符号长整数，它指定画笔的颜色。对于除了 PS_INSIDEFRAME 之外的画笔风格，如果将画笔选进设备环境中，Windows 会将 rgbColor 参数转换成设备所能表示的最相近的纯色。PS_INSIDEFRAME 风格是唯一能使用振动颜色的画笔风格，只有在宽度大于 1 时才如此。

也可以通过建立一个类型为 LOGPEN（“逻辑画笔”）的结构，并调用 CreatePenIndirect 来创建一个画笔。如果程序应用许多能在源代码中初始化的画笔，用这种方法就有效得多。首先定义一个类型为 LOGPEN 的结构变量，如 logPen：

```
LOGPEN logpen;
```

这个结构有三个成员：lopnStyle（WORD）是画笔风格，lopnWidth（POINT）是使用逻辑单位的画笔宽度，lopnColor（DWORD）是画笔的颜色。lopnWidth 成员是类型为 POINT 的结构，但 Windows 只使用 lopnWidth.x 的值作为画笔的宽度，而忽略 lopnWidth.y。然后，可以将 logPen 结构的地址传送给 CreatePenIndirect：

```
hPen = CreatePenIndirect(&logpen);
```

这样就创建了画笔。

也可以获取已存在的画笔的逻辑画笔信息。如果已经具有一个画笔的句柄，则可以用 GetObject 调用将定义逻辑画笔的数据拷贝到一个类型为 LOGPEN 的结构中：

```
GetObject(hPen,sizeof(LOGPEN),(LPSTR) &logpen);
```

注意，CreatePen 和 CreatePenIndirect 函数不需要设备环境句柄为参数。这些函数创建与设备环境没有联系的逻辑画笔。直到调用 SelectObject 之后，画笔才与设备环境有联系。因此，可以对不同的设备（如屏幕和打印机）使用相同的逻辑画笔。nWidth 不为 0 的逻辑画笔具有逻辑宽度；但是，只有在被选择到设备环境中之后，它才具有物理宽度，物理宽度取决于设备环境的映像方式。

下面是一种创建、选择然后删除画笔的方法。假设程序使用三种画笔：一种宽度为 1 的黑画笔、一种宽为 3 的红画笔、一种黑色点式画笔。先定义三个变量来存放这些画笔的句柄：

```
static HPEN nPenl,nPen2,nPen3;
```

在处理 WM_CREATE 时，可以创建这三种画笔：

```
hPen1 = CreatePen (PS_LOLID,1,0L);
hPen2 = CreatePen (PS_SOLID,3,RGB (255,0,0));
hPen3 = CreatePen (PS_DOT), 0,0L);
```

在处理 WM_PAINT（或者在它具有有效的设备环境句柄）的时候，可以选择这些画笔之一到设备环境之中，并用它画线：

```
SelectObject(hdc,hPen2);
    [LineTo、PolyLine 或 Arc 调用]
SelectObject(hdc,nPen1);
    [LineTo、PolyLine 或 Arc 调用]
```

在处理 WM_DESTROY 时，可以删除所创建的三种画笔：

```
DeleteObject(hPen1);
DeleteObject(hPen2);
DeleteObject(hPen3);
```

这是创建、选择、删除画笔的最直接的方法，但逻辑画笔在整个程序运行期间需要占用内存。为此，可能要在每个 WM_PAINT 消息期间创建画笔，并在调用 EndPaint 之后删除它们。虽然也可以在调用 EndPaint 之前删除画笔，但是要小心，不要删除设备环境中当前选择的画笔。

程序员也可能希望创建画笔，并将 CreatePen 和 SelectObject 调用组合到同一个语句中：

```
SelectObject(hdc,CreatePen(PS_DASH,0,RGB(255,0,0)));
```

现在再开始画线，使用红色虚线画笔。在画完红色虚线后，可以删除画笔。坏了！由于没有保存画笔句柄，怎样才能删除这些画笔？不要紧。请记住，SelectObject 返回设备环境中上一次选择的画笔句柄。所以，程序员可以通过调用 SelectObject 将 BLACK_PEN 选择到设备环境中，并删除从 SelectObject 返回的值：

```
DelectObject(SelectObject(hdc,GetStockObject(BLACK_DEN)));
```

下面是另一种方法，在将新创建的画笔选择到设备环境中时，保存 SelectObject 返回的画笔句柄：

```
hPen = SelectObject(hdc,CreatePen(PS_DASH,0,RGB(255,0,0)));
```

现在 hPen 是什么？如果这是在获取设备环境之后第一次调用 SelectObject，则 hPen 是 BLACK_PEN 后援对象的句柄。现在，可以将 hPen 选择到设备环境中，并删除所创建的画笔（第二次 SelectObject 调用返回的句柄），只要一条语句即可：

```
DeleteObject(SelectObject(hdc,hPen));
```

如果删除一个正在设备环境中的 GDI 对象，然后试图画线，Windows 将响应一个致命的错误。因为设备环境中不包含有效画笔。这是一种很容易跟踪的故障。删除所创建的 GDI 对象时失败，则是一种查起来十分困难的故障，因为程序似乎很正常。如果程序员为每条 WM_PAINT 消息创建了同样的逻辑画笔，则可能想导致客户区域一次次地重画，以看看可用的内存是否会下降。如果在程序终止后，HEAPWALK 显示许多小的 GDI 段，则其中的一些可能是在程序中删除失败的 GDI 对象。

14.2.3 避免与设备相关

由于显示器的分辨率不同，画笔的宽度也不会相同。后援画笔以及以宽度为 0 创建的所有画笔为 1 个像素点宽，这在高分辨率的显示器上会产生非常细的线。

如果使用 WM_TEXT，可以以 SM_CXBORDER 和 SM_CYBORDER 为参数调用 GetSystemMetrics，以获取单行窗口边框的宽度，这些值适于作画笔宽度。也可以使用某种尺度映像方式，并为程序需要创建的画笔设置特定的物理宽度。

画笔颜色也很容易与设备相关。如果在彩色显示器上开发程序而在单位显示器上运行，就会遇到一些令人不愉快的事。除了对于 PS_INSIDEFRAME 风格之外，Windows 总是使用纯色画笔；在单色系统上，画笔非黑即白。例如，在彩色的 GDI 上，程序员可能喜欢使用白色背景上的洋红画笔：

```
hPen = Create(PS_SOLID,1,RGB(255,0,255));
```

但是，由于在单色系统上与洋红最近的纯色是白色，所以画笔将是看不见的。如果想要使用彩色画笔，则应保证，对于任意缺省色为黑色的画笔，红、绿、蓝的值之和小于或等于 381(即三种原色的最大和的一半)；对于缺省色为白色的画笔，红、绿、蓝的值之和大于 381。

14.2.4 填充空隙

使用点式画笔和长划式画笔会产生一个有趣的问题：点、长划之间的空隙会怎样呢？空隙的着色取决于背景方式和设备环境中所定义的背景颜色属性。缺省背景方式为 OPAQUE。在这种方式下，Windows 用背景色来填充空隙，缺省的背景色为白色。这是与 WHITE_BRUSH 一致的，许多程序在窗口类中用 WHITE_BRUSH 删除窗口的背景。

通过如下调用：

```
SetBkColor(hdc,rgbColor);
```

可以改变 Windows 用以填充空隙的背景色。与画笔颜色所使用的 rgbColor 值一样，Windows 将这个背景色转换为纯色。可以通过调用 GetBkColor 来获取设备环境中定义的当前背景色。

也可以通过将背景色方式转换为 TRANSPARENT：

```
SetBkMode(hdc,TRANSPARANT);
```

来阻止 Windows 填充空隙。此后，Windows 将忽略背景色，并且不填充空隙。可以通过调用 GetBkColor 来获取当前背景方式(TRANSPARENT 或 OPAQUE)。

14.2.5 绘图方式

在显示器上所画的线的外观也受到设备环境中定义的绘图方式的影响。想像画一条直线，它具有的颜色不但基于画笔的颜色，而且基于画线的显示区域的原色。想像一种方法，程序员可以用同一种画笔，在白色表面上画出黑线，在黑表面上画出白线，而且不用知道表面是什么颜色。这种功能是否有用呢？采用某种绘图方式，这些都可能实现。

当 Windows 用画笔来画线时，它实际上执行一种画笔像素点与目标显示表面像素点之间的按位布尔运算。与像素点进行按位布尔运算叫做“光栅运算”，也简记为“ROP”。由于画一条直线只涉及两种像素点模式(画笔和目标)，因此这种布尔运算又称“二元光栅运算”，简记为“ROP2”。Windows 定义了 16 种 ROP2 代码，指示 Windows 如何组合画笔像素点和目标像素点。在缺省设备环境中，绘图方式定义为 R2_COPYPEN，这意味着 Windows 仅将画笔像素点拷贝至目标，这也是我们通常所熟识的。还有另外 15 种 ROP2 码。

16 种不同的 ROP2 码是怎样产生的呢？为了示范的需要，我们假定使用单色系统。目标色(窗口客户区域的颜色)为黑(将用 0 来表示)或白(将用 1 来表示)。画笔也可以为黑或白。用黑色或白色画笔在黑色或白色目标上画图有四种组合：白笔与白目标、白笔与黑目标、黑笔与白目标、黑笔与黑目标。

画笔在目标上会产生什么结果呢？一种情况是不管画笔和目标的颜色，线总是画为黑

色。这种绘图方式由 ROP2 码 R2_BLACK 指示。另一种可能是线画为黑色。除了画笔与目标都为黑色，在后面的情况下，线画为白色。这看起来有些奇怪。这种绘图方式为 R2_NOTMERGEPEN。Windows 执行目标像素点与画笔像素点的按位“或”运算，并翻转此结果。

下面列出了所有 16 种绘图方式。此表指示了原始画笔(P)与目标颜色(D)是如何组合成结果颜色的。标有“布尔运算”的那一栏用 C 的表示法显示了目标像素点与画笔像素点是如何组合的。

画笔(P)：	1 1 0 0	逻辑	
目的(D)：	1 0 1 0	操作	绘图
结果	0 0 0 0	0	R2_BLACK
	0 0 0 1	～(P | D)	R2_NOTMERGEPEN
	0 0 1 0	～P & D	R2_MASKNOTPEN
	0 0 1 1	～P	R2_NOTCOPYPEN
	0 1 0 0	P & ～D	R2_MASKPENNOT
	0 1 0 1	～D	R2_NOT
	0 1 1 0	P ^ D	R2_XORPEN
	0 1 1 1	～(P & D)	R2_NOTMASKPEN
	1 0 0 0	P & D	R2_MASKPEN
	1 0 0 1	～(P ^ D)	R2_NOTXORPEN
	1 0 1 0	D	R2_NOP
	1 0 1 1	～P | D	R2_MERGENOTPEN
	1 1 0 0	P	R2_COPYPEN(default)
	1 1 0 1	P | ～D	R2_MERGEPENNOT
	1 1 1 0	P | D	R2_MERGEPEN
	1 1 1 1	1	R2_WHITE

通过如下调用

```
SetROP2(hdc,nDrawMode);
```

可以在设备环境中设置一个新的绘图方式。其中，nDrawMode 是表中“绘图方式”一栏的某个值。可以用函数

```
nDrawMode = GetROP2(hdc);
```

获取当前绘图方式。

缺省设备环境为 R2_COPYPEN，它仅仅将画笔颜色传输到目标上。在 R2_NOTCOPOYPEN 方式下，若画笔为黑色，则画成白色；若画笔为黑色，则画成白色。R2_BLACK 方式下，不管画笔和背景色为何种颜色，总是画成黑色。与此相反，R2_WHITE 方式下总是画成白色。R2_NOP 方式就是“不操作”，让目标保持不变。

现在，我们来看一个纯单色系统。实际上，在单色显示器上，Windows 可以通过振动黑白像素点来模拟不同灰度。在振动背景上使用画笔时，Windows 简单地执行像素点基础上的像素位运算。R2_NOT 方式总是翻转目标，不管画笔的颜色是什么。在不知道背景颜色时，这种方式很有用，因为这样能保证画笔是可见的。当然，只能说是几乎可见的，因为如果背景为 50%的灰度，则画笔实际上不可见。

14.2.6 ROP2LOOK 程序

下面的 ROP2LOOK 程序让用户试用 16 种 ROP2 码。

```
# ROP2LOOK.MAK make file

rop2look.exe : rop2look.obj rop2look.def rop2look.res
     $(WINLINK) rop2look,rop2look,NUL,$(WINLIB),rop2look
     rc -t rop2look.res

rop2look.obj : rop2look.c
     $(WINCC) rop2look.c

rop2look.res : rop2look.rc
     $(WINRC) rop2look.rc
```

```
/*----------------------------------------
ROP2LOOK.C -- ROP2 Demonstration Program
----------------------------------------*/

#include <windows.h>

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "Rop2Look" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = NULL ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = szAppName ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
     }

     hwnd = CreateWindow (szAppName,"ROP2 Demonstration Program",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,CW_USEDEFAULT,
```

```
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static LOGPEN lpBlack = { PS_SOLID,1,1,RGB ( 0, 0, 0) },
        lpWhite = { PS_SOLID,1,1,RGB (255,255,255) } ;
    static short nDrawingMode = R2_COPYPEN ;
    HDC hdc ;
    HMENU hMenu ;
    HPEN hPenBlack,hPenWhite ;
    PAINTSTRUCT ps ;
    RECT rect ;
    short i ;

    switch (message)
    {
        case WM_COMMAND:
            hMenu = GetMenu (hwnd) ;
            CheckMenuItem (hMenu,nDrawingMode,MF_UNCHECKED) ;
            nDrawingMode = wParam ;
            CheckMenuItem (hMenu,nDrawingMode,MF_CHECKED) ;
            InvalidateRect (hwnd,NULL,FALSE) ;
            return 0 ;

        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;

            hPenBlack = CreatePenIndirect (&lpBlack) ;
            hPenWhite = CreatePenIndirect (&lpWhite) ;

            SetMapMode (hdc,MM_ANISOTROPIC) ;
            GetClientRect (hwnd,&rect) ;
            SetViewportExt (hdc,rect.right,rect.bottom) ;
            SetWindowExt (hdc,10,4) ;

            for (i = 0 ; i < 10 ; i += 2)
            {
                SetRect (&rect,i,0,i + 2,4) ;
```

```
                FillRect (hdc,&rect,GetStockObject (i / 2)) ;
            }
            SetROP2 (hdc,nDrawingMode) ;

            SelectObject (hdc,hPenWhite) ;
            MoveTo (hdc,1,1) ;
            LineTo (hdc,9,1) ;

            SelectObject (hdc,hPenBlack) ;
            MoveTo (hdc,1,3) ;
            LineTo (hdc,9,3) ;

            EndPaint (hwnd,&ps) ;

            DeleteObject (hPenBlack) ;
            DeleteObject (hPenWhite) ;
            return 0 ;
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; ROP2LOOK.DEF module definition file

NAME ROP2LOOK

DESCRIPTION 'ROP2 Demonstration Program (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

该程序将背景划分为五部分:分别着以白色、浅灰色、灰色、暗灰色和黑色后援刷(这一术语即将论及),然后用两支很粗的画笔画线,白色的画笔在上,黑色的画笔在下。可以在菜单中选择 16 种 ROP2 码之一。图 14.2 显示了绘图方式设置为 R2_NOTMERGEPEN 时白画笔与黑画笔所画出的效果;在这种绘图方式下,白画笔总显示为黑色,黑画笔将翻转目标。

ROP2LOOK 的白色画笔和黑色画笔都使用初始的逻辑画笔结构。注意到这两种画笔的逻辑宽度都是 1,但为什么显示出来有这样粗呢?因为程序使用的是 MM_ANISOTROPIC 映像方式,并将客户区域的宽度设置为 10 个逻辑单位,高度设置为 4 个逻辑单位。所以画笔是客户区域的 1/10 宽。

14.2.7 ROP2 与颜色

在引入颜色后,绘图方式就更有趣,也更复杂了。假设显示器能显示 8 种纯色(如在 Windows 3.0 以前版本下的 EGA 和 VGA)。画笔可以是 8 种纯色之一。这 8 种颜色是红、绿、

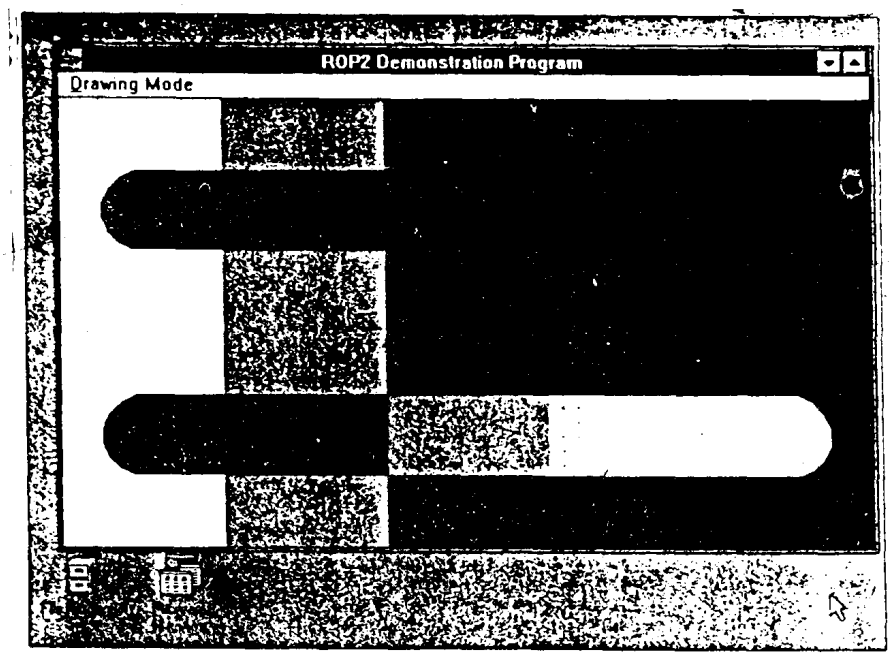


图 14.2　绘图方式设置为 R2_NOTMERGEPEN 时 ROP2LOOK 程序的显示

蓝颜色平面的位组合，如下所示。1 说明显示该颜色，0 表示没有此颜色。

红	绿	蓝	纯色
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

这三种颜色平面的每一种都分别受到光栅运算的影响。例如，假设具有青蓝色背景，洋红色画笔，且绘图方式为 R2_NOTMERGEPEN。那么，画笔实际上会画出怎样的颜色呢？对于红色，画笔为 1(有红色)，目标为 0(无红色)。对于绿色，画笔为 0(无绿色)，目标为 1(有绿色)，结果为 0(无绿色)。对于蓝色，画笔为 1(有蓝色)，目标为 1(有蓝色)，因而结果为 0(无蓝色)。因此，线没有红色，没有绿色，没有蓝色，即颜色为黑色。

14.3　画填充区域

现在再迈出一步，从画线到画图形。Windows 中七个用来画带边缘的填充图形的函数列于下表中：

函数	图形
Rectangle	直角矩形
Ellipse	椭圆
RoundRect	圆角矩形
Chord	椭圆边线上的弧，端点以弦连接
Pie	椭圆周线上的饼形图
Polygon	多边形
PolyPolygon	多个多边形

Windows 用设备环境中选择的当前画笔来画图形的边框。边框还使用当前背景方式、背景颜色和绘图方式，这跟 Windows 画线时一样。关于直线的一切也适用于这些图形的边框。

图形以当前设备环境中选择的刷子来填充。缺省情况下，使用后援对象 WHITE_BRUSH，这意味着图形内里将填为白色。Windows 定义六种后援刷子：WHITE_BRUSH、LTGRAY_BRUSH、GRAY_BRUSH、DKGRAY_BRUSH、BLACK_BRUSHT 和 NULL_BRUSH（也叫 HOLLOW_BRUSH）。前面五种刷子均被用来为 ROP2LOOK 的客户区域着色。

可以用与选择后援画笔相同的方法来选择后援刷。Windows 将 HBRUSH 定义为刷子的句柄，所以可以先定义一个刷子句柄变量：

```
HBRUSH hBrush;
```

通过调用 GetStockObject，可以获取 GRAY_BRUSH 的句柄：

```
hBrush = GetStockObject(GRAY_BRUSH);
```

使用 SelectObject 即可将它选进设备环境中：

```
SelectObject(hdc,hBrush);
```

现在，如果要画一个图形，其内里将为灰色。

如果想画一个没有边框的图形，可在设备环境中设置 NULL_PEN：

```
SelectObject(hdc,GetStockObject(NULL_PEN));
```

如果想画出图形的边框，但不填充内里，则在设备环境中选择 NULL_BRUSH：

```
SelectObject(hdc,GetStockObject(NULL_BRUSH));
```

同创建定制画笔一样，程序员也可以创建定制刷子。后面我们即将谈到这一问题。

14.3.1 边界框

Rectangle，Ellipse，RoundRect，Chord 和 Pie 函数以及 Arc 画线函数都是基于一个矩形边界框之上的。程序员要定义一个包含该对象的框，即边界框；Windows 就在这个框内画出该对象。

最简单的填充对象是矩形：

```
Rectangle(hdc,xLeft,yTop,xRight,yBottom);
```

点(xLeft,yTop)是矩形的左上角，(xRight,yBottom)是矩形的右下角；两个点都以逻辑单位表示。用函数 Rectangle 画出的图形如图 14.3 所示。在 MM_TEXT 映像方式下，xRight 必须大于 xLeft，yBottom 必须大于 yTop。在所有其他映像方式（除了 MM_ISOTROPIC 和 MM_ANISOTROPIC 方式可能例外）下，yBottom 的值应小于 yTop 的值，因为 y 轴上点的坐标随着上升而增大。

以前编过图形程序的程序员会习惯于减去一个像素点的问题。有些图形系统画的图形包含右坐标和下坐标，而有些则只画到(不包含)右坐标和下坐标。Windows 采用后一种方法，但有一种更简单的方法来考虑之。

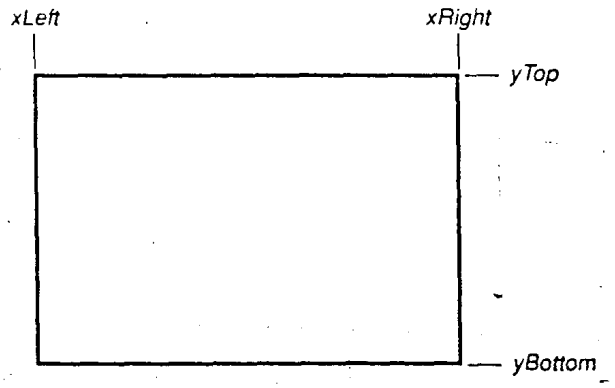


图 14.3　用 Rectangle 函数画出的图形

考虑下面的函数调用：

```
Rectangle(hdc,1,1,5,4);
```

上面我们提到，Windows 在边界框内画图。可以将显示器想像成一个网格，其中，每个像素点都在一个网格单元内。想像把边界框画在网格上，然后在边界框内画矩形。图 14.4 说明了在 MM_TEXT 映像方式下图形是怎样被画出的：

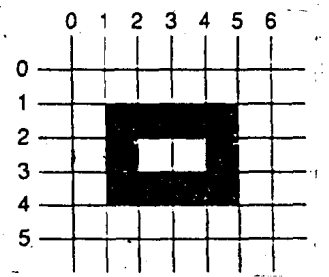


图 14.4

矩形与客户区域的分隔区域有 1 个像素点宽。Windows 用当前刷子来着色矩形的 2 个像素点。

对于除了 PS_INSIDEFRAME 之外的所有画笔风格，如果用来画边线的画笔宽度大于 1 个像素点，则画笔将置于边线的中间，以致线的另一部分可能在边界框的外面。对于 PS_INSITDEFRAME，整条线都画在边界框内。

一旦程序员知道了如何画矩形，也就知道了如何画椭圆，因为它们使用的参数相同：

```
Ellipse(hdc,xLeft,yTop,xRight,yBottom);
```

用 Ellipse 函数画出的图形如图 14.5 所示(加上了虚线构成的边界框)。

Windows 不包含有“正方形”和“圆形”函数。在除了 MM_TEXT 和 MM_ANISOTROPIC 之外的所有映像方式下，画正方形和圆形是很简单的，只要使用 Rectangle 和 Ellipse 函数，并令 xLeft 与 xRight 的差等于 yTop 与 yBottom 的差。在 MM_TEXT 方式下，画正方形和圆形稍

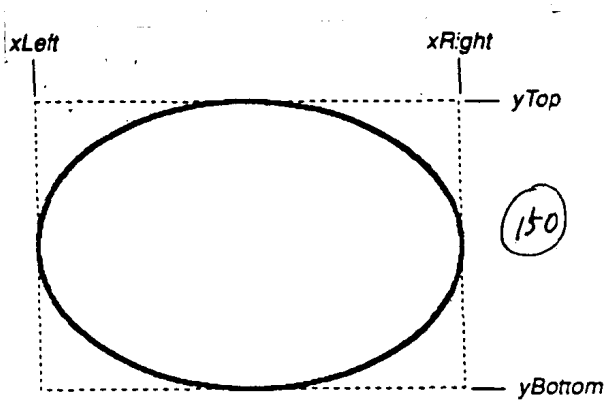


图 14.5　用 Ellipse 函数画的图形

麻烦一些。必须以 ASPECTX 和 ASPECTY 为参数调用 GetDeviceCaps，并根据像素点的方向纵横比换算尺度。在 MM_ANISOTROPIC 方式下，还必须考虑窗口和视口范围的纵横比。

画圆角矩形的函数使用与函数 Rectangle 及 Ellipse 相同的边界框，还包含另外两个参数：

```
RoundRect(hdc,xLeft,yTop,xRight,yBottom,
    xCornerEllipse,yCornerEllipse);
```

用这个函数画出的图形如图 14.6 所示。

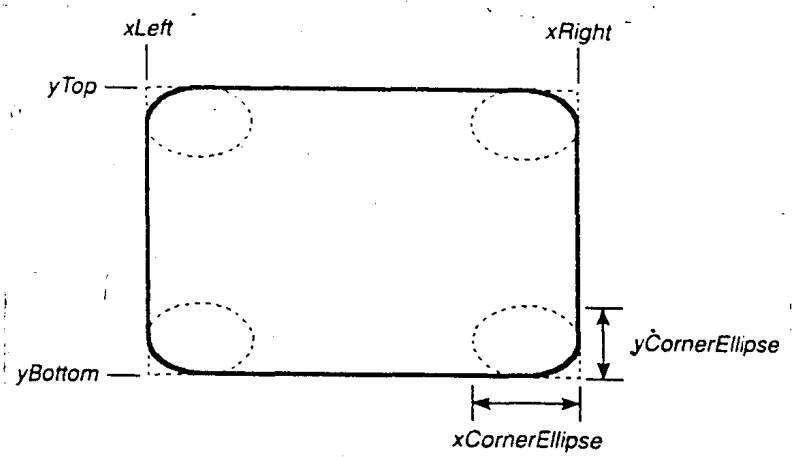


图 14.6　用 RoundRect 函数画出的图形

Windows 用一个小椭圆来画圆角。这个椭圆的宽为 xCornerEllipse，高为 yCornerEllipse，这两个值都是以逻辑单位表示的。想像 Windows 将这个小椭圆分为四个象限，对四个角，每个都使用一个象限。xCornerEllipse 和 yCornerEllipse 的值越大，角就越圆滑。如果 xCornerEllipse 等于 xLeft 与 xRight 的差，且 yCornerEllipse 等于 yTop 与 yBottom 的差，则 RoundRect 函数将画出一个椭圆。

圆角矩形是用下面的公式计算出的角椭圆尺度画出的：

```
xCornerEllipse = (xRight - xLeft) / 4;
yCornerEllipse = (yBottom - yTop) / 4;
```

这是一种简单的方法，但是结果看起来显然并不好，因为角的圆滑度是随矩形尺度的变大而增加的。要矫正这一问题，程序员很可能想让 xCornerEllipse 与 yCornerEllipse 的真实尺寸相等。

Arc、Chord 和 Pie 函数都使用同样的参数：

```
Arc(hdc,xLeft,yTop,xRight,yBottom,xStart,yStart,xEnd,yEnd);
Chord(hdc,xLeft,yTop,xRight,yBottom,xStart,yStart,xEnd,yEnd);
Pie(hdc,xLeft,yTop,xRight,yBottom,xStart,yStart,xEnd,yEnd);
```

用 Arc 函数画出的线如图 14.7 所示；用 Chord 和 Pie 函数画出的线分别如图 14.8 和图 14.9 所示。

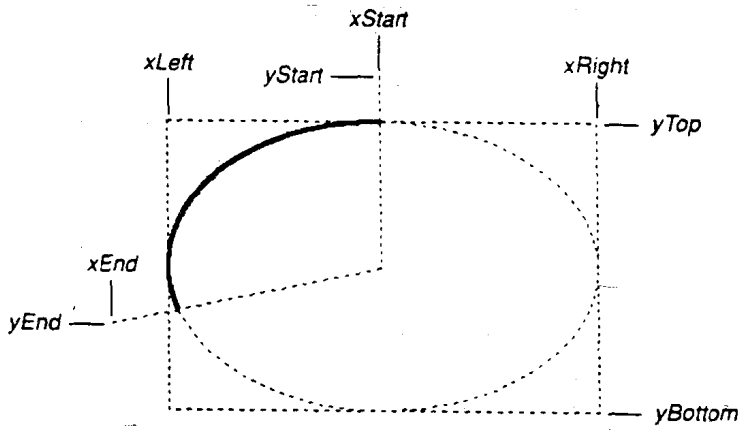


图 14.7　用 Arc 函数画出的线

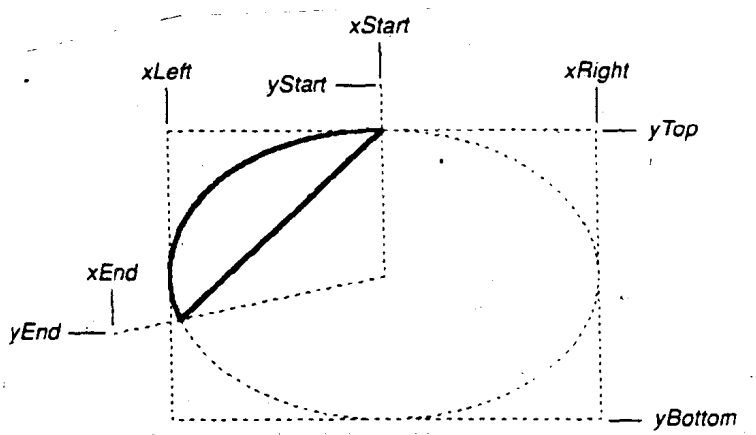


图 14.8　用 Chord 函数画出的图形

Windows 用虚线将(xStart,yStart)与椭圆的中心连接。在该线与边界框的交点，Windows 以反时针方向、沿着椭圆的边框画一条弧。Windows 还用虚线将(xEnd,yEnd)与椭圆的中心连接。在该线与边界框的交点处，Windows 停止画弧。

对于 Arc 函数，这样就结束了。因为弧只是一条椭圆形的线，而不是一个填充区域。对于 Chord 函数，Windows 连接弧线的端点。对于 Pie 函数，Windows 将弧的每个端点与椭圆的中心相连接。弦与饼形图的内里以当前刷子填充。

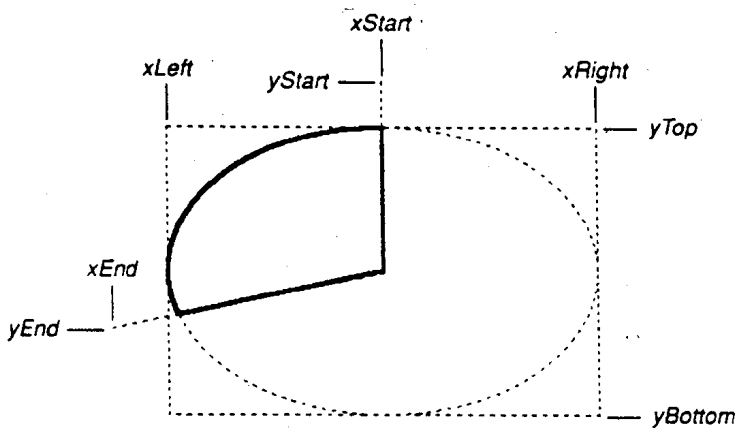


图 14.9　用 Pie 函数画出的图形

14.3.2　ARCS 程序

对于 Arc、Chord 和 Pie 函数中开始位置和终止位置的使用，读者可能会感到困惑。为什么不简单地在椭圆的边线上指定开始和终止点？是的，可以这样做。但是程序员必须计算出这些点来。Windows 的方法不要求这样精确，就能把事做成。

读者可以用下面的 ARCS 程序来尝试弧、弦、饼形图。程序用点式画笔画出边界框和椭圆。对菜单的选择将确定程序画弧、弦还是画饼形图。线或图形是用 3 像素点宽的画笔绘制的。用普通的黑笔将开始点和终止点与椭圆的中心相连接。

```
# ARCS.MAK make file

arcs.exe : arcs.obj arcs.def arcs.res
     $(WINLINK) arcs,arcs,NUL,$(WINLIB),arcs
     rc -t arcs.res

arcs.obj : arcs.c arcs.h
     $(WINCC) arcs.c

arcs.res : arcs.rc arcs.h
     $(WINRC) arcs.rc
```

```
/*------------------------------------------------------
ARCS.C -- Demonstrates Drawing Arcs,Chords,and Pies
------------------------------------------------------*/

#include <windows.h>
#include "arcs.h"

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "Arcs" ;
```

```
    HWND hwnd ;
    MSG msg ;
    WNDCLASS wndclass ;

    if (! hPrevInstance)
    {
        wndclass.style = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc = WndProc ;
        wndclass.cbClsExtra = 0 ;
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = NULL ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = szAppName ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName,"Arcs,Chords,and Pies",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static short cxClient,cyClient,x1,x2,x3,x4,y1,y2,y3,y4,
        nFigure = IDM_ARC ;
    HDC hdc ;
    HMENU hMenu ;
    HPEN hPen ;
    PAINTSTRUCT ps ;

    switch (message)
    {
        case WM_SIZE:
            x3 = y3 = 0 ;
```

```
        x4 = cxClient = LOWORD (lParam) ;
        y4 = cyClient = HIWORD (lParam) ;
        x2 = 3 * (x1 = cxClient / 4) ;
        y2 = 3 * (y1 = cyClient / 4) ;
        return 0 ;
    case WM_COMMAND:
        switch (wParam)
        {
            case IDM_ARC:
            case IDM_CHORD:
            case IDM_PIE:
                hMenu = GetMenu (hwnd) ;
                CheckMenuItem (hMenu,nFigure,MF_UNCHECKED) ;
                CheckMenuItem (hMenu,nFigure = wParam,MF_CHECKED) ;
                InvalidateRect (hwnd,NULL,FALSE) ;
                return 0 ;
        }
        break ;
    case WM_LBUTTONDOWN:
        if (! (wParam & MK_SHIFT))                    // fall through for MK_SHIFT
        {
            x3 = LOWORD (lParam) ;
            y3 = HIWORD (lParam) ;
            InvalidateRect (hwnd,NULL,TRUE) ;
            return 0 ;
        }
    case WM_RBUTTONDOWN:
        x4 = LOWORD (lParam) ;
        y4 = HIWORD (lParam) ;
        InvalidateRect (hwnd,NULL,TRUE) ;
        return 0 ;
    case WM_PAINT:
        hdc = BeginPaint (hwnd,&ps) ;
        hPen = SelectObject (hdc,CreatePen (PS_DOT,1,0L)) ;
        Rectangle (hdc,x1,y1,x2,y2) ;
        Ellipse (hdc,x1,y1,x2,y2) ;
        DeleteObject (SelectObject (hdc,CreatePen (PS_SOLID,3,0L))) ;
        switch (nFigure)
        {
            case IDM_ARC:
                Arc (hdc,x1,y1,x2,y2,x3,y3,x4,y4) ;
                break ;
            case IDM_CHORD:
                Chord (hdc,x1,y1,x2,y2,x3,y3,x4,y4) ;
                break ;
            case IDM_PIE:
                Pie (hdc,x1,y1,x2,y2,x3,y3,x4,y4) ;
                break ;
        }

        DeleteObject (SelectObject (hdc,hPen)) ;
```

```
                MoveTo (hdc,x3,y3) ;
                LineTo (hdc,cxClient / 2,cyClient / 2) ;
                LineTo (hdc,x4,y4) ;
                EndPaint (hwnd,&ps) ;
                return 0 ;
            case WM_DESTROY:
                PostQuitMessage (0) ;
                return 0 ;
        }
        return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; ARCS.DEF module definition file

NAME ARCS

DESCRIPTION 'Arc,Chord,and Pie Drawing Program (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192

/*------------------------
ARCS.RC resource script
------------------------*/

#include "arcs.h"

Arcs MENU
{
    POPUP "&Options"
    {
        MENUITEM "&Arc", IDM_ARC, CHECKED
        MENUITEM "&Chord", IDM_CHORD
        MENUITEM "&Pie", IDM_PIE
    }
}

/*--------------------
ARCS.H header file
--------------------*/

#define IDM_ARC 1
#define IDM_CHORD 2
#define IDM_PIE 3
```

用左鼠标器按钮在客户区域上点一下,ARCS 将用这一点作为开始点,即程序中的点(x3,y3)。用右鼠标器按钮在客户区域上点一下,将设置终止点(x4,y4)。使用单按钮鼠标的用户可以在按下 Shift 键的同时点鼠标,以设置终止点。

ARCS 还显示了一些一般的画笔句柄操作。程序在通过调用 BeginPaint 获取设备环境之后，创建一个点式笔，并将它选进设备环境：

```
hPen = SelectObject(hdc,CreatePen(PS_DOT,1,0L));
```

从 SelectObject 返回的 hPen 句柄是后援 BLACK_PEN 的句柄。

当 ARCS 需要画弧、弦或饼形图，它创建一个 3 像素点宽的画笔，并将它选进设备环境：

```
DeleteObject(SelectObject(hdc,CreatePen(PS_SOLID,3,0L)));
```

同时用 DeleteObject 删除从 SelectObject 返回的画笔句柄(即点式笔的句柄)。

在需要画线以连接开始和终止点与椭圆中心线时，ARCS 选择 hPen(后援 BLACK_PEN 的句柄)进入设备环境，并删除从 SelectObject 返回的 3 像素点宽的画笔：

```
DeleteObject(SelectObject(hdc,hPen));
```

现在，创建的两个画笔都已经被删除了。当前选择到设备环境中的画笔是一种后援画笔，所以设备环境可以释放。

14.3.3 饼状图的三角学

如果使用 Pie 函数创建饼形图，则每个饼形图的大小将取决于数据项的相对大小。这涉及到计算从饼形内角导出的饼形图的起点和终点。

下面温习三角几何。在笛卡尔坐标系(x 向右增加，y 向上增加)中，我们可以画一个如下所示的角：

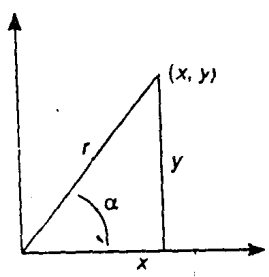


图 14.10

三角形的三条边与角 α 之间的关系由下面三个公式给出：

$$\sin(\alpha) = y/r$$

$$\cos(\alpha) = x/r$$

$$\tan(\alpha) = y/x$$

如果知道角 α(在饼形图中，它将是圆的一部分)和 r(它将是圆的半径)，就可以确定点(x,y)等于：

```
(r * cos(α),r * sin(α))
```

在 C 的库函数 sin 和 cos 中，角是用弧度来指定大小的。在 360°中，有 2×PI 个弧度。

让我们试着画一个使用 5 个数字的饼形图。为了简便，我们将它定义为常量：

```
#define NUM 5
```

在实际的程序中，它将是变量。

同样为了简便，将一个圆周的弧度定义为常量 TWO_PI：

```
#define TWO_PI (2.0 * 3.14159)
```

然后，定义以下变量：

```
static short nValues [NUM] = {3,5,2,7,4};
short i,nSum [NUM +1];
```

nValues 的初始值是我们将用来画图的数据。在实际的程序中，这些值将会是变量。nSum 数组设置为数据值的累加和，数组的第一个元素设置为 0：

```
nSum [0] = 0;
for (i = 0;i < NUM;i++)
    nSum [i+1] = nSum [i] + nValues [i];
```

数组元素 nSum[NUM]是五个值之和。

现在，我们准备开始画饼形图。将映像方式设置为 MM_ISOTROPIC，在这种方式下画圆非常容易：

```
SetMapMode (hdc,MM_ISOTROPIC);
SetWindowExt (hdc,400,400);
SetViewportExt (hdc,xClient,-yClient);
SetViewportOrg (hdc,xClient / 2,yClient / 2);
```

逻辑点(0,0)是客户区域的中心，x 和 y 坐标定义了一个普通的笛卡尔坐标系。

饼形图具有 100 逻辑单位的弧度。下面是画出五个饼段的代码：

```
for (i = 0 ; i < NUM ; i++)
    Pie (hdc,-100,100,100,-100,
        (short) (100.0 * cos (TWO_PI) * nSum [i] /nSum [NUM])),
        (short) (100.0 * sin (TWO_PI) * nSum [i] /nSum [NUM])),
        (short) (100.0 * cos (TWO_PI) * nSum [i+1] /nSum [NUM])),
        (short) (100.0 * som (TWO_PI) * nSum [i+1] /nSum [NUM])));
```

这些程序所产生的饼形图如图 14.11 所示。第一个饼楔形在饼形图的右边，它正好在 x 轴之上。其他饼楔形以逆时针方向画出。

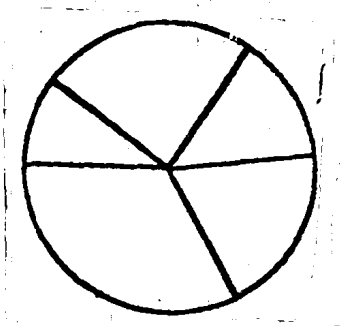


图 14.11　用 Pie 函数画出的饼形图

值

```
TWO_PI * nSum [i] / nSum [NUM]
```

和

```
TWO_PI * nSum [i+1] / nSum[NUM]
```

是各项的累加和与总和的比例，转换为从水平方向以逆时针方向度量的角。第二个公式包含了特定饼楔形所代表的项；第一个则不然。通过计算角的余弦和正弦并乘以 100，我们计算出了角的起点和终点。

14.3.4 Polygon 函数和多边形填充方式

Polygon 是第六个画带边线的填充图形的函数。该函数的调用与 PolyLine 函数相似：

```
Polygon(hdc,lpPoints,nCount);
```

lpPoints 参数是一个 far 指针，指向一个 POINT 结构数组(逻辑坐标)，nCount 是点的数目。如果该数组的最后一个点与第一个点不同，则 Windows 将会再加一条线，将最后一个点与第一个点连起来。在 PolyLine 函数中，Windows 不会这样做。

Windows 以两种方法之一且用当前刷子填充这个带边界的区域。至于选用两种方法中的哪一种，则取决于设备环境中定义的当前多边形填充方式。在缺省设置下，多边形填充方式为 ALTERNATE。这意味着 Windows 只填充那些可以通过穿越奇数号线而从多边形的外面访问的内里，其他内里不被填充。也可以将多边形填充方式设置为 WINDING，在这种情况下，Windows 一般填充所有内里区域。这两个多边形填充方式简单地用一个五角星演示。在图中，左边的星是用 ALTERNATE 方式画的，右边的星是用 WINDING 方式画的。两个图都是用下面定义的点数组组成的：

```
static POINT pt [] =
    { -59,-81,0,100,59,-81,-95,31,95,31 };
```

星的五个点是从三角函数表计算出来的。WM_PAINT 形式如下：

```
case WM_PAINT
    hdc = BeginPaint (hwnd,&ps);

    hPen = CreatePen (PS_SOLID),3,0L);
    SelectObject (hdc,hPen);
    SelectObject (hdc,GetStockObject (LIGRAY_BRUSH));
    SetMapMode (hdc,MM_ISOTROPIC);
    SetWindowExt (hdc,440,-220);
    SetViewportExt (hdc,xClient,yClient);
    SetWindowOrg (hdc,-110,110);

    SetPolyFillMode (hdc,ALTERNATE);
    Polygon (hdc,pt,sizeof (pt) / sizeof (POINT));

    SetWindowOrg (hdc,-330,110);

    SetPolyFillMode (hdc,WINDING);
    Polygon (hdc,pt,sizeof (pt) / sizeof * (POINT);

    EndPaint (hwd,&ps);
    DeleteObject (hPen);
    break;
```

PolyPolygon 函数画出多个多边形。

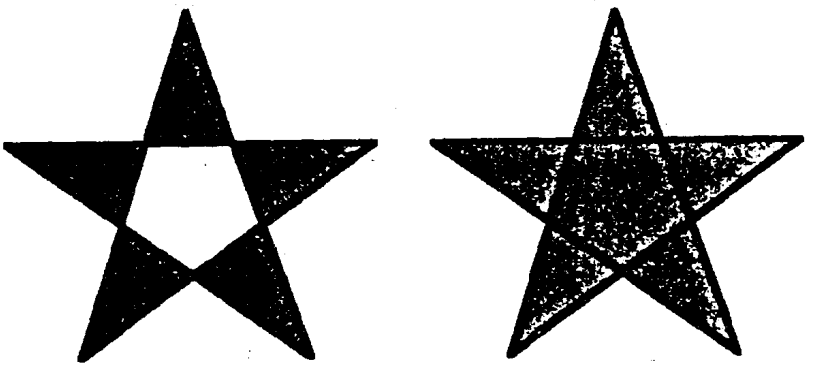


图 14.12　用两种多边形填充方式画出的图
ALTERNATE(左)和 WINDING(右)

14.3.5　用刷子填充内里

Rectangle、RoundRect、Ellipse、Chord、Pie、Polygon 和 PolyPolygon 图形的内里是用选进设备环境的当前刷子(也称"模式")来填充的。刷子是一个 8×8 的位图,它被用来填充内里区域。

当 Windows 用振动的方法来显示多于可从显示器上得到的颜色时,实际上是给颜色使用了刷子。在单色系统上,Windows 能用黑色和白色像素点的振动创建 64 种不同效果的灰色。更精确地说,Windows 能创建 64 种不同的单色刷子。对于纯黑色,8×8 位图中的所有位均为 0。第一种灰色外形有 1 位为 1,第二种灰色外形有两位为 1,以此类推,直到 8×8 位图中所有位均为 1,这就是纯白色。在彩色视频系统上,振动色也是位图,可得到更多的颜色。

我们已经使用过后援刷子。Windows 还有四个函数,使程序员可以创建逻辑刷子。用 SelectObject 将刷子选进设备环境。与逻辑画笔一样,逻辑刷子也是 GDI 对象。程序员创建的刷子都必须被删除,但是在它还在设备环境中时不能删除。

下面是第一个创建逻辑刷子的函数:

```
hBrush = CreateSolidBrush(rgbColor);
```

函数中的 Solid 这个词并不是指刷子为纯色。在将刷子选进设备环境时,Windows 为振动色创建一个 8×8 位图,并为刷子使用该位图。

还可以用由水平、垂直和斜线组成的"影线标记"来创建刷子。这种风格的刷子非常频繁地用来着色条形图的内里以及在绘图仪上绘图。创建影线刷子的函数为:

```
hBrush = CreateHatchBrush(nHatchStyle,rgbColor);
```

nHatchStyle 参数描述影线标记的外观。此参数可以是 HS_HORIZONTAL, HS_VERTICAL, HS_FDIAGONAL, HS_BDIAGONAL, HS_CROSS, HS_DIAGCROSS 中之一。图 14.13 显示了每种风格所产生的影线标记类型。

CreateHatchBrush 的 rgbColor 参数是影线的颜色。在将刷子选进设备环境时,Windows 将这种颜色转换成最近的纯色。影线之间的区域根据设备环境中定义的背景方式和背景颜色来着色。如果背景方式为 OPAQUE,则背景色(它也被转换成纯色)用来填充线之间的空间。在这种情况下,影线和填充色都不能是振动色。如果背景方式为 TRANSPARENT,则 Windows 只画出影线,不填充它们之间的区域。

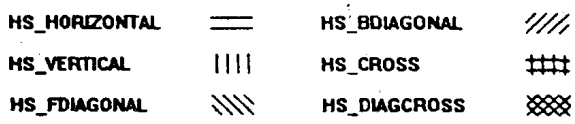


图 14.13　六种影线刷子风格

前边我们讨论过当在彩色显示器上开发的程序在单色显示器上运行时画笔颜色所遇到的问题。必须意识到，在为影线标记和刷子背景选择颜色时，会有同样的问题存在。在彩色显示器上看起来不错的着色影线标记，如果在单色显示器上，有可能会消失在背景里。

由于刷子总是 8×8 的位图，所以因显示它们的设备分辨率的不同，影线刷子的外观也会发生变化。图中所示的每种影线标志都画在一个 32×16 像素点的矩形里，这意味着 8×8 的位图在水平方向重复 4 次，在垂直方向重复 2 次。在一个 300 dpi 的激光打印机上，同样的 32×16 像素点矩阵将占据大约 1/9 寸宽、1/19 寸高的区域。

也可以用 CreatePatternBrush 创建自己的基于位图的刷子。

```
hBrush = CreatePatternBrush(hBitmap);
```

这个函数在前面已讨论过，它是一个 8×8 位图的句柄。后面将说明如何获取位图的句柄。

Windows 还提供了一个函数，它包含了其他三个创建刷子的函数（CreateSolidBrush、CreateHatchBrush 和 CreatePatternBrush）：

```
hBrush = CreateBrushIndirect(&logbrush);
```

变量 logbrush 是一个类型为 LOGBRUSH（“逻辑刷子”）的结构。该结构的三个域如下表所示。lbStyle 域的值确定了 Windows 如何解释其他两个域的值。

lbStyle	lbColor(DWORD)	lbHatch(short)
BS_SOLID	刷子的颜色	忽略
BS_HOLLOW	忽略	忽略
BS_HATCHED	影线的颜色	影线刷子风格
BS_PATTERN	忽略	位图句柄

前面我们用 SelectObject 将逻辑画笔选择到设备环境中，用 DeleteObject 删除逻辑画笔，用 GetObject 获取一个逻辑画笔的信息。对于刷子，同样能使用这三个函数。如果具有刷子的句柄，就可以用 SelectObject 将该刷子选进设备环境：

```
SelectObject(hdc,hBrush);
```

以后，可以用 DeleteObject 函数删除所创建的刷子：

```
DeleteObject(hBrush);
```

不要将一个当前选进设备环境中的刷子删掉。如果需要获取刷子的信息，可以调用 GetObject：

```
GetObject(hBrush,sizeof(LOGBRUSH),(LPSTR) &logbrush);
```

其中，logbrush 是一个类型为 LOGBRUSH 的结构。

14.3.6 刷子和位图

在使用 CreatePatternBrush 函数或者使用 CreateBrushIndirect 函数且 lpStyle 域设置为 BS_PATTERN 时，都需要一个位图句柄。位图必须至少为 8 像素点宽、8 像素点高。如果更大，Windows 只将位图的左上部分提供给刷子使用。

由于刷子和位图均是 GDI 对象，程序员必须在程序终止前删除自己创建的刷子和位图。如果基于位图创建刷子，则 Windows 制作一份位图的拷贝，以供在用该刷子绘图时使用。在调用 CreatePatternBrush（或 CreateBrushIndirect）之后立即删除位图将不会影响刷子。同样，删除刷子也不会影响位图。可以用图像编辑器创建位图文件（带扩展名 .BMP），将该文件名包含进资源描述文件的一个 BITMAP 语句中，然后将位图载入程序。LoadBitmap 函数返回一个类型为 HBITMAP 的句柄：

```
hBitmap = LoadBitmap(hInstance,lpszBitmap);
```

变量 lpszBitmap 是资源描述文件中位图的名字。

获取位图句柄的第二种方法是使用函数 CreateBitmap：

```
hBitmap = CreateBitmap(nWidth,nHeight,nPlanes,nBitsPixel,lpBits);
```

要创建一个由刷子使用的位图，nWidth 和 nHeight 应被设置为 8。要得到一个单色位图，nPlanes 和 nBitsPixel 都必须设置为 1。变量 lpBits 是一个长指针，指向包含位图像素点模式的字节数组。如果要创建一个未初始化的位图，可以设置此参数为 NULL，在这种情况下位图将包含随机数据。

获取位图句柄的第三种方法是：

```
hBitmap = CreateCompatibleBitmap(hdc,nWidth,nHeight);
```

它创建一个位图，具有与 hdc 指示的设备环境同样数目的每像素点颜色位和同样数目的颜色平面。这个函数需要 hdc 的唯一理由就是获取这些颜色信息。位图开始包含的是随机数据。

获取位图句柄的最后一种方法要求一个类型为 BITMAP 的结构（下面其变量名为 bitmap）指针：

```
hBitmap = CreateBitmapIndirect(&bitmap);
```

看一看 HBITMAP 类型与 BITMAP 类型的区别。HBITMAP 类型是位图的句柄。BITMAP 类型是描述位图的结构。BITMAP 结构有 7 个域，下面将一一列出。

其中五个域类似于 CreateBitmap 函数的参数。

bmType	设置为 0
bmWidth	位图宽度，以像素点为单位
bmHeight	位图高度，以像素点为单位
bmWidthBytes	每个光栅行的字节数
bmPlanes	颜色平面数
bmBitsPixel	每像素点邻接颜色位数
bmBits	far 指针，指向包含位图模式的字节数组

在具有了位图的句柄后，可以用 GetObject 获取位图的信息：

```
GetObject(hBitmap,sizeof(BITMAP),(LPSTR) &bitmap);
```

其中，bitmap 是类型为 BITMAP 的结构。但是，GetObject 不拷贝一个有效 far 指针到 bmBits 域中。要获得组成位图的实际位，可以调用：

```
GetBitmapBits(hBitmap,dwCount,lpBits);
```

GetBitmapBits 拷贝 dwCount 个字节到地址为 lpBits 的数组中。也可以用 SetBitmapBits 函数设置位图的各位：

```
SetBitmapBits(hBitmap,dwCount,lpBits);
```

BITMAP 结构的 bmBits 域和 CreateBitmap、SetBitmapBits、GetBitmapBits 函数的 lpBits 参数是 far 指针，指向定义位图模式的字节数组。字节数组从最上面的扫描行开始。彩色位图可能用每像素点多个位或者每扫描行多个颜色平面来组织。如果希望将位图用作刷子，且确保它能在所有设备上工作，就应该用单色位图。

字节数组的总长度等于(使用逻辑位图结构的域)：

```
bmPlanes × bmHeight × bmWidthBytes
```

bmWidthBytes 是每个扫描行的宽度(以字节为单位)。这个值必须大于或者等于

```
bmWidth × bmBitsPixel / 8
```

的最小偶数。换句话说，每一扫描行都根据需要填以字节，以获取整数个字。扫描行被填在右边，因为字节是从左到右安排的。第一个字节的最有效位是最左像素点。

14.3.7 位图刷子的创建与使用

基于位图创建和使用刷子所需的背景知识，我们已经介绍过了。现在，我们利用这些知识进行工作。假设程序员想画一个矩形，里面用看起来像小砖块的刷子填充，如图 14.14 所示。

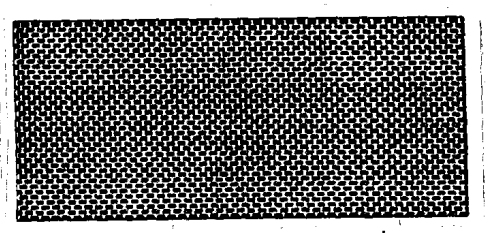


图 14.14 以定制刷子填充的图形

所需要的位图具有一个如下所示的像素点模式。

这是一个单色位图，其高度和宽度均为 8。基于这个位图，可有三种方法创建刷子。下面将一一介绍。

方法一

先在 SDKPAINT 中创建一个 8×8 的单色位图，它能模拟如上图所示的图案，并将它存于文件 BRICK.BMP 中。在资源描述文件里加入一条将该文件识别为位图并将其命名为“Brick”的语句：

```
Brick BITMAP brick.bmp
```

在程序中，定义类型为 HBITMAP(位图句柄)和 HBRUSH(刷子句柄)的变量：

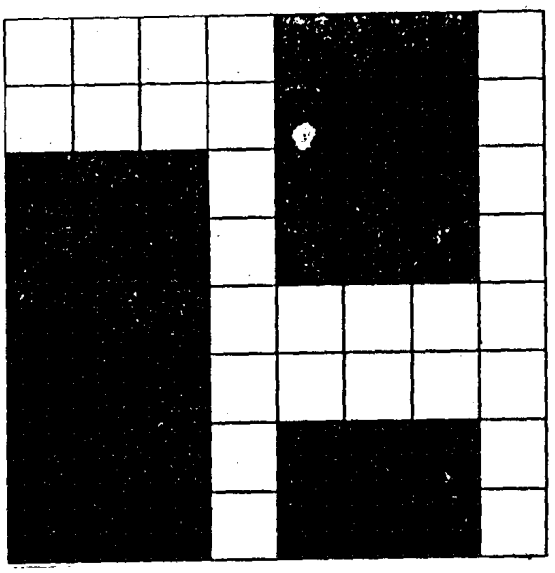


图 14.15

```
HBITMAP hBitmap;
HBRUSH hBrush;
```

这两个句柄用下面两条语句设置：

```
hBitmap = LoadBitmap(hInstance,"Brick");
hBrush = CreatePatternBrush(hBitmap);
```

如果具有一个有效的设备环境，则将此刷子选进设备环境，并显示矩形：

```
SelectObject(hdc,hBrush);
Rectangle(hdc,xLeft,yTop,xRight,yBottom);
```

在释放设备环境之后，删除刷子和位图：

```
DeleteObject(hBrush);
DeleteObject(hBitmap);
```

不一定非得等到释放设备环境之后再删除位图，实际上在创建基于位图的刷子之后，任何时候删除位图都是可以的。

方法二

这种方法在程序内将位图像素点定义为八个无符号整数组成的数组。每个整数对应于位图模式中的一个扫描行。位为 1 代表白色，位为 0 代表黑色：

```
HBITMAP hBitmap;
HBRUSH hBrush;
static WORD wBrickBits [] =
    { 0xFF,0x0C,0x0C,0x0C,0xFF,0xC0,0xC0,0xC0) ;
```

位图是用 CreateBitmap 且引用这个整数数组而创建的：

```
hBitmap = CreateBitmap (8,8,1,1,(LPSTR) wBrickBits);
```

```
hBrush = CreatePatternBrush (hBitmap);
SelectObject (hdc,hBrush);
Rectangle (hdc,xLeft,yTop,xRight,yBottom) ;
```

当不再使用刷子和位图(且刷子不被选择到有效设备环境中)之后,即可删除刷子和位图:

```
DeleteObject(hBrush);
DeleteObject(hBitmap);
```

可以不用整数数组而用无符号字符数组来作位图。不过,由于每个扫描行必须包含偶数个字节,因此必须在每个字节后面插入一个 0:

```
static unsigned char cBrickBits [] =
    { 0xFF,0,0x0C,0,0x0C,0,0x0C,0,
      0xFF,0,0xC0,0,0xC0,0,0xC0,0 };
```

方法三

该方法类似于方法二,不过这里是用逻辑位图和逻辑刷子结构来创建位图和刷子的。先定义以下变量:

```
HBITMAP hBitmap;
HBRUSH hBrush;
static BITMAP bitmap = {0,8,8,2,1,1 };
static LOGBRUSH logbrush = {BS_PATTERN,0L };
static WORD wBrickBits [] =
    { 0xFF,0x0C,0x0C,0x0C,0xFF,0xC0,0xC0,0xC0 };
```

逻辑位图结构的最后一个域未被初始化。该域必须包含一个 far 指针,指向定义位图模式的字节数组。不用初始化带有这个指针的结构,只需使用一赋值语句:

```
bitmap. bmBits = (LPSTR) wBrickBits;
```

如果将一个局部数据项的 far 地址赋给结构的该域,则一定要小心。如果程序运行于 Windows3. 0 的实模式下,且程序的数据段在内存中被移动,远程地址将变为无效(这种情况可能在 GetMessage、LocalAlloc 和 LocalReAlloc 调用之后产生)。比较好的方法是正好在使用该结构之前赋值。赋值后,就能用该结构创建位图:

```
hBitmap = CreateBitmapIndirect(&bitmap);
```

由于已经具有了位图句柄,就能用 CreateBrushIndirect 函数创建刷子:

```
logbrush. lbHatch = hBitmap ;
hBrush = CreateBrushIndirect (&logbrush);
SelectObject (hdc,nBrush);
Rectangle (hdc,xLeft,yTop,xRight,yBottom);
```

最后,删除刷子和位图:

```
DeleteObject(hBrush);
DeleteObject(hBitmap);
```

14. 3. 8 刷子的对齐

在用刷子填充某区域时,Windows 在水平和垂直两个方向重复 8×8 的位图。根据

Windows 对位图左上角相对于显示表面的对齐方式，刷子的外观会稍有不同。设备环境中确定这种布局属性为“刷子原点”。这种属性总是以屏幕坐标为准来表示的。如果用 BeginPaint 或者 GetDc 获取一个设备环境，则刷子原点最初设置为窗口客户区域的左上角。如果客户区域的左上角离屏幕左边为 30 个像素点、上边为 20 个像素点，则刷子原点设置为点(30,20)。

Windows 在客户区域中使用刷子时，刷子位图的左上角与客户区域中其 x 及 y 坐标均为 8 的倍数的设备点相吻合。让我们来看一个例子，它涉及到一个风格为 HS_FDIAGONAL 的影线刷子，外观如图 14.16 所示。

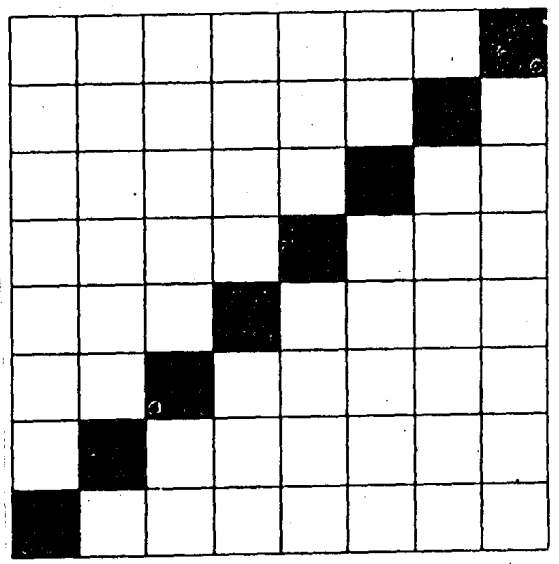


图 14.16

在 MM_TEXT 方式下，如果要画一个以 HS_FDIAGONAL 刷子来填充的矩形，且如果矩形的左上角在逻辑点(8,8)处，则向下的影线将与矩形的左上角相交。而如果矩形开始于逻辑点(8,4)，则影线将离角的上部有 4 个像素点的距离。

在大多数情况下，绘图时不需要调整刷子原点，但有时程序员可能想调整它。调整的过程包括以下三步：

(1)对刷子调用 UnrealizeObject(不能对后援刷子调用 UnrealizeObject)。

(2)用 SetBrushOrg 设置刷子原点，记住使用屏幕坐标。

(3)用 SelectObject 将刷子选进设备环境。

需要改变刷子原点的一种情况是想让子窗口的背景与父窗口的背景相融合。在获取子窗口的设备环境句柄之后，刷子改变为父窗口的左上角。这种情况下，需要改变一个设备环境中的刷子原点以便它与另一个设备环境中的刷子原点一致。

在画几个共享同一设备环境的图形时，如果程序员不想让刷子相重合，也需要改变刷子原点。例如，假设程序员想用刷子着色条形图中的条。如果不在乎刷子的布局，可以用一个如下所示的函数来画每个条：

```
void DrawBarl (hdc,xLeft,yTop,xRight,yBottom,hBrush)
    HDC hdc;
```

```
    short xLeft,yTop,xRight,yBottom;
    HBRUSH hBrushOriginal ;

    hBrushOriginal = SelectObject (hdc,hBrush);
    Rectangle (hdc,xLeft,yTop,xRight,yBottom);
    SelectObject (hdc,hBrushOriginal) ;
```

该函数仅选择作为函数参数传递的刷子句柄(保存原来所选刷子的句柄),画出矩形,然后将原刷子选回设备环境中。如果程序用该例程画几个以 HS_BDIAGONAL 刷子填充的邻接条,结果将如图 14.17 所示。注意所有三个条的影线是对准的,其效果似乎是一条线从右画到左,但效果不太好。

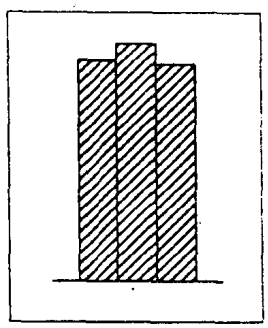


图 14.17　三个具有同样设备环境和刷子原点的条

为了避免这种结果,需要分别调整每个条的刷子,这可以用下面的函数来完成:

```
void DrawBar2 (hwnd,hdc,xLeft,yTop,xRight,yBottom,hBrush)
    HWND hwnd;
    HDC hdc;
    short xLeft,yTop,xRight,yBottom;
    HBRUSH hBrush;
{
    HBRUSH hBrushOriginal;
    POINT pt;

    UnrealizeObject (hBrush)
    pt.x = xLeft;
    pt.y = yTop;
    LPtoDP (hdc,&pt,1);
    ClientToScreen (hwnd,&pt);

    SetBrushOrg (hdc,pt.x,pt.y);

    hBrushOriginal = SelectObject (hdc,hBrush);
    Rectangle (hdc,xLeft,yTop,xRight,yBottom);
    SelectObject (hdc,hBrushOriginal);
}
```

先调用 UnrealizeObject 使刷子无效,然后将刷子原点设置为每个条的左上角。这要求将条的左上角用 LPtoDP 转换成客户区域坐标,然后用 ClientToScreen 转换为屏幕坐标。结果如图 14.18 所示。现在影线是从每个条的左上角开始画的,各条之间没有对齐。

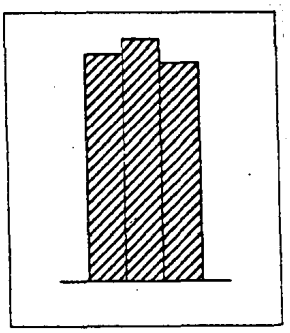


图 14.18　具有相同设备环境和不同刷子原点的条形图

14.4　矩形、区域和剪取

Windows 还提供了另外几种使用 RECT(矩形)结构和"区域"的绘图函数。区域(region)是屏幕上的一块地方,它是矩形、其他多边形和椭圆的组合。

14.4.1　矩形函数

下面三个绘图函数需要一个指向矩形结构的指针:

```
FillRect(hdc,&rect,hBrush);
FrameRect(hdc,&rect,hBrush);
InvertRect(hdc,&rect);
```

在这些函数中,rect 参数是一个 RECT 类型的结构。它包含有 4 个域:left,top,right 和 bottom。这个结构中的坐标被当作逻辑坐标。

FillRect 用指定刷子填充矩形(直到但不包含右坐标和下坐标)。该函数不需要先将刷子选进设备环境。

FrameRect 用刷子画矩形边框,但是不填充矩形。用刷子画矩形看起来有点奇怪,因为对于我们所介绍过的函数(如 Rectangle),其边线都是用当前画笔绘制的。FrameRect 允许用户画一个不一定为纯色的矩形边框。如果逻辑单位大于设备单位,则边框将会为 2 个像素点宽或者更宽。

InvertRect 将矩形中所有像素点翻转,1 转换成 0,0 转换为 1。该函数将白色区域转变成黑色,黑色区域转变为白色,绿色区域转变成洋红色。

Windows 还提供了 9 个函数,使程序员可以更容易、更清楚地操作 RECT 结构。例如,要将 RECT 结构的四个域设置为特定值,习惯的方法是使用如下的程序段:

```
rect.left = xLeft;
rect.top = xTop;
rect.right = xRight;
rect.bottom = xBottom;
```

但是,通过调用 SetRect 函数,只需要一条语句就可得到同样的结果:

```
SetRect(&rect,xLeft,yTop,xRight,yBottom);
```

在想做以下事情之一时,可以很方便地选用其他八种函数:

· 将矩形沿 x 轴和 y 轴移动几个单元：

```
OffsetRect(&rect,x,y);
```

· 增减矩形的尺寸：

```
InflateRect(&rect,x,y);
```

· 矩形各域设置为 0：

```
SetRectEmpty(&rect);
```

· 矩形拷贝给另一个矩形

```
CopyRect(&DestRect,&SrcRect);
```

· 获取两个矩形的交：

```
IntersectRect(&DestRect,&SrcRect1,&SrcRect2);
```

· 获取两个矩形的并：

```
UnionRect(&DestRect,&SrcRect1,&SrcRect2);
```

· 确定矩形是否为空：

```
bEmpty = IsRectEmpty(&rect);
```

· 确定点是否在矩形内：

```
bInRect = PtInRect(&rect,point);
```

在多数情况下，这些函数的等价代码很简单。有时候，程序员会发现，使用这些函数实际上增大了 .EXE 文件。实际上，对某些函数来讲，等价代码的源代码文件占据的空间更少。例如，可以用下列语句来代替 CopyRect 函数调用：

```
DestRect = SrcRect;
```

14.4.2 区域的创建和绘制

区域是对某段显示屏幕的描述，是矩形、其他多边形和椭圆的组合。可以用区域来绘图和剪取图形。通过选取区域进入设备环境，就能将区域用于剪取操作。

与画笔、刷子和位图一样，区域也是 GDI 对象。可以用 DeleteObject 删除所创建的区域。

创建区域时，Windows 返回一个类型为 HRGN 的句柄。最简单的区域是描述矩形的那一种。可以用以下两种方法之一创建矩形区域：

```
hRgn = CreateRectRgn(xLeft,yTop,xRight,yBottom);
```

或

```
hRgn = CreateRectRgnIndirect(&rect);
```

也可以用下边两条语句中的一种来创建椭圆区域：

```
hRgn = CreateEllipticRgn(xLeft,yTop,xRight,yBottom);
```

或

```
hRgn = CreateEllipticRgnIndirect(&rect);
```

CreateRoundRectRgn 创建圆角矩形区域。

与使用 Polygon 函数类似，可用下面的语句创建多边形区域：

```
hRgn = CreatePolygonRgn(&point,nCount,nPolyFillMode);
```

其中，point 参数是 POINT 类型结构的数组，nCount 是点数，nPolyFillMode 要么是 ALTERNATE，要么是 WINDING。也可以用 CreatePolyPolygonRgn 创建多个多边形区域。

现在读者明白了什么？这些区域有什么特殊之处？下面的函数发挥了区域的作用：

```
nRgnType = CombineRgn(hDestRgn,hSrcRgn1,hSrcRgn2,nCombine);
```

它组合两个源区域(hSrcRgn1 和 hSrcRgn2)，并导致目标区域句柄(hDestRgn)指向这个组合区域。这三个区域句柄必须都是有效的，但 hDestRgn 原来所描述的区域被删除。在使用这个函数之前，程序员可能会将 hDestRgn 初始化为指向一个小矩形区域。

nCombine 参数描述 hSrcRgn1 和 hSrcRgn2 区域是如何组合的：

nCombine 的值	新区域
RGN_AND	两个原区域的重叠部分
RGN_OR	两个源区域占据的全部地盘
RGN_XOR	除了重叠部分外，两个源区域占据的地方
RGN_DIFF	hSrcRgn1 占有、且 hSrcRgn2 不占有的地方
RGN_COPY	使 hSrcRgn1 与 hSrcRhn2 相同

从 CombineRgn 返回的 nRgnType 值为如下情形之一：

NULLREGION	指示空区域；
SIMPLEREGION	指示一个简单矩形、椭圆或多边形；
COMPLEXREGION	指示矩形、椭圆和/或多边形的组合；
ERROR	说明出现错误。

在具有区域的句柄之后，就可以在下面四个绘图函数中使用它：

```
FillRgn (hdc,hRgn,hBrush) ;
FrameRgn (hdc,hRgn,hBrush,xFrame,yFrame);
InvertRgn (hdc,hRgn);
PaintRgn (hdc,hRgn);
```

FillRgn、FrameRgn 和 InvertRgn 函数类似于 FillRect、FrameRect 和 InvertRect 函数。FrameRgn 的 xFrame 和 yFrame 参数是在区域周围所画边框的逻辑宽度和高度。PaintRgn 函数用当前选进设备环境的刷子填充该区域。所有这些函数都假设区域是以逻辑坐标定义的。

不再需要某区域之后，就可以用与删除其他 GDI 对象的函数相同的函数删除之：

```
DeleteObject(hRgn);
```

14.4.3 矩形与区域的剪取

区域在剪取操作中也很有用。InvalidateRect 函数使显示的一个矩形区域失效，并产生一个 WM_PAINT 消息。我们经常用 InvalidateRect 函数删除客户区域，并产生一个 WM_PAINT 消息：

```
InvalidateRect(hwnd,NULL,TRUE);
```

通过调用 GetUpdateRect 可以获取失效矩形的坐标，用 validateRect 函数可以使客户区域的矩形有效。在接收一个 WM_PAINT 消息后，无效矩形的坐标可以从 PAINTSTRUCT 结构得到，该结构是用 BeginPaint 函数填充的。这个无效矩形还定义了一个“剪取区域”。不能在剪取区域外绘图。

Windows 有两个使用区域的函数，它们类似于 InvalidateRect 和 ValidateRect，它们是：

```
InvalidateRgn(hwnd,hRgn,bErase);
```

和

```
ValidateRgn(hwnd,hRgn);
```

当接收到作为无效区域结果的 WM_PAINT 消息时，剪取区域将不必为矩形。

用以下两个函数之一

```
SelectObject(hdc,hRgn);
```

或

```
SelectClipRgn(hdc,hRgn);
```

将一个区域选进设备环境，可以创建自己的剪取区域。剪取区域设置为使用设备坐标。

GDI 制作一份剪取区域的拷贝，所以在将它选择到设备环境之后，用户可以删除它。Windows 还提供了几个对剪取区域进行操作的函数，如 ExcludeClipRect 用于将一个矩形从剪取区域里排除掉，IntersectClipRect 用于创建一个新的剪取区域，它是前一个剪取区域与矩形的交，OffsetClipRgn 用于将剪取区域移动到客户区域的另一部分。

14.4.4 CLOVER 程序

CLOVER 程序用四个椭圆形成一个区域，将这个区域选进设备环境中，然后画出从窗口客户区域的中心出发的一系列直线。这些直线只在区域所定义的部分出现。结果显示如图 14.19 所示。

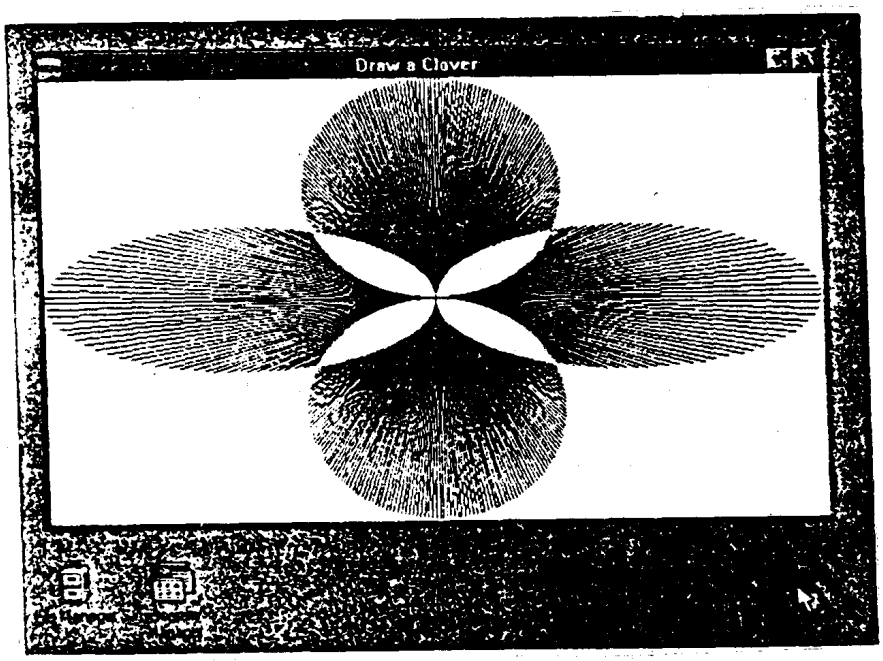


图 14.19 CLOVER 用复杂的剪取区域画出的图形

要用常规的方法画出这个图形，就必须根据椭圆的边线所涉及的公式计算出每条直线的端点。通过使用一个复杂的剪取区域，可以直接画出这些直线，而让 Windows 确定端点。CLOVER 程序如下所示。

```
# CLOVER.MAK make file
```

```
clover.exe : clover.obj clover.def
     $(WINLINK) clover,clover,NUL, $(WINLIB),clover
     rc -t clover.exe

clover.obj : clover.c
     $(WINCC) clover.c

/*------------------------------------------------
CLOVER.C -- Clover Drawing Program using Regions
------------------------------------------------*/

#include <windows.h>
#include <math.h>
#define TWO_PI (2.0 * 3.14159)

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "Clover" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = NULL ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = NULL ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
     }

     hwnd = CreateWindow (szAppName,"Draw a Clover",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,CW_USEDEFAULT,
          CW_USEDEFAULT,CW_USEDEFAULT,
          NULL,NULL,hInstance,NULL) ;

     ShowWindow (hwnd,nCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg,NULL,0,0))
```

```
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static HRGN hRgnClip ;
    static short cxClient,cyClient ;
    double fAngle,fRadius ;
    HCURSOR hCursor ;
    HDC hdc ;
    HRGN hRgnTemp [6] ;
    PAINTSTRUCT ps ;
    short i ;

    switch (message)
    {
        case WM_SIZE:
            cxClient = LOWORD (lParam) ;
            cyClient = HIWORD (lParam) ;

            hCursor = SetCursor (LoadCursor (NULL,IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            if (hRgnClip)
                DeleteObject (hRgnClip) ;

            hRgnTemp [0] = CreateEllipticRgn (0,cyClient / 3,
                cxClient / 2,2 * cyClient / 3) ;
            hRgnTemp [1] = CreateEllipticRgn (cxClient / 2,cyClient / 3,
                cxClient,2 * cyClient / 3) ;
            hRgnTemp [2] = CreateEllipticRgn (cxClient / 3,0,
                2 * cxClient / 3,cyClient / 2) ;
            hRgnTemp [3] = CreateEllipticRgn (cxClient / 3,cyClient / 2,
                2 * cxClient / 3,cyClient) ;
            hRgnTemp [4] = CreateRectRgn (0,0,1,1) ;
            hRgnTemp [5] = CreateRectRgn (0,0,1,1) ;
            hRgnClip = CreateRectRgn (0,0,1,1) ;

            CombineRgn (hRgnTemp [4],hRgnTemp [0],hRgnTemp [1],RGN_OR) ;
            CombineRgn (hRgnTemp [5],hRgnTemp [2],hRgnTemp [3],RGN_OR) ;
            CombineRgn (hRgnClip, hRgnTemp [4],hRgnTemp [5],RGN_XOR) ;

            for (i = 0 ; i < 6 ; i++)
                DeleteObject (hRgnTemp [i]) ;

            SetCursor (hCursor) ;
```

```
            ShowCursor (FALSE) ;
            return 0 ;
        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;

            SetViewportOrg (hdc,cxClient / 2,cyClient / 2) ;
            SelectClipRgn (hdc,hRgnClip) ;

            fRadius = hypot (cxClient / 2.0,cyClient / 2.0) ;

            for (fAngle = 0.0 ; fAngle < TWO_PI ; fAngle += TWO_PI / 360)
            {
                MoveTo (hdc,0,0) ;
                LineTo (hdc,(short) ( fRadius * cos (fAngle) + 0.5),
                    (short) (-fRadius * sin (fAngle) + 0.5)) ;
            }
            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_DESTROY:
            DeleteObject (hRgnClip) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; CLOVER.DEF module definition file

NAME CLOVER

DESCRIPTION 'Clover Drawing Using Regions (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

由于区域总是使用设备坐标，CLOVER 程序必须在每次接收到 WM_SIZE 消息时重新创建区域。这可能需要几秒钟。CLOVER 以创建四个椭圆区域开始，这四个椭圆存放在 hRgnTemp 数组的头四个元素中。然后程序创建三个“哑”区域：

```
hRgnTemp[4] = CreateRectRgn(0,0,1,1);
hRgnTemp[5] = CreateRectRgn(0,0,1,1);
hRgnClip = CreateRectRgn(0,0,1,1);
```

客户区域左右的两个椭圆区域被组合：

```
CombineRgn(hRgnTemp[4],hRgnTemp[0],hRgnTemp[1],RGN_OR);
```

同样，客户区域上下两个椭圆区域被组合：

```
CombineRgn(hRgnTemp[5],hRgnTemp[2],hRgnTemp[3],RGN_OR);
```

最后，两个组合后的区域再被组合到 hRgnClip 中：

```
CombineRgn(hRgnClip,hRgnTemp[2],hRgn[5],RGN_XOR);
```

RGN_XOR 标识符用于从结果区域中排除重叠部分。最后，删除 6 个临时区域：

```
for (i = 0;1 < 6;i++)
    DeleteObject(hRgnTemp[i];
```

考虑到结果，WM_PAINT 的处理很简单。视口原点被设置为客户区域的中心(使直线画起来容易些)，在 WM_SIZE 消息期间创建的区域被选择为设备环境的剪取区域：

```
SetViewportOrg(hdc,xClient / 2,yClient / 2);
SelectClipRgn(hdc,hRgnClip);
```

现在，剩下的就是画直线，共 360 条，每隔一度画一条。每条线的长度为变量 fRadius，这是从中心到客户区域的角落的距离：

```
fRadius = hypot (xClient / 2.0,yClient / 2.0);

for (fAngle = 0.0 ; fAngle < TWO_PI ; fAngle += TWO_PI / 360)
{
    MoveTo (hdc,0,0);
    LineTo (hdc,(short) (fRadius * cos (fAngle) + 0.5),
        (short) (-fRadius * sin (fAngle) + 0.5));
}
```

在处理 WM_DESTROY 时删除该区域：

```
DeleteObject(hRgnClip);
```

14.5 其他 GDI 函数

另外一些绘图函数不太好归入哪一类。它们包括 FloodFill、DrawIcon、ScrollWindow、ScrollDC、LineDDA 和 LineProc。

FloodFill 用当前刷子填充一个区域，其语法为：

```
FloodFill (hdc,xStart,yStart,rgbColor);
```

这是在 Windows PAINTBRUSH 程序中当使用着色滚筒时使用的一个函数。此函数从逻辑点(xStart,yStart)开始填充一个区域，直到遇到 rgbColor 的边界。

如果点(xStart,yStart)就是 rgbColor，或者如果该点在剪取区域的外面，FloodFill 就不会工作。如果在通常重画客户区域期间使用 FloodFill，程序员可能想在调用 BeginPaint 之前使整个客户区域无效，以确保(xStart,yStart)在剪取区域中。

ExtFloodFill 函数的语法如下：

```
ExtFloodFill(hdc,xStart,yStart,rgbColor,wFill);
```

ExtFlood 是 FloodFill 的扩展版本，它可以根据最后一个参数确定是填充到边界，还是在表面填充颜色 rgbColor。

DrawIcon 在显示器上画一个图标：

```
DrawIcon(hdc,xStart,yStart,hIcon);
```

这个函数只在 MM_TEXT 映像方式下使用。

ScrollWindow 和 ScrollDC 都用来滚动窗口区域的一部分。

它不是 GDI 函数，所以它总是使用设备单位。一般语法为：

```
ScrollWindow(hwnd,xScroll,yScroll,&rectScroll,&rectClip);
```

注意，第一个参数是窗口句柄，而不是设备环境句柄。rectScroll 矩形结构指示的客户区域段被向右移动 xScroll 个像素点，向下移动 yScroll 个像素点。如果用这个函数向左或向上移动，则参数可以为负。只有 rectClip 矩形内的区域受到滚动的影响，所以将 rectClip 设置为 rectScroll、一个较大的矩形或 NULL 都是有意义的。如果想要 rectScroll 内的区域被滚动，而不影响矩形外的一切，则将 rectClip 设置为 rectScroll。如果 rectScroll 内的区域移动到矩形外面是可接受的，则设置 rectClip 为 NULL。因滚动暴露出来的区域失效，该区域产生一条 WM_PAINT 消息。由于 ScrollWindow 不是 GDI 函数，它不能进行剪取，并且也将滚动对话框或子窗口，这可能覆盖客户区域的表面。

ScrollDC 是 GDI 函数，其原理不同。语法为：

```
ScrollDC(hdc,xScroll,yScroll,&rectScroll,&rectClip,
    hRgnUpdate,&rectUpdate);
```

如果最后两个函数参数不设置为 NULL，则由它们给程序返回信息。如果程序员在第六个参数中给 Windows 传送一个区域句柄，则 Windows 将区域设置为滚动所暴露的失效区域。如果程序员给最后一个参数传送一个指向 RECT 结构的指针，则 Windows 将结构设置为指示滚动所暴露出的失效矩形。

LineDDA 函数计算直线上的所有点，该直线连接两个给定点。DDA 代表“digital differential analyzer”(数字差分分析器)。其语法为：

```
LineDDA(xStart,yStart,xEnd,yEnd,lpfnLineProc,lData);
```

LineDDA 需要一个如下形式的回调函数：

```
void FAR PASCAL _export LineProc(int x,int y,(LONG)lData)
{
    [其他程序语句]
}
```

传递给 LineDDA 的 lpfnLineProc 参数必须是一个 MakeProcInstance 调用的结果：

```
lpfnLineProc = MakeProcInstance(LineProc,hInstance);
```

在调用 LineDDA 时，Windows 对连接点(xStart,yStart)和(xEnd,yEnd)的直线上的每个点调用 LineProc。点由 LineProc 的 x 和 y 参数指示。LineProc 也接收指向程序员所定义的数据的指针，该指针由程序员在 LineDDA 调用中给出。

注意，LineDDA 不要求设备环境的句柄。此函数只是一个直线计算算法，将它所计算的每一点告知程序。对这些点怎样处理则取决于程序员。在 LINEDDA 程序中，我们画了一个矩形，并将矩形的顶点与客户区域的顶点连接。由于对 Windows 允许程序员很轻松地画出的普通点式线和长划线式风格不满意，我们将线画为一系列小椭圆。结果如下所示。

```
# LINEDDA.MAK make file

linedda.exe : linedda.obj linedda.def
    $(WINLINK) linedda,linedda,NUL, $(WINLIB),linedda
    rc -t linedda.exe

linedda.obj : linedda.c
    $(WINCC) linedda.c
```

```
/*---------------------------------------
LINEDDA.C -- LineDDA Demonstration
---------------------------------------*/

#include <windows.h>

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;
void FAR PASCAL _export LineProc (int,int,LONG) ;

HANDLE hInst ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
        LPSTR lpszCmdLine,int nCmdShow)
{
    static char szAppName[] = "LineDDA" ;
    HWND hwnd ;
    MSG msg ;
    WNDCLASS wndclass ;

    if (! hPrevInstance)
    {
        wndclass.style = CS_HREDRAW | CS_VREDRAW ;
        wndclass.lpfnWndProc = WndProc ;
        wndclass.cbClsExtra = 0 ;
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = NULL ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;

        RegisterClass (&wndclass) ;
    }

    hInst = hInstance ;

    hwnd = CreateWindow (szAppName,"LineDDA Demonstration",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
```

```
    }
    return msg. wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static FARPROC lpfnLineProc ;
    static short cxClient,cyClient,xL,xR,yT,yB ;
    HDC hdc ;
    PAINTSTRUCT ps ;

    switch (message)
    {
        case WM_CREATE:
            lpfnLineProc = MakeProcInstance ((FARPROC) LineProc,hInst) ;
            return 0 ;
        case WM_SIZE:
            xR = 3 * (xL = (cxClient = LOWORD (lParam)) / 4) ;
            yB = 3 * (yT = (cyClient = HIWORD (lParam)) / 4) ;
            return 0 ;
        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;

            LineDDA (xL,yT,xR,yT,lpfnLineProc,(LONG) hdc) ;
            LineDDA (xR,yT,xR,yB,lpfnLineProc,(LONG) hdc) ;
            LineDDA (xR,yB,xL,yB,lpfnLineProc,(LONG) hdc) ;
            LineDDA (xL,yB,xL,yT,lpfnLineProc,(LONG) hdc) ;

            LineDDA (0, 0, xL,yT,lpfnLineProc,(LONG) hdc) ;
            LineDDA (cxClient,0, xR,yT,lpfnLineProc,(LONG) hdc) ;
            LineDDA (cxClient,cyClient,xR,yB,lpfnLineProc,(LONG) hdc) ;
            LineDDA (0, cyClient,xL,yB,lpfnLineProc,(LONG) hdc) ;

            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

void FAR PASCAL _export LineProc (int x,int y,LONG lData)
{
    static short nCounter = 0 ;

    if (nCounter == 2)
        Ellipse ((HDC) lData,x - 2,y - 2,x + 3,y + 3) ;

    nCounter = (nCounter + 1) % 4 ;
```

```
}

; LINEDDA. DEF module definition file

NAME LINEDDA

DESCRIPTION 'LineDDA Demonstration (c) Charles Petzold, 1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

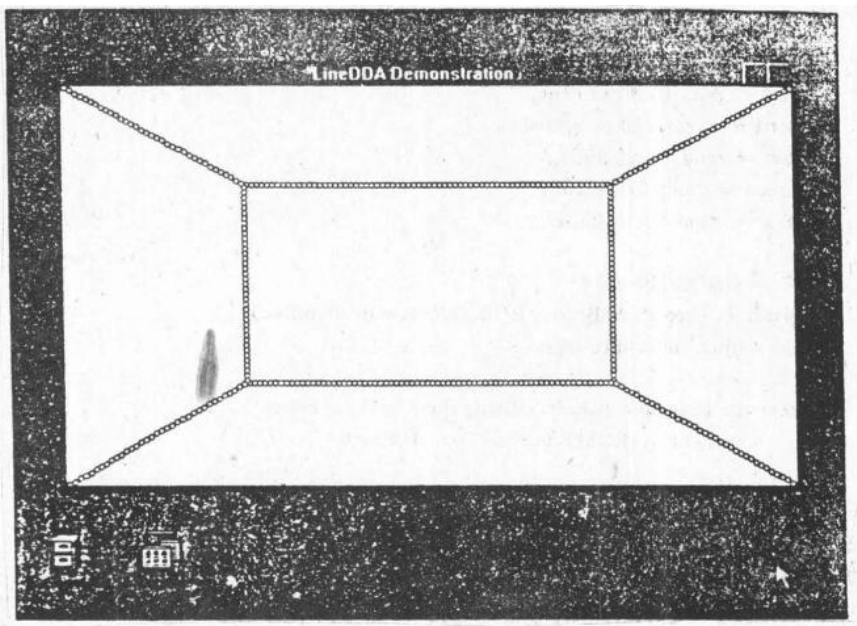


图 14.20　LINEDDA 程序的显示结果

在处理 WM_PAINT 消息期间，LineDDA 函数调用了八次，对每条线各调用一次。lpData 参数是设备环境句柄的地址。

LineProc 函数很短：

```
void FAR PASCAL _export LineProc (int x, int y, (LONG) lData)
{
    statoc short nCounter = 0;

    if (nCounter == 2)
        Ellipse ((HDC) lData.x - 2, y - 2, x + 3, y + 3);

    nCounter = (nCounter + 1) % 4;
}
```

注意，nCounter 变量定义为静态的，以便它的值能在 LineProc 调用之前被保存。它在值

0,1,2,3之间循环。LineProc以该点为中心画一个椭圆。

14.6 画图程序示例

在所有图形系统中,有一个有趣的程序,它"不停"地运行,用随机的大小和颜色画出没有意义的一系列矩形。程序员可以在Windows中创建一个这样的程序,但它并不像乍看起来那样容易编写。到现在为止,程序员应该知道,绝对不能用下面的程序来完成这一工作:

```
case WM_PAINT;                                    // Very,very bad code !!!

    hdc = BeginPaint (hwnd,&ps);

    while (TRUE)
    {
        xLeft = rand () % xClient;
        xRight = rand () % xClient;
        yTop = rand () % yClient;
        yBottom = rand () % yClient;
        nRed = rand () % 255;
        nGreen = rand () % 255;
        nBlue = rand () % 255;

        hdc = GetDC (hwnd);
        hBrush = CreateSolidBrush (RGB (nRed,nGreen,nBlue));
        SelectObject (hdc,hBrush);

        Rectangle (hdc,min (xLeft,xRight),min (yTop,yBottom),
            max (xLeft,xRight),max (yTop,yBottom));
    }
    EndPaint (hwnd, * ps);
    return 0;
```

的确,它能够运行,但什么样的程序都能运行。由于Windows是一个无优先权的多任务环境,程序将无法进入这样的无限循环。这种循环将停止系统中所有其他进程。

一种可以接受的方法是设置一个Windows定时器,给窗口函数发送WM_TIMER消息。对于每条WM_TIMER消息,程序员用GetDC获取一个设备环境,画一个随机的矩形,然后用ReleaseDC释放设备环境。但这样又减轻了程序的趣味性,因为程序不能尽可能快地画随机矩形。它必须等待WM_TIMER消息。

在Windows中必须有足够的"确定时间",在这个时间内,所有消息队列为空,Windows只停在一个小循环中等待键盘或鼠标输入。我们能否想出一个办法,在这个确定时间内,在将消息加入程序的消息队列之后才释放控制呢?这是PeekMessage函数的目的之一。

下面是PeekMessage调用的一个例子:

```
PeekMessage(&msg,NULL,0,0,PM_REMOVE);
```

前面四个参数(一个指向MSG结构的指针、一个窗口句柄、两个值指示消息范围)与GetMessage的参数相同。将第二、三、四个参数设置为NULL或0时,表明我们想让PeekMessage返回程序中所有窗口的所有消息。与GetMessage相似,如果消息等待在其他程序的消息队列中,PeekMessage将控制转交给其他程序。与GetMessage相似的还有,

PeekMessage 只为进行这个函数调用的程序所创建的窗口返回消息。

如果要从消息队列中删除消息，则将 PeekMessage 的最后一个参数设置为 PM_REMOVE。如果不删除消息，则将该参数设置为 PM_NOREMOVE。这就是 PeekMessage 仅“偷看”而不“获取”的原因，它使得程序可以检查程序队列的下一条消息，而不实际删除之。GetMessage 不将控制返回给程序，直到从程序的消息队列中获取消息。但 PeekMessage 能在两种条件下返回控制：

- 在程序的消息队列里有消息时，PeekMessage 的返回值为 TRUE(非 0)。
- 当所有运行于 Windows 下的程序的消息队列中都没有消息时，PeekMessage 的返回值为 FALSE(0)。

使用 PeekMessage 而不是 GetMessage 的消息循环主要是能告诉 Windows，“让其他程序运行一会儿，但一旦它们的消息队列为空后，就将控制返回给我——我的工作还没有完成呢”。如果两个或者更多的程序正在运行，且都用 PeekMessage 循环获取消息，则 Windows 用类似循环赛的方法相继将控制返回给用 PeekMessage 等待的程序。

这使得我们可以替换普通的消息循环，也就是如下所示的循环：

```
while (GetMessage (&msg,NULL,0,0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
return msg. wParam ;
```

而用如下所示的循环：

```
while (TRUE)
{
    if (PeekMessage (&msg,NULL,0,0,PM_REMOVE))
    {
        if (msg,message == WM_QUIT)
        break;

        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    else
    {
        [other program lines to do some work]
    }
}
return msg. wParam ;
```

注意，WM_QUIT 消息是显式检查的。在普通的消息循环中不用这样做，因为如果 GetMessage 接收到一条 WM_QUIT 消息，它将返回 0。但是 PeekMessage 用它的返回值来指示是否得到一条消息，所以需要对 WM_QUIT 进行检查。

如果 PeekMessage 的返回值为 TRUE，则消息如通常那样处理。如果返回值为 FALSE，则程序在将控制返回给 Windows 之前，还可以做一点工作(如显示另一个随机矩形)。

尽管 Windows 文档上说，程序员不能用 PeekMessage 从消息队列中删除 WM_PAINT 消息，但这并不是一个真正的问题。毕竟，GetMessage 也不能从消息队列中删除 WM_PAINT 消

息。从队列中删除 WM_PAINT 消息的唯一方法是令窗口客户区域的失效区域变得有效，这可以用 ValidateRect 和 ValidateRgn 或者 BeginPaint 和 EndPaint 对来完成。如果在用 PeekMessage 从队列中取出 WM_PAINT 消息后，同平常一样处理它，就不会有问题。所不能做的是使用如下所示的代码来清除消息队列中的所有消息：

```
while (PeekMessage (&msg,NULL,0,0,PM_REMOVE));
```

这条语句从消息队列中移出并丢弃所有消息，除了 WM_PAINT 之外。如果队列中有一条 WM_PAINT 消息，程序就会永远处在 while 循环里。

Windows 的 TERMINAL 程序在从通信线接收到输入后，使用一个 PeekMessage 循环。这使 TERMINAL 可以"不间断地"检查输入数据。PRINT MANAGER 程序用这种技术在打印机或绘图仪上打印；并且，一个 Windows 打印程序也包含有一个带 PeekMessage 循环的函数。有了 PeekMessage 函数，我们现在就可以编写一个不停地显示随机矩形的程序了。这个程序(RANDRECT)如下所示。

```
# RANDRECT.MAK make file

randrect.exe : randrect.obj randrect.def
     $(WINLINK) randrect,randrect,NUL, $(WINLIB),randrect
     rc -t randrect.exe

randrect.obj : randrect.c
     $(WINCC) randrect.c

/*----------------------------------------
RANDRECT.C -- Displays Random Rectangles
----------------------------------------*/

#include <windows.h>
#include <stdlib.h>

#define min(a,b) (((a) < (b)) ? (a) : (b))
#define max(a,b) (((a) > (b)) ? (a) : (b))

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;
void DrawRectangle (HWND) ;

short cxClient,cyClient ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName[] = "RandRect" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
```

```
        wndclass.cbClsExtra = 0 ;
        wndclass.cbWndExtra = 0 ;
        wndclass.hInstance = hInstance ;
        wndclass.hIcon = NULL ;
        wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wndclass.lpszMenuName = NULL ;
        wndclass.lpszClassName = szAppName ;

        ! RegisterClass (&wndclass) ;
    }

    hwnd = CreateWindow (szAppName,"Random Rectangles",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (TRUE)
    {
        if (PeekMessage (&msg,NULL,0,0,PM_REMOVE))
        {
            if (msg.message == = WM_QUIT)
                break ;

            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
        }
        else
            DrawRectangle (hwnd) ;
    }
    return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    switch (message)
    {
        case WM_SIZE:
            cxClient = LOWORD (lParam) ;
            cyClient = HIWORD (lParam) ;
            return 0 ;
        case WM_DESTROY:
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
```

```
}

void DrawRectangle (HWND hwnd)
{
     HBRUSH hBrush ;
     HDC hdc ;
     short xLeft,xRight,yTop,yBottom,nRed,nGreen,nBlue ;

     xLeft = rand () % cxClient ;
     xRight = rand () % cxClient ;
     yTop = rand () % cyClient ;
     yBottom = rand () % cyClient ;
     nRed = rand () & 255 ;
     nGreen = rand () & 255 ;
     nBlue = rand () & 255 ;

     hdc = GetDC (hwnd) ;
     hBrush = CreateSolidBrush (RGB (nRed,nGreen,nBlue)) ;
     SelectObject (hdc,hBrush) ;

     Rectangle (hdc,min (xLeft,xRight),min (yTop,yBottom),
          max (xLeft,xRight),max (yTop,yBottom)) ;

     ReleaseDC (hwnd,hdc) ;
     DeleteObject (hBrush) ;
}

; RANDRECT. DEF module definition file

NAME RANDRECT

DESCRIPTION 'Random Rectangle Drawing Program (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

第十五章　位　图

位图是有多种用途的图像。有了它便可在各种应用程序中画出图案。

比如在应用程序 Paintbrushes 中，就需要用到位图（参看前面讲过的关于输出的一章）。

15.1　位图的类型

Windows 3.0 版和更高版本有两种位图：

- 依赖于设备的位图
- 与设备无关的位图（DIB）

1. 依赖于设备的位图

依赖于设备的位图由内存中的位模式构成，由专门的函数将这些位模式传给输出设备。这种位图在内存中的位与设备上的像素之间有专门的对应关系。

2. 与设备无关的位图（DIB）

与设备无关的位图不具备与设备相关的格式。函数 DIB 将这种与设备无关的图像数据转化为当前输出设备可理解的格式。

15.2　依赖于设备的位图

依赖于设备的位图可看作屏幕内存的一个直接拷贝。图形卡不同，位图的结构也不同，这就是术语“依赖于设备”的含义。

15.2.1　建立一个位图

依赖于设备的位图可以用下面几种方法建立：

(1)由图形资源建立

(2)由空位图建立

(3)由位组建立

由图形资源建立的位图

位图可以用基于 Windows 的图形工具，比如 Paintbrush 或 IMAGEDIT（SDK 图像编辑器）来建立。IMAGEDIT. EXE 产生 256×256 的位图。产生的图像必须保存到位图文件中（即

扩展名为 .BMP 的文件)。

文件名由 BITBMAP 资源描述语句传递给资源描述文件(进一步可参看"资源"一章)。

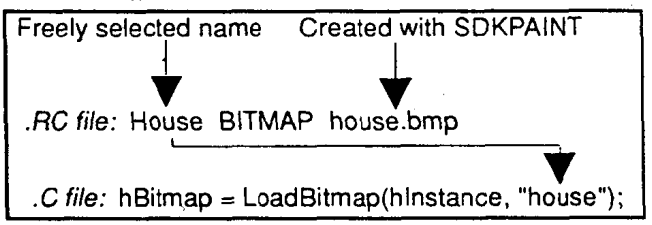


图 15.1 文件名

函数 LoadBitmap 根据文件名把位图调入内存,并返回输出所需的句柄。

创建位图也可以不需要资源,而使用 GDI 函数或一个数组就可以建立。下面四个函数可用来建立空位图,然后用刚才的两种方法进行填充。

```
HBITMAP CreateBitmap( nWidth,nHeight,nPlanes,nBitCount,lpBits )
HBITMAP CreateBitmapIndirect( lpBitmap )
HBITMAP CreateCompatibleBitmap( hDC,nWidth,nHeight )
HBITMAP CreateDiscardableBitmap( hDC,nWidth,nHeight )
```

前两个函数的参数有很大区别。函数 HBITMAP CreateBitmap 直接传递所需参数,而函数 HBITMAP CreateBitmapIndirect 传递一个 BITMAP 结构的指针。

首先讨论函数 HBITMAP CreateBitmap。位图的颜色可以分为两类:第一类使用颜色平面(每个颜色平面代表一种颜色)。每个具体像素在平面内被定义为一个位,所有平面被一个接一个地放在内存中。

如果输出设备支持八种颜色,则需要三个颜色平面(红、绿、蓝颜色平面)。黄色像素实际上包含绿色平面上的一位和蓝色平面上的一位。使用这种方法,必须把参数 nPlanes(颜色平面的个数)赋值为 3,而把参数 nBits(每个像素有几位)赋值为 1。

第二种方法仅需要一个颜色平面,在唯一的颜色平面内由多个位控制颜色。比如,显示八种颜色时,每个像素需要三位。在这种方式下,参数 nPlanes 赋值为 1,而参数 nBits 赋值为 3。使用哪种方式决定于所选的输出设备。

让我们先看一下八种颜色的 4×4 位图。

位图的图像显示

第一行:洋红
第二行:黄
第三行:绿
第四行:红

位图编码

每像素一位		每像素三位
三个颜色平面	或	一个颜色平面
1 1 1 1		101 101 101 101

```
1 1 1 1                    110 110 110 110
0 0 0 0                    001 001 001 001
1 1 1 1                    100 100 100 100
0 0 0 0
1 1 1 1
0 0 0 0
0 0 0 0

1 1 1 1
0 0 0 0
1 1 1 1
0 0 0 0
```

函数 CreateCompatibleBitmap 和 CreateDiscardableBitmap 不包含参数 nPlanes 和 nBitCount。Windows 把设备环境中指定为句柄的选项传给第一个参数。

如果没有选择设备环境，函数 CreateDiscardableBitmap 所建立的位图可以从内存中删除。函数 SelectObject 试图在滞后一些时间后选择此位图，该函数的返回值为 0。

用 GDI 函数产生位图

空位图可以用 GDI 输出函数填入矩形等图形。这种位图必须放在内存设备环境中(参看函数 PatBlt 的例子)。

用位组产生位图

如果用函数 CreateBitmap 建立新位图，则需向最后一个参数传递一个包含预定义位模式的数组。此数组可用来初始化位图。否则，也可以用函数 SetBitmapBits 填充位图。

15.2.2 输出函数

三个函数可用来输出与设备有关的位图：

- PatBlt
- BitBlt
- StretchBlt

术语 Blt(读作“blit”)是 block transfer 的缩写。函数 GetDeviceCaps 和参数 RASTERCAPS 可判断输出设备是否支持这三个位图函数。

```
if (GetDeviceCaps (hDC,RASTERCAPS) & RC_BITBLT)
    TRUE => 设备可显示位图
```

如果返回值与 RC_BITBLT 匹配，此设备就支持与设备无关的位图。第二个和第三个函数需要一个附加的设备环境(通常是内存设备环境)。位图按左上角对齐格式写到内存设备环境中。

最后，位图还可以从源(内存设备环境)拷贝到目标设备环境(即显示环境)，从而显示在输出设备上。输出结束后，内存设备环境释放位图并恢复到初始状态。

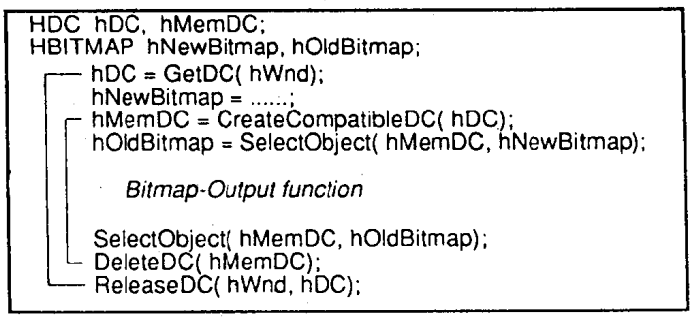


图 15.2

函数的结果并不一定有用。比如，没有设备环境时也可以建立位图。函数 PatBlt 把一个位模式传给使用当前设备环境句柄的设备。其他函数仅需要一个设备环境。

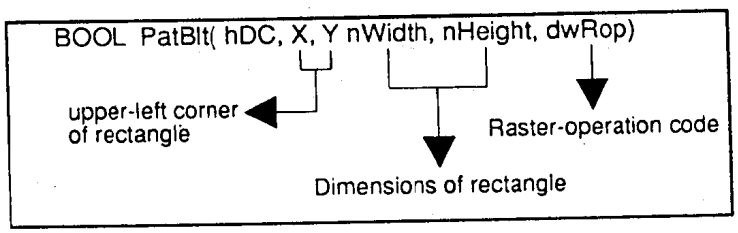


图 15.3 函数 PatBlt

第二个至第四个参数指示写到输出设备的范围。这些参数使用逻辑单位。在此矩形块中的绘图风格是当前刷子(存于设备环境中)和模式(已为输出设备准备好)的组合。最后一个参数(光栅操作码)指示这些位怎样相互作用。

例如，若参数为 WHITENESS，则整个方块被涂成白色。这种情况下，当前刷子和可用模式都是无关紧要的。在使用上述四种函数时，这个参数可产生一个空白位图，这个空白位图可用来删除一个已定义的新位图的内容。

```
HDC hDC,hMemDC;
HBITMAP hNewBitmap,hOldBitmap;

hNewBitap = CreateBitmap(16,16,1,1,NULL,);

hDC = GetDC (hWnd );
hMemDC = CreateCompatibleDC( hDC );
hOldBitmap = SelectObject( hMemDC,hNewBitmap );

PatBlt( hMemDC,0,0,16,16,WHITENESS );

Selectobject( hMemDC,hOldBitmap );
DeleteDC( hMemDC );
Release( hWnd,hDC );
```

这个函数的结果很像函数 BitBlt 和 StretchBlt 的结果。用当前刷子填充特定方块的光栅操作码 PATCOPY 也可产生同样的效果。

```
hNewBrush = GetStockObject( WHITE_BRUSH );
```

```
hOldBrush = SelectObject ( hMemDC,hNewBrush );

PatBlt( hMemDC( 0,0,16,16,PATCOPY );

SelectObject( hMemDC,hOldBrush );
```

函数 BitBlt 把位图从源设备环境(用句柄 hSrcDC 和内存设备环境标识)传输到目标设备环境。因为位图的大小未变,所以它只能被使用一次。

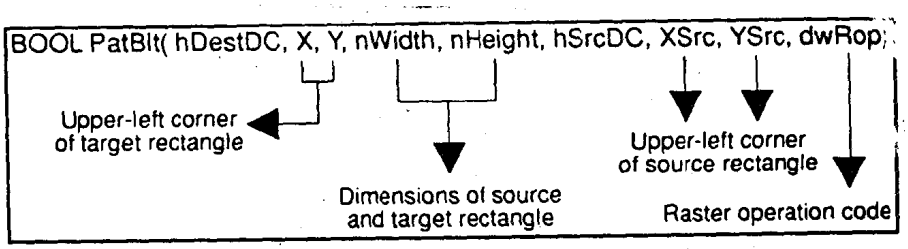
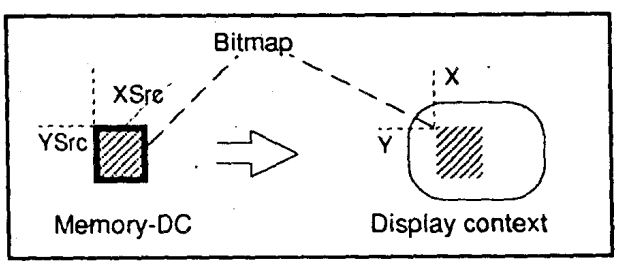


图 15.4 函数 BitBlt

目标块和源块的左上角坐标一致。最后一个参数向函数 PatBlt 和其他函数提供光栅操作码。



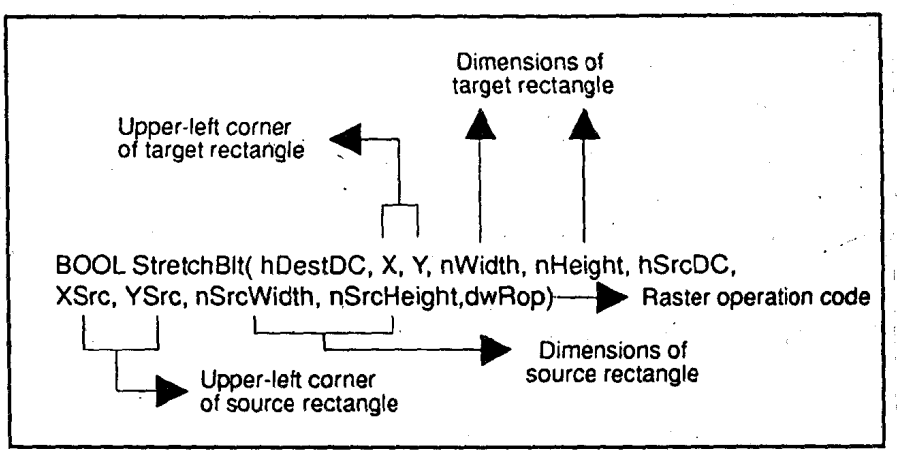


图 15.5 函数 StretchBlt

仅当需改变位图在目标设备环境中的大小时才需要用到函数 StretchBlt(参看上图)。函数 StretchBlt 的两个参数指示目标块的高度和宽度。

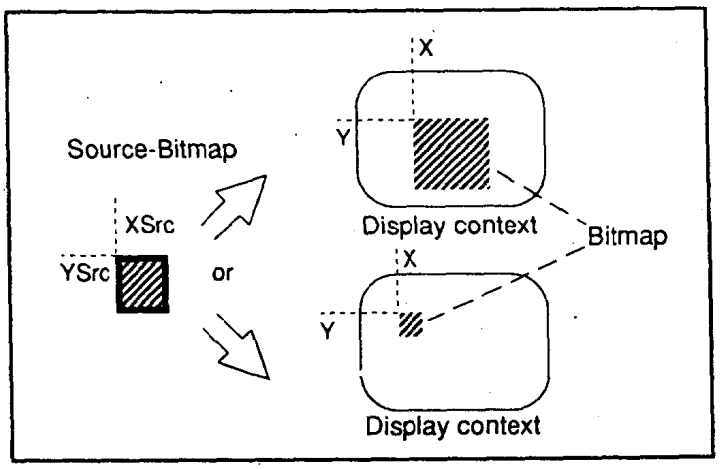


图 15.6　在目标设备环境中改变位图的大小

此函数也可以对位图进行镜像变换，只要把高度和宽度描述为负数即可。

展开模式

压缩位图意味着把几行或几列合并为一行或一列。这样做损失了位图的一些信息。但是，展开模式增加了像素的组合。函数 SetStretchBltMode 可激活所需的模式。

三个可用的选项是：

1. BLACKONWHITE

BLACKONWHITE 模式是缺省模式，用逻辑运算 AND 增加所需的像素组。这意味着仅当原来的像素都是白色时结果像素才是白色。

2. WHITEONBLACK

如果选用了 WHITEONBLACK 模式，则使用逻辑运算 OR 来增加所需的像素组。因此在这种模式下白色像素较多。

3. COLORONCOLOR

仅当所有的像素都是黑色时，显现的像素才是黑色的。如果想压缩一个彩色位图，则需使用 COLORONCOLOR 模式。这时不应该用函数 StretchBlt 删除掉组合像素，而只需要选择性地删除几行或几列。否则，可能会遇到不可预测的颜色问题。

三种位图输出函数的光栅操作码详细描述了 GDI 怎样将源方块和目标方块的像素与目标设备环境的当前刷子组合起来。组合的结果贮存于目标方块之中。

三种大小的逻辑组合提供了 256 种可能的组合。50 种最实用的组合以其名字记录下来，此名字的值可以为 SRCINVERT，PATCOPY，BLACKNESS 和 SRCCOPY 等；它在文件 WINDOWS. H 中被定义。

最常用的值是 SRCCOPY。这个光栅操作码仅仅用于显示源位图。

15.2.3　进一步了解位图

函数 SetPixel 可在屏幕的指定位置，用指定的颜色设置一个像素。但是 Windows 的开发

者很少使用这个函数。他们用 GetPixel 函数来控制像素的颜色。

位图

函数 GetObject 可提供位图的大小，以便填充类型为 BITMAP 的逻辑位图结构。

bmType	逻辑位图：0
bmWidth	位图宽度(单位：像素)
bmHeight	位图高度(单位：像素)
bmWidthBytes	一个光栅行的字节数
bmPlanes	色面数
bmBitPixel	颜色位数/像素
bmBits	指向字节数组

上面的值定义了位图的高度、宽度、颜色平面的个数和每个像素点占几位。但是，此函数并不填充最后一个数组。必须调用函数 GetBitmapBit 来找到位模式。

```
BITMAP bm;
BYTE Bits[200];
HPITMAP hBitmap;

hBitmap = LoadBitmap( hInstance,"Koala" );
GetObject( hBinmap,sizeof( BITMAP ),(LPSTR) &bm );
GetBitmapBits( hBitmap,bm. bmHeight * bm. bmWidthBytes,Bits );
```

这个例子产生了只有一个颜色平面的位图。如果想增加颜色平面，必须把第二个参数乘以颜色平面的个数。

单色位图也可显示为彩色图像，只要指定好前景颜色和背景颜色。前景颜色代表置位(white)位的颜色，背景色即非置位(black)位的颜色。

```
SetTextColor( hDC,RGB( 0,0,255 ) );
SetBkColor( hDC,RGB(255,0,0 ) )
```

图 15.7 显示了一个单色位图的例子。

在上图的缺省颜色下，如果用 BitBlt 或 StretchBlt 建立一个单色位图，则为“蓝框红底”。

在位图使用完毕(或应用程序结束)时应删除位图，并释放位图所占的内存。同时，任何一个设备环境都不应再包含此位图。函数 DeleteObject 完成这些任务。

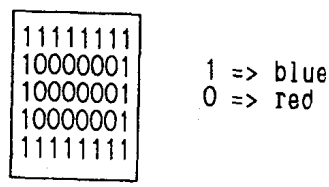


图 15.7

15.2.4 与设备有关的位图的例子

本例使用两个位图，图像在客户区中作正弦曲线运动。有一个位图看起来像一只小狗，它是从 .RC 文件中调出的。

另一个位图由 Pie GDI 函数产生，它的尺寸不断地增大。消息 WM_TIMER 控制着这个

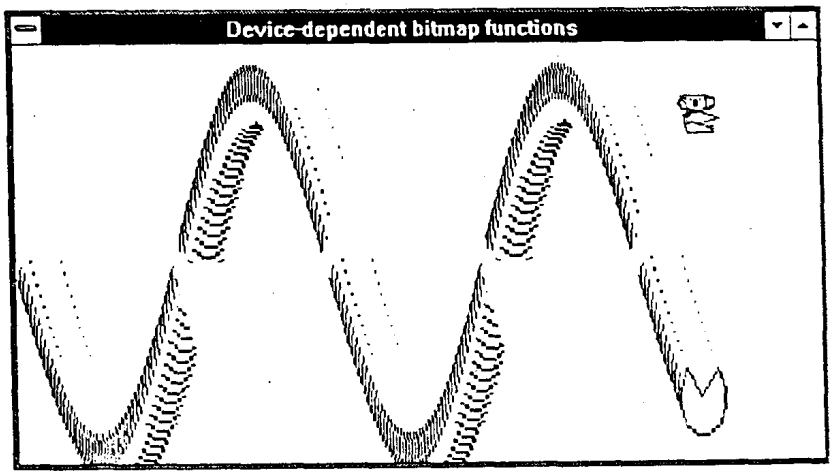


图 15.8

连续的运动。

新消息	描述
WM_SIZE	发送大小变化消息给窗口

新函数	描述
BitBlt	把位图从源设备环境拷贝到目标设备环境
CreateBitmap	建立一个位图
CreateCompatibleDE	建立一个内存设备环境
DeleteDC	删除一个内存设备环境
GetObject	在缓冲区中拷贝逻辑对象数据
PatBlt	建立一个位模式
StretchBlt	从源设备环境中把位图拷贝到目标设备环境中，以备最后修改其大小

新结构	描述
BITMAP	位图数据结构

源代码：BIT1.C

```
/* * BIT1.C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates device dependent bitmap functions * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                          // Include windows.h header file
#include "math.h"                             // Include math.h header file

#if _MSC_VER >= 700                           // Running Microsoft C 7.00 or
    #pragma warning (disable:4028)            // higher? Disable warning 4028
```

```
#endif

BOOL Bit1Init ( HANDLE );
long FAR PASCAL Bit1WndProc( HWND,unsigned,WORD,LONG);

/* * WinMain * * * * * * * * * * * * * * * * * * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                    // Current & previous instance
LPSTR lpszCmdLine;                                 // Ptr to string after prg name
int cmdShow;                                       // App. window's appearance
{
    MSG msg;                                       // Message handle
    HWND hWnd;                                     // Window handle

    if (! hPrevInstance)                           // Initialize first instance
    {
        if (! Bit1Init( hInstance ))
            return FALSE;
    }

    hWnd = CreateWindow("Bit1","Device-dependent bitmap functions",
        WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,0,CW_USEDEFAULT,0,
        NULL,NULL,hInstance,NULL);

    while (! SetTimer(hWnd,1,100,NULL))
        if (IDCANCEL == MessageBox(hWnd,"Too many timers running",
            "Timer",MB_RETRYCANCEL))
        return FALSE;

    ShowWindow( hWnd,cmdShow );
    UpdateWindow( hWnd );

    while (GetMessage(&msg,NULL,0,0))              // Message reading
    {
        TranslateMessage(&msg);                    // Message translation
        DispatchMessage(&msg);                     // Send message to Windows
    }

    return (int)msg. wParam;                       // Return wParam of last message
}

/* * Specify window class * * * * * * * * * * * * * * * * * * */

BOOL Bit1Init( hInstance )
HANDLE hInstance;
{
    WNDCLASS wcBit1Class;

    wcBit1Class. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcBit1Class. hIcon = LoadIcon( NULL,IDI_APPLICATION);
```

```
    wcBit1Class.lpszMenuName = NULL;
    wcBit1Class.lpszClassName = "Bit1";
    wcBit1Class.hbrBackground = GetStockObject( WHITE_BRUSH );
    wcBit1Class.hInstance = hInstance;
    wcBit1Class.style = CS_HREDRAW | CS_VREDRAW;
    wcBit1Class.lpfnWndProc = Bit1WndProc;
    wcBit1Class.cbClsExtra = 0 ;
    wcBit1Class.cbWndExtra = 0 ;

    if (! RegisterClass( &wcBit1Class ) )
        return FALSE;

    return TRUE;
}

/* * Bit1WndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL Bit1WndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    HANDLE hInst;
    HDC hDC,hMem;
    static HBITMAP hBit1,hBit2,hOld;
    static BITMAP bm;
    static int xPos,y1Pos,y2Pos,i;
    static int xClient,yClient;
    static double y,x;

    switch (message)
    {
        case WM_CREATE:                                    // Create window
            hInst = GetWindowWord( hWnd,GWW_HINSTANCE);
            hBit1 = LoadBitmap(hInst,"koala");
            GetObject(hBit1,sizeof(BITMAP),(LPSTR)&bm);

            hBit2 = CreateBitmap(20,20,1,1,NULL);
            hDC = GetDC(hWnd);
            hMem = CreateCompatibleDC(hDC);
            hOld = SelectObject(hMem,hBit2);
            PatBlt( hMem,0,0,20,20,WHITENESS);
            Pie(hMem,0,0,20,20,2,0,18,0);
            SelectObject(hMem,hOld);
            DeleteDC(hMem);
            ReleaseDC(hWnd,hDC);
            break;
        case WM_SIZE:
```

```
        xClient = LOWORD(lParam);
        yClient = HIWORD(lParam);
        yClient = yClient - bm.bmHeight;
        y1Pos = yClient/2;
        y2Pos = yClient/2;
        x = xPos = 0;
        y = 0.0;
        i = 0;
        break;
    case WM_TIMER:                                          // Timer running?
        hDC = GetDC(hWnd);                          // PatBlt draws BMP in background color
        hMem = CreateCompatibleDC(hDC);
        PatBlt(hDC,xPos,y1Pos,bm.bmWidth,
            bm.bmHeight,WHITENESS);
        x = i * 3.14159/50;
        xPos = i * (xClient/200);
        i++;
        if (xPos > xClient)
            i = 0;
        y = sin(x);

    /** Draw koala bitmap **********************/

        hOld = SelectObject(hMem,hBit1);
        y1Pos = (int)(-y * ((double)yClient)
            /2 + ((double)yClient)/2);
        BitBlt(hDC,xPos,y1Pos,bm.bmWidth,bm.bmHeight,
            hMem,0,0,SRCCOPY);

    /** Enlarge pie bitmap **********************/

        SelectObject(hMem,hBit2);
        y2Pos = (int)(y * ((double)yClient)
            /2 + ((double)yClient)/2);
        StretchBlt(hDC,xPos,y2Pos,30,50,
            hMem,0,0,20,20,SRCCOPY);

        SelectObject(hMem,hOld);
        DeleteDC(hMem);
        ReleaseDC(hWnd,hDC);

        break;
    case WM_DESTROY:                                        // Destroy window
        KillTimer( hWnd,1);
        DeleteObject(hBit1);
        DeleteObject(hBit2);
        PostQuitMessage(0);
        break;
    default:
        return(DefWindowProc(hWnd,message,wParam,lParam));
        break;
```

```
    }
    return(0L);
}
```

模块定义文件:BIT1.DEF

```
NAME Bit1

DESCRIPTION 'Device-dependent bitmap functions'

EXETYPE WINDOWS

STUB 'WINSTUB.EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS Bit1WndProc
```

资源描述文件:BIT1.RC

```
koala BITMAP koala.bmp
```

MAKE 文件:BIT1.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = koala.bmp

all: BIT1.EXE

BIT1.RES: BIT1.RC $(RESFILES)

BIT1.OBJ: BIT1.C

BIT1.EXE: BIT1.OBJ BIT1.RES BIT1.DEF
    link /NOD/A:16 BIT1,BIT1,nul,libw slibcew,BIT1.def
    rc BIT1.res BIT1.exe
```

像下面这样运行 NMAKE 便可产生应用程序 BIT1:

```
nmake bit1.mak
```

BIT1.EXE 是怎样工作的

WinMain 例程中的函数 SetTimer 启动计时器,如果 16 个系统计时器都在使用中,则不

能激活另外的计时器。应用程序显示一个对话框来指示这种情况。

对话框中包含有重试操作或取消操作的按钮。按下 Retry 按钮时再一次启动计时器。

消息 WM_CREATE 提供两个位图句柄。位图 KOALA 是用 IMAGEDIT. EXE 建立的，保存在文件 KOALA. BMP 中。资源描述将这个文件定义为资源。

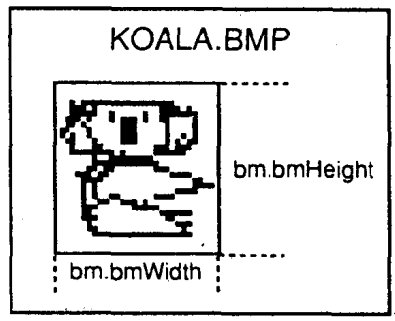


图 15. 9

KOALA 是函数 LoadBitmap 的第二个参数所定义的资源名字。函数 GetObject 将位图的大小和其他值赋给 BITMAP 结构变量 bm。

第二个位图是用函数 CreateBitmap 建立的。这个位图的高度和宽度各为 20 个像素。必须选择一个内存设备环境来访问这个位图。

函数 CreateCompatibleDC 提供内存设备环境句柄。函数 DeleteDC 释放掉这个句柄。

位图中包含有一些随机值。这可防止函数 CreateBitMap 初始化数组。因此，在执行函数 GDI Pie 之前整个图形是一片白色(函数 PatBlt)。

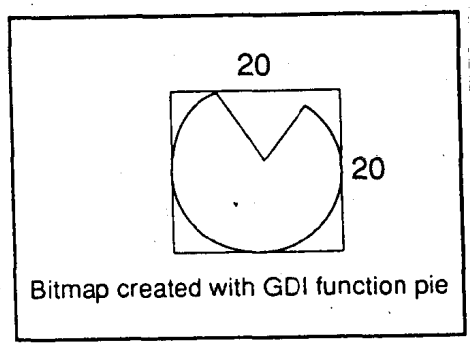


图 15. 10

消息 WM_SIZE 指示窗口的大小改变了。参数 lParam 的低位字为新的宽度，高位字为新的高度。

KOALA 位图把窗口高度作为自身的高度。

这是绘制两个位图时计算 y 值的基础。每隔 8. 1s 通过消息 WM_TIMER 输出一次。

位图 KOALA 在新位置上重画之前，必须从客户区删掉。函数 PatBlt 通过以背景色重画旧位图实现这一功能。

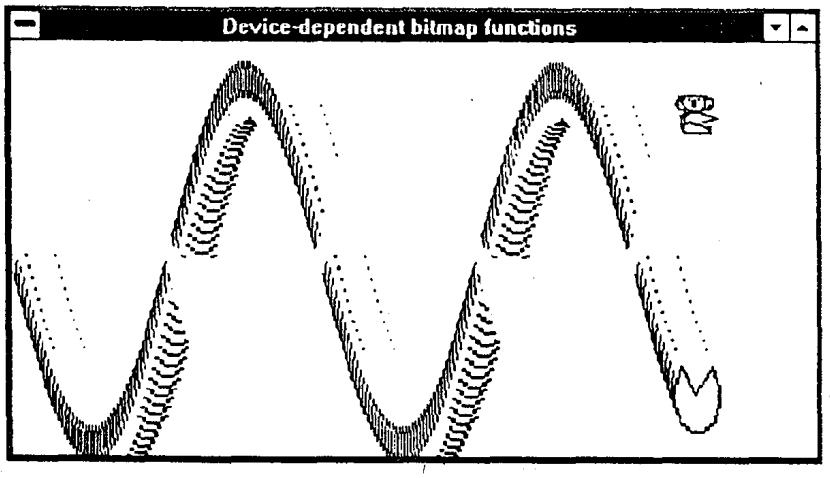


图 15.11

正弦曲线的 x 坐标值为 0 到 2p，在客户区将显示两个正弦曲线。位图总共要画 200 次。当画到右边界时，全部被消去，x 值重设为 0。

输出时需要一个选择位图的内存设备环境。此设备环境有时就是源设备环境。目标设备环境为客户区的显示环境。

在输出时，必须通过正弦计算从 x 坐标值求得 yClient，在(x，yClient/2)点画出位图。应用程序使用函数 BitBlt 画出位图 KOALA。

函数 StretchBlt 用来放大位图 Pie。当应用程序执行完毕时，所有位图和计时器都应被消除。其他计时器也可以被消除。

15.3 依赖于设备的位图(DIB)

当图像从一台机器移植到另一台上时，与设备有关的位图就会发生问题。因此，Windows 也提供了不依赖于设备的位图。使用这种位图的应用程序是 Paintbrush (PBRUSH. EXE)和 IMAGEDIT. EXE(SDK 图像编辑器)。

15.3.1 DIB 结构

Paintbrush(PBRUSH. EXE)和 IMAGEDIT. EXE 建立与设备无关的位图文件。这种文件包含一个称为 BITMAPFILEHEADER 的数据结构、一个 BITMAPINFOHEADER 结构和一个位模式数据。

比较一下依赖于设备的位图文件与 DIB 文件，就可发现与设备无关的位图文件含有更多的数组。

所有与设备无关的位图文件都以字母“BM”开头。

旧的位图文件(即用 ICONEDIT. EXE 建立的位图)可转换为 DIB 文件，只要用 IMAGEDIT. EXE 调出它们后再存盘即可。

IMAGEDIT. EXE 增加了必要的信息。函数 GetObject 可读取文件中的变量(除了数组

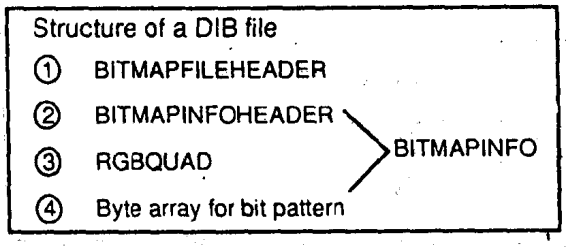


图 15.12 DIB 文件的结构

bmWidthBytes 例外),并从数组 biWidth 和变量 biBitCount 计算出 bmWidthBytes 变量的值(以像素为单位)。

变量 biBitCount 指示出显示一个像素需要多少位。由于两个位串可包含 256 个入口,它还充当数组 bmiColors 的一部分。对于单色位图,两个入口就足够了。

若位模式中的某一位为 0,则传送第一个入口的颜色,否则传送第二个入口的颜色。彩色位图必须用三种不同的选项来提供可见颜色的最大个数。

每像素占位	bmiColros 表的入口数	描述
1	2	单色位图
	Bit = 1→	像素使用第一个入口的颜色
	Bit = 0→	像素使用第二个入口的颜色
4	16	位图最多有 16 种颜色,每个像素用 4 个位来索引 bmiColors 表中的颜色
8	256	位图最多有 256 种颜色,每个像素用 8 个位来索引 bmiColors 表中的颜色
24	NULL	位图最多有 16M 种颜色,位图数组中每三位代表一个像素的红、蓝、绿色的亮度

数组 bmiColors 可被看作 16 位整数的数组,这样可以显示出当前逻辑调色板的序号并使得 RGB 值更有条理一些。在这种情况下,函数 DIB 必须通过参数 wUsage 向 DIB_PAL_COLORS 发消息。

除了结构 BITMAPINFO 外,还有一种结构叫作 BITMAPCOREINFO。它可描述 Presentation Manager(OS/2)格式下的与设备无关的位图。这使得 Windows 可以访问这种类型的位图。

这是依赖于设备的位图和与设备无关位图的另一重要区别。与设备无关位图的原点在图形的左下角,而不是像依赖于设备的位图那样在图形的左上角。

15.3.2 DIB 函数

Windows 3.1 版有四个控制与设备无关位图的函数:

- CreateDIBitmap
- GetDIBits
- SetDIBits
- SetDIBitsToDevice

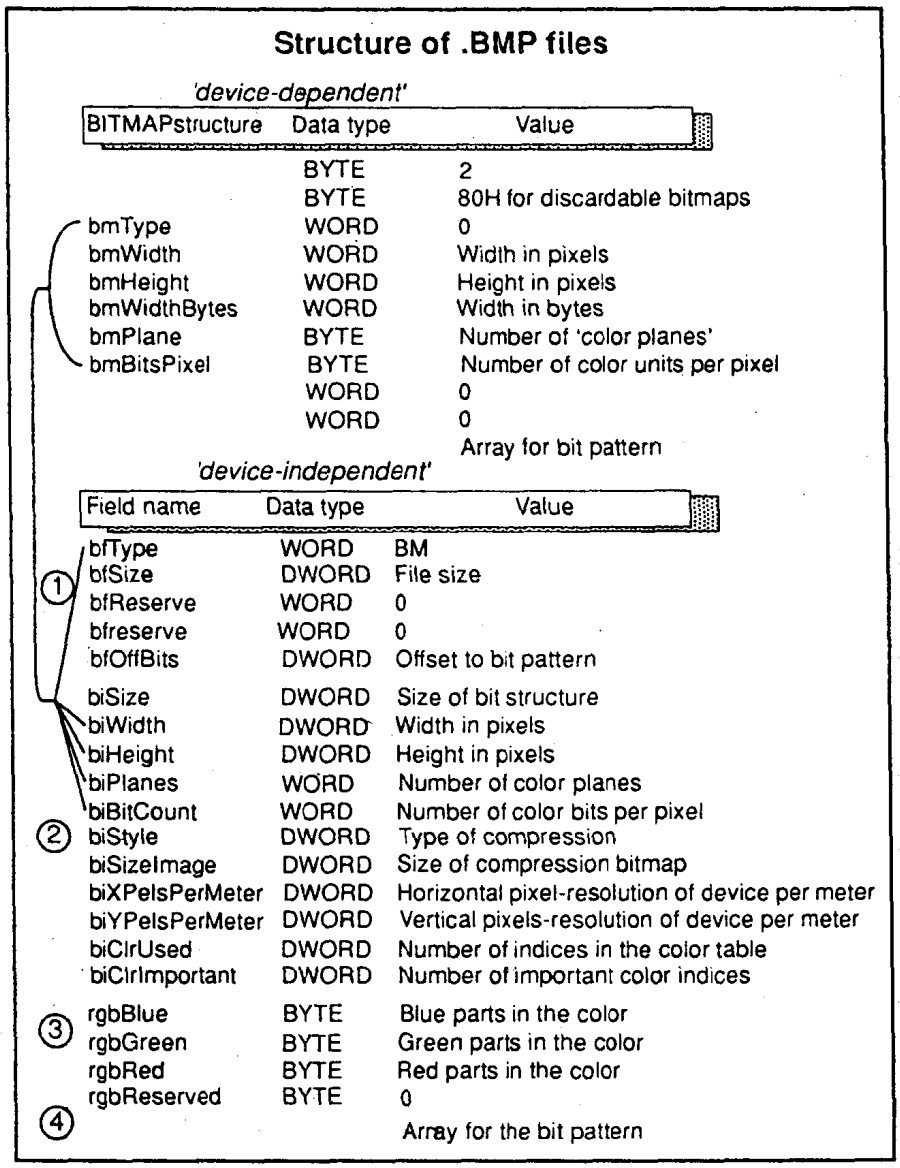


图 15.13 BMP 文件的结构

函数 CreateDIBitmap 用一个 DIB 的详细说明 BITMAPINFO 来建立一个大小合适的、与设备无关的彩色位图。

当参数 dwUsage 被设置为 CBM_INIT 时，位图被按位初始化。这些位由最后一个参数的指针寻址。

此函数用来把与设备无关的位图转化为依赖于设备的位图格式，它的第一个参数就是被转化成的设备环境。

依赖于设备的位模式被拷贝到内存中，通过它的内存句柄便可访问它。

如果没有任何一位在初始化时被传送，则仍可用函数 SetDIBits 写一个空位图。同时一个已存在的位图也可以用新的位模式填充。

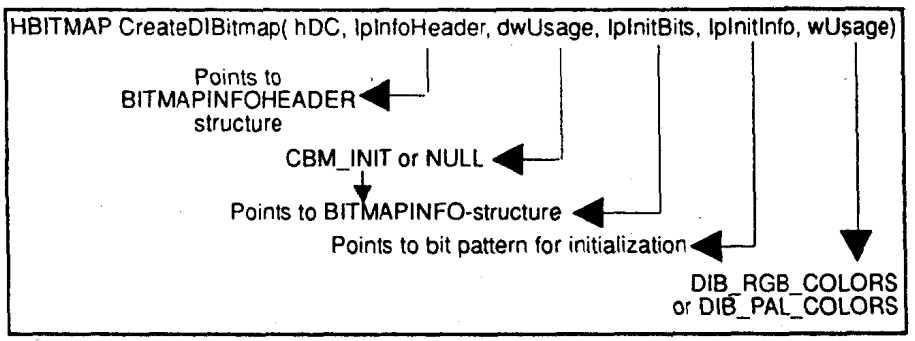


图 15.14　函数 CreateDIBitmap

因此,不能在设备环境中选择位图。结构 BITMAPINFO 中的变量决定了哪些字节应被设置,以及被怎样解释。

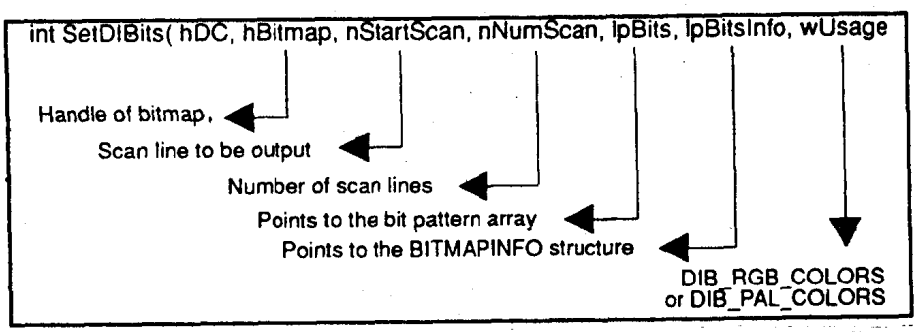


图 15.15　函数 SetDIBits

函数 GetDIBits 的参数与函数 SetDIBits 的相同。GetDIBits 把指针所指的位组按与设备无关的格式拷贝到模式中。返回值决定于传输了多少个光栅行。

int GetDIBits (hDC,hBitmap,nStartScan,nNumScan,lpBits,
lpBitsInfo,wUsage)

为了输出一个与设备无关的位图,必须把 SetDIBitsToDevice 写到设备接口中去。这时才需要内存设备环境。位图被定义为 BITMAPINFO 结构和一个位组(而不是一个句柄)。

SrcX,SrcY,nWidth 和参数 nHeight,nStartScan,nNumScan 的值传送整个或部分要显示的图形。

与设备无关的单色位图仍应使用依赖于设备的位图函数。

由于参考点的不同,有时可能要对图形进行垂直方向的镜面反射。这需要用函数 GetBitmapBits 改变光栅行的顺序。当位模式串已经被赋值后,就不应用函数 GetDIBits 而应该用 GetBitmapBits。

15.3.3　DIB 函数示例

这个例子使用前一个例子中的单色 KOALA 位图。这样做的原因是与设备无关的位图

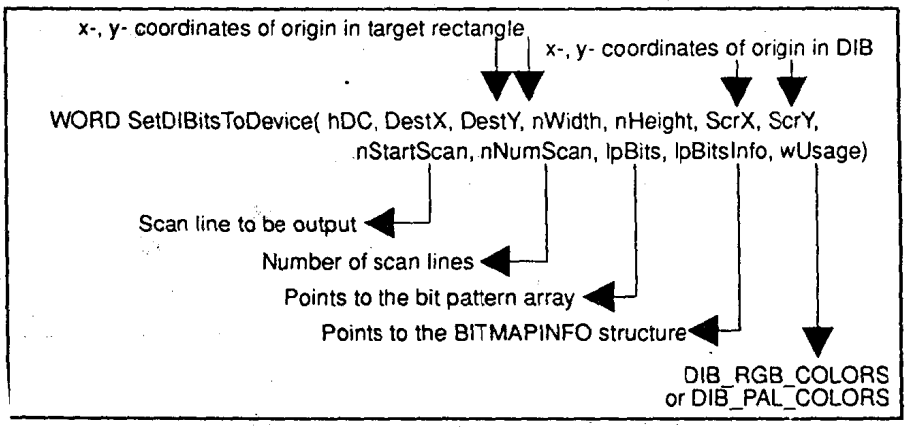


图 15.16　函数 SetDIBitsToDevice

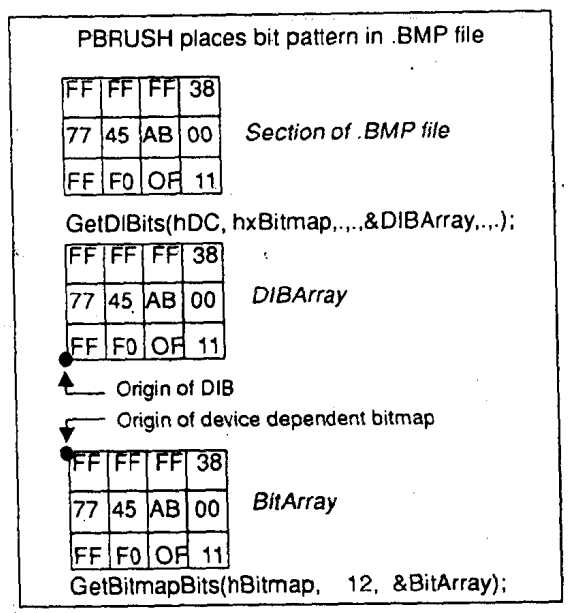


图 15.17　PBRUSH 放在 .BMP 文件中的位模式

例程使用的一系列光栅行不同于依赖于设备的位图。

函数 SetDIBitsToDevice 将显示两次图形，函数 BitBlt 将显示三次。函数 GetDIBits 和函数 GetBitmapBits 用两个数组转化位图。

新函数	描述
CreateDIBitmaps	根据 DIB 描述创建位图
GetBitmapBits	拷贝位图中的位到数组中
GetDIBits	拷贝 DIB 格式的位图位到数组中
SetDIBitsToDevice	直接显示 DIB

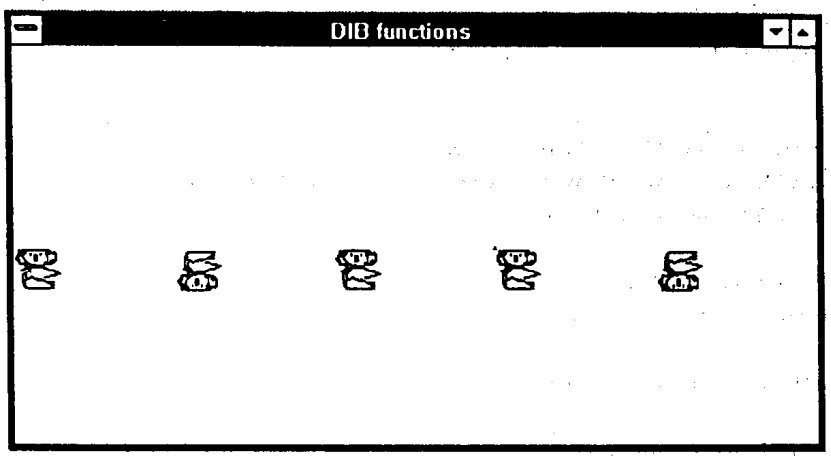


图 15.18

新结构	描述
BITMAPINFO	DIB 信息,分为:
BITMAPINFOHEADER	(DIB 格式信息)
RGBQUAD	(RGB 颜色结构)

源代码:BIT2.C

```
/* * BIT2.C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays the same bitmap in different ways * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                                    /* Include windows.h header file
#include "math.h"                                       // Include math.h header file
#include "string.h"                                     // Include string.h header file

#if _MSC_VER >= 700                                     // Running Microsoft C 7.00 or
    #pragma warning (disable:4028)                      // higher? Disable warning 4028
#endif

BOOL Bit2Init ( HANDLE );
long FAR PASCAL Bit2WndProc( HWND,unsigned,WORD,LONG);

/* * WinMain (main function for all Windows applications) * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;                         // Current & previous instance
LPSTR lpszCmdLine;                                      // Long ptr - string after prg name
int cmdShow;                                            // App. window's appearance
{
    MSG msg;                                            // Message handle
    HWND hWnd;                                          // Window handle

    if (! hPrevInstance)                                // Initialize first instance
```

```
    {
        if (! Bit2Init( hInstance ))
            return FALSE;
    }

    hWnd = CreateWindow("Bit2","DIB Functions",
        WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,0,CW_USEDEFAULT,0,
        NULL,NULL,hInstance,NULL);

    ShowWindow( hWnd,cmdShow );
    UpdateWindow( hWnd );

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int)msg.wParam;
}

BOOL Bit2Init( hInstance )
HANDLE hInstance;
{
    WNDCLASS wcBit2Class;

    wcBit2Class.hCursor = LoadCursor( NULL,IDC_ARROW );
    wcBit2Class.hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcBit2Class.lpszMenuName = NULL;
    wcBit2Class.lpszClassName = "Bit2";
    wcBit2Class.hbrBackground = GetStockObject( WHITE_BRUSH );
    wcBit2Class.hInstance = hInstance;
    wcBit2Class.style = CS_HREDRAW | CS_VREDRAW;
    wcBit2Class.lpfnWndProc = Bit2WndProc;
    wcBit2Class.cbClsExtra = 0 ;
    wcBit2Class.cbWndExtra = 0 ;

    if (! RegisterClass( &wcBit2Class ) )
        return FALSE;

    return TRUE;
}

long FAR PASCAL Bit2WndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    HANDLE hInst;
    PAINTSTRUCT ps;
```

```
HDC hDC,hMem;
static HBITMAP hBitmap,hBit,hDIB,hMixed,hOld;
static BITMAP bm;
static BITMAPINFO DIBInfo;
static BYTE DIBArray[200],BitArray[200];
static short xClient,yClient;

switch (message)
{
     case WM_CREATE:
          hInst = GetWindowWord( hWnd,GWW_HINSTANCE);
          hBitmap = LoadBitmap(hInst,"koala");
          GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bm);

          memset(&DIBInfo. bmiHeader,0,sizeof(BITMAPINFOHEADER));
          DIBInfo. bmiHeader. biSize = sizeof(BITMAPINFOHEADER);
          DIBInfo. bmiHeader. biWidth = bm. bmWidth;
          DIBInfo. bmiHeader. biHeight = bm. bmHeight;
          DIBInfo. bmiHeader. biPlanes = 1;
          DIBInfo. bmiHeader. biBitCount = 1;

          DIBInfo. bmiColors[0]. rgbRed = 0;
          DIBInfo. bmiColors[0]. rgbGreen = 0;
          DIBInfo. bmiColors[0]. rgbBlue = 0;
          DIBInfo. bmiColors[0]. rgbReserved = 0;
          DIBInfo. bmiColors[1]. rgbRed = 0xff;
          DIBInfo. bmiColors[1]. rgbGreen = 0xff;
          DIBInfo. bmiColors[1]. rgbBlue = 0xff;
          DIBInfo. bmiColors[1]. rgbReserved = 0;

          hDC = GetDC( hWnd);
          // Handle bitmap as DIB: Lower-left origin
          GetDIBits(hDC,hBitmap,0,bm. bmHeight,
               DIBArray,&DIBInfo,DIB_RGB_COLORS);

          // Create device-dependent bitmap
          hDIB = CreateDIBitmap(hDC,&DIBInfo. bmiHeader,
               CBM_INIT,DIBArray,
               &DIBInfo,DIB_RGB_COLORS);
          ReleaseDC(hWnd,hDC);

          hMixed = CreateBitmap(32,32,1,1,(LPSTR)DIBArray);

          // Handle bitmap as DIB: upper-left origin
          GetBitmapBits(hBitmap,bm. bmHeight * bm. bmWidthBytes,
               BitArray);

          // Create device-dependent bitmap
          hBit = CreateBitmap(32,32,1,1,(LPSTR)BitArray);
          break;
     case WM_SIZE:
```

```
            xClient = LOWORD(lParam);
            yClient = HIWORD(lParam);
            break;
        case WM_PAINT:
            hDC = BeginPaint(hWnd,&ps);
            SetDIBitsToDevice(hDC,0,yClient/2,
                bm.bmWidth,bm.bmHeight,0,0,0,32,
                DIBArray,&DIBInfo,DIB_RGB_COLORS);

        // Stand koala on its head
            SetDIBitsToDevice(hDC,1 * xClient/5,yClient/2,
                bm.bmWidth,bm.bmHeight,
                0,0,0,32,BitArray,&DIBInfo,
                DIB_RGB_COLORS);

            hMem = CreateCompatibleDC(hDC);
            hOld = SelectObject(hMem,hBit);
            BitBlt(hDC,2 * xClient/5,yClient/2,32,32,
                hMem,0,0,SRCCOPY);

            SelectObject(hMem,hDIB);
            BitBlt(hDC,3 * xClient/5,yClient/2,32,32,
                hMem,0,0,SRCCOPY);

            SelectObject(hMem,hMixed);
            // Stand koala on its head
            BitBlt(hDC,4 * xClient/5,yClient/2,
                32,32,hMem,0,0,SRCCOPY);
            SelectObject(hMem,hOld);
            DeleteDC(hMem);

            EndPaint(hWnd,&ps);
            break;
        case WM_DESTROY:                                    // Destroy window
            DeleteObject(hBitmap);
            DeleteObject(hBit);
            DeleteObject(hDIB);
            DeleteObject(hMixed);
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,
                wParam,lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:BIT2.DEF

```
NAME Bit2
```

```
DESCRIPTION 'Device-independent bitmap functions'

EXETYPE WINDOWS

STUB 'WINSTUB.EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS Bit2WndProc
```

资源描述文件:BIT2.RC

```
koala BITMAP koala.bmp
```

MAKE 文件:BIT2.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = koala.bmp

all: BIT2.EXE

BIT2.RES: BIT2.RC $(RESFILES)

BIT2.OBJ: BIT2.C

BIT2.EXE: BIT2.OBJ BIT2.RES BIT2.DEF
    link /NOD/A:16 BIT2,BIT2,nul,libw slibcew,BIT2.def
    rc BIT2.res BIT2.exe
```

像下面这样运行 NMAKE 便可产生应用程序 BIT2:

```
nmake bit2.mak
```

BIT2.EXE 是怎样工作的

这个例子提供了依赖设备的和与设备无关的位图函数的五个组合。

消息 WM_CREATE 用下面的 DIB 函数给 BITMAPINFO 结构变量 DIBinfo 赋值。这些变量是由 .RC 文件从 KOALA.BMP 中读出的。颜色表 bmiColors 包含两个入口,这保证了位图按照单色位图处理。

位模式被函数 GetDIBits 拷贝到数组 BitArray 中,被依赖设备的函数 GetBitmapBits 拷贝到数组 DIBArray。

函数 CreateBitmap 被调用了两次，以访问两个数组之一。函数 CreateDIBitmap 建立了第三个位图。

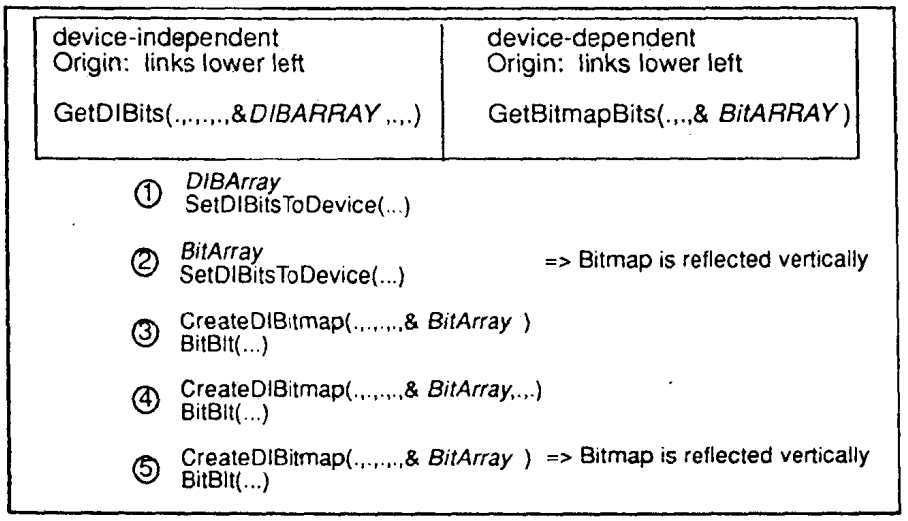


图 15.19

客户区的大小(被消息 WM_SIZE 定义)和消息 WM_PAINT 控制着按顺序显示五个 Koala 图像。

最初位图被 DIB 函数和数组 DIBArray 直接写到屏幕上。由于数组 BitArray 还需要其他数组，这一行开头的图形需要与设备无关的位图和依赖设备的位图的函数组合来显示。

函数 BitBlt 需要一个内存设备环境来依次地显示后面三个图形。第二个位图是由一个依赖于设备的位图函数和一些 DIB 数组联合建立的。这个位图被垂直方向镜面反射地显示出来。

第十六章　位、块传输和元文件

位图和元文件代表着两种不同的存储图像信息的方法。位图是图像完全数字化后的表示形式。图像中的每一个像素点对应于位图中的一个或多个位。单色位图中每个像素点只需要一位；彩色位图则需要更多的位，以指示每个像素点的颜色。

元文件作为表示图像的另一种方法，它将图像信息存储为一系列记录，这些记录直接对应于 GDI 调用，如 MoveTo、Rectahgle、TextOut 等。因此，元文件不是图像的数字表示，而是对图像的描述。

在计算机图形学中，位图和元文件各有其作用。位图非常频繁地用来表示来源于真实世界的复杂图像，如数字化的照片。元文件也比较适合于描述由人或计算机生成的图像，如建筑图纸等。位图和元文件都能存在于内存中，或者作为文件存放在磁盘上，两者都可以利用剪贴板在 Windows 应用程序间传输。

程序员可以利用 Windows 3.1 提供的 PAINTBRUSH 程序“手工”地构造位图。然后，可以将位图作为资源包含到资源描述文件中，并用 LoadBitmap 函数将它载入到程序中。

元文件与 Windows 的绘图程序（如 Micrografx Designer）以及其他 CAD（计算机辅助设计）程序有很密切的联系。这些程序的用户用直线、矩形、椭圆、文本以及其他图形原语绘图。尽管它们通常用一种专用的格式来将图像存储到文件中，但它们总是能将图像以元文件的格式传输到剪贴板上。

位图有两个主要缺陷。首先，它极其容易涉及设备相关性的问题。最明显的与设备相关的特性是颜色。在单色设备上显示彩色位图通常是不会满意的。另一个问题是，一个位图隐含着一个特定的分辨率和图像的纵横比。尽管位图可以延展或压缩，但这一过程通常涉及到复制或减除像素点的行或列，从而引起缩放后图像的失真。元文件几乎可以缩放到任意尺寸，而不会导致失真。

位图的第二个主要缺陷是，它需要的存储空间太大。例如，表示一个完整的 640×480 像素点、16 色 VGA 屏幕上的位图需要的空间超过 150KB。元文件需要的存储空间总是比位图少得多。位图的存储空间是由图像的大小和它所包含的颜色数目决定的，而元文件的存储空间是由图像的复杂程度和它所包含的单个 GDI 指令的数目决定的。

位图较元文件的优越之处在于速度。将位图拷贝到视频显示器上比起重新执行元文件总是要快得多。

有两种类型的设备环境不代表真实的设备，它们是内存设备环境和元文件设备环境。在说明位图和元文件并介绍创建、使用和操作它们的方法的同时，我们将看到这两种设备环境是如何使用的。文本、位图和元文件是可以由应用程序通过剪贴板共享数据的三种基本形式。

16.1 颜色和位图

图像中的每个像素点对应于位图中的一个或多个位。单色图像中，每个像素点只需要一位。彩色图像中的每个像素点需要多位。一个位图可以表示的不同颜色数等于以 2 为底数，以每像素点位数为指数计算出的幂。例如，16 色位图中，每个像素点需要 4 位，256 色位图中，每个像素点需要 8 位。

在 Windows 3.0 之前，Windows 所支持的唯一位图是 GDI 对象，它可以用位图句柄来引用。这些位图是单色的，或者与真实图形输出设备（如视频显示器）有同样的颜色组织。例如，一个与 VGA 兼容的位图有四个颜色平面。

问题是，这些彩色位图不能在保存之后又在具有不同颜色组织的图形输出设备上使用，如 VGA 的位图不能在 IBM 8514/A 上显示，因为 IBM 8514/A 能显示 256 种颜色，每个像素点使用 8 位。

对于 Windows 3.0，定义了一种新的位图格式，叫做设备无关的位图，简写为 DIB。DIB 包含它自己的颜色表，它显示像素点位如何与 RGB 颜色相对应。DIB 可以在任意光栅输出设备上显示。唯一的问题是，DIB 的颜色必须被转换成设备所能表示的最相近的颜色。

16.2 与设备无关的位图(DIB)

在前几章讨论位图的使用时，我们用函数（如 CreateBitmap 和 LoadBitmap）创建了 GDI 位图对象。这些函数返回一个类型为 HBITMAP 的位图句柄。实际位图驻留在由 GDI 控制的内存中。自 Windows 1.0 以来，这些位图对象就是 Windows 所支持的。它们必须或者是单色位图，或者是彩色位图，且与视频显示器或者打印机兼容。

DIB 格式之所以被称为“与设备无关的”，是因为它包含有一个颜色表。颜色表描述了像素点值是如何与 RGB 颜色值相对应的。这个颜色表不一定与特定图形输出设备兼容。

DIB 格式是 OS/2 1.1 Presentation Manager（图像管理器）所支持的位图格式的扩展。WINDOWS.H 头文件包含有一些与 OS/2 位图有关的结构。OS/2 2.0 图像管理器支持的位图是 DIB 的扩展。与 DIB 相对应，有时称 GDI 位图对象为“与设备相关的位图”。之所以说它们与设备相关，是因为它们必须与一种特定的图形输出设备兼容。DIB 不是 GDI 对象。GDI 不能存储 DIB。DIB 必须存储在 GDI 之外的文件中或者全局内存块中。一旦 GDI 获取一个 DIB 之后，它就会将 DIB 转化成与设备相关的位图，以与某种真实设备兼容。

DIB 主要用于程序之间的交换。通过将 DIB 存储到文件中，或者将它们拷贝到剪贴板上，均可以在程序之间传送。

DIB 可以转换成一个与设备相关的 GDI 位图对象；在这种的情况下，DIB 与设备无关的颜色信息就丢失了。GDI 位图对象也能用以构造 DIB，在这种情况下，DIB 将包含一个与图形输出设备兼容的颜色表，而 GDI 位图对象也与该输出设备兼容。

如果需要以一种与设备无关的格式将位图信息存放到文件中、读取位图文件或者将位图信息传送进/出剪贴板，就必需使用 DIB。但是，如果只需要创建或使用单色位图，或者只需要在自己的程序中使用与视频显示器兼容的位图，则使用与设备相关的 GDI 位图对象通

常要容易得多。

16.2.1 DIB 文件

程序员可以用 Microsoft Image Editor(图像编辑器)、Borland Resource Workshop 或者销售的 Windows 产品中提供的 PAINTBRUSH 程序来创建与设备无关的位图,并将它保存到磁盘文件中。该文件通常具有扩展名 .BMP,尽管有些 DIB 是使用扩展名 .DIB 的。

DIB 文件开始于一个文件头段,它由 BITMAPFILEHEADER 结构定义。这个结构有五个字段:

字段	长度	描述
bfType	UINT	字节"BM"(代表位图)
bfSize	DWORD	文件的总尺寸
bfReserved1	UINT	设置为 0
bfReserved2	UINT	设置为 0
bfOffBits	DWORD	位图位离文件头的偏移地址

后跟另一个头段,它由 BITMAPINFOHEADER 结构定义。这个结构有 11 个字段:

biSize	DWORD	这个结构的长度,以字节为单位
biWidth	LONG	位图的宽度,单位为像素点
biHeight	LONG	位图的高度,单位为像素点
biPlanes	WORD	设置为 1
biBitCount	WORD	每像素点所需颜色位(1、4、8、24)
biSizeImage	LONG	位图位的长度,以字节为单位(只有在使用压缩时才需要)
biXPelsPerMeter	LONG	水平分辨率,单位为像素点/m
biYPelsPerMeter	LONG	垂直分辨率,单位为像素点/m
biClrUsed	DWORD	图像中使用的颜色数
biClrImportant	DWORD	图像中重要颜色的数目

在 biBitCount 字段之后的所有字段可能以 0 为缺省值(或者甚至不在文件中出现)。因此,结构的长度可以小至只有 16 字节。也可以在表中所列字段之外增加其他字段。

如果 biClrUsed 设置为 0,且每像素点的颜色位数为 1,4 或 8,则 BITMAPINFOHEADER 结构后跟一颜色表,颜色表由两个或更多的 RGBQUAD 结构组成。RGBQUAD 结构定义了一个 RGB 颜色值:

字段	长度	描述
rgbBlue	BYTE	蓝色的密度
rgbGreen	BYTE	绿色的密度
rgbRed	BYTE	红色的密度
rgbReserved	BYTE	设置为 0

RGBQUAD 结构的数目通常由 biBitCount 字段确定:1 个颜色位需要两个 RGBQUAD 结构,四个颜色位需要 16 个 RGBQUAD 结构,八个颜色位需要 256 个 RGBQUAD 结构。但是,如果 biClrUsed 字段不为 0,则 biClrUsed 字段包含了颜色表中 RGBQUAD 结构的数目。

颜色表后跟以定义位图图像的位数组。这个数组从底行像素点开始。每行从最左边的像素点开始。每个像素点对应于 1,4,8 或 24 位。

对于每像素点一个颜色位的单色位图,每行的每一个像素点由每行第一个字节的最有效的位表示。如果这一位为 0,则像素点的颜色可以从颜色表中的第一个 RGBQUAD 结构中获取。如果该位为 1,则颜色由颜色表中的第二个 RGBQUAD 结构给出。

对于 16 色位图(每像素点 4 个颜色位),每行的第一个像素点由每行第一个字节中最有效的 4 位来表示。每个像素点是通过用 4 位值作为颜色表中 16 个表项的索引来获取的。

对于 256 色的位图,每个字节对应于一个像素点。像素点的颜色用这个 8 位值作为颜色表的 256 个表项的索引来获取。

如果位图图像中每像素点为 24 个颜色位,每三个字节是一个像素点的 RGB 值。没有颜色表(除非 BITMAPINFOHEADER 结构的 biClrUsed 字段不为 0)。

在每种情况下,位图数据的每行包含 4 的倍数个字节。该行被填充至右边以确保这一点。

OS/2 1.1 及其以后的版本所支持的位图格式是相似的。它开始于一个 BITMAPFILEHEADER 结构,但后跟一个 12 字节的 BITMAPCOREHEADER 结构。通过查看该结构的第一个字段,程序员能确定位图文件是使用这种格式还是 Windows 格式。颜色表由 RGBTRIPLE 结构(而不是 RGBQUAD 结构)组成。

16.2.2 压缩 DIB 内存格式

如果程序员遇到一个 DIB 文件,想将它读入 Windows 程序中,可以直接将它读进一个全局内存块中。这种内存块经常会具有超过 64KB 的长度。这被称为"压缩 DIB 内存格式"。它包含 DIB 文件中除了 BITMAPFILEHEADER 结构之外的所有数据。

因此,一个包含 DIB 的全局内存块以一个 BITMAPINFOHEADER 结构开始,后跟一个颜色表(如果给出的话),再后跟位图位。压缩 DIB 内存格式用于将 DIB 拷贝到剪贴板或将 DIB 拷贝出剪贴板。

也可以使用 SetDIBitsToDevice 或 StretchDIBits 函数以压缩 DIB 内存格式在屏幕上显示一个位图,以创建一个基于 DIB 的刷子(CreatePatternBrush),或根据 DIB 创建一个与设备相关的 GDI 位图对象(用 CreateDIBitmap)。

在每种情况下,都必须计算位图位在全局内存块中的开始地址。通过检查 BITMAPINFOHEADER 结构,以确定颜色表的大小,就可以计算出来。这个过程如下节的 SHOWDIB 程序所示。

16.2.3 显示 DIB

Windows 有两个函数,它们能用压缩 DIB 内存块显示 DIB。最通用的是 StretchDIBits,它允许位图被延伸或压缩,可使用各种光栅操作。SetDIBitsToDevice 函数要简单一些;它显示位图,不扩展也不压缩它,它也不能使用任意光栅操作。

在下面给出的 SHOWDIB 程序中演示了这两个函数以及从文件中将 DIB 读入全局内存块的过程。

```
# SHOWDIB. MAK make file
```

```
showdib.exe : showdib.obj showdib.def showdib.res
     $(WINLINK) showdib,showdib,NUL, $(WINLIB),showdib
     rc -t showdib.res

showdib.obj : showdib.c showdib.h
     $(WINCC) showdib.c

showdib.res : showdib.rc showdib.h
     $(WINRC) showdib.rc

/*---------------------------------------
SHOWDIB.C -- Show DIB in Window
---------------------------------------*/

#include <windows.h>
#include <windowsx.h>
#include <commdlg.h>
#include <stdlib.h>
#include "showdib.h"

#define MIN(a,b) ((a) < (b) ? (a) : (b))

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

char szAppName [] = "ShowDib" ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
          LPSTR lpszCmdLine,int nCmdShow)
{
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = NULL ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = szAppName ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
     }

     hwnd = CreateWindow (szAppName,"Show DIB in Window",
```

```
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,CW_USEDEFAULT,
        CW_USEDEFAULT,CW_USEDEFAULT,
        NULL,NULL,hInstance,NULL) ;

    ShowWindow (hwnd,nCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg,NULL,0,0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam ;
}

// Functions for extracting information from DIB memory blocks

DWORD GetDibInfoHeaderSize (BYTE huge * lpDib)
{
    return ((BITMAPINFOHEADER huge * ) lpDib)->biSize ;
}

WORD GetDibWidth (BYTE huge * lpDib)
{
    if (GetDibInfoHeaderSize (lpDib) == = sizeof (BITMAPCOREHEADER))
        return (WORD) (((BITMAPCOREHEADER huge * ) lpDib)->bcWidth) ;
    else
        return (WORD) (((BITMAPINFOHEADER huge * ) lpDib)->biWidth) ;
}

WORD GetDibHeight (BYTE huge * lpDib)
{
    if (GetDibInfoHeaderSize (lpDib) == = sizeof (BITMAPCOREHEADER))
        return (WORD) (((BITMAPCOREHEADER huge * ) lpDib)->bcHeight) ;
    else
        return (WORD) (((BITMAPINFOHEADER huge * ) lpDib)->biHeight) ;
}

BYTE huge * GetDibBitsAddr (BYTE huge * lpDib)
{
    DWORD dwNumColors,dwColorTableSize ;
    WORD wBitCount ;

    if (GetDibInfoHeaderSize (lpDib) == = sizeof (BITMAPCOREHEADER))
    {
        wBitCount = ((BITMAPCOREHEADER huge * ) lpDib)->bcBitCount ;

        if (wBitCount ! = 24)
            dwNumColors = 1L << wBitCount ;
        else
```

```
            dwNumColors = 0 ;

        dwColorTableSize = dwNumColors * sizeof (RGBTRIPLE) ;
    }
    else
    {
        wBitCount = ((BITMAPINFOHEADER huge * ) lpDib)->biBitCount ;

        if (GetDibInfoHeaderSize (lpDib) >= 36)
            dwNumColors = ((BITMAPINFOHEADER huge * ) lpDib)->biClrUsed ;
        else
            dwNumColors = 0 ;

        if (dwNumColors == 0)
        {
            if (wBitCount != 24)
                dwNumColors = 1L << wBitCount ;
            else
                dwNumColors = 0 ;
        }

        dwColorTableSize = dwNumColors * sizeof (RGBQUAD) ;
    }

    return lpDib + GetDibInfoHeaderSize (lpDib) + dwColorTableSize ;
}

// Read a DIB from a file into memory

BYTE huge * ReadDib (char * szFileName)
{
    BITMAPFILEHEADER bmfh ;
    BYTE huge * lpDib ;
    DWORD dwDibSize,dwOffset,dwHeaderSize ;
    int hFile ;
    WORD wDibRead ;

    if (-1 == (hFile = _lopen (szFileName,OF_READ | OF_SHARE_DENY_WRITE)))
        return NULL ;

    if (_lread (hFile,(LPSTR) &bmfh,sizeof (BITMAPFILEHEADER)) !=
        sizeof (BITMAPFILEHEADER))
    {
        _lclose (hFile) ;
        return NULL ;
    }

    if (bmfh.bfType != * (WORD * ) "BM")
    {
        _lclose (hFile) ;
        return NULL ;
```

```
    }

    dwDibSize = bmfh.bfSize - sizeof (BITMAPFILEHEADER) ;

    lpDib = (BYTE huge * ) GlobalAllocPtr (GMEM_MOVEABLE,dwDibSize) ;

    if (lpDib == NULL)
    {
        _lclose (hFile) ;
        return NULL ;
    }

    dwOffset = 0 ;

    while (dwDibSize > 0)
    {
        wDibRead = (WORD) MIN (32768ul,dwDibSize) ;

        if (wDibRead != _lread (hFile,(LPSTR) (lpDib + dwOffset),wDibRead))
        {
            _lclose (hFile) ;
            GlobalFreePtr (lpDib) ;
            return NULL ;
        }

        dwDibSize -= wDibRead ;
        dwOffset += wDibRead ;
    }

    _lclose (hFile) ;

    dwHeaderSize = GetDibInfoHeaderSize (lpDib) ;

    if (dwHeaderSize < 12 || (dwHeaderSize > 12 && dwHeaderSize < 16))
    {
        GlobalFreePtr (lpDib) ;
        return NULL ;
    }

    return lpDib ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
        LONG lParam)
{
    static char szFileName [_MAX_PATH],
                szTitleName [_MAX_FNAME + _MAX_EXT] ;
    static char * szFilter[] = { "Bitmap Files (*.BMP)", "*.bmp",
                "DIB Files (*.DIB)", "*.dib",
                "" } ;
    static BYTE huge * lpDib ;
```

```
static OPENFILENAME ofn ;
static short cxClient,cyClient ;
static WORD wDisplay = IDM_ACTUAL ;
HDC hdc ;
HMENU hMenu ;
BYTE huge * lpDibBits ;
PAINTSTRUCT ps ;
short cxDib,cyDib ;

switch (message)
{
    case WM_CREATE:
        ofn.lStructSize = sizeof (OPENFILENAME) ;
        ofn.hwndOwner = hwnd ;
        ofn.lpstrFilter = szFilter [0] ;
        ofn.lpstrFile = szFileName ;
        ofn.nMaxFile = _MAX_PATH ;
        ofn.lpstrFileTitle = szTitleName ;
        ofn.nMaxFileTitle = _MAX_FNAME + _MAX_EXT ;
        ofn.lpstrDefExt = "bmp" ;

        return 0 ;
    case WM_SIZE:
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
        return 0 ;
    case WM_COMMAND:
        hMenu = GetMenu (hwnd) ;

        switch (wParam)
        {
            case IDM_OPEN:
                if (GetOpenFileName (&ofn))
                {
                    if (lpDib ! = NULL)
                    {
                        GlobalFreePtr (lpDib) ;
                        lpDib = NULL ;
                    }

                    lpDib = ReadDib (szFileName) ;

                    if (lpDib == = NULL)
                        MessageBox (hwnd,szAppName,
                            "Could not open DIB file",
                            MB_ICONEXCLAMATION | MB_OK) ;

                    InvalidateRect (hwnd,NULL,TRUE) ;
                }
                return 0 ;
            case IDM_ACTUAL:
```

```
                case IDM_STRETCH:
                    CheckMenuItem (hMenu,wDisplay,MF_UNCHECKED) ;
                    wDisplay = wParam ;
                    CheckMenuItem (hMenu,wDisplay,MF_CHECKED) ;
                    InvalidateRect (hwnd,NULL,TRUE) ;
                    return 0 ;
            }
            break ;
        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;
            if (lpDib != NULL)
            {
                lpDibBits = GetDibBitsAddr (lpDib) ;
                cxDib = GetDibWidth (lpDib) ;
                cyDib = GetDibHeight (lpDib) ;

                SetStretchBltMode (hdc,COLORONCOLOR) ;

                if (wDisplay == IDM_ACTUAL)
                    SetDIBitsToDevice (hdc,0,0,cxDib,cyDib,0,0,
                        0,cyDib,(LPSTR) lpDibBits,
                        (LPBITMAPINFO) lpDib,
                        DIB_RGB_COLORS) ;
                else
                    StretchDIBits (hdc,0,0,cxClient,cyClient,
                        0,0,cxDib,cyDib,
                        (LPSTR) lpDibBits,
                        (LPBITMAPINFO) lpDib,
                        DIB_RGB_COLORS,SRCCOPY) ;
            }
            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_DESTROY:
            if (lpDib != NULL)
                GlobalFreePtr (lpDib) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

/* ------------------------
SHOWDIB.RC resource script
------------------------ */

#include "showdib.h"

ShowDib MENU
{
    POPUP "&File"
    {
```

```
        MENUITEM "&Open…",IDM_OPEN
    }
    POPUP "&Display"
    {
        MENUITEM "&Actual Size", IDM_ACTUAL,CHECKED
        MENUITEM "&Stretch to Window", IDM_STRETCH
    }
}

/*-----------------------
SHOWDIB.H header file
-----------------------*/

#define IDM_OPEN 10
#define IDM_ACTUAL 20
#define IDM_STRETCH 21

; SHOWDIB.DEF module definition file

NAME SHOWDIB

DESCRIPTION 'Show DIB in Window (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

当用户从 File 菜单中选择 Open 后，SHOWDIB 显示通用对话框库中的 File Open 对话框。然后它调用函数 ReadDib。

ReadDib 函数将一个 DIB 文件读入全局内存块，内存块是用 WINDOWSX.H 中定义的宏 GlobalAllocPtr 分配的。在打开文件后，函数将文件的开头读入一个 BITMAPFILEHEADER 结构中。检查前两个字节(bfType 字段)是否为"BM"标记，然后用 bfSize 字段计算文件的大小(小于结构 BITMAPFILEHEADER 的大小)。结果为压缩 DIB 内存块的大小。

然后，ReadDib 分配一个这样大小的全局内存块。注意，用于存放这个内存块的指针被定义为"huge"指针。许多 DIB 的长度超过 64KB，程序员必须使用 huge 指针，以便编译器生成正确的代码来处理 DIB 的多个段。现在，文件的剩下部分被简单地以一次 32KB 块的方式读入内存块。

最后有一个校验检查，即 ReadDib 调用 GetDibInfoHeaderSize(它只取出内存块的第一个 DWORD)，并检查它是 12 字节长(在这种情况下，程序所处理的是 OS/2 1.x 位图文件)还是最少 16 字节长(在这种情况下，它是一个 Windows DIB)。

SHOWDIB 中的 GetDibWidth 和 GetDibHeight 函数从内存块取出位图宽度和高度。GetDibBitsAddr 函数要复杂得多。它需要 BITMAPINFOHEADER 结构中的一些信息，以计算 DIB 图位的内存地址。

在 WndProc 中处理 WM_PAINT 消息，用这三个函数将参数传送给 SetDIBitsToDevice 或 StretchDIBits 函数。这两个函数都需要一个指向 BITMAPINFOHEADER 结构的指针(DIB 内

存块的开头)和一个指向位图位的指针。StretchDIBits 函数特别通用,它允许程序员在位图内以指定逻辑宽度和高度在显示器上显示任何矩形。

16.2.4 创建 DIB

如果使用 16 色或 256 色视频显示器读入一个全色位图,每像素点 24 位,显示位图所需要的时间会很长。这是因为设备驱动程序必须对位图中每个像素点执行最相近颜色搜索。

用 CreateDIBitmap 函数将 DIB 转换成与设备相关的 GBI 位图对象就能提高显示处理的速度。最相近颜色的搜索必须只执行一次,然后位图对象将成为与视图显示器相同的格式。在 SHOWDIB 中使用这些变量,函数调用将为:

```
hBitmap = CreateDIBitmap (hdc,
          (LPBITMAPINFOHEADER) lpDib,
          CBM_INIT,(LPSTR) lpDibBits,
          (LPBITMAPINFO) lpDib,
          DIB_RGB_COLORS);
```

与函数的名字相反,CreateDIBitmap 不创建 DIB。它用一个 DIB 说明创建与设备相关的 GDI 位图对象,它返回该对象的句柄。这个 GDI 位图对象与图形输出设备兼容,设备的句柄作为第一个参数传送给函数。与显示 DIB 时相似,GDI 必须将与设备无关的颜色转换成设备的颜色。

在这个调用之后,通过将位图句柄选进一个内存设备环境,并使用 BitBlt 就能显示位图。

也能用 CreateDIBitmap 创建一个未初始化的 GDI 位图对象:

```
hBitmap = CreateDIBitmap(hdc,&bmih,0L,NULL,NULL,0);
```

两个函数可用于设置和获取位图的位。第一个函数设置位:

```
SetDIBits(hdc,hBitmap,nStart,nNum,lpBits,&bmi,wUsage);
```

后面三个参数与 CreateDIBitmap 函数的参数相同。nStart 参数指定开始扫描行地址(lpBits),它能从 0(代表最下面的扫描行)至位图的高度(单位为像素点)减 1(代表最上面的扫描行)。nNum 参数指示设置到位图中的扫描行数。

GetDIBits 函数具有同样的参数:

```
GetDIBits(hdc,hBitmap,nStart,nNum,lpBits,&bmi,wUsage);
```

但是在这种情况下,lpBits 指向一个接收位图位的缓冲区。函数设置 BITMAPINFO 结构的字段,以指示位图和颜色表的尺度。

16.3 GDI 位图对象

源于 Windows 1 的早期位图格式限制很多,它高度依赖于位图为之创建的输出设备。应该使用 DIB 格式,而不应使用早期位图格式在磁盘上存储位图文件。尽管如此,如果所需位图只在程序内使用,则采用与设备相关的位图要容易得多,性能也会更好。

16.3.1 在程序中创建位图

Windows 提供了五个函数,让程序员在程序中创建与设备相关的 GDI 位图对象。第一种

是上面讨论的 CreateDIBitmap 函数。其他函数是：

```
hBibmap = CreateBitmap (cxWidth,cyHeight,nPlanes,nBitsPixel,lpBits);
hBitmap = CreateBitmapIndirect (&bitmap);
hBitmap = CreateCompatibleBitmap (hdc,cxWidth,cyHeight);
hBitmap = CreateDiscardableBitmap (hdc,cxWidth,cyHeight);
```

CreateDiscardableBitmap 函数很少使用，这里不予推荐。

在所有情况下，cxWidth 和 cyHeihgt 参数都是位图的宽度和高度(单位为像素点)。在 CreateBitmap 中，nPlanes 和 nBitsPixel 参数分别是颜色平面数目和位图中每像素点的颜色位数。两个参数中至少有一个为 1。如果两个参数都为 1，则函数创建的是单色位图。下面我们将简单讨论颜色平面和颜色位是如何表示颜色的。

如果要创建的是未初始化的位图，则可以将 CreateBitmap 函数的 lpBits 设置为 NULL。结果位图中包含随机数据。在 CreateCompatibleBitmap 和 CreateDiscardableBitmap 函数中，Windows 使用 hdc 所指示的设备环境来获取颜色平面数和每像素点的颜色位数。这些函数创建的位图都未初始化。

CreateBitmapIndirect 与 CreateBitmap 类似(除了它使用 BITMAP 类型的位图结构定义位图外)。下表显示了这个结构的各个字段：

字段	类型	描述
bmType	int	设置为 0
bmWidth	int	位图宽度，以像素点为单位
bmHeight	int	位图高度，以扫描行为单位
bmWidthBytes	int	位图宽度必须为偶数，以字节为单位
bmPlanes	BYTE	颜色平面数
bmBitsPixel	BYTE	每像素点颜色位数
bmBits	void FAR *	far 指针，指向位数组

bmWidthBytes 字段必须是偶数——存储一个扫描行所需的最小偶字节数。bmBits 所指示的位数组必须根据 bmWidthBytes 字段来组织。如果 bm 是一个类型为 BITMAP 的结构变量，则可以用下面的语句计算 bmWidthBytes 字段：

```
bm.bmWidthBytes = (bm.bmWidth * bm.bmBitsPixel + 15) / 16 * 2;
```

如果 Windows 不能创建位图(一般是因为没有足够的内存)，它将返回 NULL。必须检查从位图创建函数返回的值，特别是在创建大位图时。

位图的句柄不是全局内存块的句柄，所以不要试图对它使用 GlobLock 函数。该句柄是一个 GDI 模块数据段的局部句柄。该句柄指向 GDI 中一个小局部内存块，该内存块包含 BITMAP 结构中的信息以及实际位。

一旦创建位图之后，就不能改变其大小、颜色平面数或像素点的颜色位数。必须另创建一个位图，并将原位图的位传输给新的位图。

如果程序员具有位图句柄，就可以用如下语句获取尺寸和颜色结构：

```
GetObject(hBitmap,sizeof (BITMAP),(LPSTR) &bitmap);
```

这样就将位图信息拷贝到了一个类型为 BITMAP 的结构 bitmap 中。这个函数不填充 bmBits 字段。为了访问位图的实际位，必须调用：

```
GetBitmapBis(hBitmap,dwCount,lpBits);
```

这样将 dwCount 位拷贝到 far 指针 lpBits 所指的字符数组中。为了确保位图的所有位都拷贝进这个数组，可以根据位图结构的各字段计算 dwCount 参数：

dwCount = (DWORD) bitmap.bmWidthBytes * bitmap.bmHeight * bitmap.bmPlanes;

也可以用以下函数指示 Windows 将一个包含有位图位的字符数组拷贝到已存在的位图中：

SetBitmapBits(hBitmap,dwCount,lpBits),

由于位图是 GDI 对象，所以必须删除所创建的位图：

DeleteObject(hBitmap);

16.3.2 单色位图格式

单色位图的位格式相对而言较简单，几乎可以直接从程序员所要创建的图像推出来。例如，假设程序员想创建一个外形如下所示的位图：

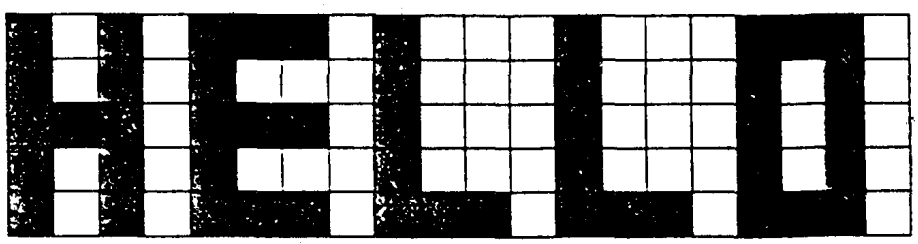


图 16.1

程序可以写下一系列位(0 代表黑，1 代表白)，直接对应于这个网格。从左到右读这些位，然后，可以以每 8 位一组指定一个十六进制字节。如果位图的宽度不是 16 的整数倍，则在字节的右部填充 0，以得到偶数个字节：

0 1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 = 51 77 10 00
0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 = 57 77 50 00
0 0 0 1 0 0 1 1 0 1 1 1 0 1 1 1 0 1 0 1 = 13 77 50 00
0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 = 57 77 50 00
0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 = 51 11 10 00

宽度(单位为像素点)为 20，高度为 5 个扫描行，字节宽度为 4。可以用下面的语句为位建立一个 BITMAP 结构：

```
static BITMAP bitmap = {0,20,5,4,1,1,};
```

可以将这些位存放到一个 BYTE 数组中：

```
static BYTE ByBits [] = { 0x51,0x77,0x10,0x00,
        0x57,0x77,0x50,0x00,
        0x13,0x77,0x50,0x00,
        0x57,0x77,0x50,0x00,
        0x51,0x11,0x10,0x00 };
```

用 CreateBitmapIndirect 创建位图时需要两条语句：

```
bitmap.bmBits = (LPSTR) byBits;
```

```
hBitmap = CreateBitmapIndirect (&bitmap);
```

在使用指向 byBits 的指针时应该小心。在将这个 far 地址赋给 bmBits 字段之后，立即调用 CreateBitmapIndirect 是可以的，但在实模式下，如果 Windows 移动程序局部数据段，这个字段将变得无效。

为了避免这个问题，程序员可能会选择下面的方法：

```
hBitmap = CreateBitmapIndirect (&bitmap);
SetBitmapBits(hBitmap,(CDWORD) sizeof byBits,byBits);
```

也可以完全避免使用位图结构，用一条语句来创建位图：

```
hBitmap = CreateBitmap(20,5,1,1,byBits);
```

16.3.3 彩色位图格式

旧式风格的彩色位图稍微复杂一些，它非常依赖于设备。彩色位图的组织使得将位传输到特定的输出设备很容易。位图是组织为一系列颜色平面还是组织为每像素点含多个颜色位取决于位图所相配的设备。

我们先来看一看一个 bmBitsPixel 值为 1(这意味着每像素点具有 1 个颜色位)、bmPlanes 值大于 1 的位图。EGA 或 VGA 的彩色位图就是这样。Windows 使用 EGA 或 VGA 的四种颜色平面，显示 16 种颜色，所以 bmPlanes 为 4。位数组开始于最上面的扫描行。每个扫描行的颜色平面是顺序存储的——先存放红色平面、绿色平面、蓝色平面，最后是密度平面。然后，接着是位图的第二个扫描行。

位图也可以用每像素点多个位来表示颜色。假设一种设备(如 IBM 8514/A)可以用每像素点 8 个颜色位(1 字节)来表示 256 种颜色。对于每个扫描行，一个字节代表紧左像素点的颜色，第二字节表示下一像素点的颜色，以此类推。BITMAP 结构中的 bmWidthBytes 值反映了每个扫描行填充后的字节宽度，但 bmWidth 值仍是每个扫描行的像素点数。

问题在于，位图中没有任何信息指定这多个颜色平面或多个颜色位是如何对应于实际的显示颜色的。特定的彩色位图仅适合于显示内存组织类似于位图的输出设备。例如，假设我们有一个设备，它用每像素点 8 位来存储颜色信息，但是该设备解释 256 个值的方式与 8514/A 不一样。这是完全合法的，但是为 8514/A 创建的位图在其他设备上将产生不正确的颜色。

因此，用 CreateBitmap 或 CreateBitmapIndirect 创建初始化位图不是我们所推荐的方法。只应该用这两个函数创建初始化的或者不初始化的单色位图。

要创建彩色位图，应使用 CreateCompatibleBitmap，它能保证格式与真实图形输出设备兼容。通过将位图选进内存设备环境并在该设备环境上画出位图，或者通过使用 BitBlt 函数，就可将彩色位图变成图像。

要创建不必与真实图形输出设备兼容的彩色位图，可使用 DIB。

16.3.4 位图的尺寸

另两个与位图有联系的函数是一些造成混乱的来源。这两个函数是：

```
SetBitmapDimension (hBitmap,xDimension,yDimension);
```

和

```
dwDimension = GetBitmapDimension(hBitmap);
```

xDimension 和 yDimension 的值(分别是从 GetBitmapDimension 返回的 dwDimension 值的低位字和高位字)是位图的宽度和高度,单位为 0.1mm,对应于 MM_LOMETRIC 映射方式下的逻辑单位。GDI 不使用这些尺寸。它们既不是 BITMAP 结构的一部分,也不是位图文件格式的一部分。当然,两个互相协作的应用程序可以用这些尺寸,它们对位图的缩放有所帮助,位图可以通过剪贴板或以其他方法在应用程序之间交换。

16.4 内存设备环境

SetDIBitsToDevice 和 StretchDIBits 这两个函数允许程序员在输出设备上显示位数组。但是,如果程序员有位图句柄,却没有函数将该位图画在设备环境的显示表面上。程序员可能会徒劳地搜索如下所示的函数:

```
DrawBitmap (hdc,hBitmap,xStart,yStart);                    //无此函数!!!
```

该函数能将位图拷贝到 hdc 所代表的设备环境上,从逻辑点(xStart,yStart)开始。我们将编写自己的 DrawBitmap 函数。不过,读者首先必须熟悉几个概念,先从内存设备环境开始。

内存设备环境是具有仅存在于内存中的"显示表面"的设备环境。可以用下面的函数

```
hdcMem = CreateCompatibleDC(hdc);
```

创建一个内存设备环境。hdc 句柄是一个已存在的有效设备环境的句柄。该函数返回一个内存设备环境句柄。在创建内存设备环境后,所有属性均被设置为普通的缺省值。对于内存设备环境,程序员几乎可以做想做的一切事情,包括将属性设置为非缺省值、获取属性的当前设置以及选择画笔、刷子和区域进入其中,甚至可以在它上面绘图。但是对这一切它不会立即产生反应,下面是其原因。

程序员刚开始创建内存设备环境后,它具有一个刚好包含一个单色像素点的"显示表面"。这个显示表面很小。不要用 GetDeviceCaps 来获取这些信息。hdcMem 的 HORZSIZE、VERTSIZE、HORZERES、VERTRES、BITSPIXEL 和 PLANES 值都将被设置为与原 hdc 相关的值。如果 GetDeviceCaps 真的返回与内存设备环境相关的正确值(它创建时的值),则 HORZRES、VERTRES、BITSPIXEL 和 PLANES 都将返回 1。程序员需要将内存设备环境的显示表面变大些。通过将一个位图选进该设备环境中,就可以做到这一点。

```
SelectObject(hdcMem,hBitmap);
```

现在,hdcMem 的显示表面与 hBitmap 所代表的位图具有同样的宽度、高度和颜色结构。在缺省的窗口和视口原点下,内存设备环境的逻辑点(0,0)对应于位图的左上角。

如果位图上有某种图形,则该图形现在是内存设备环境显示表面的一部分。对该位图所做的任意改变(例如,通过使用 SetBitmapBits 给位图设置另一个位数组)都反映在这个显示表面中。简言之,位图就是内存设备环境的显示表面。

前面我们讨论过不同的创建位图的函数,其中一个是:

```
hBitmap = CreateCompatibleBitmap(hdc,xWidth,yHeight);
```

如果 hdc 是屏幕或打印机的普通设备环境句柄,则位图的颜色平面数目和像素点位数与设备的这些信息相同。但是,如果 hdc 是内存设备环境的句柄(而且还没有位图被选进内存设备环境),则 CreateCompatibleBitmap 返回一个单色位图,其宽和高分别为 xWidth 个像素

点和 yHeight 个像素点。

位图是六种 GDI 对象之一。不能用 SelectObject 将画笔、刷子或区域和字模这四个 GDI 对象选进设备环境，只能将其选进内存设备环境。

如果不再需要使用内存设备环境，必须删除之：

```
DeleteDC(hdcMem);
```

现在，读者可能会说："这一切都很好啊！"但是，我们还没有解决将位图输出到显示器上的问题。已做的只是将它选进了内存设备环境。现在要做什么呢？我们必须学习将位从一个设备环境"blt"（拼读为"blit"）到另一个设备环境中。

16.5 强大的 BLT

计算机图形学涉及到将像素点输出到显示设备上。论及强有力的像素点操作，Windows 中没有赶得上 BitBlt 及其"近亲"PatBlt 和 StretchBlt 的。BitBlt（拼为"bitblit"）代表"bit-block transfer"（位块传输）。BitBlt 是一个像素点移动程序，或称光栅擦除程序。"传输"并不能完全说明 BitBlt。它不仅仅做传输工作，实际上它用 256 种不同类型的光栅操作之一对三组像素点执行逻辑组合。

16.5.1 PatBlt 函数

PatBlt（"pattern block transfer"，模式块传输）是三种"blt"函数中最简单的。它实际上与 BitBlt 和 StretchBlt 非常不同，因为它只使用一种设备环境。但是从 PatBlt 开始，它仍然是合适的。

绘图方式的设备环境属性可以设置为 16 种二进制光栅运算（ROP2）码之一。在画线时，绘图方式确定 Windows 对画笔像素点和设备环境目标像素点所执行的逻辑操作类型。PatBlt 类似于绘图方式，不过它改变设备环境目标的矩形区域，而不只是线。它对这个矩形内的像素点与"模式"执行逻辑操作。"模式"不是什么新东西，它只是刷子的另一个名字。对于这个模式，PatBlt 使用当前选进设备环境的刷子。

PatBlt 的语法为：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeight,dwRop);
```

xDest、yDest、xWidth 和 yHeight 参数使用逻辑单位。逻辑点（xDest，yDest）指定矩形的左上角。矩形宽为 xWidth 个单位，高为 yHeight 个单位。下一节"Blt 坐标"对这些值有更精确的定义。这是 PatBlt 所改变的矩形区域。PatBlt 对刷子和目标设备环境执行的逻辑操作由 dwROP 参数确定，它是双字（32 位整数）ROP 码，不是绘图方式所用的 ROP2 码之一。

Windows 有 256 种 ROP 码。它们定义了源显示区域、目标显示区域与模式（即刷子）间所有可能的逻辑组合。通过使用排序（sort）"编译器"，视频显示器的设备驱动程序支持所有 256 种光栅操作。这个编译器用 32 位 ROP 码在堆栈上创建一系列机器语言指令，这套指令能完成对显示器像素点的逻辑操作；然后执行这些指令。32 位 ROP 码的高位字是 0 到 255 之间的数。低位字是一个数，它帮助设备驱动程序"编译"构成逻辑操作的机器码。256 种 ROP 码中有 15 种有名字。

由于 PatBlt 函数只使用一个目标设备环境和一个模式（不使用源设备环境），所以它只

是这些256个ROP码的子集，也就是只使用目标设备环境和模式的16个ROP码。PatBlt支持的16种光栅操作如下表所示。

模式(P)：	1 1 0 0	布尔	ROP
目的地(D)：	1 0 1 0	操作	代码/名称
结果	0 0 0 0	0	000042/BLACKNESS
	0 0 0 1	～(P | D)	0500A9
	0 0 1 0	～P & D	0A0329
	0 0 1 1	～P	0F0001
	0 1 0 0	P & ～D	500325
	0 1 0 1	～D	550009/DSTINVERT
	0 1 1 0	P ^ D	5A0049/PATINVERT
	0 1 1 1	～(P & D)	5F00E9
	1 0 0 0	P & D	A000C9
	1 0 0 1	～(P ^ D)	A50065
	1 0 1 0	D	AA0029
	1 0 1 1	～P | D	AF0229
	1 1 0 0	P	F00021/PAATCOPY
	1 1 0 1	P | ～D	F50225
	1 1 1 0	P | D	FA0089
	1 1 1 1	1	FF0062/WHITENESS

对于单色设备环境，位为1对应于白色像素点，位为0对应于黑色像素点。在开始考虑PatBlt时，目标和模式为纯黑或纯白是最容易考虑的。例如，如果调用：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeight,0x5F00E9L);
```

则仅当目标点为空，且将WHITE_BRUSH选进设备环境时，从逻辑点(xDest,yDest)开始、宽为xWidth个像素点、高为yHeight个像素点的矩形区域将被着色为黑。否则，目标将着色为白。当然，即使在单色设备环境中，目标和刷子也都可以是黑色和白色像素点的振动组合。在这种情况下，Windows以像素点为单位执行逻辑组合，这可能产生奇怪的结果。例如，如果目标已经用GRAY_BRUSH着色，且GRAY_BRUSH也是当前选进设备环境的刷子，则：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeight,PATINVERT);
```

将设置目标为纯白或纯黑，究竟哪种颜色则取决于目标的振动像素点与刷子的振动像素点如何相组合。

彩色引入了许多复杂性。Windows根据设备内存的组织方式分别对每个颜色平面或分别对每个颜色位集执行逻辑操作。

PatBlt的一些较通常的用法如下。如果想画一个黑色矩形，调用：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeight,BLACKNESS);
```

要画白色矩形，则用：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeight,WHITENESS);
```

函数：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeiht,DSTINVERT);
```

总将矩形的颜色翻转。如果 SHITE_BRUSH 被选进当前设备环境，则函数：

```
PatBlt(hdc,xDest,yDest,xWidth,yHeight,PATINVERT);
```

也翻转成矩形。

应该还记得，FillRect 函数用刷子填充矩形区域：

```
FillRect(hdc,&rect,hBrush);
```

FillRect 函数与下面的程序段等价：

```
hBrush = SelectObject (hdc,hBursh);
PatBlt (hdc,rect,left,rect. top,
        rect. right - rect. left,
        rect. bottom - rect. top,PATCOPY);
SelectObject (hdc,hBrush);
```

事实上，Windows 就是用这段程序来执行 FillRect 函数的。在调用

```
InvertRect(hdc,&rect);
```

时，Windows 将它转换成如下函数：

```
PatBlt(hdc,rect. left,rect. top,
    rect. right - rect. left,
    rect. bottom - rect. top,DSTINVERT);
```

16.5.2 Blt 坐标

在介绍 PatBlt 函数的语法时，我们说过，点(xDest,yDest)指定矩形的左上角，矩形宽为 xWidt 个单位，高为 yHeight 个单位。实际上，尽管 Windows 文档也是这样说的，但这种陈述并不精确。在深入介绍之前，我们需要澄清关于 blt 函数和坐标的一些含混概念。

BitBlt、PatBlt 和 StretchBlt 是 GDI 绘图函数中仅有的三个指定离顶点的逻辑宽度和逻辑高度的矩形函数。所有其他使用矩形边界框的 GDI 函数都要求坐标以左上角和右下角为准指定。对于 MM_TEXT 映像方式，上面对 PatBlt 参数的描述是精确的。对于度量映像方式，却并非如此。如果 xWidth 和 yHeight 使用负值，则点(xDest,yDest)将是矩形的左下角。如果想让(xDest,yDest)作为矩形的左上角，yHeight 参数必须设置为矩形高度的负值。

为了更精确，PatBlt 所着色的矩形具有一个由 xWidth 的绝对值给出的逻辑宽度和由 yHeight 的绝对值给出的逻辑高度。这两个参数均可为负值。矩形是由逻辑点(xDest,yDest)和(xDest+xWidth,yDest+yHeight)为顶点定义的。矩形的左上角总是包含在 PatBlt 要修饰的区域内。右下角在矩形外面。根据映像方式和 xWidth 及 yHeight 参数的符号，矩形左上角的坐标有四种可能：

```
(xDest,yDest)
(xDest,yDest + yHeight)
(xDest + xWidth,yDest)
(xDest + xWidth,yDest + yHeight)
```

如果将映像方式设置为 MM_LOENGLISH，并想对客户区域左上角面积为一平方寸的地方使用 PatBlt，可以使用下列之一：

```
PatBlt(hdc,0,0,100,-100,dwROP);
PatBlt(hdc,0,-100,100,dwROP);
```

```
PatBlt(hdc,100,0,-100,-100,dwROP);
PatBlt(hdc,100,-100,-100,100,dwROP);
```

要将 PatBit 的参数设置正确，最简单的方法是将 xDest 和 yDest 设置为矩形的左上角。如果映像方式将 y 轴增长方向定义为向上，则 yHeight 参数的值要使用负数。如果映射方式将 x 坐标值增长方向定义为向左(这样做还闻所未闻)，则 xWidth 参数应使用负数值。

16.5.3 用 BitBlt 传输位

从某种程度上，BitBlt 是 PatBlt 的超集。PatBlt 能完成的工作它都能完成，它在逻辑运算中引入第二个设备环境。下面是一般的语法：

```
BitBlt (hdcDest,xDest,yDest,xWidth,yHeight,
    hdcSrc,xSrc,ySrc,dwROP);
```

BitBlt 调用修改由逻辑点(xDest,yDest)和以逻辑单位给出的 xWidth 和 yHeight 参数所定义的矩形内的目标设备环境(其句柄为 hdcDest)。这些参数定义了一个上一节所描述的矩形。BitBlt 也使用一个源设备环境(其句柄为 SrcDC)中的矩形。该矩形开始于逻辑点(xSrc，ySrc)，宽为 xWidth 个逻辑单位，高为 yHeight 个逻辑单位。

BitBlt 对三个元素执行逻辑组合，这三个元素为：选进目标设备环境中的刷子、源设备环境矩形中的像素点和目标设备环境矩形中的像素点。结果写到目标设备环境矩形中。可以使用 256 个 ROP 码的任何一个作 BitBlt 的 dwROP 参数。

看看显示逻辑组合结构的 8 个 0 和 1。对应于这些位的两位数字的十六进制数是 ROP 码的高位字。如果我们可以创建一个想从模式、源和目标那里得到的结果表，那么我们就能从 ROP 码表中很容易地确定 ROP 码。后面我们会这样做。如果使用“PatBlt 函数”那一节的表中 16 个 ROP 码之一，则可以用 PatBlt 代替 BitBlt，因为没有引用源设备环境。

程序员可以将 hdcSrc 和 hdcDest 设置为相同的设备环境句柄，在这种情况下，BitBlt 将对目标矩形、源矩形和选进设备环境的当前刷子执行逻辑组合。但是，在客户区域设备环境中，这样做是有一点冒险的：Windows 将使用这个窗口的像素点为源 。Windows 不知道客户区域中另一个窗口下面是什么。

不过，源和目标使用相同设备环境的 BitBlt 函数例子很容易找到。函数：

```
BitBlt(hdc,100,0,50,100,hdc,0,0,SRCCOPY);
```

将从逻辑点(0,0)开始且宽度为 50 个逻辑单位、高度为 100 个逻辑单位的矩形拷贝到从逻辑点(100,0)开始的矩形区域中。

16.5.4 使用不同的 ROP 码

SRCCOPY 确实是 BitBlt 最常用的 dwROP 参数。要发现其他 255 种 ROP 码的使用是很困难的。所以，本节给出了两个例子，这两个例子显示了其他 ROP 码的特征。

在第一个例子中，我们有一个单色位图，想将它传输到屏幕上。但是，我们不想因显示位图而使黑色(0)位影响当前存在于客户区域上的显示。而且，我们想要所有白色(1)位用刷子着色客户区域，刷子也许是用 CreateSolidBrush 创建的彩色刷子。那么，怎样做呢？

假定工作于 MM_TEXT 映像方式，想要从客户区域中点(xStart,yStart)开始输出位图。我们已经有了一个单色位图句柄(hBitmap)和彩色刷子句柄(hBrush)。我们知道位图的宽度

和高度，并将它们存放在一个 BITMAP 结构 bm 中。下面就是这些代码：

```
hdcMem = CreateCompatibleDC (hdc);
SelectObject (hdcMem,hBitmap);
hBrush = SelectObject (hdc,hBrush);

BitBlt (hdc,xStart,yStart,bm. bmbmWidth,bm. bmHeight,
        hdcMem,0,0,0xE20746L);

SelectObject (hdc,hBrush);
DeleteDC (hdcMem);
```

BitBlt 对目标设备环境(hdc)、源设备环境(hdcMem)和当前选进目标设备环境的刷子执行逻辑组合。所以我们创建一个内存设备环境，将位图选进内存设备环境，将彩色刷子选进客户区域显示环境，然后调用 BitBlt。然后，我们将原刷子选进显示设备环境，并删除内存设备环境。

这段代码唯一令人迷惑的地方是 ROP 代码 0xE20746。这个 ROP 码导致 Windows 执行以下逻辑操作：

((Destination ^ Pattern) ε Source) ^ Destination

还不明显吗？试试这种方法：拷贝“用 BitBlt 传输位”一节中表的这一部分。

Pattern:	1	1	1	1	0	0	0	0
Source:	1	1	0	0	1	1	0	0
Destination:	1	0	1	0	1	0	1	0
Result:	?	?	?	?	?	?	?	?

图 16.2

对于位图(将被选进源内存设备环境)中的每一个黑色位，我们不想改变目标设备环境。这意味着凡是源为 0 的地方，我们就希望结果位与目标相同。

Pattern:	1	1	1	1	0	0	0	0
Source:	1	1	0	0	1	1	0	0
Destination:	1	0	1	0	1	0	1	0
Result:	?	?	1	0	?	?	1	0

图 16.3

现在完成一半了。对于位图中每个白色位，我们想用模式来着色目标设备环境。选进目标设备环境的刷子就是这个模式。所以凡是源为 1 的地方，我们都希望结果等于模式。

Pattern:	1	1	1	1	0	0	0	0
Source:	1	1	0	0	1	1	0	0
Destination:	1	0	1	0	1	0	1	0
Result:	1	1	1	0	0	0	1	0

图 16.4

这意味着 ROP 码的高位字为 0xE2。我们可以在《程序员参考》的 ROP 表中查到它的完

整 ROP 码为 0xE20746。

也许现在读者会发现，自己在用 SDKPAAINT 创建位图时，混用了白色位和黑色位。这很容易修正。这只是另一种逻辑操作。

Pattern:	1	1	1	1	0	0	0	0
Source:	1	1	0	0	1	1	0	0
Destination:	1	0	1	0	1	0	1	0
Result:	1	0	1	1	1	0	0	0

图 16.5

现在，ROP 码的高位字为 0xB8，整个 ROP 码为 0xB8074A，它执行以下逻辑操作：

((Destination ^ Pattern) ε Source) ^ Pattern

下面是第二个例子。有两个组成图标和光标的位图，这两个位图的使用允许这些图形从某种程度上讲是透明的，或者翻转下面的显示表面。对于单色图标或光标，这两个位图编码如下：

Bitmap 1:	0	0	1	1
Bitmap 2:	0	1	0	1
Result:	Black	White	Screen	Inverse Screen

图 16.6

Windows 将位图 1 选进内存设备环境，并用 BitBlt(带一个 ROP 码 SRCAND)来将位图传输到显示器上。这个 ROP 码执行如下逻辑操作：

Destination & Source

在位图 1 中，位为 1 则目标不改变，位为 0 则设置为 0。然后 Windows 将位图 2 选进设备环境，并以 SRCINVERT 为参数使用 BitBlt。逻辑操作为：

Destination ^ Source

在位图 2 中，对于所有为 0 的位，目标不改变；对于所有为 1 的位，翻转目标。

看看表的第一和第二列。使用 SRCAND 的位图 2 使所有位变黑，使用 SRCINVERT 的位图 2 通过反转黑色位，将选择位变为白色。这些操作设置组成图标或光标的黑色位和白色位。现在看看表的第三和第四列。使用 SRCAND 的位图 2 不改变显示，使用 SRCINVERT 的位图翻转选择位的颜色。这些操作使图标或光标透明，或者翻转下面的颜色。

16.5.5 再论内存设备环境

我们已经用内存设备环境将存在的位图传输到显示器上。也能用内存设备环境在位图的表面上绘图。首先，创建一个内存设备环境：

```
hdcMem = CreateCompatibleDC(hdc);
```

然后，创建期望尺寸的位图。如果想创建单色位图，可以使之与 hdcMem 兼容：

```
hBitmap = CreateCompatibleBitmap(hdcMem,xWidth,yHeight);
```

如果要使位图与视频显示器具有相同的颜色结构，可以使位图与 hdc 兼容：

```
hBitmap = CreateCompatibleBitmap(hdc,xWidth,yHeight);
```

现在，可以将位图选进内存设备环境：

```
SelectObject(hdcMem,hBitmap);
```

现在，可以用 GDI 函数以及设备环境来画图。一开始创建位图时，它包含随机位，所以程序员可能想用 PatBlt、一个 WHITENESS 或 BLACKNESS ROP 码，以删除内存设备环境的背景。

完成在内存设备环境上画图之后，需要删除它：

```
DeleteDC(hdcMem);
```

现在，程序员具有一个位图，它包含在被选进内存设备环境时所画的一切东西。

下面显示的 SCRAMBLE 程序用一个内存设备环境作为临时存放 BitBlt 操作的空间，BitBlt 操作交换显示器上两个矩形的内容。

```
# SCRAMBLE.MAK make file

scramble.exe : scramble.obj scramble.def
     $(WINLINK) scramble,scramble,NUL,$(WINLIB),scramble
     rc -t scramble.exe

scramble.obj : scramble.c
     $(WINCC) scramble.c

/*------------------------------------------------
SCRAMBLE.C -- Scramble (and Unscramble) Screen
------------------------------------------------*/

#include <windows.h>
#include <stdlib.h>

#define NUM 200

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
     static short nKeep [NUM][4] ;
     HDC hdc = CreateDC ("DISPLAY",NULL,NULL,NULL) ;
     HDC hdcMem = CreateCompatibleDC (hdc) ;
     short cxSize = GetSystemMetrics (SM_CXSCREEN) / 10 ;
     short cySize = GetSystemMetrics (SM_CYSCREEN) / 10 ;
     HBITMAP hBitmap = CreateCompatibleBitmap (hdc,cxSize,cySize) ;
     short i,j,x1,y1,x2,y2 ;

     SelectObject (hdcMem,hBitmap) ;

     srand (LOWORD (GetCurrentTime ())) ;

     for (i = 0 ; i < 2 ; i++)
          for (j = 0 ; j < NUM ; j++)
          {
```

```
            if (i == 0)
            {
                nKeep [j] [0] = x1 = cxSize * (rand () % 10) ;
                nKeep [j] [1] = y1 = cySize * (rand () % 10) ;
                nKeep [j] [2] = x2 = cxSize * (rand () % 10) ;
                nKeep [j] [3] = y2 = cySize * (rand () % 10) ;
            }
            else
            {
                x1 = nKeep [NUM - 1 - j] [0] ;
                y1 = nKeep [NUM - 1 - j] [1] ;
                x2 = nKeep [NUM - 1 - j] [2] ;
                y2 = nKeep [NUM - 1 - j] [3] ;
            }
            BitBlt (hdcMem,0,0,cxSize,cySize,hdc, x1,y1,SRCCOPY) ;
            BitBlt (hdc, x1,y1,cxSize,cySize,hdc, x2,y2,SRCCOPY) ;
            BitBlt (hdc, x2,y2,cxSize,cySize,hdcMem,0,0,SRCCOPY) ;
        }

    DeleteDC (hdcMem) ;
    DeleteDC (hdc) ;
    DeleteObject (hBitmap) ;

    return FALSE ;
}

; SCRAMBLE. DEF module definition file

NAME SCRAMBLE

DESCRIPTION 'Screen Scrambler (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

SCRAMBLE 不具有窗口过程。在 WinMain 中，它获取整个屏幕的设备环境

```
hdc = CreateDC("DISPLAY",NULL,NULL,NULL);
```

和一个内存设备环境

```
hdcMem = CreateCompatibleDC(hdc);
```

然后，确定整个屏幕的尺寸，并除以 10：

```
xSize = GetSystemMetrics(SM_CXSRCEEN) / 10;
ySize = GetSystemMetrics(SM_CYSCREEN) / 10;
```

程序用这些尺寸创建一个位图：

```
hBitmap = CreateConpatibleBitmap(hdc,xSize,ySize);
```

并将它选进内存设备环境中：

```
SelectObject(hdcMem,hBitmap);
```

SCRAMBLE 使用普通的 C rand("随机数生成程序")函数产生 4 个是 xSize 和 ySize 的整数倍的随机数：

```
x1 = xSize * (rand() % 10);
y = xSize * (rand() % 10);
x2= xSize * (rand() % 10);
x2= xSize * (rand() % 10);
```

程序使用三个 BitBlt 函数交换显示区域的两个矩形块。

第一个 BitBlt 将从点(x1,y1)开始的矩形拷贝到内存设备环境中：

```
BitBlt(hdcMem,0,0,xSize,ySize,hdc,x1,y1,SRCCOPY);
```

第二个 BitBlt 将从点(x2,y2)开始的矩形拷贝到从点(x1,y1)开始的位置：

```
BitBlt(hdc,x1,y1,xSize,ySize,hdc,x2,y2,SRCCOPY);
```

第三个 BitBlt 将内存设备环境中的矩形拷贝到从点(x2,y2)开始的区域：

```
BitBlt(hdc,x2,y2,xSize,ySize,hdcMem,0,0,SRCCOPY);
```

这个过程很有效地交换了显示器上两个矩形区域的内容。SCRAMPLE 这样做了 200 次，在此之后屏幕将完全混乱。不过，不用怕，因为 SCRAMPLE 保存了这些混乱的过程，在退出前会使屏幕回到原样。

也可以用内存设备环境将一个位图的内容拷贝给另一个。例如，假设程序员想创建一个位图，该位图包含另一个图的左上部四分之一。如果源位图具有句柄 hBitmap，程序员可以将尺度拷贝到一个 BITMAP 类型的结构中：

```
GetObject(hBitmap,sizeof(BITMAP),(LPSTR) &bm);
```

并创建一个新的四分之一大小的未初始化的位图：

```
hBitmap2 = CreateBitmap(bm.bmWidth / 2,bm.bmHeight / 2,
    bm.bmPlanes,bm.bmBitsPixel,NULL);
```

现在，创建两个内存设备环境，并分别将原位图和新创建的位图选进两个内存设备环境中：

```
hdcMem1 = CreateCompatibleDC (hdc);
hdcMem2 = CreateCompatibleDC (hdc);
SelectObject1 (hdcMem1,hBitmap);
SelectObject1 (hdcMem2,hBitmap);
```

最后，将头一个位图的左上部四分之一拷贝给第二个位图：

```
BitBlt(hdcMem2,0,0,bm.bmWidth / 2,bm.bmHeight / 2,
    hdcMem1,0,0,SRCCOPY);
```

此后将清除之：

```
DeleteDC(hdcMem1);
DeleteDC(hdcMem2);
DeleteDC(hBitmap);
```

16.5.6 颜色转换

如果在 BitBlt 调用中目标和源设备环境的颜色特性不同，Windows 必须将位图从一种颜

色格式转换成另一种。最好的颜色转换是当源位图为单色时发生的转换。Windows 用目标设备环境的文本颜色和背景颜色属性来进行转换：

单色 DC(源)	颜色 DC(目标)
0(黑)	文本色(缺省为黑)
1(白)	背景色(缺省为白)

背景颜色属性是 Windows 用来填充点式或长划式线之间的空隙以及影线刷子中影线之间区域的颜色。可以用 SetBkColor 来改变背景颜色。文本颜色确定了文本的色彩。可以用 SetTextColor 改变文本颜色。在缺省设置下，单色位图转换成彩色设备环境上的黑白位图。

将彩色设备环境中的源位图转换成单色设备环境中的目标位图不那么令人满意：

彩色 DC(源)	单色 DC(目标)
像素点 ！ = 背景色	0(黑)
像素点 == 背景色	1(白)

在这种情况下，Windows 用源设备环境的背景色来确定哪个颜色转换为白色。而所有其他颜色都转换为黑色。

下面是另一个与颜色有关的问题。Windows 需要位图中的特定颜色组合(在不同平面上的或同一平面上的)与背景色的 24 位颜色值相等。这意味着彩色设备环境必须代表一个真实设备或者基于真实设备的内存设备环境。例如，假设有一个单色设备驱动程序。我们在屏幕设备环境的基础上创建一个内存设备环境，并选择一个彩色位图到该内存设备环境中。现在，试着将该位图传输给一个单色设备环境。这是不能成功的，因为 Windows 不知道内存设备环境位图中每像素点的多个颜色平面或多个位与真实颜色间的关系。

16.5.7 映像方式转换

BitBlt 调用的源和目标设备环境需要不同的起始坐标，但是只需要一个宽度和一个高度：

```
BitBlt(hdcDest,xDest,yDest,xWidth,yHeight,
    hdcSrc,xSrc,ySrc,dwROP);
```

xWidth 和 yHeight 的值使用逻辑单位，它们既适用于源设备环境的矩形，又适用于目标设备环境的矩形。在调用驱动程序文件执行实际操作之前，BitBlt 必须将所有坐标和大小转换成设备坐标。由于源设备环境和目标设备环境都使用 xWidth 和 yHeight，它们必须分别转换为每个设备环境的设备坐标(像素点)。

如果源设备环境与目标设备环境相同，或者两个设备环境都使用 WM_TEXT 映像方式，则以设备单位计算的矩形尺寸在两个设备环境中将是相同的。然后，Windows 可以做简单的像素点对像素点传输。但是，如果以设备单位计算，矩形的尺寸在两个设备环境中不同，则 Windows 将工作交给更有用的 StretchBlt 函数。

16.5.8 用 StrechBlt 伸展位图

StrechBlt 调用与 BitBlt 调用相比，多了两个参数：

```
StrechBlt(hdcDest,xDest,yDest,xDestWidth,yDestHeight,
    hdcSrc,xSrc,ySrc,xSrcWidth,ySrcHeight,dwROP);
```

由于 StretchBlt 的源矩形和目标矩形接受不同的宽度和高度参数，所以它允许将源设备环境中的位图伸展或压缩，以适合目标设备环境中较大或较小的区域。

与 BitBlt 的功能是 PatBlt 的功能的超集一样，StrechBlt 与 BitBlt 有同样的关系。它允许用户分别指定源矩形和目标矩形的宽度和高度。与 PatBlt 和 BitBlt 一样，StretchBlt 中所有坐标值均使用逻辑单位。我们已经在两个程序中使用过 StrechBlt。

StretchBlt 还允许程序员在垂直或水平方向倒转图像。如果 xSrcWidth 和 xDestWidth 的符号(当它们转换成设备单位时)不同，则 StretchBlt 创建一个镜像，左变为右，右变为左。如果 ySrcHeight 和 yDestHeight 不同，则 StretchBlt 将图像颠倒。可以用 BLOWUP1 程序来证实这一点，令图像从右上角开始(宽为负)、从左下角开始(高为负)、右下角开始(宽、高均为负)。

如果读者使用过 BLOWUP1，很可能会发现 StretchBlt 很慢，特别是在对大位图操作时。StretchBlt 还有一些源自缩放位图的问题和困难。在扩展位图时，StretchBlt 必须重复像素点的行或列。但如果扩展不是整数倍，则该过程将导致图像的某种失真。

在压缩位图时，StretchBlt 必须将两个或更多个行或列像素点合并成一行或一列。这样做有三种方法，使用哪种取决于设备环境里的伸展方式或属性。可以用 SetStretchBltMode 函数改变这个属性：

```
SetStretchBltMode(hdc,nMode);
```

nMode 的值可以是下面三个值之一：

- BLACKONWHITE(缺省设置)——如果必须将两个或多个像素点组合成一个像素点，StretchBlt 对像素点执行逻辑“与”操作。仅当所有原像素点为白色时，结果像素点才为白色，这在实践中意味着黑色像素点比白色像素点优先。
- WHITEONBLACK——必须将两个或多个像素点执行逻辑“或”操作。仅当所有原像素点为黑色时，结果像素点才为黑色。这在实践中意味着白色像素点优先。
- COLORONCOLOR——StretchBlt 简单除去几行或几列像素点，不进行逻辑组合。这通常是对彩色位图的最好的方法，因为用其他两种方式能导致颜色失真。

16.5.9 动画

GDI 只支持静态图形。虽然 Windows 确实没有通常的动画支持(如反转视频页、查看视频信号的垂直踪迹或构造可旋转的画片的能力)，但这并不意味着我们不能在显示器上移动图像。现在，我们将给出一个弹球程序。下面所示的 BOUNCE 程序构造一个球，它在窗口的客户区域里弹来弹去。程序用定时器来令球弹动；它用内存设备环境中的简单“位块传输”画球。

```
# BOUNCE.MAK make file

bounce.exe : bounce.obj bounce.def
     $(WINLINK) bounce,bounce,NUL, $(WINLIB),bounce
     rc -t bounce.exe

bounce.obj : bounce.c
     $(WINCC) bounce.c

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

```
BOUNCE. C -- Bouncing Ball Program
---------------------------------------- */

#include <windows. h>

#define min(a,b) (((a) < (b)) ? (a) : (b))
#define max(a,b) (((a) > (b)) ? (a) : (b))

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName [] = "Bounce" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass. style = CS_HREDRAW | CS_VREDRAW ;
          wndclass. lpfnWndProc = WndProc ;
          wndclass. cbClsExtra = 0 ;
          wndclass. cbWndExtra = 0 ;
          wndclass. hInstance = hInstance ;
          wndclass. hIcon = NULL ;
          wndclass. hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass. hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass. lpszMenuName = NULL ;
          wndclass. lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
     }

     hwnd = CreateWindow (szAppName,"Bouncing Ball",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,CW_USEDEFAULT,
          CW_USEDEFAULT,CW_USEDEFAULT,
          NULL,NULL,hInstance,NULL) ;

     if (! SetTimer (hwnd,1,50,NULL))
     {
          MessageBox (hwnd,"Too many clocks or timers!",
               szAppName,MB_ICONEXCLAMATION | MB_OK) ;
          return FALSE ;
     }

     ShowWindow (hwnd,nCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg,NULL,0,0))
```

```
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg. wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
            LONG lParam)
{
    static HANDLE hBitmap ;
    static short cxClient,cyClient,xCenter,yCenter,cxTotal,cyTotal,
        cxRadius,cyRadius,cxMove,cyMove,xPixel,yPixel ;
    HBRUSH hBrush ;
    HDC hdc,hdcMem ;
    short nScale ;

    switch (message)
    {
        case WM_CREATE:
            hdc = GetDC (hwnd) ;
            xPixel = GetDeviceCaps (hdc,ASPECTX) ;
            yPixel = GetDeviceCaps (hdc,ASPECTY) ;
            ReleaseDC (hwnd,hdc) ;
            return 0 ;
        case WM_SIZE:
            xCenter = (cxClient = LOWORD (lParam)) / 2 ;
            yCenter = (cyClient = HIWORD (lParam)) / 2 ;

            nScale = min (cxClient * xPixel,cyClient * yPixel) / 16 ;

            cxRadius = nScale / xPixel ;
            cyRadius = nScale / yPixel ;

            cxMove = max (1,cxRadius / 4) ;
            cyMove = max (1,cyRadius / 4) ;

            cxTotal = 2 * (cxRadius + cxMove) ;
            cyTotal = 2 * (cyRadius + cyMove) ;

            if (hBitmap)
                DeleteObject (hBitmap) ;

            hdc = GetDC (hwnd) ;
            hdcMem = CreateCompatibleDC (hdc) ;
            hBitmap = CreateCompatibleBitmap (hdc,cxTotal,cyTotal) ;
            ReleaseDC (hwnd,hdc) ;

            SelectObject (hdcMem,hBitmap) ;
            Rectangle (hdcMem,-1,-1,cxTotal + 1,cyTotal + 1) ;
```

```
            hBrush = CreateHatchBrush (HS_DIAGCROSS,0L) ;
            SelectObject (hdcMem,hBrush) ;
            SetBkColor (hdcMem,RGB (255,0,255)) ;
            Ellipse (hdcMem,cxMove,cyMove,cxTotal - cxMove,
                cyTotal - cyMove) ;
            DeleteDC (hdcMem) ;
            DeleteObject (hBrush) ;
            return 0 ;
        case WM_TIMER:
            if (! hBitmap)
                break ;

            hdc = GetDC (hwnd) ;
            hdcMem = CreateCompatibleDC (hdc) ;
            SelectObject (hdcMem,hBitmap) ;

            BitBlt (hdc,xCenter - cxTotal / 2,
                    yCenter - cyTotal / 2,cxTotal,cyTotal,
                    hdcMem,0,0,SRCCOPY) ;

            ReleaseDC (hwnd,hdc) ;
            DeleteDC (hdcMem) ;

            xCenter += cxMove ;
          yCenter += cyMove ;

            if ((xCenter + cxRadius >= cxClient) ||
                    (xCenter - cxRadius <= 0))
                cxMove = -cxMove ;

            if ((yCenter + cyRadius >= cyClient) ||
                    (yCenter - cyRadius <= 0))
                cyMove = -cyMove ;
            return 0 ;
        case WM_DESTROY:
            if (hBitmap)
                DeleteObject (hBitmap) ;

            KillTimer (hwnd,1) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

; BOUNCE.DEF module definition file

NAME BOUNCE

DESCRIPTION 'Bouncing Ball Program (c) Charles Petzold,1992'
EXETYPE WINDOWS
```

```
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

一旦程序获取一个 WM_SIZE 消息，BOUNCE 就重新构造球。球的直径取客户区域的高和宽中较短者的 1/16。不过，程序创建的位图比球大，在位图的四边，它在球的尺度之外扩展球半径的 1/4。

在位图选进内存设备环境后，它将被着以白色：

```
Rectangle(hdcMem,-1,-1,xTotol + 1,yTotal + 1);
```

一个斜影线刷子被选进内存设备环境，球在位图的中央画出：

```
Ellipse(hdcMem,xMove,yMove,xTotal - xMove,yTotal - yMove);
```

在球移动时，球边缘的边界有效地删除了前一个球的图像。在另一个位置重画此球，只需使用 ROP 码 SRCCOPY 进行一切 BitBlt 调用：

```
BitBlt(hdc,xCenter - xTotal / 2,
    yCenter - yTotal / 2,
    xTotal,yTotal,hdcMem,0,0,SRCCOPY);
```

BOUNCE 演示了在显示器上移动图像的最简单的方法。但这种方法对于一般目的并不令人满意。如果程序员对动画有兴趣，可以找到其他一些 ROP 码(如 SRCINVERT)以对源和目标执行异或运算。

16.6 元文件

元文件是 GDI 函数的集合，以二进制形式编码。要创建元文件，首先要创建元文件设备环境。然后，就能用大多数 GDI 绘图函数在这个元文件设备环境上画图。不过，这些 GDI 调用并不会真的在什么东西上画图。相反，它们被存储在元文件内。在关闭元文件设备环境之后，就获取一个元文件句柄。然后，就可以在真实设备环境上"运行"这个元文件，并执行元文件中的 GDI 调用。

元文件大多用于程序间通过剪贴板共享图片。因为元文件将图片描述为 GDI 函数调用的集合，它们既比位图占用的空间少得多，又比位图更与设备无关。我们从几个简单例子开始对元文件进行讨论，然后再给出理论上的考虑。

16.6.1 内存元文件的简单应用

假设某公司的标志包含一个矩形，画有两条对角线，中间有一个蓝色的图形。我们需要经常在程序窗口的客户区域或在打印机上画出这个标志。让我们将标志制作为元文件。

先定义几个必需的变量：

```
static HANDLE hmf;
HANDLE hBrush;
HDC hdcMeta;
```

在处理 WM_CREATE 消息时，就能创建元文件。我们调用 CreateMetaFile，以获取一个元

文件设备环境的句柄：

```
hMetaDC = CreateMetaFile(NULL);
```

NULL 参数指示这将是一个“内存”元文件；也就是说，元文件将存储在内存中，而不是存放在磁盘文件里。

现在，我们可以在这个元文件设备环境上画我们的标志了。假设我们决定其宽和高均为 100 个单位：

```
Rectangle (hdcMeta,0,0,100,100)
MoveTo (hdcMeta,0,0);
LineTo (hdcMeta,100,100);
MoveTo (hdcMeta,0,100);
LineTo (hdcMeta,100,0);

hBrush = CreateSolidBrush (RGB (0,0,255));
SelectObject (hdcMeta,hBrush);
Ellipse (hdcMeta,20,20,80,80);
```

在画图完成后，我们调用 CloseMetaFile 关闭元文件设备环境。这个调用返回元文件的句柄：

```
hmf = CloseMetaFile(hdcMeta);
```

现在，我们可以删除所创建的刷子：

```
DeleteObject(hBrush);
```

创建元文件就完成了。hmf 变量被定义为静态的，所以在处理其他消息时，它仍然存在。

我们画标志所用的宽和高均为 100。它们是逻辑单位还是设备单位呢？在这种情况下，它们既不是逻辑单位又不是设备单位，只是单位。只有当我们运行文件时，它们才有意义。现在，我们来运行之。在处理 WM_PAINT 消息时，我们想用 100 个标志来填充客户区域，并不关心它们怎样伸展。我们假设已经获得了代表客户区域宽度和高度的 cxClient 和 cyClient 的值，并且已经定义了这里要用到的一些变量。

我们获取客户区域设备环境的句柄，并设置一个水平和垂直方向均为 1000 个逻辑单位的 MM_ANISOTROPIC 映像方式：

```
hdc = BeginPaint (hwnd,&ps);
SetMapMode (hdc,MM_ANISOTROPIC);
SetWindowExt (hdc,1000,1000);
SetViewportExt (hdc,cxClient,cyClient);
```

现在可以调用 PlayMetaFile，将元文件运行 100 次，每次都改变窗口的原点，以将元文件移到新位置：

```
for (x = 0; x < 10; x++)
    for (y = 0; y < 10; y++);
    {
        SetWindowOrg (hdc,100 * x,-100 * y);
        PlayMetaFile (hdc,hmf);
    }
```

调用 PlayMetaFile，实际上相当于重复我们在处理 WM_CREATE 消息期间原来创建元文件时在 CreateMetaFile 与 CloseMetaFile 之间所做的一切调用。

在完成这个处理后，我们用通常的方法结束 WM_PAIT 消息：

```
EndPaint(hwnd,&ps);
```

还有一个任务。在创建元文件后，句柄就是 GDI 模块的特性。程序员在退出程序之前，必须先用 DeleteMetaFile 显式地删除它。可以在处理 WM_DESTROY 消息时删除它：

```
case WM_DESTROY:
    DeleteMetaFile (hmf);
    PostQuitMessage (0);
    return 0;
```

16.6.2 元文件存盘

在上面的例子中，CreateMetaFile 的 NULL 参数意味着想创建存储在内存里的元文件。我们也可以创建与普通文件一样存放在磁盘上的元文件。这种方法对于大的元文件比较合适，因为这样做用的内存空间小。Windows 必须在内存中保持一个相对较小的区域，以存放包含元文件的文件名。另一方面，将元文件存放在磁盘上需要在再次运行元文件时进行磁盘访问。

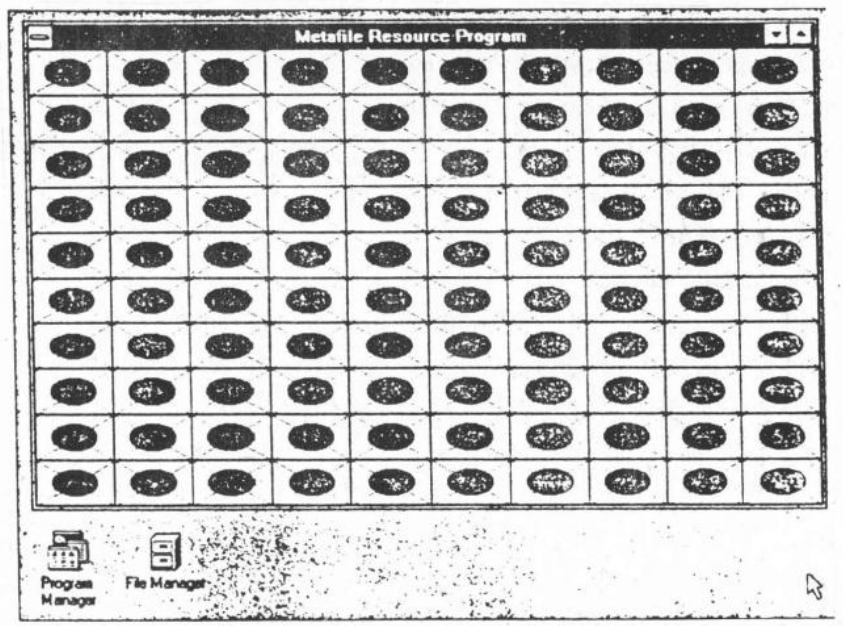


图 16.7 用 PlayMetaFile 函数画的多个图形

我们再次使用公司标志的例子。除了上面已给出的变量外，我们还需要一个存放元文件名的变量：

```
static char szFileName [80];
```

在这个例子中，我们使用一个临时文件。在处理 WM_CREATE 消息期间，我们可以用 Windows 的 GetTempFileName 函数为临时文件创建一个文件名：

```
GetTempFileName(0,MF,0,szFileName);
```

Windows 先检查 WS_DOS 环境的 TEMP 变量，以便为该文件选择一个磁盘和子目录。如果在 MS_DOS 环境中没有 TEMP 变量，则 Windows 使用第一个硬盘的根目录。文件名以否定号(～)开始，后跟我们在 GetTempFileName 函数中指定的字符(MF)，然后是一个独特

的数；扩展名为 .TMP。在从调用返回后，文件名就存放在 szFileName 数组里。

我们用文件名来创建元文件设备环境：

```
hMetaDC = CreateMetaFile(szFileName);
```

我们可以与在原来的例子中一样在设备环境上画图，然后关闭元文件设备环境，以获取元文件句柄：

```
hmf = CloseMetaFile(hdcMeta);
```

WM_PAINT 消息的处理与原来的例子一样。不过，在处理 WM_DESTROY 消息时，必须加点东西。下面的语句：

```
DeleteMetaFile(hmf);
```

删除元文件句柄，它指向磁盘文件的内存区域，但磁盘文件仍然存在。必须用普通的 C 函数删除这个文件：

```
unlink(szFileName);
```

下面是使用基于磁盘的元文件的另一种方法。这种方法不需要将 hmf 保存为静态变量。首先，我们获取一个临时文件名，并跟以前一样创建元文件设备环境：

```
GetTempFileName(0,MF,szFileName);
hdcMeta = CreateMetaFile(szFileName);
```

现在，在元文件设备环境上画图。完成之后，关闭元文件设备环境，并获取元文件句柄：

```
hmf = CloseMetaFile(hdcMeta);
```

但是现在我们删除元文件：

```
DeleteMetaFile(hmf);
```

我们难道真的想这样做？也许的确如此。删除基于磁盘的元文件使得元文件句柄失效，释放了元文件需要的内存，而磁盘文件原封不动。在处理 WM_PAINT 消息期间，我们可以通过调用 GetMetaFile 获取这个磁盘文件的元文件句柄：

```
hmf = GetMetaFile(szFileName);
```

现在，可以跟前面一样运行元文件。在处理 WM_PAINT 消息结束时，可以删除元文件句柄：

```
DeleteMetaFile(hmf);
```

在处理 WM_DESTROY 消息时，不一定要删除元文件，因为已经在 WM_CREATE 消息和每次 WM_PAINT 消息的结束删除了它。但是，仍然要删除磁盘文件：

```
unlink(szFileName);
```

16.6.3 现存元文件的使用

我们在上面的例子中所做的似乎隐含着，我们能在一个程序里创建基于磁盘的元文件，并在另一个程序里通过调用 GetMetaFile 使用它。的确可以。图中所示的 MFCREATE("metafilecreate"，元文件创建)程序是本书最短的 Windows 程序。它所做的只是创建一个名字为 MYLOGO.WMF 的基于磁盘的元文件。.WMF 扩展名代表“Windows metafile”(Windows 元文件)，它是以磁盘文件存放的元文件的常用扩展名。

```
# MFCREATE.MAK make file
```

```
mfcreate.exe : mfcreate.obj mfcreate.def
    $(WINLINK) mfcreate,mfcreate,NUL, $(WINLIB),mfcreate
    rc -t mfcreate.exe

mfcreate.obj : mfcreate.c
    $(WINCC) mfcreate.c

/*----------------------------------------
MFCREATE.C -- Metafile Creation Program
----------------------------------------*/

#include <windows.h>

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
    HBRUSH hBrush = CreateSolidBrush (RGB (0,0,255)) ;
    HDC hdcMeta = CreateMetaFile ("MYLOGO.WMF") ;

    Rectangle (hdcMeta,0,0,100,100) ;
    MoveTo (hdcMeta,0,0) ;
    LineTo (hdcMeta,100,100) ;
    MoveTo (hdcMeta,0,100) ;
    LineTo (hdcMeta,100,0) ;
    SelectObject (hdcMeta,hBrush) ;
    Ellipse (hdcMeta,20,20,80,80) ;

    DeleteMetaFile (CloseMetaFile (hdcMeta)) ;
    DeleteObject (hBrush) ;

    MessageBeep (0) ;

    return FALSE ;
}

; MFCREATE.DEF module definition file

NAME MFCREATE

DESCRIPTION 'Metafile Creation Program (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

在 WinMain 函数中，MFCREATE 用文件名 MYLOGO.WMF 创建了一个元文件设备环境：

```
hMetaDC = CreateMetaFile("MYLOGO.WMF");
```

然后，在这个设备环境上画图。在画图结束后，用一条语句关闭元文件设备环境，并删除

元文件句柄：

```
DeleteMetaFile(CloseMetaFile(hdcMeta));
```

程序发出蜂鸣声，以指示已结束，然后退出 WinMain。

现在，我们可以在另一个程序中使用这个元文件。下面是 WM_PAINT 的全部程序。要获取基于磁盘的文件的句柄，只需调用 GetMetaFile。在不再需要元文件之后，调用 DeleteMetaFile：

```
hdc = BeginPaint (hwnd,&ps);
SetMapMode (hdc,MM_ANISTROSPIC);
SetWindowsExt (hdc,1000,1000);
SetViewportExt (hdc,xClient,yClient);

hmf = GetMetaFile ("MYLOGO.WMF");
for (x = 0; x < 10; x++)
    for (y = 0; y < 10; y++)
    {
        SetWindowOrg (hdc,-100 * x,-100 * y);
        PlayMetaFile (hdc,hmf);
    }
DeleteMetaFile (hmf);
EndPaint (hwnd,&ps);
```

另外，也可以定义 hmf 为静态变量，在处理 WM_CREATE 期间调用 GetMetaFile 一次，在处理 WM_DESTROY 期间调用 DeleteMetaFile 一次。当然，这种方法有一些问题。程序假定 MYLOGO.WMF 是在当前目录或者在 PATH 环境变量所列出的目录中，Windows 将显示一个消息框，要求用户将 MYLOGO.WMF 盘插入 A 驱动器中。用户对这类消息的通常反应是很冷淡的。应该在调用 GetMetaFile 之前，先查找这个文件。

现在，我们来试试另一种方法。

16.6.4 元文件作为资源

“用户定义资源”是一个文本块。现在，我们将元文件转变为用户定义资源。MYLOGO.WMF 将变成需要它的程序的 .EXE 文件的一部分。下面所示的 MFRESORC（“metafile resource”，元文件资源）用 MFCREATE 创建的 MYLOGO.WMF 元文件完成了这一过程。

```
# MFRESORC.MAK make file

mfresorc.exe : mfresorc.obj mfresorc.def mfresorc.res
    $(WINLINK) mfresorc,mfresorc,NUL,$(WINLIB),mfresorc
    rc -t mfresorc.res

mfresorc.obj : mfresorc.c
    $(WINCC) mfresorc.c

mfresorc.res : mfresorc.rc mylogo.wmf
    $(WINRC) mfresorc.rc

/*----------------------------------------
MFRESORC.C -- Metafile Resource Program
```

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

#include <windows.h>

long FAR PASCAL _export WndProc (HWND,UINT,UINT,LONG) ;

int PASCAL WinMain (HANDLE hInstance,HANDLE hPrevInstance,
                    LPSTR lpszCmdLine,int nCmdShow)
{
     static char szAppName [] = "MFResorc" ;
     HWND hwnd ;
     MSG msg ;
     WNDCLASS wndclass ;

     if (! hPrevInstance)
     {
          wndclass.style = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc = WndProc ;
          wndclass.cbClsExtra = 0 ;
          wndclass.cbWndExtra = 0 ;
          wndclass.hInstance = hInstance ;
          wndclass.hIcon = NULL ;
          wndclass.hCursor = LoadCursor (NULL,IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName = NULL ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
     }

     hwnd = CreateWindow (szAppName,"Metafile Resource Program",
          WS_OVERLAPPEDWINDOW,
          CW_USEDEFAULT,CW_USEDEFAULT,
          CW_USEDEFAULT,CW_USEDEFAULT,
          NULL,NULL,hInstance,NULL) ;

     ShowWindow (hwnd,nCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg,NULL,0,0))
     {
          TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

long FAR PASCAL _export WndProc (HWND hwnd,UINT message,UINT wParam,
                    LONG lParam)
{
     static HANDLE hmf ;
```

```
    static short cxClient,cyClient ;
    HANDLE hInstance,hResource ;
    HDC hdc ;
    PAINTSTRUCT ps ;
    short x,y ;

    switch (message)
    {
        case WM_CREATE:
            hInstance = ((LPCREATESTRUCT) lParam) -> hInstance ;
            hResource = LoadResource (hInstance,
                    FindResource (hInstance,"MyLogo","METAFILE")) ;

            LockResource (hResource) ;
            hmf = SetMetaFileBits (hResource) ;
            UnlockResource (hResource) ;
            return 0 ;
        case WM_SIZE:
            cxClient = LOWORD (lParam) ;
            cyClient = HIWORD (lParam) ;
            return 0 ;
        case WM_PAINT:
            hdc = BeginPaint (hwnd,&ps) ;

            SetMapMode (hdc,MM_ANISOTROPIC) ;
            SetWindowExt (hdc,1000,1000) ;
            SetViewportExt (hdc,cxClient,cyClient) ;

            for (x = 0 ; x < 10 ; x++)
                for (y = 0 ; y < 10 ; y++)
                {
                    SetWindowOrg (hdc,-100 * x,-100 * y) ;
                    PlayMetaFile (hdc,hmf) ;
                }

            EndPaint (hwnd,&ps) ;
            return 0 ;
        case WM_DESTROY:
            DeleteMetaFile (hmf) ;
            PostQuitMessage (0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd,message,wParam,lParam) ;
}

/* ----------------------------
MFRESORC. RC resource script
---------------------------- */
MyLogo METAFILE "mylogo. wmf"

; MFRESORC. DEF module definition file
```

```
NAME MFRESORC

DESCRIPTION 'Metafile Resource Program (c) Charles Petzold,1992'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 1024
STACKSIZE 8192
```

资源描述文件只有一行。也许 METAFILE 看起来像一个普通的资源描述关键字(如 MENU 和 DIALOG),但它不是。我们在定义这种资源类型。我们给予这个 METAFILE 的特定资源类型的名字是"MyLogo"。

在处理 WM_CREATE 消息期间,MFRESORC 必须首先调用 FindResource 和 LoadResource,以获取这个资源的句柄:

```
hResource = LoadResource(hInstance,
    FindResource(hInstance,"MyLogo","METAFILE"));
```

然后给该资源加锁:

```
LockResource(hResource);
```

通常,必须给资源加锁,以获取内存块的指针。不过,LockResource 也完成实际将资源载入内存的工作。这是我们所需要做的。现在,我们可以用 SetMetaFileBits 将这个全局内存块转换成元文件,然后资源被解锁:

```
hmf = SetMetaFileBits(hResource);
GlobalUnlock(hResource);
```

SetMetaFileBits 有一个对应的函数 GetMetaFileBits,它将元文件句柄转换成全局内存句柄。GetMetaFileBits 只能与内存元文件一起使用。

作为资源载入的元文件是内存元文件。如果想使用基于磁盘的元文件,可以拷贝之。下面是在 GlobalUnlock 语句之后使用的程序代码:

```
GetTempFileName (0,MF,0,szFileName);
hmf2 = CopyMetaFile (hmf,szFileName);
DeleteMetaFile (hmf);
hmf = hmf2;
```

hmf2 句柄不需要定义为静态变量。MFRESORC 显示了在处理 WM_DESTROY 时对元文件句柄的删除操作:

```
DeleteMetaFile(hmf);
```

也必须删除临时文件:

```
unlike(szFileName);
```

CopyMetaFile 函数也可以用来将已存在的元文件拷贝给内存元文件:

```
hmf2 = CopyMetaFile(hmf,NULL);
```

通过看一看 MYLOGO.WMF 的内容可能会对元文件是什么和不是什么有一个较明确的概念。元文件开始于一个 18 字节的头记录,后跟一系列元文件记录,每条记录包含三个或更多的双字节字。第一个字是记录中包括第一个字在内的字的数目,第二个字为 0;第三个

字是指示该字要求的 GDI 调用的代码。

这些代码在《程序员参考》和 WINDOWS. H 里有记录，标识符以 META 打头。这个字的低位字节标识特定的 GDI 调用；高位字节通常指示作为该调用参数的多个字。这个代码后的若干字是调用的实际参数，以倒序排列，不包括 hdc 参数。例如，GDI 调用：

```
Rectangle(hdcMeta,0,0,100,100);
```

在元文件中表现为一个七字记录。用十六进制表示，这些字为：

```
0007 0000 041B 0064 0064 0000 0000
```

字 041B 说明调用为 Rectangle，有四个参数，除去 hdc。后面跟着参数。

这条规则唯一的例外是 SeletObject 调用。Windows 必须保存函数选进元文件设备环境的对象。例如，调用：

```
hBrush = CreateSolidBrush (RGB (0,0,255));
```

完全不影响元文件设备环境。甚至可以在用 CreateMetaFile 创建元文件设备环境之前进行这个调用。但是，在将刷子选进元文件设备环境时必须产生两个记录。第一个记录是 CreateBrushIndirect 的：

```
0007 0000 FC02 0000 0000 00FF 0000
```

识别码 FC02 后跟的四个字是 LOGBRUSH 结构的元素，顺序与结构中元素的顺序相同（lpStyle 在前）。

调用：

```
SelectObject(hdcMeta,hBrush);
```

编码为记录：

```
0004 0000 012D 0000
```

唯一的参数 0000 指示它是元文件中第一个创建的对象的句柄。后面对其他 GDI 对象的任意 SelectObject 调用都将具有递增的参数。

对于基于内存的元文件，这些记录存放在全局内存块中。可以用 EnumMetaFile 函数来获取记录。对于基于硬盘的元文件，这些记录存放在磁盘文件里。基于磁盘的元文件句柄指向一个小的全局内存块，其中包含驱动器、目录和文件名。

第十七章　文件管理

很多 Windows 应用程序可产生不同类型的数据，这些数据被放进文件中，以备将来调用或显示。有些文件用于初始化应用程序(即在应用程序开始运行时传送初始值)。如果由于版本更新而需要改变这些值，则需要建立一个新的初始化文件而不是进行软件更新。

下面的章节将讨论 MS-DOS 文件和初始化文件。

17.1　MS-DOS 文件

Windows 可实现多任务(即同时运行几个应用程序或一个应用程序的几个实例)，但是，必须遵守几条规则。

首先，文件在应用程序调用函数 GetMessage 或 PeekMessage 时不能被打开。在某些情况下，这样做会导致文件被删除，而且容易造成打开的文件个数超过 MS-DOS 设置的界限。因此，任何文件都应使用同一条消息来打开或关闭。

文件必须遵照 DOS 文件句柄的约定，文件名不超过八个字符，并有可选的三个字符的扩展名。文件名必须使用 OEM(系统固有的)字符集，而不是 ANSI(Windows)字符集。

如果一个文件使用了临时文件，则每个实例都必须有自己的文件，参数 GetTempFileName 用来指定互异的文件名。

在任务切换时，如果还有文件被打开着而消息框不能出现在另一个应用程序中，那么每个消息框都将被当作系统模态对话框对待。

文件访问必须通过 Windows 函数而不是 C 语言函数进行。Windows 函数与模式无关。

17.1.1　打开文件

可用函数 OpenFile 或_lopen 打开文件。函数 OpenFile 提供了更多的选择(例如打开一个新文件)。函数_lcreate 也可用来打开一个新文件。

函数 OpenFile 的第一个参数是指向文件名的指针。如果当前目录下不存在这个文件，函数 PATH 将指示 OpenFile 去查找这个文件。

如果使用函数_lopen 和_lcreate，就必须用函数 AnsiToOem 把包含 ANSI 字符的文件名转化为 OEM 字符串。

函数 OpenFile 的第二个参数是指向 OFSTRUCT 结构变量的指针。在第一次调用此函数时，结构中各元素被赋值。只要标志 OF_REOPEN 被置位，结构变量就可用于对同一个文件的一系列 OpenFile 函数调用中。

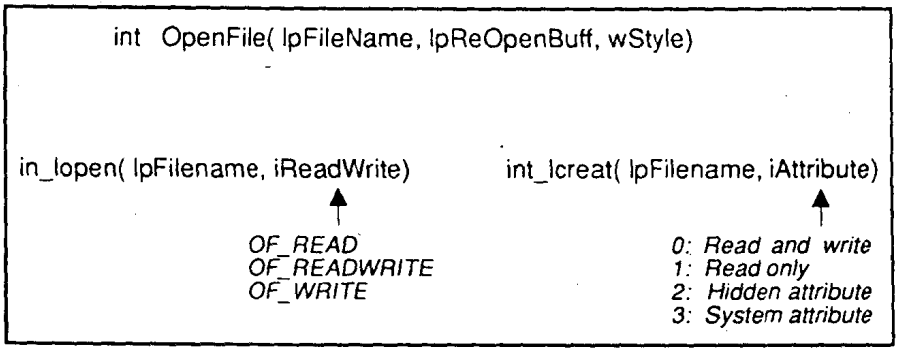


图 17.1 函数 OpenFile

结构 OFSTRUCT

OFSTRUCT 结构如下：

数据组	数据类型	描述
cBytes	BYTE	结构的大小(以字节为单位)
fFixedDisk	BYTE	读写硬盘上的文件
nErrCode	WORD	MS-DOS 错误码
reserved[4]	BYTE	保留
szPathName[120]	BYTE	全文件名

如果函数 OpenFile 调用成功，szPathName 就被赋值以完整的路径名(包括驱动器号和路径)。

OpenFile 的最后一个参数指出访问文件的类型，可用逻辑操作 OR 来组合这些访问类型。

OF_CREATE	建立新文件
OF_DELETE	删除文件
OF_REOPEN	用 OFSTRUCT 结构打开文件
OF_READ	用只读方式打开文件
OF_READWRITE	用读/写方式打开文件
OF_WRITE	用只写方式打开文件
OF_EXIST	检查文件是否存在
OF_PARSE	分析 OFSTRUCT 结构
OF_VERIFY	比较当前文件与老文件的日期
OF_PROMPT	若文件不存在，则显示一个消息框
OF_CANCEL	在消息框中加入按钮 Cancel

当 OF_CREATE 被设置时，应用程序就准备用指定的名字建立一个新文件。如果有同名文件存在，这个文件的长度将被置为零。

设置标志 OF_EXIST 可防止数据丢失。这个标志检查是否有同名文件存在，如果有，则函数 OpenFile 返回－1。

打开已存在的文件需要更多的说明。文件的打开方式可以是只读、只写或读写。OpenFile的返回值是打开文件的句柄。所有文件都使用句柄进行访问，如果文件不能被打开，则返回值为－1。

OF_PROMPT 产生一个对话框以通知用户文件不存在，并提示用户在 A：驱动器中插入软盘。OF_CANCEL 在 OF_PROMPT 对话框中放置一个 Cancel 按钮。如果文件不存在，用户可以按此按钮以取消文件调用。

通常可以用 OF_REOPEN 重新打开文件，这时并不需要文件名，因为在第一次打开文件时，完整的文件名已经存入 OFSTRUT 结构变量中了。重新打开文件使得在改变了当前目录或者所需的文件不在 DOS PATH 中时，应用程序还能找到文件。

函数 OF_DELETE 可以删除指定了名称的文件。如果删除失败，则返回－1。

17.1.2 关闭文件

一旦用完了一个文件，就应立即用函数_lclose 关闭它。

```
int _lclose (hFile);
```

函数_lclose 只需要一个参数，即文件句柄。如果关闭成功，则_lclose 返回 0；如果发生了错误，则返回－1。

17.1.3 读文件

如果文件被打开时设置了读属性，则可以从打开的文件中读出数据。Windows 用下面这个函数读文件：

```
int _lread (hFile,lpBuffer,wBytes);
```

参数 hFile 指示要读哪一个文件，缓冲区接收文件数据以备进一步处理。

因为文件常常是很大的，所以不能一次读进内存。参数 lpBuffer 指示了存储文件数据的缓冲区大小，参数 wBytes 为一次读入的字符数。

函数_lread 的返回值为实际读入的字节数。注意，此值不一定等于参数 wBytes 的值。在文件尾被读入时，返回值会比参数 wBytes 小。如果发生了错误，函数_lread 返回－1。

17.1.4 写文件

读文件与写文件所用的参数完全相同。Windows 用下面这个函数写文件：

```
int _lwrite (hFile,lpBuffer,wBytes)
```

参数 hFile 为文件句柄，参数 lpBuffer 为存储数据的缓冲区的大小，参数 wBytes 为将要传送的字节数。

函数_lwrite 的返回值为实际写入文件的字节数。

17.1.5 设置文件指针

每次读写文件之后，文件指针都会自动地调整到新位置。

每次打开文件，文件指针都将指向文件头。如果想把数据插入到文件的另一位置（例如想在文件尾添加数据），则必须修改文件指针。

Windows 用函数_llseek 完成此任务：

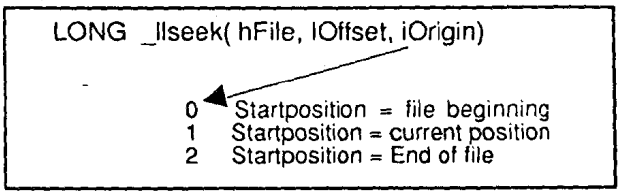


图 17.2 函数_llseek

文件指针位于何处依赖于三个初始点:文件头、文件尾和文件当前位置。第二个参数为所需的初始点,第三个指针为文件指针所移动的字节数。如果初始点为文件尾,则文件指针在定位时向文件头方向移动。

函数_llseek 的返回值是新的文件指针位置,即从文件头开始的字节数。

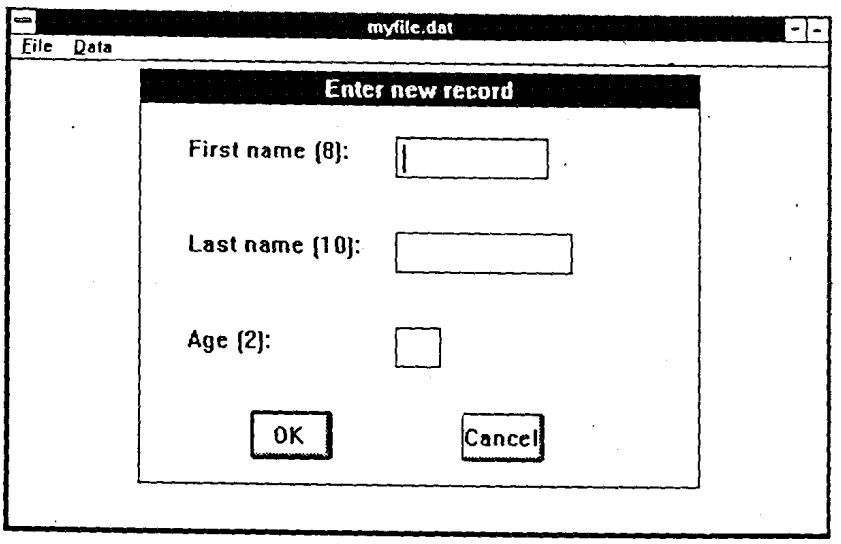


图 17.3

17.1.6 文件访问示例

这个应用程序演示简单的文件操作。它的菜单可允许用户建立新文件,打开已有的文件,删除文件和向文件中输入数据。当用户选中 New 或者 Open 菜单项时,将出现一个带有编辑控件的对话框,用户可以在这个对话框中输入不指定路径的文件名。

打开文件后,选中 Enter 菜单项便可向文件中输入文本了。屏幕上将显示一个对话框,其中有"first name,last name,age"的编辑控件。按下 Save 按钮便可把数据存入文件中,并在主窗口中显示出来。

新消息	描述
EM_LIMITTEXT	为编辑窗口设置最大的文本大小
新函数	描述
_lclose	关闭文件

_llseek	设置文件指针
_lread	从文件中读取数据
_lwrite	向文件中写数据
OpenFile	打开、建立并删除一个文件

源代码:FILE.C

```
/* * FILE.C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Simple database,storing names and ages. This application uses dialog * */
/* * boxes and message boxes. * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"
#include "file.h"
#include "string.h"

#if _MSC_VER >= 700                                    // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                         // higher? Disable warning 4028
#endif

BOOL FileInit ( HANDLE );
long FAR PASCAL FileWndProc( HWND,unsigned,WORD,LONG);
BOOL FAR PASCAL OpenNewDlgProc( HWND,unsigned,WORD,LONG);
                                                       // Function - Open/New dialog box
BOOL FAR PASCAL EntryDlgProc( HWND,unsigned,WORD,LONG);
                                                       // Function - Entries dialog box
static char szNameText[15],szDataRecord[24];

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow)
HANDLE hInstance,hPrevInstance;
LPSTR lpszCmdLine;
int cmdShow;
{
    MSG msg;
    HWND hWnd;

    if (! hPrevInstance)
    {
        if (! FileInit( hInstance ))
            return FALSE;

/* * Specify main application window * * * * * * * * * * * * * * * */

    }
    hWnd = CreateWindow("File",
        "File Management Demo",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,0,
        CW_USEDEFAULT,0,
        NULL,NULL,hInstance,NULL);
```

```
    ShowWindow( hWnd,cmdShow );
    UpdateWindow( hWnd );

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int)msg. wParam;
}

BOOL FileInit( hInstance )
HANDLE hInstance;

/* * Specify window class * * * * * * * * * * * * * * * * * * * */
{
    WNDCLASS wcFileClass;

    wcFileClass. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcFileClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcFileClass. lpszMenuName = (LPSTR)"MenuFile";
    wcFileClass. lpszClassName = (LPSTR)"File";
    wcFileClass. hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcFileClass. hInstance = hInstance;
    wcFileClass. style = CS_VREDRAW | CS_HREDRAW;
    wcFileClass. lpfnWndProc = FileWndProc;
    wcFileClass. cbClsExtra = 0 ;
    wcFileClass. cbWndExtra = 0 ;

    if ( ! RegisterClass( &wcFileClass ) )
        return FALSE;
    return TRUE;
}

long FAR PASCAL FileWndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    static HANDLE hInst;
    char lpBuffer[400],lpBuff[1];
    FARPROC lpprocDia;
    static OFSTRUCT OfStruct;
    int hFile,iNumber,i;
    BOOL bExist = FALSE;
    static BOOL bDFile;
    HDC hDC;
    PAINTSTRUCT ps;
    RECT rect;
```

```
switch (message)
{
    case WM_CREATE:
        hInst = GetWindowWord(hWnd,GWW_HINSTANCE);
        bDFile = FALSE;
        memset(szNameText,' ',15);
        break;
    case WM_COMMAND:                                          // Menu items
        switch(wParam)
        {
            case ID_DEL:                                      // Delete file & data
                GetWindowText(hWnd,szNameText,15);
                OpenFile(szNameText,&OfStruct,OF_DELETE);
                SetWindowText(hWnd,"File Management Demo");
                bDFile = FALSE;
                memset(szNameText,' ',15);
                InvalidateRect(hWnd,NULL,TRUE);
                break;
            case ID_NEW:                                      // Create new file
                lpprocDia = MakeProcInstance (OpenNewDlgProc,hInst);
                if (DialogBox(hInst,"OPENNEW",hWnd,lpprocDia))
                {
                    if (OpenFile(szNameText,&OfStruct,OF_EXIST) )= 0)
                    {
                        bExist = TRUE;
                        if (MessageBox(hWnd,"Replace existing file?",
                                "File",MB_OKCANCEL |
                                MB_ICONHAND) == IDOK)
                        {
                            bExist = FALSE;
                        }
                    }
                    if (bExist)
                        memset(szNameText,' ',15);
                    if (! bExist)
                    {
                        hFile = OpenFile(szNameText,&OfStruct,
                                OF_READWRITE | OF_CREATE);
                            if (hFile )= 0)
                            {
                                lpBuff[0] = '\0';
                                _lwrite(hFile,lpBuff,1);
                                _lclose(hFile);
                                SetWindowText(hWnd,szNameText);
                                bDFile = TRUE;
                            }
                            else
                                MessageBox(hWnd,"Illegal filename",
                                    "Attention",MB_OK);
                    }
                }
```

```
            InvalidateRect(hWnd,NULL,TRUE);
            FreeProcInstance (lpprocDia);
            break;
        case ID_OPEN;// Open existing file
            lpprocDia = MakeProcInstance (OpenNewDlgProc,hInst);
            if (DialogBox(hInst,"OPENNEW",hWnd,lpprocDia))
            {
                if (OpenFile(szNameText,&OfStruct,OF_EXIST) == -1)
                {
                    MessageBox(hWnd,"File not found","Error",MB_OK);
                    bDFile = FALSE;
                    memset(szNameText,' ',15);
                }
                else
                {
                    bDFile = TRUE;
                    SetWindowText(hWnd,szNameText);
                    InvalidateRect(hWnd,NULL,TRUE);
                }
            }
            FreeProcInstance (lpprocDia);
            break;
        case ID_ENT;                          // Enter data record for placement in file
            lpprocDia = MakeProcInstance (EntryDlgProc,hInst);
            if (DialogBox(hInst,"ENTRIES",hWnd,lpprocDia))
            {
                GetWindowText(hWnd,szNameText,15);
                hFile = OpenFile(NULL,&OfStruct,
                        OF_READWRITE | OF_REOPEN);
                if (hFile >= 0)
                {
                    i=0;
                    lpBuff[0]= 'a';
                    while (lpBuff[0] ! = '\0')
                    {
                        _llseek(hFile,i,0);
                        i++;
                        _lread(hFile,lpBuff,1);
                    }
                    i--;
                    _llseek(hFile,i,0);
                    _lwrite(hFile,szDataRecord,24);
                    _lclose(hFile);
                    InvalidateRect(hWnd,NULL,TRUE);
                }
                if (hFile == -1)
                    MessageBox(hWnd,"No file open","Error",MB_OK);
            }
            FreeProcInstance (lpprocDia);
            break;
    }
```

```
            break;
        case WM_PAINT:
            hDC = BeginPaint(hWnd,&ps);
            if (bDFile)
            {
                GetWindowText(hWnd,szNameText,15);
                if (OpenFile(szNameText,&OfStruct,OF_EXIST) == -1)
                {
                    bDFile = FALSE;
                    SetWindowText(hWnd,"File management demo");
                    memset(szNameText,' ',15);
                }
                hFile = OpenFile(szNameText,&OfStruct,OF_READ |
                        OF_PROMPT | OF_CANCEL);
                if (hFile >= 0)
                {
                    memset(lpBuffer,' ',400);
                    iNumber = _lread( hFile,lpBuffer,400);
                    if (iNumber >= 0)
                    {
                        GetClientRect(hWnd,&rect);
                        DrawText(hDC,lpBuffer,-1,&rect,DT_CENTER);
                    }
                    _lclose(hFile);
                }
            }
            EndPaint(hWnd,&ps);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}

/** Generate OPEN/NEW dialog box **********/

BOOL FAR PASCAL OpenNewDlgProc( hDlg,message,wParam,lParam )
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
    static HWND hName;

    switch (message)
    {
        case WM_INITDIALOG:
```

```
            hName = GetDlgItem(hDlg,ID_NAME);
            SendMessage(hName,EM_LIMITTEXT,12,0L);
            SetFocus(hName);
            return(FALSE);
            break;
        case WM_COMMAND:
            switch (wParam)
            {
                case IDOK:                                          // OK,open file
                    GetDlgItemText( hDlg,ID_NAME,szNameText,15);
                    EndDialog (hDlg,1);
                    return(TRUE);
                    break;
                case IDCANCEL:                                      // Cancel,don't open file
                    EndDialog (hDlg,0);
                    return(TRUE);
                    break;
                default:
                    return(FALSE);
                }
        default:
            return(FALSE);
    }
}

/* * Generate Enter data dialog box * * * * * * * * * * * * * * * */

BOOL FAR PASCAL EntryDlgProc( hDlg,message,wParam,lParam )
HWND hDlg;
unsigned message;
WORD wParam;
LONG lParam;
{
    static char szName1Text[9],szName2Text[11],szName3Text[3];
    HWND hFirstN,hLastN,hAge;

    switch (message)
    {
        case WM_INITDIALOG:                                         // Dialog controls
            hFirstN = GetDlgItem(hDlg,ID_FIRSTNAME);
            hLastN = GetDlgItem(hDlg,ID_LASTNAME);
            hAge = GetDlgItem(hDlg,ID_YOURAGE);
            SendMessage(hFirstN,EM_LIMITTEXT,8,0L);
            SendMessage(hLastN,EM_LIMITTEXT,10,0L);
            SendMessage(hAge,EM_LIMITTEXT,2,0L);
            SetFocus(hFirstN);
            return(FALSE);
            break;
        case WM_COMMAND:                                            // Buttons
            switch (wParam)
            {
```

```
                case IDOK:                                          // OK,store record
                    memset(szName1Text,' ',9);
                    memset(szName2Text,' ',11);
                    memset(szName3Text,' ',3);
                    GetDlgItemText( hDlg,ID_FIRSTNAME,szName1Text,9);
                    GetDlgItemText( hDlg,ID_LASTNAME,szName2Text,11);
                    GetDlgItemText( hDlg,ID_YOURAGE,szName3Text,3);
                    memset(szDataRecord,' ',24);
                    memcpy(szDataRecord,szName1Text,strlen(szName1Text));
                    memcpy(szDataRecord+9,szName2Text,strlen(szName2Text));
                    memcpy(szDataRecord+20,szName3Text,strlen(szName3Text));
                    memcpy(szDataRecord+22,"\n",2);
                    EndDialog (hDlg,1);
                    return(TRUE);
                    break;
                case IDCANCEL:                                      // Cancel,don't store record
                    EndDialog (hDlg,0);
                    return(TRUE);
                    break;
                default:
                    return(FALSE);
            }
        default:
            return(FALSE);
    }
}
```

模块定义文件:FILE. DEF

```
NAME File

DESCRIPTION 'File example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS FileWndProc @4
    OpenNewDlgProc @8
    EntryDlgProc @9
```

资源描述文件:FILE. RC

```
#include "file. h"
#include "windows. h"
```

```
#include "opennew.dlg"
#include "entries.dlg"

MenuFile MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New", ID_NEW
        MENUITEM "&Open…", ID_OPEN
        MENUITEM "&Delete", ID_DEL
    END

    POPUP "&Data"
    BEGIN
        MENUITEM "&Entry…", ID_ENT
    END
END
```

头文件:FILE.H

```
#define ID_NEW 10
#define ID_OPEN 11
#define ID_DEL 12
#define ID_ENT 13

#define ID_NAME 20

#define ID_FIRSTNAME 30

#define ID_LASTNAME 31

#define ID_YOURAGE 32
```

对话框文件:ENTRIES.DLG

```
ENTRIES DIALOG LOADONCALL MOVEABLE DISCARDABLE 52,49,162,111
CAPTION "Enter new record"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "First name (8):",-1,"static",SS_LEFT
        | WS_CHILD,14,9,49,8
    CONTROL "Last name (10):",-1,"static",SS_LEFT
        | WS_CHILD,14,37,58,8
    CONTROL "Age (2):",-1,"static", SS_LEFT
        | WS_CHILD,14,65,45,8
    CONTROL "",ID_FIRSTNAME,"edit",ES_LEFT | ES_AUTOHSCROLL
        | WS_BORDER | WS_TABSTOP | WS_CHILD,74,9,44,12
    CONTROL "",ID_LASTNAME,"edit",ES_LEFT | ES_AUTOHSCROLL
        | WS_BORDER | WS_TABSTOP | WS_CHILD,74,37,51,12
    CONTROL "",ID_YOURAGE,"edit",ES_LEFT | ES_AUTOHSCROLL
        | WS_BORDER | WS_TABSTOP | WS_CHILD,74,65,13,12
    CONTROL "OK",IDOK,"button",BS_DEFPUSHBUTTON | WS_TABSTOP
```

```
        | WS_CHILD,32,90,24,14
    CONTROL "Cancel",IDCANCEL, "button", BS_PUSHBUTTON
        | WS_TABSTOP | WS_CHILD,93,90,24,14
END
```

对话框文件:OPENNEW. DLG

```
OPENNEW DIALOG LOADONCALL MOVEABLE DISCARDABLE 53,68,131,78
CAPTION "Open / New"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "File name:", -1,"static",SS_LEFT | WS_GROUP
        | WS_CHILD,31,15,49,8
    CONTROL "",ID_NAME,"edit",ES_LEFT | ES_AUTOHSCROLL
        | WS_BORDER | WS_TABSTOP | WS_CHILD,30,25,61,12
    CONTROL "OK",IDOK,"button",BS_DEFPUSHBUTTON | WS_GROUP
        | WS_TABSTOP | WS_CHILD,18,53,24,14
    CONTROL "Cancel",IDCANCEL,"button",BS_PUSHBUTTON
        | WS_CHILD,69,53,37,14
END
```

MAKE 文件:FILE. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = entries.dlg opennew.dlg
H = FILE.H

all: FILE.EXE

FILE.RES: FILE.RC $(RESFILES)

FILE.OBJ: FILE.C $(H)

FILE.EXE: FILE.OBJ FILE.RES FILE.DEF
    link /NOD/A:16 FILE,FILE,nul,libw slibcew,FILE.def
    rc FILE.res FILE.exe
```

像下面这样运行 NMAKE 可产生应用程序 FILE:

```
nmake file.mak
```

FILE.EXE 是怎样工作的

对话框 OPENNEW 和 ENTRIES 使用户能够通过编辑控件与应用程序交互。这些控件被赋以头文件 FILE.H 中定义的 ID 值。在两个对话框例程 OpenNewDlgProc 和 EntryDlgProc 中,每个编辑控件所能输入的文本数都由消息 EM_LIMITTEXT 限制。

头文件 WINDOWS. H 定义了对话按钮 IDOK 和 IDCANCEL。当用户按下 OK 按钮时，函数 GetDlgItemText 将指定文件中的数据读入相应的缓冲区中。

在 First name、Last name、Age(2)域中输入的数据被作为一个记录存入 szDataRecord 串中。每个记录由一个回车(CR)和"\0"结尾。两个对话框文件都由语句 #include 加入到资源描述文件 FILE. RC 中。

消息 WM_CREATE 用函数 GetWindowWord 传递当前实例的句柄，有几个函数需要句柄信息。布尔变量 bDFile 的值为 FALSE，以指示文件还未打开(消息 WM_PAINT 需要此信息)。字符串 szNameText 包含对话框函数 OpennewDlgProc 中输入的文件名，这个串在初始化时为空白字符。

有四个菜单项可供用户选择。选中 New 则建立一个新文件，这个菜单项将调用 OPENNEW 对话框，用户可在此框中输入新文件名。当用户按下了 OK 按钮时，函数 DialogBox 将接收到 EndDialog(hDlg, 1)的返回值 1，并将此值作为本身的返回值。函数 OpenFile 的参数 OF_EXIST 被用来检查有无同名的文件存在，如果有，它的值将不等于－1。

这时将出现一条消息，寻问是否删掉已存在的同名文件。如果用户想替换老文件或输入了一个新文件名，则函数 OpenFile 将建立这个文件。消息 WM_PAINT 调用函数 DrawText 时，将在文件名的尾部插入"\0"。函数 SetWindowText 将窗口标题换为当前文件名。如果函数 OpenFile 的返回值为－1，可能是由于输入的文件名长度超过了应用程序通常允许的范围。

此应用程序也可以打开已有的文件进行输入。选中 Open 菜单项时出现的对话框与选中 New 时的一样。如果输入的文件名已经有同名文件存在，文件名就会出现在窗口标题中。布尔变量 bDFile 被置为 TRUE，消息 WM_PAINT 将产生一个窗口以显示文件的内容。

选中 Delete 菜单项将删除当前的文件。函数 GetWindowText 从窗口标题获得当前文件名。函数 OpenFile 以 OF_DELETE 为参数完成删除工作。WM_PAINT 消息不再显示文件的内容了。

建立或打开一个文件之后，用户便可选中 Data Enter 来向文件中输入数据。如果文件必须被重新打开，函数 OpenFile 带上参数 OF_REOPEN 便可完成，这时才需要文件名，只需在选中 New 或 Open 前传送变量 ofstruct 的地址即可。

函数_llseek 总是把文件指针后移一个字节，直到文件指针遇到字符"\0"(文件尾)为止，这个字符是用函数_lwrite 写文件时下一个记录的开始点。读文件时，每次要把文件指针后移一个字节。写文件时，要把文件指针前移一个字节。

如果用户在输入数据前忘了打开文件，就将出现一个消息框，提示"No file open"。如果布尔变量 bDFile 被置为 False，对消息 WM_PAINT 的处理将是仅消除窗口的内容，而不显示任何新内容。

当用户选中 Delete 菜单项删除文件时就会发生上述情况。也可用 Windows 的其他应用程序或同一应用程序的另一实例来删除文件。

负值结果将删除标题条中的文件名并将变量 bDFile 置为 FALSE。任何其他的结果都将重新打开指定的文件。lpBuffer 指向的 400 个字节的缓冲区被清空，然后把文件数据调入此缓冲区中。函数 DrawText 将显示调入此缓冲区的记录。每个记录以一个回车结尾。应用程序从文件中读到"\0"时使结束输出。

17.2 初始化文件

初始化文件的扩展名为 .INI，它们包含着其他文件的参数信息。例如，初始化文件可以给不同用户配置不同的 Windows 环境。文件中的值随时可以改变。

```
[Name of Range]
Keyword = Current Value
```

图 17.4

Windows 有两个初始化文件：WIN.INI 和 SYSTEM.INI（INI 有时读作“Innie”）。这些 .INI 文件是按一定的顺序排列好的非格式化文本文件。下面是一个 .INI 文件结构和从文件 WIN.INI 中的摘录的内容：

```
[windows]
BorderWidth = 3
[CorelDraw]
Dir = C:\APPS\CORELDRW
```

这种格式也可用于用户自己的初始化文件。类属名（被方括号括住的一行）必须是有意义的，即若类属于应用程序，则应用程序名应该是用户。

每个类属有一系列入口，这些入口定义了关键字和关键字的值，这些值由文本、整数或引号中的文本构成。引号中的文本中的空白字符也不应被跳过。以分号（;）开头的行为注释行。

17.2.1 标准初始化文件

文件 WIN.INI 包含应用程序信息和系统配置数据。文件 SYSTEM.INI 包含控制设备的配置数据，这些数据不能被函数访问。

WIN.INI

此文件包含不超过 11 个系统类属和应用程序定义的类属：

- Windows
- Desktop
- Kernel
- Ports
- Colors
- PrinterPorts
- Devices
- Fonts
- intl（International）
- Extensions
- PIF

用户可通过 Windows 控制面板（Windows Control Panel）修改此文件中的大多数选项。Windows 控制面版进行的修改只影响当前 Windows 会话。因为 WIN.INI 文件控制着所有应用程序，所以当消息 WM_WININICHANGE 被送出时，WIN.INI 将把参数通知给文件中所列出的所有应用程序。

函数 SendMessage 的参数 lParam 是一个将要修改的类属名。如果函数的第一个参数被赋为 0×FFFF 而不是一个窗口句柄，则直接发送消息给最顶层的窗口。

下列三个函数可直接从文件 WIN.INI 中读写数据：

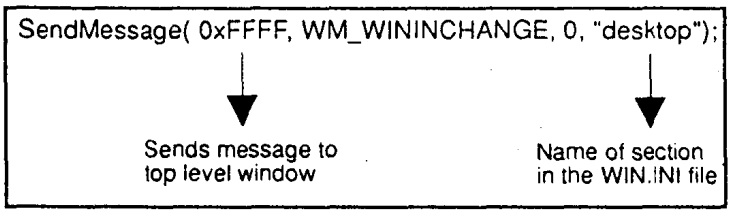


图 17.5

int GetProfileInt(lpApplicationName,lpKeyName,nDefault)

int GetProfileString(lpApplicationName,lpKeyName,lpDefault,lpReturnedString,nSize)

BOOL WriteProfileString(lppApplicationName,lpKeyName,lpString)

参数	指向
lpApplicationName	类属名和应用程序名
lpKeyName	关键字名

函数 GetProfileString 从文件 WIN. INI 中拷贝一个字符串到参数指定的缓冲区中。函数查找被指定的类属和关键字。如果找到了关键字,就把相应的字符串拷贝到缓冲区中。

在其他实例中,变量 lpDefault 指定的串被拷贝到变量 lpReturnedString 中,这里的查找不区分大小写。

如果已知哪些关键字位于 WIN. INI 的一个指定的类属下,则参数 lpKeyName 可被置为 NULL。函数 GetProfileString 列出在变量 lpReturnedString 中找到的所有入口,每两个关键字之间以字符"\0"隔开。

函数 GetProfileInt 把在 WIN. INI 中读出的数据看作整数。如果关键字只包含字符串,函数将返回零值。如果关键字在一系列字符前有一个整数,则只读取这个整数。

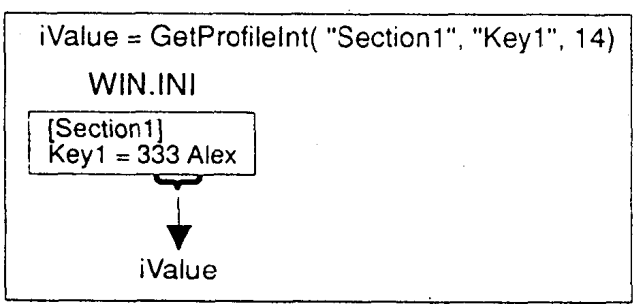


图 17.6 函数 GetProfileInt

函数 WriteProfileString 允许在应用程序中修改文件 WIN. INI。因为所有的初始化文件都是文本文件,所以函数不能使用整数值作参数,而应该用 C 函数 itoa 进行转换。

如果一个类属名或关键字不存在,函数 WriteProfileString 将不会把它写到文件 WIN. INI 中。此函数也可用来删除文件 WIN. INI 中的某些行。如果最后一个参数值为

NULL，则关键字所指定的那一行将被删掉。如果关键字被置为 NULL，函数将删除第一个参数中标出的整个类属。

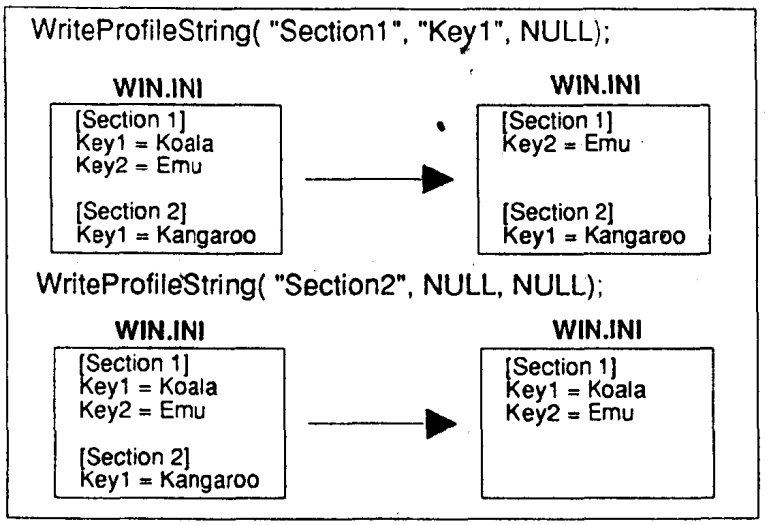


图 17.7　函数 WriteProfileString

SYSTEM. INI

此初始化文件包含了 Windows 初始化时所需的所有全程系统信息，用户不能从控制面板中修改这些数据。与文件 WIN. INI 不同的是，没有读写此文件的函数。

SYSTEM. INI 中的配置类属

文件 SYSTEM. INI 包含下面四个类属：

- Boot
- BootDescription
- Win386
- Keyboard

如果某个类属发生了改变(例如[Keyboard])，在下一个 Windows 会话开始时将生效。

17.2.2　建立自己的初始化文件

应用程序也可以访问用户定义的初始化文件。用下面的函数可建立初始化文件：

```
int GetPrivateProfileInt(lpApplicationName,lpKeyName,nDefault,lpFileName)
int GetPrivateProfileString(lpApplicationName,lpKeyName,lpDefault,lpReturnedString,
nSize,lpFileName )
BOOL WritePrivateProfileString(lpApplicationName,lpKeyName,lpString,lpFileName)
```

参数	指向
lpApplicationName	类属名和应用程序名

lpKeyName	关键字名
lpFileName	初始化文件名

这些函数与 WIN. INI 中的函数的使用方法相同，只是附加的参数被写入了私有初始化文件中。

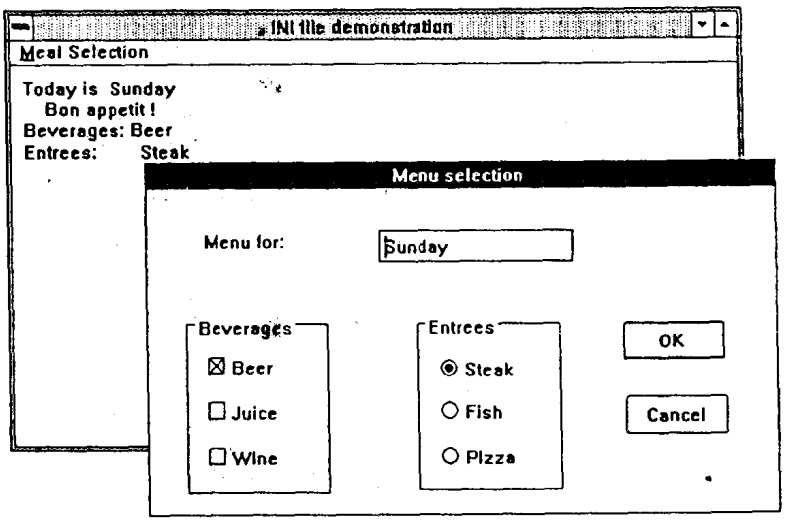


图 17.8

17.2.3 初始化文件示例

这个示例建立一个私有初始化文件，文件的内容显示在客户区中。用户可通过对话框输入初始化文件的数据。

对话框的任务是安排午餐，用户可以选择三道主菜之一和不超过三种饮料。菜单根据当天是周几而改变。

新函数	描述
GetPrivateProfileInt	从一个私有 . INI 文件中取出数字
GetPrivateProfileString	从一个私有 . INI 文件中取出数据
GetProfileString	从文件 WIN. INI 中取出数据
lstrcpy	把一件事加到另一件事后
WritePrivateProfileString	向一个私有 . INI 文件中写数据
WriteProfileString	向文件 WIN. INI 中写数据

源文件：FINIT. C

```
/* * FINIT.C * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Demonstrates initialization file creation using a simple menu planning * * /
/* * application. * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#define NOMINMAX
```

```
#include "windows.h"
#include "finit.h"
#include <string.h>
#include <stdlib.h>

#if _MSC_VER >= 700
#define itoa(v,s,r) _itoa(v,s,r)
#pragma warning (disable:4028)
#endif

BOOL FinitInit ( HANDLE );
long FAR PASCAL FinitWndProc( HWND,unsigned,WORD,LONG);
BOOL FAR PASCAL MealsDlgProc( HWND,unsigned,WORD,LONG);

char szReturnDay[11];
BOOL bJuice,bWine,bBeer;
char szEntree[6];
WORD wEntree;

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow)
HANDLE hInstance,hPrevInstance;
LPSTR lpszCmdLine;
int cmdShow;
{
    MSG msg;
    HWND hWnd;

    if (! hPrevInstance)
    {
        if (! FinitInit( hInstance ))
            return FALSE;
    }
    hWnd = CreateWindow("Finit",
            "INI file demonstration",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,0,CW_USEDEFAULT,0,
            NULL,NULL,hInstance,NULL);

    ShowWindow( hWnd,cmdShow );
    UpdateWindow( hWnd );

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int)msg.wParam;
}

BOOL FinitInit( hInstance )
HANDLE hInstance;
```

```
{
    WNDCLASS wcFinitClass;

    wcFinitClass.hCursor = LoadCursor( NULL,IDC_ARROW );
    wcFinitClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcFinitClass.lpszMenuName = (LPSTR)"Meals";
    wcFinitClass.lpszClassName = (LPSTR)"Finit";
    wcFinitClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcFinitClass.hInstance = hInstance;
    wcFinitClass.style = CS_VREDRAW | CS_HREDRAW;
    wcFinitClass.lpfnWndProc = FinitWndProc;
    wcFinitClass.cbClsExtra = 0 ;
    wcFinitClass.cbWndExtra = 0 ;

    if (! RegisterClass( &wcFinitClass ) )
        return FALSE;
    return TRUE;
}

long FAR PASCAL FinitWndProc( hWnd,message,wParam,lParam )
HWND hWnd;
unsigned message;
WORD wParam;
LONG lParam;
{
    static HANDLE hInst;
    FARPROC lpprocDia;
    HDC hDC;
    PAINTSTRUCT ps;
    int i;

    static char szScrnOutput[4][26] = {"Today is ",
            " Bon appetit ! ",
            "Beverages: ",
            "Entrees: "};
    static char szSecName1[] = "Date";
    static char szSecName2[] = "Food";
    static char szSecName3[] = "Drink";
    static char szKeyName11[] = "Day";
    static char szKeyName21[] = "Entree";
    static char szKeyName31[] = "Beer";
    static char szKeyName32[] = "Juice";
    static char szKeyName33[] = "Wine";
    static char szDefaultStr[] = " ";
    static char szReturn[10];
    static char szInitFileName[13];
    char szName[12];
    char lpValue[2];

    switch (message)
    {
```

```
case WM_CREATE:
    hInst = GetWindowWord(hWnd,GWW_HINSTANCE);

    szDefaultStr[0] = 0;
    GetProfileString("Finit","MealsInit",szDefaultStr,szInitFileName,20);
    if ( *szInitFileName == 0)
    {
        WriteProfileString("Finit","MealsInit","meals.ini");
        lstrcpy( szInitFileName,"meals.ini");
    }

    szDefaultStr[0] = 0;
    GetPrivateProfileString("Init","Meals",szDefaultStr ,szName,13,szInitFileName);
    if (lstrcmp(szName,szInitFileName))
    {
        WritePrivateProfileString("Init","Meals","meals.ini",szInitFileName);
        WritePrivateProfileString(szSecName1,szKeyName11,"Sunday",
                szInitFileName);
        WritePrivateProfileString(szSecName2,szKeyName21,"Steak",szInitFileName);
        WritePrivateProfileString(szSecName3,szKeyName31,"1",szInitFileName);
        WritePrivateProfileString(szSecName3,szKeyName32,"0",szInitFileName);
        WritePrivateProfileString(szSecName3,szKeyName33,"0",szInitFileName);
    }

    /** Default information for MEALS.INI **********/
    GetPrivateProfileString(szSecName1,szKeyName11,"Sunday",szReturnDay,11,
            szInitFileName);
    GetPrivateProfileString(szSecName2,szKeyName21,"Steak",szEntree,6,
            szInitFileName);

    /** Change entree **********************/

    switch (szEntree[0])
    {
        case 'S':
            wEntree = ID_STEAK;
            break;
        case 'F':
            wEntree = ID_FISH;
            break;
        case 'P':
            wEntree = ID_PIZZA;
            break;
        default:
            wEntree = ID_STEAK;
            break;
    }

    bBeer = GetPrivateProfileInt(szSecName3,szKeyName31,1,szInitFileName);
    bJuice = GetPrivateProfileInt(szSecName3,szKeyName32,0,szInitFileName);
    bWine = GetPrivateProfileInt(szSecName3,szKeyName33,0,szInitFileName);
```

```
        break;
    case WM_COMMAND:
        switch(wParam)
        {
            case ID_MEALS:
                lpprocDia = MakeProcInstance (MealsDlgProc,hInst);
                if (DialogBox(hInst,"Meals",hWnd,lpprocDia))
                {
                    switch (wEntree)
                    {
                        case ID_STEAK:
                            lstrcpy( szEntree,"Steak");
                            bresk;
                        case ID_FISH:
                            lstrcpy( szEntree,"Fish");
                            break;
                        case ID_PIZZA:
                            lstrcpy( szEntree,"Pizza");
                            break;
                        default:
                            lstrcpy( szEntree,"Steak");
                            break;
                    }

                    WritePrivateProfileString(szSecName1,szKeyName11,
                        szReturnDay,szInitFileName);
                    WritePrivateProfileString(szSecName2,szKeyName21,
                        szEntree,szInitFileName);
                    itoa(bBeer,lpValue,10);
                    WritePrivateProfileString(szSecName3,szKeyName31,
                        lpValue,szInitFileName);
                    itoa(bJuice,lpValue,10);
                    WritePrivateProfileString(szSecName3,szKeyName32,
                        lpValue,szInitFileName);
                    itoa(bWine,lpValue,10);
                    WritePrivateProfileString(szSecName3,szKeyName33,
                        lpValue,szInitFileName);

                    InvalidateRect(hWnd,NULL,TRUE);
                }
                FreeProcInstance (lpprocDia);
                break;
            default:
                break;
        }
        break;
    case WM_PAINT:                                          // Draw window
        hDC = BeginPaint(hWnd,&ps);

        lstrcpy(&szScrnOutput[0][10],szReturnDay);
        lstrcpy(&szScrnOutput[3][16],szEntree);
```

```
            i = 0;

            /* * Display beverage * * *
```

```
            SendDlgItemMessage(hDlg,ID_WINE,BM_SETCHECK,bWine,(LONG)0);
            return(FALSE);
            break;
        case WM_COMMAND:
            switch (wParam)
            {
                case ID_STEAK:
                case ID_FISH:
                case ID_PIZZA:
                    wEntree = wParam;
                    return (TRUE);
                    break;

                    /* * Place information in MEALS. INI * * * * * * * * * * * * * * */

                case IDOK:
                    GetDlgItemText( hDlg,ID_DAY,szReturnDay,11);
                    bBeer = (BOOL)SendDlgItemMessage(hDlg,ID_BEER,BM_GETCHECK,0,0L);
                    bJuice = (BOOL)SendDlgItemMessage(hDlg,ID_JUICE,BM_GETCHECK,0,0L);
                    bWine = (BOOL)SendDlgItemMessage(hDlg,ID_WINE,BM_GETCHECK,0,0L);
                    EndDialog (hDlg,1);
                    return(TRUE);
                    break;

                /* * Cancel,do not place information in MEALS. INI * * * * * * * * * */

                case IDCANCEL:
                    EndDialog (hDlg,0);
                    return(TRUE);
                    break;
                default:
                    return(FALSE);
            }
        default:
            return(FALSE);
    }
}
```

模块定义文件:FINIT. DEF

```
NAME Finit

DESCRIPTION 'Initialization file demo'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
```

```
STACKSIZE 4096

EXPORTS FinitWndProc @4
    MealsDlgProc @5
```

资源描述文件:FINIT.RC

```
#include "finit.h"
#include "windows.h"

#include "meals.dlg"

Meals MENU
BEGIN
    POPUP "&Meal Selection"
    BEGIN
        MENUITEM "&Menu Plan…", ID_MEALS
    END
END
```

头文件:FINIT.H

```
#define ID_MEALS 10

#define ID_BEER 20
#define ID_JUICE 21
#define ID_WINE 22

#define ID_DAY 30

#define ID_STEAK 40
#define ID_FISH 41
#define ID_PIZZA 42
```

对话框文件:MEALS.DLG

```
MEALS DIALOG LOADONCALL MOVEABLE DISCARDABLE 51,47,235,123
CAPTION "Menu selection"
STYLE WS_BORDER | WS_CAPTION | WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Menu for:", -1,"static",SS_LEFT | WS_GROUP
        | WS_CHILD,21,16,65,9
    CONTROL "",ID_DAY,"edit",ES_LEFT | ES_AUTOHSCROLL
        | WS_BORDER | WS_TABSTOP | WS_CHILD,87,16,73,12
    CONTROL "Beverages",-1,"button",BS_GROUPBOX | WS_GROUP
        | WS_CHILD,14,47,54,68
    CONTROL "Beer",ID_BEER,"button",BS_AUTOCHECKBOX
        | WS_TABSTOP | WS_CHILD,22,61,30,12
    CONTROL "Juice",ID_JUICE,"button",BS_AUTOCHECKBOX
        | WS_CHILD,22,78,28,12
    CONTROL "Wine",ID_WINE, "button",BS_AUTOCHECKBOX
        | WS_CHILD,22,95,34,12
    CONTROL "Entrees",-1,"button",BS_GROUPBOX | WS_GROUP
```

```
        | WS_CHILD,101,47,54,67
    CONTROL "Steak", ID_STEAK,"button",BS_AUTORADIOBUTTON
        | WS_TABSTOP | WS_CHILD,110,62,36,12
    CONTROL "Fish", ID_FISH,"button",BS_AUTORADIOBUTTON
        | WS_CHILD,110,78,34,12
    CONTROL "Pizza", ID_PIZZA,"button",BS_AUTORADIOBUTTON
        | WS_CHILD,110,95,34,12
    CONTROL "OK",IDOK,"button",BS_DEFPUSHBUTTON | WS_GROUP
        | WS_TABSTOP | WS_CHILD,178,51,38,14
    CONTROL "Cancel",IDCANCEL,"button",BS_PUSHBUTTON
        | WS_TABSTOP | WS_CHILD,179,79,38,14
END
```

MAKE 文件：FINIT. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = meals.dlg
H = FINIT.H

all: FINIT.EXE

FINIT.RES: FINIT.RC $(RESFILES)

FINIT.OBJ: FINIT.C $(H)

FINIT.EXE: FINIT.OBJ FINIT.RES FINIT.DEF
    link /NOD/A:16 FINIT,FINIT,nul,libw slibcew,FINIT.def
    rc FINIT.res FINIT.exe
```

像下面这样运行 NMAKE 便得到应用程序 FINIT：

```
nmake fint.mak
```

FINIT. EXE 是怎样工作的

应用程序首先用函数 GetProfileString 检查 WIN. INI 中列出的一个现有的初始化文件。这个文件名列在关键字 FoodInit 下的类属[Finit]中。

如果不存在，则 Windows 把缺省串(由"\0"构成)拷贝到 szInitFileName 中。在这种情况下，函数 WriteProfileString 在文件 WIN. INI 中插入一个新的类属，并给关键字赋与名字"MEALS. INI"。若想使用其他文件名而不是"MEALS. INI"，则应该修改"Finit"。

下面这些入口被放到文件 WIN. INI 中：

文件 MEALS. INI 包含四个类属。第一个类属是[Init]，它包含关键字 Meals。如果它不是关键字，则文件中必然存在错误。

例如，这个文件可能是空的。如果是这种情况，可用函数 WritePrivateProfileString 重新初

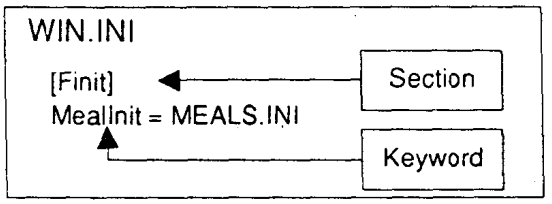


图 17.9 WIN. INI 中的入口

始化这个文件(即把所有类属写到这个文件中,文件 MEALS. INI 应该如下所示)。

```
[Init]
Meals=meals. ini
[Date]
Day=Sunday
[Food]
Entree=Steak
[Drink]
Beer=1
Juice=0
Wine=0
```

[Drink]关键字的值是整数,这些整数值被转化为 ASCII 字符,之后被函数 WritePrivateProfileString 插入。

不论文件 MEALS. INI 是否存在,函数 GetPrivateProfileString 和 GetPrivateProfileInt 都将读取信息并写入缓冲区。

键入主菜的第一个字母便可指明当前主菜,每种主菜和饮料都由对话按钮和 ID 值描述。变量 Entree 接受当前与主菜对应的值。

类属[Drink]包含每个关键字的值(0 和 1),函数 GetPrivateProfileInt 将这些整型值赋给相应的布尔变量。

选中 Menu Plan 菜单项将显示上图的对话框,对话框中的内容是从初始化文件中提取的值。消息 WM_INITDIALOG 在对话框函数中配置按钮和编辑控件。

当用户选中了一个新的单选按钮时,变量 wEntree 将记录下这个按钮的 ID 值。按下 OK 按钮确认三个关于饮料的布尔型变量为当前状态。编辑控件的内容将被写入变量 szReturnDay 中。

关闭这个对话框时,变量 wEnfree 的 ID 值所指示的"Steak"、"Fish"或" Pizza"将被拷贝到变量 szEntree 中,这个文本串和其他值都由函数 WritePrivateProfileString 写入文件 MEALS. INI 中。

函数 itoa 把三个关于饮料的布尔型变量转换为 ASCII 字符。函数 InvalidateRect 产生一个显示新选项的消息 WM_PAINT。

资源描述文件用一个弹出式菜单和一个菜单项描述用户刚定义好的菜单。语句 #include把这些特征加进应用程序中,头文件 FINIT. H 把所有的对话框控件描述为 ID 值。

第十八章　帮助系统

帮助(help)系统为用户提供了当前所运行的应用程序的联机信息。由于此联机帮助为程序提供了用户友好的界面，所以几乎所有 Windows 应用程序都包含基于 Windows Help 应用程序的帮助系统。

为生成这样的 Help 系统，以下两步是必要的：

(1)信息必须经过收集、编辑和处理；

(2)应用程序经适当编程。

大多数 Windows 应用程序用菜单中的 Help 项激活 Help 应用程序。

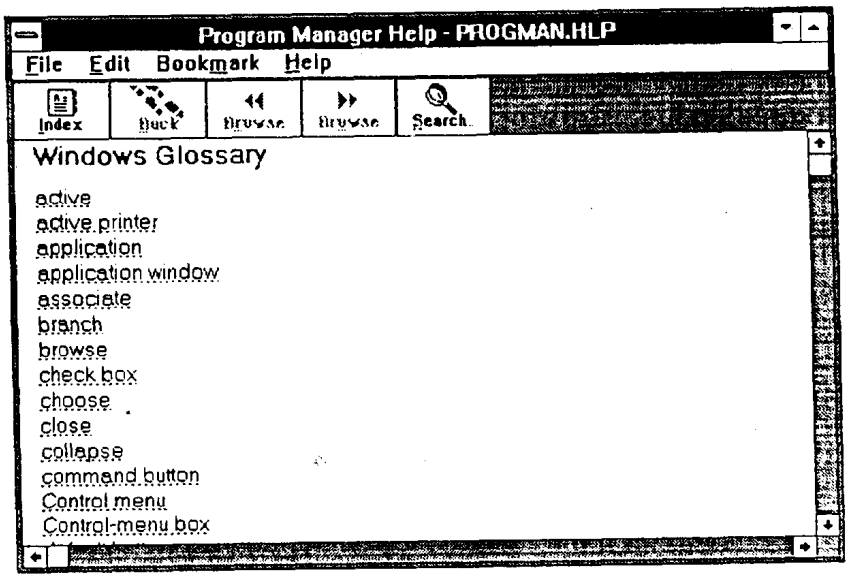


图 18.1　Help 系统举例

该菜单项可开辟一新窗口，并列出 Help 信息文本。Help 窗口都有相同的特点，即只显示相关部分的内容。

同其他窗口一样，可使用系统菜单关闭 Help 窗口。当应用程序结束之后，Help 窗口从屏幕上消失。

18.1　Help 应用程序

应用程序 WINHELP. EXE 存在于 Windows 软件包中。一旦用户选择了 Help 菜单或 Help

项，屏幕上就会显示 Help 窗口。该窗口中有五个按钮，用户可对其进行选择，以查询帮助信息。

这些按钮是：

Index Back 〈〈 〉〉 Search

而在有些系统中可能是下面这种情况：

Contents Search Back History 〈〈 〉〉

Help 文本确定可使用哪个按钮以及哪个可被加灰显示。Help 窗口有自己的弹出式菜单：File、Edit、Bookmark 和 Help。本节将对这些菜单及它们的菜单项进行讨论。

18.1.1 文件(File)

该菜单有四个菜单项，其中有两个用于显示信息，有一个用于关闭 Help 窗口。选择 Open …菜单项可以从其他 Windows 应用程序中装入 Help 文件。

18.1.2 编辑(Edit)

其中，Copy 项可将 Help 窗口中的当前文本存入剪贴板中。为了向已存在的 Help 文本中补充用户自己的注解，可使用 Annotate…项。使用带有编辑件可以控制显示，可在编辑控件的控制下输入用户自己的注解。

当用鼠标按钮选择 OK 后，对话框就消失了。“裁纸”图标插入当前文本中，表明这一主题包含用户的解释，选择 Annotate…项即可将它显示出来。

18.1.3 书签(Bookmark)

当然也可将 Bookmark 填加到任何主题中，然后利用这个 Bookmark 直接跳到主题处。Bookmark 由用户自己命名。

为了显示所有已存在的 Bookmark，可选用 Bookmark 弹出菜单。当用鼠标按钮选中某一 Bookmark 时，菜单会显示相应的主题。

18.1.4 帮助(Help)

WINHELP. EXE 应用程序也有自身的 Help 文件，它提供关于“Help”这一主题的基本信息。在选择 Using Help 菜单项后，Help 窗口将显示文本。

18.2 生成 Help 系统

本节将举例说明怎样生成 Help 系统，该应用程序只能用于启动 Help 系统和系统中的各项。下面将分节解释各项。

18.2.1 规划

Help 系统应具备向用户提供完整信息的功能。所以，在规划 Help 系统时，第一步就是收集信息。一般情况下，大部分信息是现成的。例如，可检测应用程序或从用户手册中提取可用信息，并加以收集整理。

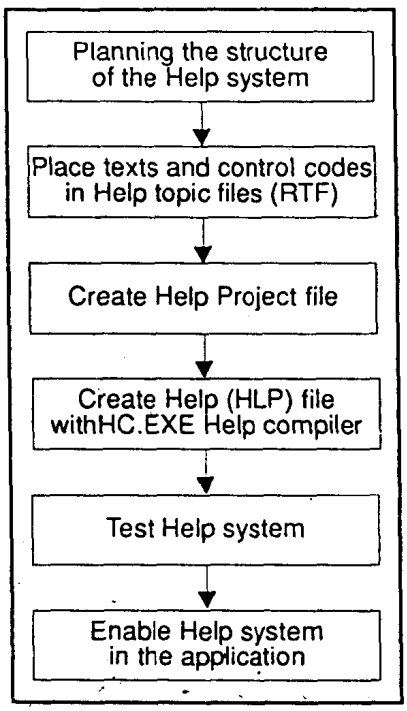


图 18.2　生成可执行 Help 系统

一旦拥有足够的基本信息，就可规划 Help 系统的结构了。独立主题以十六进制进行安排，因为 Help 索引总放在顶层。用户可以从顶层顺序移到底层。十六进制的层号决定于主题号和整个 Help 系统的大小。

例如，在我们的 Help 系统中，层次结构看起来如图 18.3 所示。

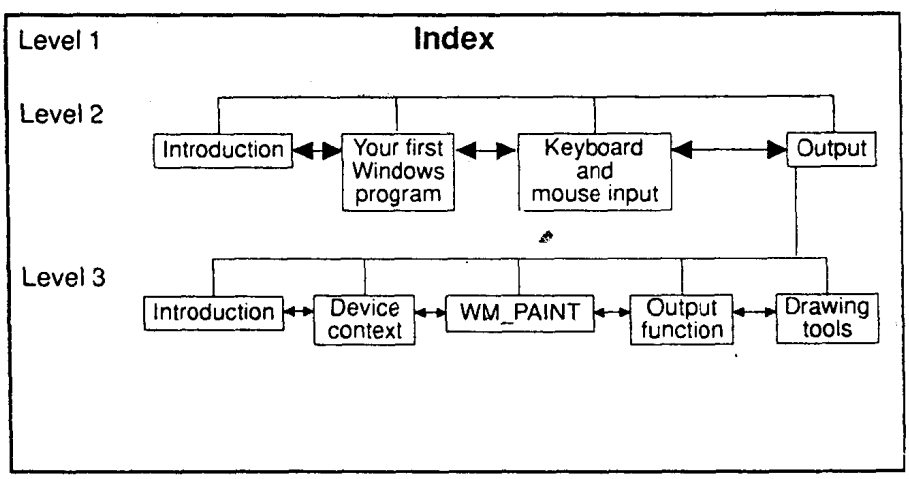


图 18.3　Help 系统示例中的层次

另外，如果主题在相同的层上，则用户也可直接从一个主题移动到下一个主题。接下来，要规划包含主题的文件的结构以及文件号。

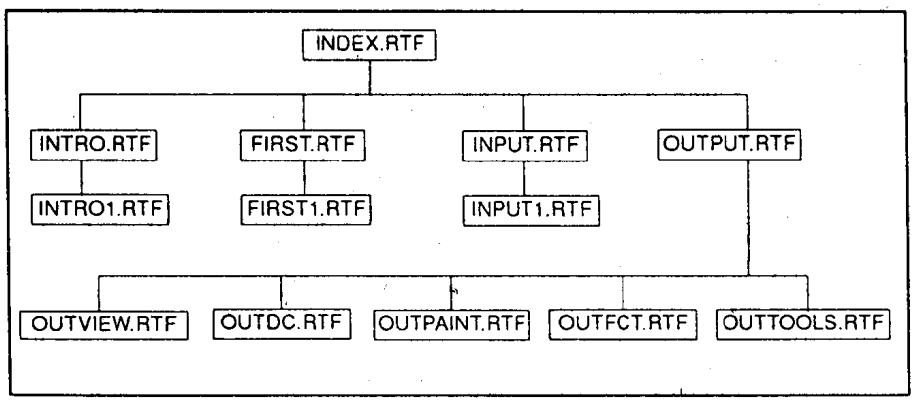


图 18.4　Help 系统索引

每个 Help 主题都描述了应用程序的一个方面，而且可存放在不同的文件中。如果几个主题混放一个独立的文件中，则可使用分页符将它们分离开来。

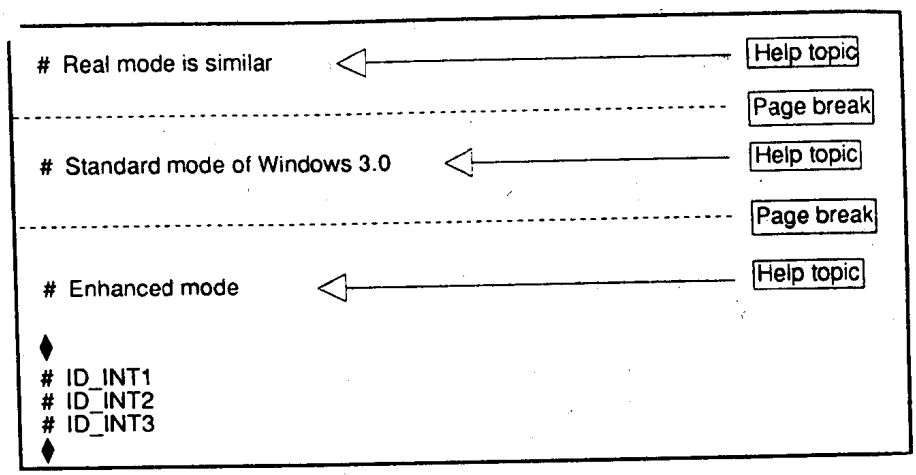


图 18.5　三个独立定义

在例子中，除了三个定义外，每个主题都使用独立文件。

所有文件都要以 RTF(Rich Text Format)格式存储。用户需要支持 Windows 的字处理器。

18.2.2　Help 主题文件

在开发 Help 系统时，时间主要花费在生成 Help 主题文件上。首先应以正确的文件结构存放文本，然后设置控制代码，以确定用户在 Help 系统范围内如何进行移动。当然，也可以在编辑文本或向已存在的文本加入补充信息时插入这些代码。

为了生成 Help 主题文件，就需要可以以 RTF 格式存储文件的编辑器。另外，还应该使用

注解、下划线和重点符号等形式进行编辑，这是由于它们是组成控制代码的属性。例如，Word 5.0 可用于生成一个 Help 例子，该字处理器满足这些要求。

上下文串

Help 系统中的每一主题都需要它自己的上下文串进行标识。合法的上下文串最长可达 255 个字母、数字和下划线，大小写字母均可。这个串用于在两个主题之间转移(见本章开头)。

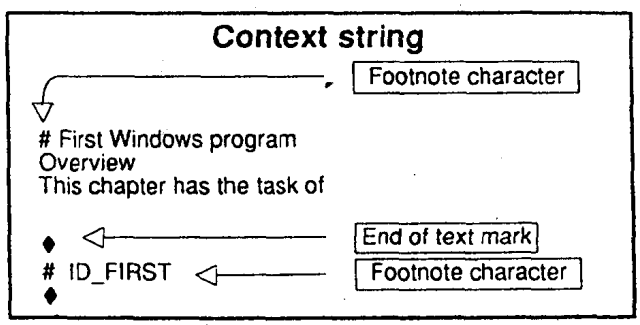


图 18.6 上下文串

"#"等注解符可以标志主题的开始。将一个上下文串分配给一个 Help 主题，然后输入空格和作为注解文本的上下文串。在"#"字符和串之间只能有一个空格。

标题

许多 Help 主题都有标题。本例中指的是在 Help 系统中所用的标题，而不是一般的文本第一行中的标题。

例如，在定义一个 Bookmark 时，用户提出一个标题作为模型。当对关键字的搜索成功后，它将在 Search 对话框的 Go to 表中显示出来。

为了定义一个标题，可在相应主题的文本开始处敲入 $ 注解符，然后把它作为主题注解文本写入标题中，当然它不能被格式化。

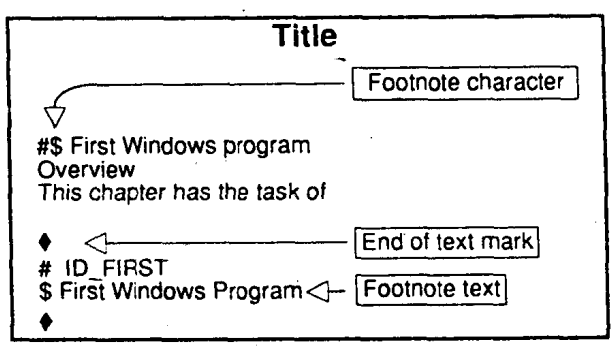


图 18.7 定义一个标题

尽管标题最长(包括空格)可为 127 个字符,但是建议用户将其长度限制在 40 个字符之内,这样可确保能在 Go to 表中显示出该标题。

关键字

使用 Help 系统时,最好的办法之一是搜寻某一特定项,然后直接跳到包含该项的段落处,关键字解决了这个问题。

用户可按需要为主题定义许多关键字。在选择 Search 按钮后,关键字会在列表框中显示出来。

另外,应用程序不区分大小写字母。如果几个主题有相同的关键字,"Go to"表将按字母顺序显示所有标题。

然后,用户就可选择自己需要的主题了。关键字的注解字符是大写字母"K"。敲入由连字符分隔的所有关键字作为主题的注解文本,该文本也可分为几行。

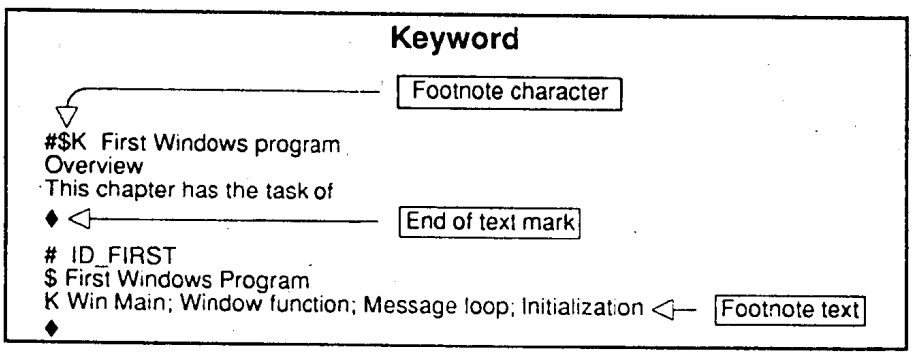


图 18.8 关键字

浏览序列号

"《"和"》"按钮使用户可以在相关主题间来回移动。

用以表示主题的顺序称为浏览序列。Help 系统的生成程序按序列号确定浏览序列。

注解符"+"用于浏览序列。序列号通常包括一个序列名和一个用冒号分开的数字。

所有同名序列号都属于一个浏览序列。若不指定名字,就要在空表中注解该主题。序列号确定了浏览的先后顺序。通常,序列号不按顺序分配。

如图所示,由于三个主题具有同一序列名,用户就可以从一个主题跳到下一个主题。

一旦达到序列表的开始处或结尾处,相关的按钮就自动置为 DISABLED 状态,这样就不能再使用它了。

交叉参考

使用交叉参考可以从最高层序列达到最底层,使用分离主题的上下文串可从一层跳到下一层。

首先,应确定想要跳转到的文本,该文本应该用"穿透"字符格式进行格式化。

然后,作为隐含文本,输入要转移的主题的上下文串,而隐含文本应直接同"穿透"文本

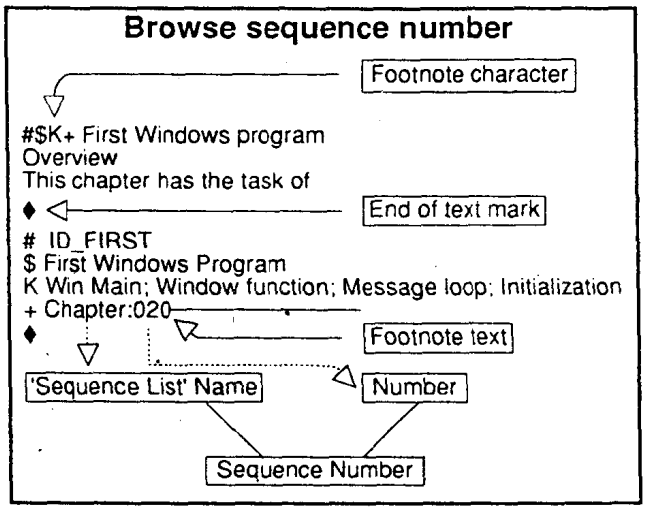


图 18.9　浏览序列号

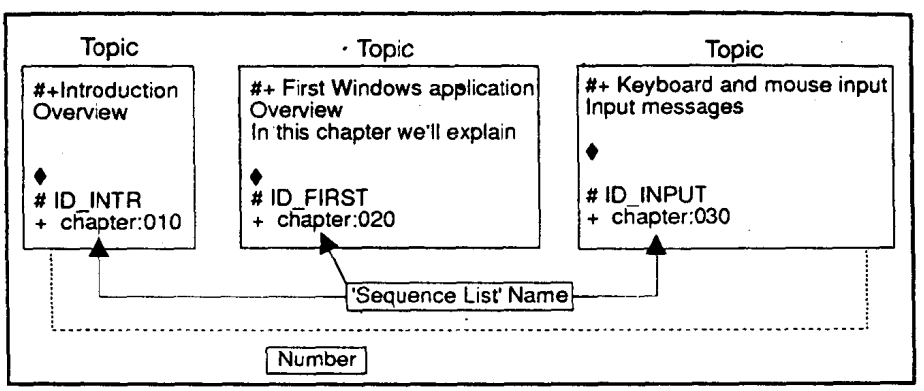


图 18.10　序列

连接。

在 Word 5.0 中，以穿透字符格式化的文本在显示时仍然是正常文本。在下面的例子中，穿透字符是可见的。

该例表明如何通过下划线文本进行转移。Help 编译器将穿透字符格式转换为下划线格式。

定义

主题文件通常包含定义过的单词或表达式(如外语单词)。

为了避免定义对文本的干扰，Help 系统使用户能在必要时才显示这些定义，且在包含定义的表达式下面加上着重线。

Introduction

Overview of Window 3.1

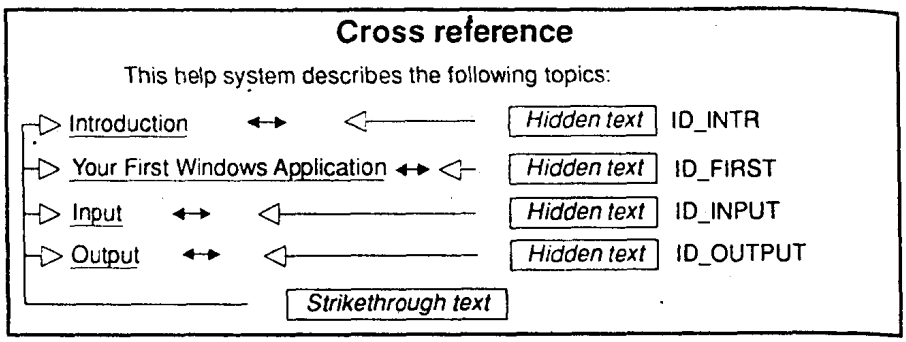


图 18.11 交叉参考

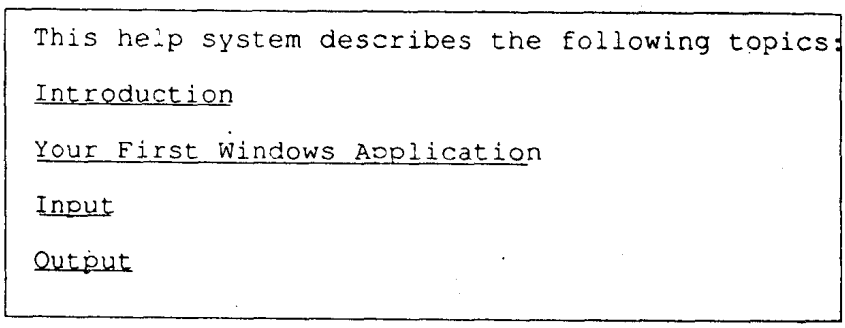


图 18.12

Windows 3.1 was introduced in 1992. The Windows 3.1 graphic user interface is considered a user environment of DOS that is capable of multitasking

Windows Modes

Windows 3.1 has two different modes: Standard mode and 386 Enhanced mode.

当用户用这种方法选择(用鼠标)带下划线的文本时,会显示一个附加信息框。在释放鼠标按钮时,附加文本将会消失。

在 Help 编译器中,这些附加定义是独立主题。所以,它们也需要上下文串。这些定义可以被组合到一个独立文件中(与它们所引用的主题文件同名),也可将每个定义放到独立的文件中。

具有定义的表达式按下划线字符格式显示,而提供该定义的上下文串以隐含文本格式存储。

在 Word 5.0 中,下划线格式文本可正常显示在屏幕上。因此,在上图中,这些字符是可见的。

BUILDTAG

BUILDTAG 是带有名字的开关,它所带的名字可插入到主题中。这样,Help 编译器就可在 Help 文件内建立主题。

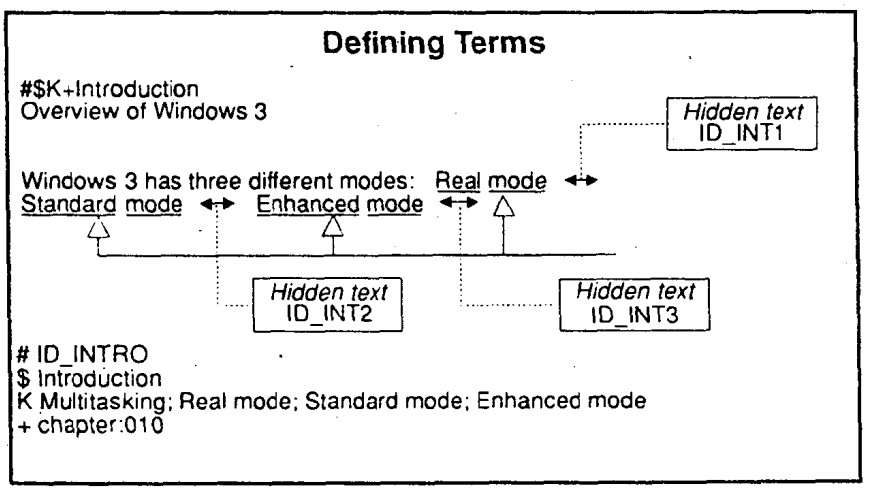


图 18.13 定义项

在为应用程序的不同版本或有更多属性的版本生成 Help 文件时，可以使用该选项。

注解符“*”位于主题的开始处，它为 BUILDTAG 所专用。将该注解符插入到所有其他注解符的前面可使 Help 编译器更有效地工作。

在插入“*”注解符时，此开关设置成 TRUE。另外，Help 项目文件中的所有开关都在[BUILDTAG]段中描述。现在，可以改变文件中的开关状态，此处所说的文件在[Options]段中的 BUILD 关键字下面(参见下一节)。

只有 BUILDTAG 为 TRUE 的主题才能被 Help 编译器编译到 Help 文件中。

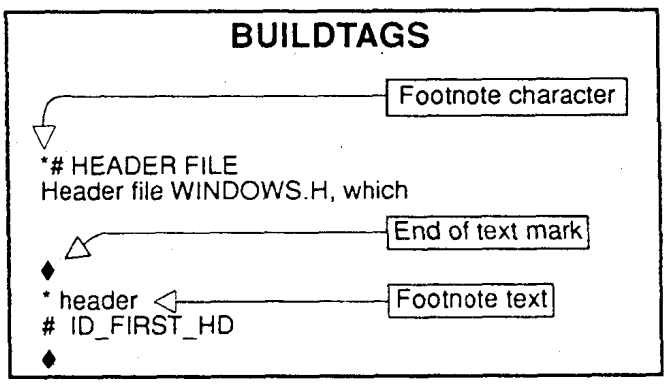


图 18.14 BUILDTAGS

所有无开关的主题被自动写入到 Help 文件中。因此，开关主要用于从 Help 文件中消除特定主题。为了消除一个主题，可将开关置为 FALSE，下例就是通过将开关设置为 FALSE 来消除文件主题的。

18.2.3 Help 项目文件

Help 项目文件包含 Help 编译器将 Help 主题文件转换成与 WINHELP.EXE 兼容的二进制 Help 文件的全部信息。

它是具有 .HPJ 扩展名的文本文件，由五段组成：

OPTIONS	列出选项
FILES	列出所需的所有主题文件
BUILDTAGS	给有效的建立标志赋值
MAP	添加上下文串
ALIAS	赋多个主题给同一上下文串

在补充的注解内容前应加一个分号。当 Help 编译器遇到分号时，会将当前分号之后的文本忽略掉。

[OPTIONS]

[OPTIONS]最多包含四个不同的选项。

WARNING 可指出 Help 编译器输出错误信息的级别。一共有 3 类出错信息(1,2,3)。当出错级别为 1 时，只输出严重出错信息。若选择级别 3，则显示所有出错信息。

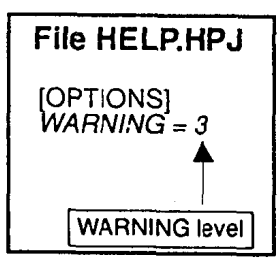


图 18.15 出错信息级别 3

根选项确定了 Help 项目的根目录。Help 编译器在根目录中搜寻 Help 主题文件。若不包含根目录，则编译器假定为当前目录。

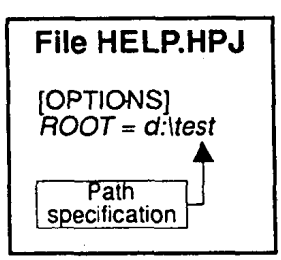


图 18.16 指定路径

假如 Help 项目文件只包含[FILES]部分，那么 Help 编译器输出最高层的第一个文件的

主题。

在这种情况下，索引不需要上下文串。当然也可以用 Index 选项设置索引，然后输入所需主题的上下文串。

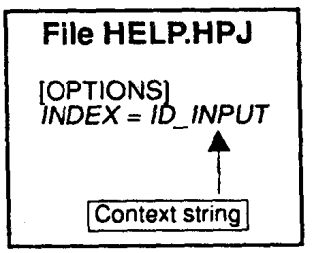


图 18.17　上下文串

为了找到索引主题，可在 Help 窗口中按 Index 按钮。

如果已在 Help 主题文件中建立了建立标志，则可以使用 Build 选项连接所需的主题。

利用逻辑操作符 OR(|)、AND(&)和 NOT(~)可以将所有开关连接在一起。在生成 Help 文件时，Help 编译器只承认开关值为 TRUE 的主题。

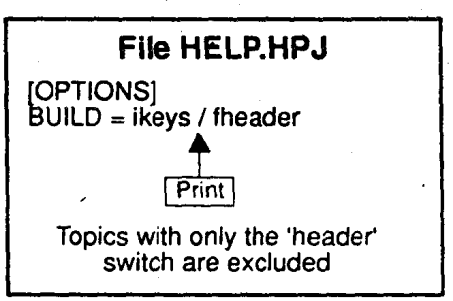


图 18.18

[FILES]

任何 Help 项目文件都有这一部分。[FILES]部分列出了开发者想要转换成二进制 Help 文件的 Help 主题文件。Help 编译器搜寻当前目录和由这些文件的 Root 选项指定的路径。如果未能找到文件，编译器将显示出错信息。下面是 HELP. HPJ 文件的例子。

```
[FILES]
INDEX. RTF ; Highest hierarchy level
INPUT. RTF ; Chapter: Keyboard and Mouse Input
OUTPUT. RTF ; Chapter: Output
FIRST. RTF ; Chapter: Your First Windows Application
```

一个大型 Help 系统可包含许多相互作用的文件。通常，这些文件的名字被存入头文件中并进行分类，这样就可以用包含语句(include)将该文件连接到[FILES]段中。下面是一个 HELPINCL. H 头文件的例子，修改过的 HELP. HPJ 文件可从头文件接收数据。

```
; HELPINCL. H
```

```
INDEX. RTF ; Highest hierarchy level
INDEX. RTF ; Chapter: Keyboard and Mouse Input
OUTPUT. RTF ; Chapter: Output
FIRST. RTF ; Chapter: Your First Wiindows Application

; HELP. HPJ
[FILES]
#include (HELPINCL. H)
```

[BUILDTAGS]

要使用 BUILDTAGS,Help 项目文件必须包含[BUILDTAGS]以及 Build 选项。这一部分列出了合法开关的名字,开关最多可有 30 个。

下面是 HELP. HPJ 文件中的[BUILDTAGS]部分:

```
[BUILDTAG]
iKey ;Topic:Keyboard input
fheader ;Topic:Header file
```

[MAP]

如果需要 Help 系统支持上下文敏感的帮助,就应该在 Help 项目文件中包含[MAP]部分。

上下文敏感的 Help 为客户区以外的主题提供 Help 文本。例如,用鼠标对菜单进行选择,可得到有关系统菜单的信息,当然,也可以按组合键访问这些信息。

为了调出上下文敏感的 Help,可按 Shift+F1 键。在按下了该组合键后,常规的指针光标中添加了一个问号,它指示当前处于 Help 控制下。

程序员应事先设计好光标转换以及在客户区之外按鼠标引出的信息。

Help 文本仅为相应的主题提供,使用上下文号可以连接相应的主题。

在[MAP]部分,上下文串由上下文号组合而成。然而,为了显示某特定主题,上下文号必须与运行应用程序时发送的上下文号匹配。

这些号可以为十进制,也可为十六进制,用户也可用 #define 语句来进行定义,#define 语句应写入独立的头文件中,使用 #include 语句就可将头文件连入程序中。

[ALIAS]

当同一上下文串被分配给多个主题时,就要使用[ALIAS]部分。例如,当需要用一个新主题替换两个旧主题时,就要使用[ALIAS]。

如果要删除旧主题而不使用[ALIAS]部分,必须在所有其他主题中搜寻非法交叉参考。为了避免这种情况,最好使用[ALIAS]。

被删除的主题的上下文串由新主题串覆盖。新上下文串也可以使用多次。

下例删除了三个主题为 Device、Context 和 Output 的文件,ID_DUT_DC,ID_DUT_FUNC 和 ID_DUT_TOOLS 三个上下文串分别标识这三个文件。现在要用 NEW. RTF 文件(其主题为 NEW,上下文串为 ID_NEW)来替代这三个文件。

这样,我们只需改变 Help 工程文件,而不用改变其他主题文件。如果在[MAP]部分使用了别名,则 Help 项目文件中的[ALIAS]部分应放在[MAP]部分之前。

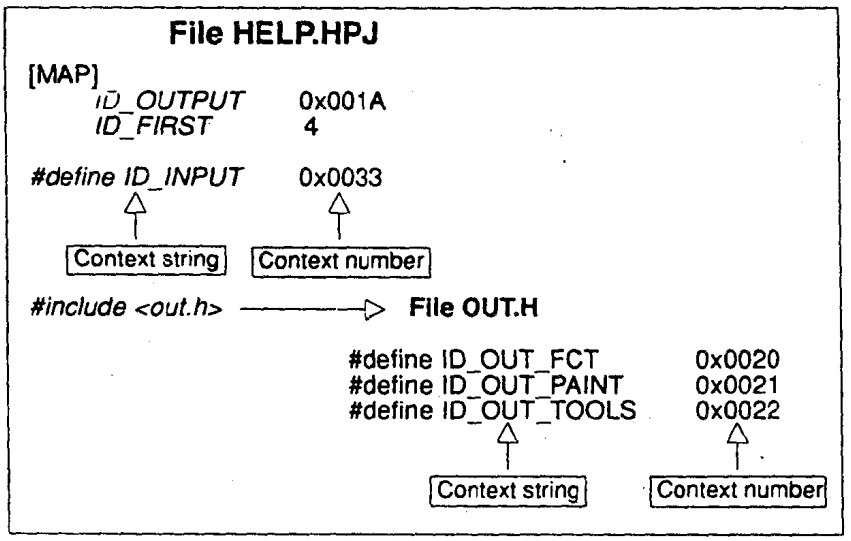


图 18.19 头文件

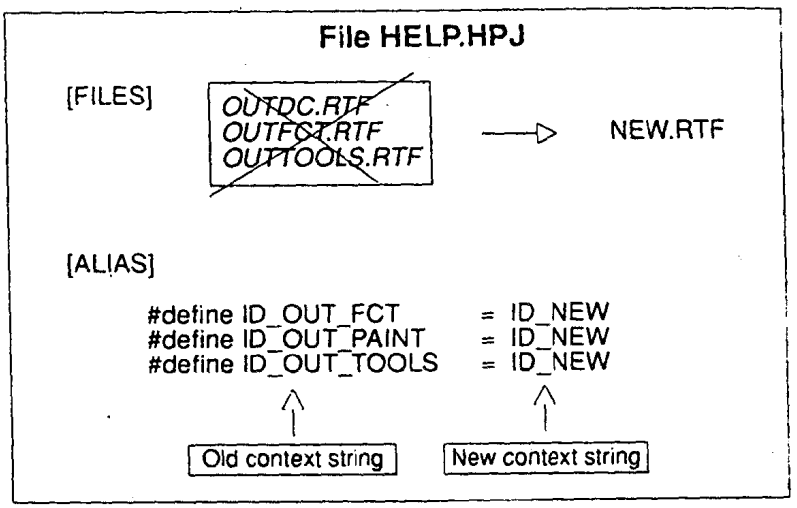


图 18.20 使用新上下文串替换旧上下文串

Help 编译器 HC31. EXE

生成 Help 项目文件之后，可以以文件名作参数调用 Help 编译器：

hc31 filename. hpj

生成的 Help 文件与 Help 项目文件同名，但扩展名为 . HLP。可以用 WINHELP. EXE 应用程序装入并读出这个新文件。

18. 2. 4 应用编程

为了从应用程序中激活 Help 系统，应该使用 WinHelp 函数。

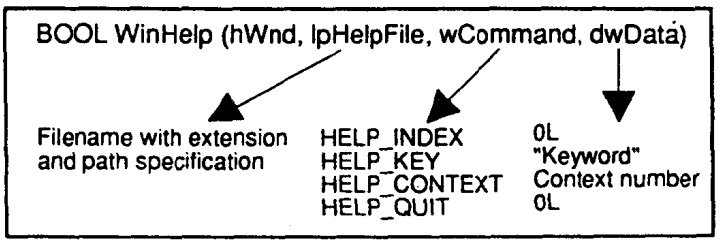


图 18.21 WinHelp 函数

一般地，提供 Help 服务的应用程序有一个 Help 弹出式菜单，该菜单包含 Index 菜单项。当用鼠标选择该项时，就会在 Help 窗口中显示索引主题。

WinHelp 函数的第三个参数值为 HELP_INDEX。dwData 参数值为 NULL。

```
WinHelp (hwnd,"help. hlp",HELP_INDEX,OL);
```

弹出式菜单通常还包含其他一些选项，用来转移到其他主题。在下例中，wCommand 参数值为 HELP_KEY。主题本身由一个关键字指定，该关键字应在适当的 Help 主题文件中输入。

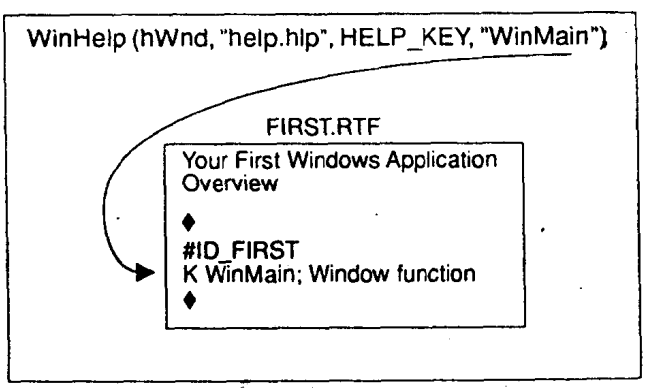


图 18.22 dwData 参数

如果在 Help 系统中已安装了上下文敏感的帮助，则 wCommand 参数值为 HELP_CONTEXT。应使用 Help 项目文件的[MAP]中定义的上下文号作为最后的参数。

如果要退出 Windows，应通知该 Help 应用程序。为了达到这一目的，应使用 WinHelp 函数。此时，wCommand 参数应设置为 HELP_QUIT。若没有其他应用程序使用 Help，就关闭 WINHELP. EXE。

18.3 Help 系统举例

前面已解释了应用程序的各个部分及其 Help 系统，下面的例子在 Windows 3.1 下进行工作(各种版本稍有不同)。

该应用程序包含一个 Help 弹出式菜单，其中有四个菜单项：Index，Your 1st Application，

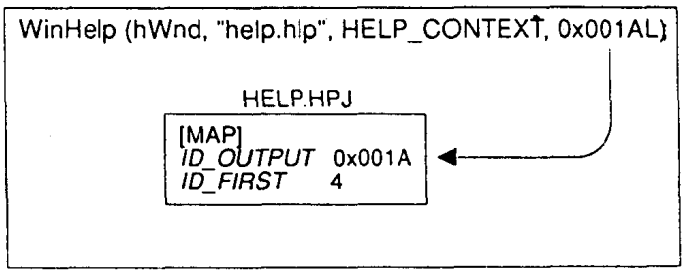


图 18.23 自由选择上下文号

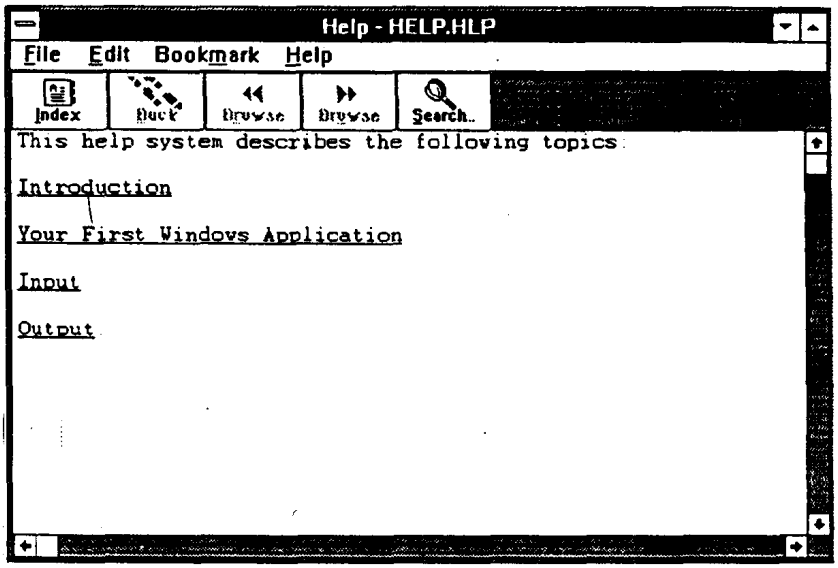


图 18.24

Input 和 Output。

源代码:HELP. C

```
/* * HELP. C * * *
```

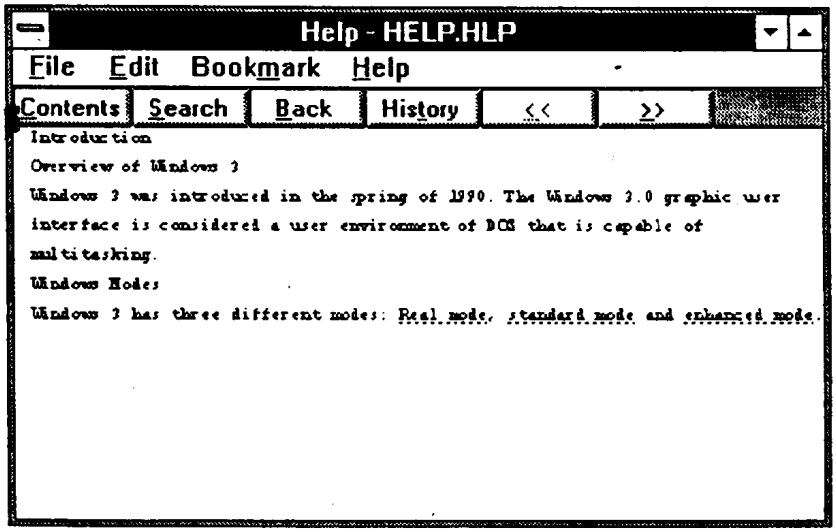


图 18.25

```
/* * WinMain (main function for every Windows application) * * * * * * * */

int PASCAL WinMain(hInstance,hPrevInstance,lpCmdLine,CmdShow)
HANDLE hInstance;                                    // Current instance
HANDLE hPrevInstance;                                // Previous instance
LPSTR lpCmdLine;                                     // Long ptr to string after name
int CmdShow;                                         // Specifies app. window
{
    MSG msg;                                         // Message variable
    HWND hWnd;                                       // Window handle
    HANDLE hAccel;                                   // Accelerator handle

    if (! hPrevInstance)                             // Initialize first instance
    {
        if (! HelpInit(hInstance))                   // If initialization fails
            return (FALSE);                          // return FALSE
    }

    /* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Help",                      // Window class name
            "Help system demo",                      // Window caption
            WS_OVERLAPPEDWINDOW,                     // Overlapped window
            CW_USEDEFAULT,                           // Default upper-left x pos.
            0,                                       // Upper-left y pos.
            CW_USEDEFAULT,                           // Default initial x size
            0,                                       // y size
            NULL,                                    // No parent window
            NULL,                                    // Window menu used
            hInstance,                               // Application instance
            NULL);                                   // No creation parameters
```

```
    ShowWindow(hWnd,CmdShow);                                   // Make window visible
    UpdateWindow(hWnd);                                         // Update window

    hAccel = LoadAccelerators(hInstance,"Help");                // Load kbd accelerators

    while (GetMessage(&msg,NULL,NULL,NULL))                     // Message reading
    {
        if (! TranslateAccelerator(hWnd,hAccel,&msg))
        {
            TranslateMessage(&msg);                             // Message translation
            DispatchMessage(&msg);                              // Send message to Windows
        }
    }
    return (msg. wParam);                                       // Return wParam of last message
}

BOOL HelpInit(hInstance)                                        // Initialize instance handle
HANDLE hInstance;                                               // Instance handle
{

    /* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcHelpClass;                                       // Main window class

    wcHelpClass. style = NULL;                                  // Horiz. & vert. redraw
    wcHelpClass. lpfnWndProc = HelpWndProc;                     // Window function
    wcHelpClass. cbClsExtra = 0;                                // No extra bytes
    wcHelpClass. cbWndExtra = 0;                                // No extra bytes
    wcHelpClass. hInstance = hInstance;                         // Instance
    wcHelpClass. hIcon = LoadIcon(NULL,IDI_APPLICATION);        // Icon
    wcHelpClass. hCursor = LoadCursor(NULL,IDC_ARROW);          // Cursor
    wcHelpClass. hbrBackground = GetStockObject(WHITE_BRUSH);   // Background
    wcHelpClass. lpszMenuName = "HelpMenu";                     // Help menu
    wcHelpClass. lpszClassName = "Help";                        // Window class

    if (! RegisterClass(&wcHelpClass))                          // Register window class
        return FALSE;
    return (TRUE);
}

/* * HelpWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL HelpWndProc(hWnd,message,wParam,lParam)
HWND hWnd;                                                      // Window handle
unsigned message;                                               // Message handle
WORD wParam;
LONG lParam;
{
    char chFileName[10] = "help. hlp";                          // Help file's name
```

```
    switch (message)                                          // Process messages
    {
        case WM_COMMAND:
            switch (wParam)
            {
                case IDM_HELP_INDEX:                                  // Get Index
                    WinHelp(hWnd,chFileName,HELP_INDEX,0L);
                    break;
                case IDM_HELP_FIRST:                          // Get First app. help
                    WinHelp(hWnd,chFileName,HELP_KEY,(DWORD)(LPSTR)"WinMain");
                    break;
                case IDM_HELP_INPUT:                           // Get Input help
                    WinHelp(hWnd,chFileName,HELP_CONTEXT,0x0001L);
                    break;
                case IDM_HELP_OUTPUT:                          // Get output help
                    WinHelp(hWnd,chFileName,HELP_CONTEXT,0x000aL);
                    break;
                default:                                  // Send other msgs here
                    return (DefWindowProc(hWnd,message,wParam,lParam));
            }
            break;
        case WM_DESTROY:                                          // Destroy window
            WinHelp(hWnd,chFileName,HELP_QUIT,0L);
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc(hWnd,message,wParam,lParam));
    }
    return (0L);
}
```

模块定义文件:HELP. DEF

```
NAME Help

DESCRIPTION 'Help example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE

DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS HelpWndProc
```

资源描述文件:HELP. RC

```
#include "windows. h"
```

```
#include "help.h"

HelpMenu MENU
BEGIN
    POPUP "&Help"
    BEGIN
        MENUITEM "&Index", IDM_HELP_INDEX
        MENUITEM "Your 1st &Application", IDM_HELP_FIRST
        MENUITEM "I&nput", IDM_HELP_INPUT
        MENUITEM "&Output", IDM_HELP_OUTPUT
    END
END

Help ACCELERATORS
BEGIN
    VK_F1, IDM_HELP_INDEX, VIRTKEY
END
```

头文件:HELP.H

```
#define IDM_HELP_INDEX 101
#define IDM_HELP_FIRST 102
#define IDM_HELP_INPUT 103
#define IDM_HELP_OUTPUT 104
```

在.RC 文件中,F1 功能键定义为加速键,按该键可激活 Help 系统。另外,加速键的两个必要的功能被插入程序文本中。在这两个功能中,WinHelp 函数的 wCommand 参数设置为 HELP_CONTEXT,这时 Help 并不是上下文敏感的。

另外,同 HELP_KEY 一样,也可以使用 HELP_CONTEXT,这样,只要为一个主题定义一个上下文号就可直接跳到该主题上。

用 Word 5.0 生成 13 个 Help 主题,然后将其转换为 RTF 格式。所有这些主题(除了索引)都有一个标题,其中一些还有关键字。

我们已在"规划"部分中说明了独立主题间的关系。在 Word 5.0 中,下划线格式的文本按正常文本显示。

Help 项目文件包括[FILES],[MAP],[BUILDTAGS]和[OPTIONS]等部分,定义在该文件中的两个[BUILDTAGS]用在 INPUT.RTF 和 FIRST1.RTF 文件中。由于[OPTIONS]部分的头文件开关设置为 FALSE,所以,该开关所包含的主题包含在 Help 文件中。

Help 项目文件:HELP.HPJ

```
[OPTIONS]
BUILD=ikeys | fheader

[FILES]
INDEX.RTF
OUTVIEW.RTF
OUTDC.RTF
OUTPAINT.RTF
INPUT.RTF
```

INDEX.RTF:

This Help system describes the following topics:

~~Introduction~~ID_INTR

~~Your First Windows Application~~ID_FIRST

~~Input~~ID_INPUT

~~Output~~ID_INPUT

♦

FIRST.RTF:

#$K+Your First Windows Application
Overview
In this chapter we'll explain the important

~~HEADER file~~ID_FRSTHD

♦

ID_FIRST
$ Your First Windows Application
K WinMain; Window function; Message loop; Initialization
+ chapter:020

♦

FIRST1.RTF:

*#Header file
The header file WINDOWS.H, found in

♦

*fheader
ID_FRSTHD

♦

图 18.26①

OUTPUT. RTF
OUTTOOLS. RTF
OUTFCT. RTF
INTRO. RTF
INTRO1. RTF
INPUT1. RTF
FIRST. RTF
FIRST1. RTF

[MAP]
#define ID_INPUT 0x0001
ID_OUTPUT 10

[BUILDTAGS]
ikeys ; Keyboard input

INTRO.RTF:

```
#$K+Introduction
Overview of Windows 3.1
Windows 3.1 was introduced in 1992. The Windows 3.1 graphic

Windows Modes
Windows 3.1 has two different modes: Standard modeID_INT1 and 386
Enhanced modeID_2
♦

# ID_INTR
$ Introduction
K Multitasking;Standard mode;386 Enhanced mode
+ chapter:010
♦
```

INTRO1.RTF:

```
# Standard mode exceeds the 1 Meg limit set by MS-DOS and
----------
# In 386 Enhanced mode, which can be used only on the 80386 or
----------

♦

# ID_INT1
# ID_INT2
♦
```

INPUT.RTF:

```
#$K+ Keyboard & mouse input
Input messages
In this chapter we'll describe the messages

~~Keyboard input~~ID_IN_KBD

♦

# ID_FIRST_HD
$ Keyboard and mouse input
K Queue; Mouse; Timer; Keyboard
+ chapter:030
♦
```

图 18.26②

fheader ; Header files

MAKE 文件:HELP.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc
```

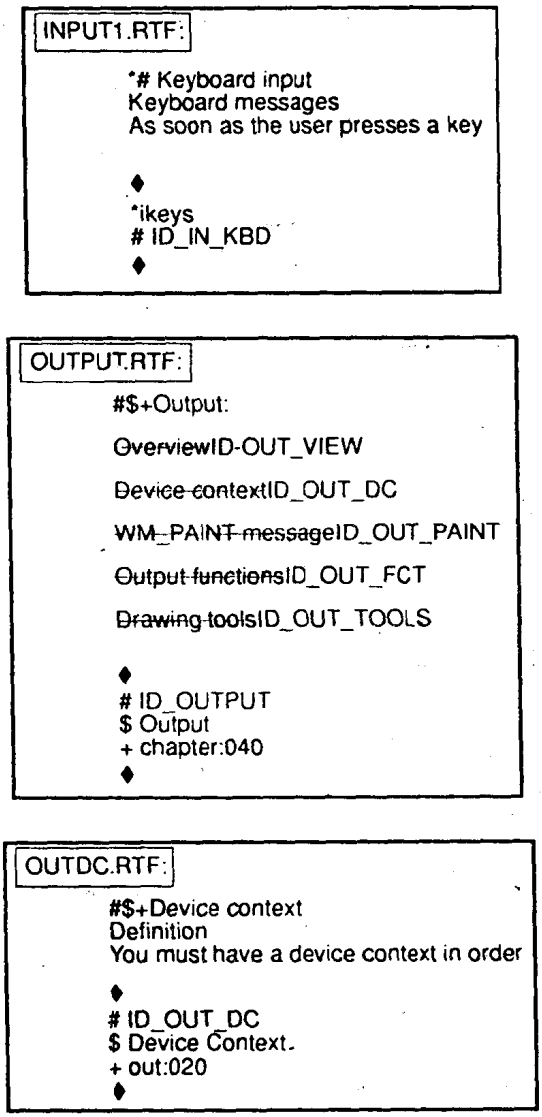


图 18.26③

RESFILES =

H = HELP.H

HELPFILES = intro.rtf intro1.rtf first.rtf first1.rtf index.rtf input.rtf input1.rtf outfct.rtf outview.rtf outdc.rtf outpaint.rtf output.rtf outtools.rtf

all: HELP.EXE

HELP.RES: HELP.RC $(RESFILES)

HELP.OBJ: HELP.C $(H)

HELP.EXE: HELP.OBJ HELP.RES HELP.DEF

OUTPAINT.RTF:

#$+KThe WM_PAINT message
In Windows applications you should

♦
ID_OUT_PAINT
$ WM_PAINT
+ out:030
K WM_PAINT; Queue
♦

OUTVIEW.RTF:

#$+Overview
In this chapter we'll examine

♦
ID_OUT_VIEW
$ Overview
+ out:010
♦

OUTFCT.RTF:

#$+Output functions
K The simpler output functions are divided into the basic text and

♦
ID_OUT_FCT
$ Output functions
+ out:040
K Text; Graphics; Device context; Mapping mode; Pixel; TextOut;
DrawText; DC
♦

OUTTOOLS.RTF:

#$+Drawing tools
Basics
It's possible to change some GDI

♦
ID_OUT_TOOLS
$ Tools
+ out:050
♦

图 18.26④

link /NOD/A:16 HELP,HELP,nul,libw slibcew,HELP.def
rc HELP.res HELP.exe

第十九章　多文档界面

多文档界面(MDI)是在单个应用程序中表示和处理多个文档的界面标准。

一个 MDI 应用程序包括一个主窗口，在主窗口里，用户可以打开若干个文档。每个文档出现在应用程序主窗口的子窗口内，子窗口包含一个系统菜单。

子窗口由各分级结构(如主窗口、客户窗口和子窗口)送来的信息定位。客户窗口取代了框架窗口中的客户区域，并由主窗口控制，它承担许多管理任务。

19.1　MDI 应用程序的结构

图 19.1 说明了 MDI 应用程序的基本结构。

类似于注册子窗口或者关于 MDICREATESTRUT 结构的要点将在下面几节逐一说明。

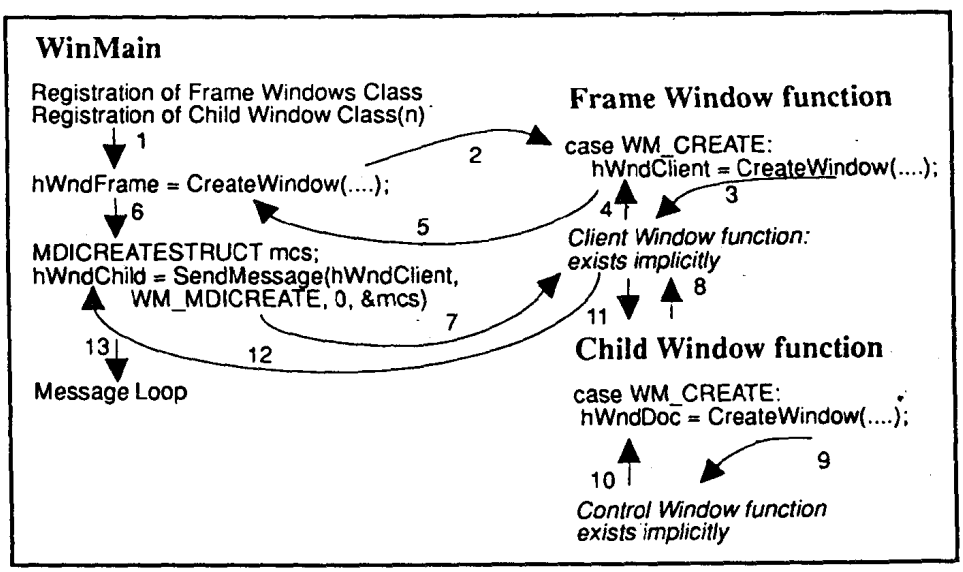


图 19.1　MDI 程序的基本结构

如果 MDI 子窗口要完成如“edit”和“list box”类控件一样的控制任务，那么就要安置一个拥有正确分类的控制窗口到 MDI 子窗口中。

在上图中，只建立了第一个子窗口，其他子窗口是由框架窗口的窗口函数产生的。客户窗口也可由 WinMain 函数建立。

19.2 补充和修改

在以前的 Windows 版本中(Windows 3.0 以前),建立 MDI 应用程序非常困难,因为程序员必须处理所有的细节。Windows 3.1 自己管理 MDI 窗口,但与通常的应用程序相比需要更多的时间和工作量来连接 MDI 功能。

19.2.1 消息循环

子窗口在系统菜单中也有加速键,不同的是,子窗口加速键使用的是 Control 键,而不是通常窗口的 ALT 键。例如,用户可以用 Ctrl+F4 加速键关闭子窗口。当用户准备编辑下一个文档时,可用 Ctrl+F6 组合键把它调到窗口中。

要使这些加速键有效,用户必须用 TranslateMDISysAccel 函数扩充消息循环,这个函数可以检查是否激活了子窗口的加速键。如果此函数在正确的键被按下之后接收到了 WM_KEYDOWN 消息,它就将它们翻译成 WM_SYSCOMMAND 消息,并将这些消息发送给活动子窗口。

然后此函数返回 TRUE 值,否则返回值为 False。TranslateAccelerator 函数能够检查是否激活了用户为框架窗口定义的加速键。

```
while( GetMessage ( &msg,NULL,0,0 ) )
{
    if( ! TranslateMDISysAccel ( hWndMDIClient,&msg ) &&
        ! TranslateAccelerator( hWndFrame,hAccel,&msg ) )
    {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
}
```

19.2.2 框架窗口

主窗口在 MDI 应用程序的框架窗口中被命名并且总是含有一个菜单。当用户注册框架窗口类时,没有任何特殊点需要考虑。当在 WinMain 函数中创建框架窗口时,用户应当将 WS_CLIPCHILDREN 风格参数传给 CreateWindow 函数。否则,客户窗口在输出时会被重写。

框架窗口的窗口函数不把未处理的所有消息传递给 DefWindowProc 缺省窗口函数,相反,它把它们传递给 DefFrameProc 函数,这个函数具有五个参数,而不是四个参数。

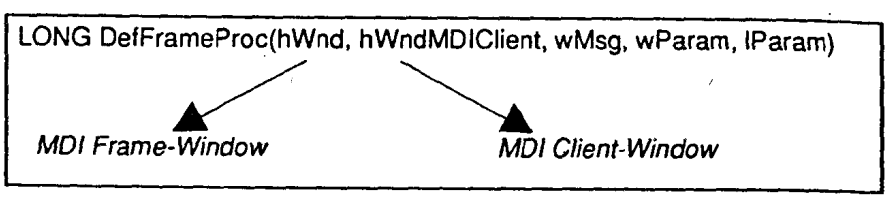


图 19.2 DefFrameProc 例程

由于是多文档界面,DefFrameProc 函数必须为一些消息完成其他任务,例如,如果函数

接收到 WM_SIZE 消息，它就重新设置客户窗口的大小以适应新的客户区。

此函数将 WM_SETFOCUS 消息传送给客户窗口函数，客户窗口将为活动的子窗口设置输入焦点，这就是必须把这个消息传递给 DefFrameProc 函数的原因（不管它们是否已经得到处理）。因选择子窗口标题而产生的所有 WM_COMMAND 消息也应传送给缺省函数，以便用户能通过菜单启动子窗口。

框架窗口也有一个菜单，在此菜单中 MDI 子窗口标题作为一项被列出来。在应用程序运行时，此菜单可以通过传送 WM_MDISETMENU 消息给 MDI 客户窗口得到交换。lParam 参数包含菜单和弹出式菜单的两个句柄，在菜单中子窗口标题应显示出来。

19.2.3 MDI 客户窗口

从用户的观点来看，客户窗口提供了显示子窗口的主窗口背景。对于程序员来说，这是一个命名为“mdiclient”的预定义类的窗口。因此，客户窗口也有预定义的窗口函数。

只有在创建框架窗口后才能创建客户窗口，因为框架窗口的句柄是 CreateWindow 函数的 hWndParent 参数，所以，客户窗口既可以在框架窗口创建后立即由 WinMain 函数创建，也可以由框架窗口函数的 WM_CREATE 消息创建。

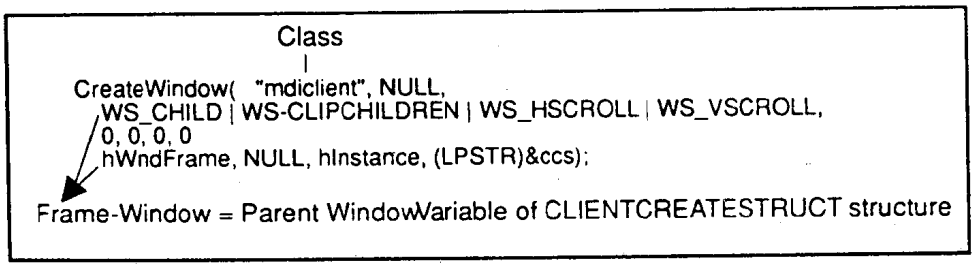


图 19.3

客户窗口根据两种风格参数（WS_HSCROLL 和 WS_VSCROLL）创建滚动条，但是，滚动条只有在框架窗口太小而不足以显示整个子窗口时才是可见的。

最后一个参数 lParam 指向 CLIENTCREATESTRUCT 结构变量，此结构包括两个字段：

hWindowMenu：附接 MDI 子窗口标题的菜单的句柄

idFirstChild：第一个 MDI 子窗口的 ID 值

弹出式菜单的句柄被赋给第一个字段，在此菜单中，子窗口标题应作为其选项之一。例如，用户可用 GetSubMenu 函数获得菜单句柄。

现在，用户可以从弹出式菜单中选择需要的标题，并在前台编辑文档。

第一个创建的 MDI 子窗口的 ID 值由第二个字段给定。对于后面的每个子窗口，其 ID 值都加 1。当一个子窗口关闭后，Windows 对其他窗口赋予新值，以免值域内产生空白。

由于子窗口标题像正常菜单项一样对待，因而当用户选择它们时，就产生一条 WM_COMMAND 消息，此消息被传送给框架窗口函数。

菜单项提供的 ID 值与 MDI 子窗口的 ID 值是相同的，所以，当给第一个窗口赋以 ID 值时，应保证这个 ID 值与其他菜单的 ID 值不发生冲突。

19.2.4 MDI 子窗口

MDI 应用程序里的子窗口与正常的主窗口非常相似，它们有系统菜单、最小化框和最大化框，并能重新设置窗口的大小。唯一的区别是用户可以从框架窗口中移走 MDI 子窗口。

一个 MDI 应用程序可为 MDI 子窗口设置一个或若干个类。子窗口的数目由应用程序任务的多少决定。

由于框架窗口中的子窗口都可以变成图标显示出来，所以用户应该传送一个图标句柄给类结构中的 hIcon 字段，也可以用 cbWndExtra 字段为特定窗口的数据保留一个特殊区域。

使用 GetWindowWord 和 GetWindowLong 函数可以从保留区中获得数据。要存储数据，则可使用 SetWindowWord 和 SetWindowLong 函数。除了处理数据之外，这些函数也可用来访问窗口和类结构中的特定字段。这两个函数处理长整型的 Long syllable。

因此，它们使用一个每次增加四个字节的偏移量。另外两个函数保存和装载 WORD 数据，它们使用一个双字节的偏移量，这个偏移量总是从它自己的数据的 Null 开始。

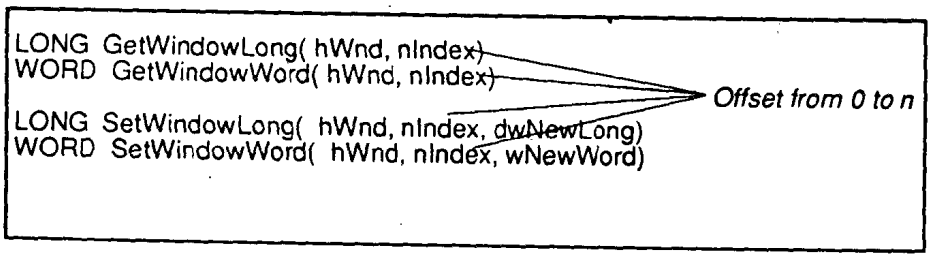


图 19.4

特殊的 MDI 消息控制 MDI 应用程序中的子窗口。通常，客户窗口接收到 MDI 消息，这个消息是在创建、关闭和启动窗口时产生的。例如，当从框架窗口的菜单中选择某个菜单项时即可产生消息。

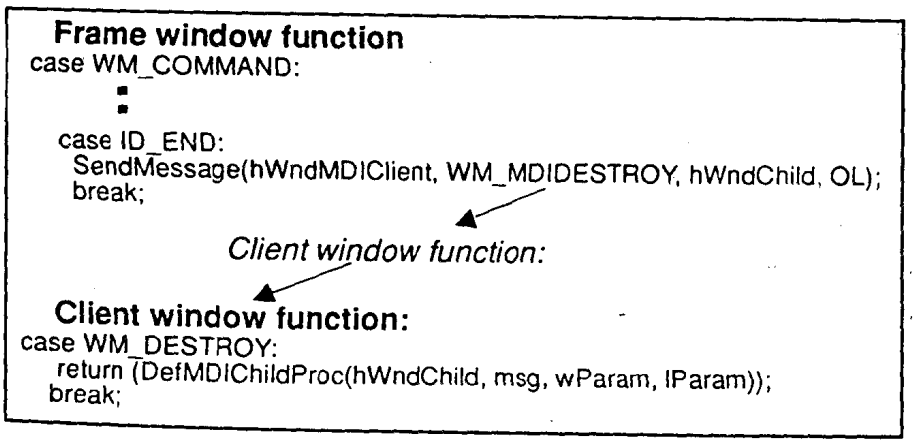


图 19.5 框架窗口函数

通常，消息只影响当前被激活的窗口，然而，有三个 MDI 消息影响所有的 MDI 子窗口，

因为它们重新排列了窗口。

- WM_MDICASCADE
- WM_MDIICONARRANGE
- MDI_TILE

wParam 和 lParam 两个参数总是置为 NULL。消息 WM_MDICASCADE 从左上角到右下角顺序安排所有子窗口。

但是，如果传递 WM_TILE 消息给客户区域，则所有子窗口并排排列。消息 WM_MDIICONARRANGE 在客户窗口中重新安排表示 MDI 子窗口的所有图标。

CreateWindow 函数不能创建 MDI 子窗口，相反，WM_MDICREATE 消息被传送给客户窗口，lParam 参数指向 MDICREATESTRUCT 结构变量。

szClass	指向已登录的 MDI 子窗口类
szTile	指向 MDI 子窗口标题
hOwner	实例句柄
x	MDI 子窗口左上角的 x 坐标
y	MDI 子窗口左上角的 y 坐标
cx	MDI 子窗口的初始宽度
cy	MDI 子窗口的初始高度
style	MDI 子窗口的风格

这个结构里的字段与 CreateWindow 函数的参数非常相似，这里需要一个指向前一个登录的类的指针和一个窗口标题。字段 hOwner 被赋值为当前实例句柄。

初始坐标和大小经常置为 CW_USEDEFAULT 值，所以 Windows 可以重叠地显示子窗口。最后一个字段可以是下面指定的四种风格参数的组合，但不能同时使用 WS_MAXIMIZE 和 WS_MINIMIZE，因为它们都指定了窗口的一个特殊的初始状态。

隐式 MDI 子窗口风格

WS_CHILD	WS_CAPTION
WS_CLIPSIBLINGS	WS_THICKFRAME
WS_SYSMENU	WS_MAXIMIZEBOX
WS_CLIPCHILDREN	WS_MINIMIZEBOX

任选 MDI 子窗口风格

WS_MAXIMIZE	WS_HSCROLL
WS_MINIMIZE	WS_VSCROLL

像 WS_CHILD 和 WS_SYSTEMMENU 一样的风格参数，在 CreateWindow 函数中都是必需的，当由 WM_MDICREATE 消息创建一个子窗口时，会自动使用这两个参数。

用户可以在任何时候把 WM_MDIMAXIMIZE 消息传送给客户窗口，从而将现有子窗口的客户区域扩大到整个客户窗口，然后 Windows 把子窗口的系统菜单放在框架窗口菜单条的左边，并把子窗口名写入框架窗口的标题条中。传送 WM_MDIRESTORE 消息可以使子窗口回到初始状态。

wParam 参数包含子窗口句柄，WM_MDINEXT 消息可以激活窗口列表中的下一个子窗口，并把激活的窗口放在所有其他子窗口后面。

- WM_MDIMAXIMIZE
- WM_MDIRESTORE
- WM_MDINEXT

由于 Windows 控制 MDI 应用程序运行，它在内部更多地使用消息，而不是应用程序本身。

WM_MDIACTIVE 消息可以传送给客户窗口或子窗口。当要启动另一个子窗口时，客户窗口就接收它，比如，当用户键入 Ctrl+F6 进入下一个文档时，这种情况就会发生。

启动子窗口后，客户窗口把消息传送给非活动窗口和新的活动窗口。lParam 参数包含两个窗口句柄。当子窗口为活动窗口时，wParam 参数为 TRUE。当消息传送给子窗口时，wParam 参数包含应当被激活的子窗口的句柄。

对于客户窗口：

wParam = 将被激活的子窗口的句柄

lParam = NULL

对于子窗口：

wParam = TRUE 或 FALSE

lParam = HIWORD(hWndNewActive)

LOWORD(hWndOldActive)

在子窗口的窗口函数中，既可以让活动子窗口接收到 WM_MDIACTIVATE 消息，从而把句柄保存下来，也可以把 WM_MDIGETACTIVE 消息传送给客户窗口，从而确定窗口句柄。

```
子窗口函数( hWnd,msg,wParam,lParam )
case WM_MEIACTIVATE:
    if( wParam )
        hWndActive = hWnd;
    break;
```

或

```
hWndActive=LOWORD(SendMessage(hWndClient,WM_MDIGETACTIVE,0,OL));
```

例如，如果按鼠标按钮使框架窗口处于活动状态，那么前面用 WM_MDIACTIVATE 消息启动的 MDI 子窗口就不会接收 MDI 消息。在相反的情况下，它只接收 WM_MDIACTIVATE 消息。

还有一个预定义的窗口函数叫作 DefMDIChildProc，它有四个参数：

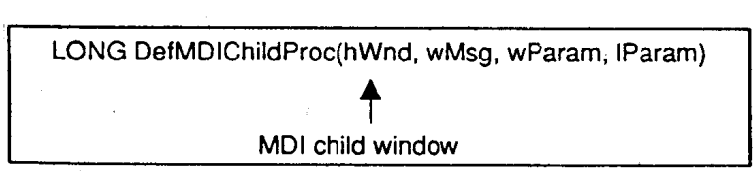


图 19.6　函数 DefMDIChildProc

WM_SIZE,WM_MOVE,WM_CLOSE 等特定消息必须传送给这个函数,MDI 应用程序才能正常运行。

例如,WM_CLOSE 消息使 DefMDIChildProc 函数把 WM_MDIDESTROY 消息传送给客户窗口,程序员在需要关闭一个 MDI 子窗口时也应传送这个 MDI 消息给客户窗口,否则,它会干扰 MDI 管理。

19.3 MDI 示例

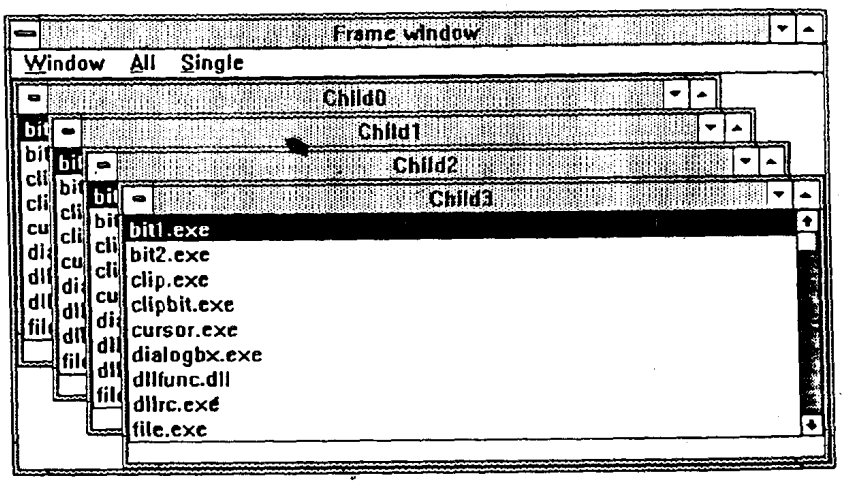


图 19.7

下面的应用程序是 MDI 的一个例子,它说明了 MDI 的基本过程。MDI 子窗口有它自己的子窗口,它们属于 Listbox 中预定义的类,并且完全覆盖了 MDI 子窗口。

当前目录中的所有文件都显示在列表框中。框架窗口中有一个菜单,可以用来改变 MDI 子窗口的状态。这个菜单允许用户创建、关闭和重新安排子窗口。

虽然整个菜单可以被另一个所替代,但现有子窗口标题将作为菜单项。

新函数	描述
LB_GETCURSEL	提供从列表框中选出的项的下标
LB_SETCURSEL	选择列表框中的串
WM_MDIACTIVATE	激活 MDI 子窗口
WM_MDICASCADE	重叠放置 MDI 子窗口
WM_MDIDESTROY	创建 MDI 子窗口
WM_MDICLOSE	关闭 MDI 子窗口
WM_MDIGETACTIVE	提供活动子窗口句柄
WM_MDIICONARRANGE	安放所有 MDI 子窗口图标
WM_MDIMAXIMIZE	设置 MDI 子窗口为最大状态
WM_MDINEXT	激活下一个 MDI 子窗口
WM_MDIRESTORE	恢复 MDI 子窗口为前一个大小

WM_MDISETMENU	修改框架窗口菜单
WM_MDITILE	并列放置 MDI 子窗口

新函数	描述
DlgDirList	用指定目录填充列表框
DrawMenuBar	重画菜单条
GetWindow	在 Window Manager List 中查找句柄
MoveWindow	引发 WM_SIZE 消息
SetWindowWord	修改窗口属性
TranslateMDISysAccel	转换 MDI 子窗口的加速键

新结构	描述
CLIENTCREATESTRUCT	包含 MDI 客户窗口的参数
MDICREATESTRUCT	包含 MDI 子窗口的参数

源代码:MDI.C

```
/* MDI.C ************************/
/* Demonstrates MDI (Multiple Document Interface) access by generating */
/* child windows, frame windows and list boxes */
/************************/

#include "windows.h"                                    // Include windows.h header file
#include "mdi.h"                                        // Include mdi.h header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

HWND hwndMDIClient = NULL;                              // Window handle to MDI client
                                                        // window

BOOL MDIInit(HANDLE);
LONG FAR PASCAL FrameWndProc(HWND,unsigned,WORD,LONG);
LONG FAR PASCAL MDIChildWndProc(HWND,unsigned,WORD,LONG);

/* WinMain (main function for every Windows application ********/

int PASCAL WinMain(hInstance,hPrevInstance,lpszCmdLine,CmdShow)
HANDLE hInstance;                                       // Current instance
HANDLE hPrevInstance;                                   // Previous instance
LPSTR lpszCmdLine;                                      // Ptr to string after prg. name
int CmdShow;                                            // Specify app. window appearance
{
    MSG msg;                                            // Message variable
    HANDLE hAccel;                                      // Accelerator handle
    HWND hwndChild,hwndFrame = NULL;                    // Window handle
```

```
MDICREATESTRUCT mcs;                                    // Create structure variable

if (! hPrevInstance)                                    // Initialize first instance
{
    if (! MDIInit(hInstance))                           // If initialization fails
        return FALSE;                                   // return FALSE
}

/* * Specify appearance of application's main window * * * * * * * * * */

hwndFrame = CreateWindow ("Frame",                      // Window class name
        "Frame window",                                 // Window caption
        WS_OVERLAPPEDWINDOW |                           // Overlapped window style
        WS_CLIPCHILDREN,                                // child window style
        CW_USEDEFAULT,                                  // Default upper-left x pos.
        0,                                              // Upper-left y pos.
        CW_USEDEFAULT,                                  // Default initial x size
        0,                                              // y size
        NULL,                                           // No parent window
        NULL,                                           // Window menu used
        hInstance,                                      // Application instance
        NULL);                                          // No creation parameters

if (! hwndFrame)                                        // If frame window fails
    return FALSE;                                       // return FALSE

if (! hwndMDIClient)                                    // If MDI client window fails
    return FALSE;                                       // return FALSE

hAccel = LoadAccelerators (hInstance,"MDIMenu");        // Get accelerators
                                                        // for MDI
ShowWindow (hwndFrame,CmdShow);                         // Make window visible
UpdateWindow (hwndFrame);                               // Update window

/* * MDICREATESTRUCT variables * * * * * * * * * * * * * * * * * * */

mcs.szTitle = (LPSTR)"Child0";                          // MDI child window's title
mcs.szClass = "Child";                                  // MDI child window's class name
mcs.hOwner = hInstance;                                 // Instance handle
mcs.x = mcs.cx = 0;                                     // x = MDI child window's
                                                        // upper-left x position;
                                                        // cx = MDI child window's
                                                        // initial width
mcs.y = mcs.cy = 0;                                     // y = MDI child window's
                                                        // upper-left y position;
                                                        // cy = MDI child window's
                                                        // initial height
mcs.style = WS_MAXIMIZE;                                // Additional MDI child window
                                                        // styles (maximize window)

hwndChild = (WORD)SendMessage (hwndMDIClient,           // Generate and send
```

```
            WM_MDICREATE,                                   // message to child
            0,                                                  // window
            (LONG)
            (LPMDICREATESTRUCT)
            &mcs);

    while (GetMessage (&msg,NULL,0,0))                      // Message reading
    {
        if ( ! TranslateMDISysAccel (hwndMDIClient,&msg) &&
            ! TranslateAccelerator (hwndFrame,hAccel,&msg))
                                                        // Read accelerator data
        {
            TranslateMessage (&msg);                        // Message translation
            DispatchMessage (&msg);                     // Send message to Windows
        }
    }
    return 0;
}

BOOL MDIInit(HANDLE hInstance)                          // Initialize instance handle
{
    /* * Specify window classes * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcMDIClass;                                    // Main window class

    wcMDIClass.style = 0;                                   // Horizontal & vertical
                                                        // redraw of client area
    wcMDIClass.lpfnWndProc = FrameWndProc;                  // Window function
    wcMDIClass.cbClsExtra = 0;                              // No extra bytes
    wcMDIClass.cbWndExtra = 0;                              // No extra bytes
    wcMDIClass.hInstance = hInstance;                           // Instance
    wcMDIClass.hIcon = LoadIcon(NULL,IDI_APPLICATION);          // Icon
    wcMDIClass.hCursor = LoadCursor(NULL,IDC_ARROW);            // Cursor
    wcMDIClass.hbrBackground = GetStockObject(WHITE_BRUSH);     // Background
    wcMDIClass.lpszMenuName = "MDIMenu";                        // Menu
    wcMDIClass.lpszClassName = "Frame";                     // Window class

    if (! RegisterClass (&wcMDIClass) )                     // Register window class
        return FALSE;                               // Return FALSE if registration fails

    /* * Child window information * * * * * * * * * * * * * * * * * */

    wcMDIClass.lpfnWndProc = MDIChildWndProc;               // Window function
    wcMDIClass.hIcon = LoadIcon(NULL,IDI_HAND);                 // Icon
    wcMDIClass.lpszMenuName = NULL;                         // No menu
    wcMDIClass.cbWndExtra = 4;                              // Extra bytes
    wcMDIClass.lpszClassName = "Child";                     // Window class

    if (! RegisterClass(&wcMDIClass))                       // Register window class
        return FALSE;                               // Return FALSE if registration fails
                                                        // If registration is successful,
```

```
    return TRUE;                                                    // Return TRUE
}

/* * FrameWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

LONG FAR PASCAL FrameWndProc ( hwnd,message,wParam,lParam )
HWND hwnd;                                                          // Window handle
unsigned message;                                                   // Message type
WORD wParam;                                                        // Message-dependent 16 bit value
LONG lParam;                                                        // Message-dependent 32 bit value
{
    CLIENTCREATESTRUCT ccs;                                         // Client creation structure
    MDICREATESTRUCT mcs;                                            // MDI creation structure
    HWND hWndChild;                                                 // Handle to child window(s)
    static HANDLE hInst;                                            // Instance handle
    static int i = 1;                                               // i variable
    char wntitle[10];                                               // Window title
    static HMENU hMenu1,hMenu2;                                     // Menu handles

    switch (message)                                                // Process messages
    {
        case WM_CREATE:                                             // Create window
            hInst = GetWindowWord(hwnd,GWW_HINSTANCE);
            hMenu2 = LoadMenu(hInst,"MDI2Menu");                    // Second menu
            hMenu1 = GetMenu(hwnd);                                 // First menu
            ccs.hWindowMenu = GetSubMenu (hMenu1,0);
            ccs.idFirstChild = ID_FIRST;

            /* * Specify appearance of applications child windows * * * * * * * * */
            hwndMDIClient =
                CreateWindow ("mdiclient",                          // Window class name
                    NULL,                                           // Window caption
                    WS_CHILD|                                       // Child window
                    WS_CLIPCHILDREN|                                // Repaint main window
                                                                    // without deleting
                                                                    // child windows
                    WS_VSCROLL|                                     // Vertical scroll bar
                    WS_HSCROLL,                                     // Horizontal scroll bar
                    0,0,                                            // Upper-left x and y
                    0,0,                                            // Initial x and y sizes
                    hwnd,                                           // Parent window
                    NULL,                                           // No menu
                    hInst,                                          // Application instance
                    (LPSTR)&ccs);                                   // Creation parameters

            ShowWindow (hwndMDIClient,SW_SHOW);// Make window visible
            break;
        case WM_COMMAND:                                            // Menu and command access
            switch (wParam)
```

```
{
    case ID_NEW:                                    // [New Window] menu item
                                                    // Specifies new win. parameter

        wsprintf(wntitle,"Child%i",i++);
        mcs.szTitle = (LPSTR)wntitle;
        mcs.szClass = "Child";
        mcs.hOwner = hInst;
        mcs.x = mcs.cx = CW_USEDEFAULT;
        mcs.y = mcs.cy = CW_USEDEFAULT;
        mcs.style = 0;

        hWndChild = (WORD)SendMessage
                (hwndMDIClient,
                WM_MDICREATE,
                0,
                (LONG)
                (LPMDICREATESTRUCT)&mcs);
        break;
    case ID_TILE:                                   // [Tile] menu item
        SendMessage (hwndMDIClient,// Send msg
                WM_MDITILE,                         // to tile
                0,                                  // existing
                0L);                                // clients
        break;
    case ID_CASC:                                   // [Cascade] menu item
        SendMessage (hwndMDIClient,// Send msg
                WM_MDICASCADE,// cascade
                0,                                  // existing
                0L);                                // clients
        break;
    case ID_ICON:                                   // [Arrange Icons] menu item
        SendMessage (hwndMDIClient,// Send msg
                WM_MDIICONARRANGE,                  // to
                0,                                  // arrange
                0L);                                // minimized
        break;                                      // icons
    case ID_MAXI:                                   // [Maximize] menu item
        hWndChild = LOWORD(SendMessage
                (hwndMDIClient,
                WM_MDIGETACTIVE,
                0,
                0L));
        SendMessage (hwndMDIClient,
                WM_MDIMAXIMIZE,
                hWndChild,
                0L);
        break;
    case ID_REST:                                   // [Restore] menu item
        hWndChild = LOWORD(SendMessage
                (hwndMDIClient,
                WM_MDIGETACTIVE,
```

```
                0,
                0L));
        SendMessage (hwndMDIClient,
                WM_MDIRESTORE,
                hWndChild,
                0L);
        break;
    case ID_NEXT:                                   // [Next] menu item
        SendMessage (hwndMDIClient,
                WM_MDINEXT,
                0,
                0L);
        break;
    case ID_MENU1:                                  // [First Menu] menu item
        SendMessage (hwndMDIClient,
                WM_MDISETMENU,
                0,
                MAKELONG(hMenu2,
                    GetSubMenu
                    (hMenu2,0)));
        DrawMenuBar(hwnd);
        break;
    case ID_MENU2:                                  // [Toggle] menu item
        SendMessage (hwndMDIClient,
                WM_MDISETMENU,
                0,
                MAKELONG(hMenu1,
                    GetSubMenu
                    (hMenu1,0)));
        DrawMenuBar(hwnd);
        break;
    case ID_CLOSE:                                  // [Close] menu item
        ShowWindow(hwndMDIClient,SW_HIDE);
                                                    // Hide client window to avoid
                                                        // unnecessary repainting

        while (hWndChild = GetWindow
                (hwndMDIClient,
                GW_CHILD))
                                                // Search for MDI child windows
                                            // and destroy using WM_MDIDESTROY
            SendMessage (hwndMDIClient,
                    WM_MDIDESTROY,
                    (WORD)hWndChild,
                    0L);
                                                    // Window? Destroy window
        ShowWindow( hwndMDIClient,SW_SHOW);
                                                // Make client window visible again
        break;
    default:
        DefFrameProc(hwnd,
```

```
                        hwndMDIClient,
                        WM_COMMAND,
                        wParam,
                        0L);
            }
            break;
        case WM_DESTROY:                                        // End
            PostQuitMessage (0);
            break;
        default:
            return DefFrameProc (hwnd,
                    hwndMDIClient,
                    message,
                    wParam,
                    lParam);
    }
    return 0;
}

/* * Child window function * * * * * * * * * * * * * * * * * * */

LONG FAR PASCAL MDIChildWndProc ( hwnd,message,wParam,lParam )
HWND hwnd;                                                  // Window handle
unsigned message;                                           // Message type
WORD wParam;                                    // Message-dependent 16 bit value
LONG lParam;                                    // Message-dependent 32 bit value
{
    HANDLE hInst;                                           // Instance handle
    HWND hwndList;                                          // Window list handle
    WORD Index;                                             // Index variable

    switch (message)                                        // Process messages
    {
        case WM_CREATE:                                     // Create window
            hInst = GetWindowWord(hwnd,GWW_HINSTANCE);
            hwndList = CreateWindow ("listbox",             // Win class name
                    NULL,                                   // Caption
                    WS_CHILD|                               // Child window
                    WS_VISIBLE|                             // styles
                    LBS_STANDARD,                           // Common styles
                    0,                                      // Upper-left x
                    0,                                      // Upper-left y
                    0,                                      // Initial x
                    0,                                      // Initial y
                    hwnd,                                   // Parent window
                    ID_LIST,                                // Menu used
                    hInst,                                  // App. instance
                    NULL);                                  // No creation parameters

            /* * List directory and set focus to this window * * * * * * * * * * */
```

```
            DlgDirList(hwnd," * . * ",ID_LIST,0,0x10 | 0x4000);
            SetWindowWord (hwnd,0,(WORD)hwndList);
            SetWindowWord (hwnd,2,(WORD)0);
            SetFocus(hwndList);                                    // Set input focus
            break;
        case WM_MDIACTIVATE:                                       // MDI activation message
            if (wParam)
            {
                Index = GetWindowWord (hwnd,2);
                SetWindowWord(hwnd,2,(WORD)Index+1);
                hwndList = GetWindowWord (hwnd,0);
                SendMessage(hwndList,
                        LB_SETCURSEL,
                        Index+1,
                        0L);
            }
            break;
        case WM_SIZE:                                              // Sizing message
            hwndList = GetWindowWord (hwnd,0);                     // Get control ID
            MoveWindow (hwndList,0,0,LOWORD(lParam),
                    HIWORD(lParam),TRUE);                          // Move controls
            return DefMDIChildProc (hwnd,                          // Pass unprocessed
                    message,                                       // messages to this
                    wParam,                                        // function
                    lParam);
            break;
        case WM_SETFOCUS:                                          // Set input focus
            hwndList= GetWindowWord (hwnd,0);
            SetFocus (hwndList);
            break;
        case WM_COMMAND:                                           // Commands
            switch (wParam)
            {
                case ID_LIST:                                      // List menu
                    Index = (WORD)SendMessage(GetWindowWord
                            (hwnd,0),
                            LB_GETCURSEL,
                            0,
                            0L);
                    SetWindowWord(hwnd,2,(WORD)Index);
                    break;
                default:
                    break;
            }
            break;
        default:
            return DefMDIChildProc (hwnd,message,wParam,lParam);
    }
    return FALSE;
}
```

模块定义文件:MDI. DEF

```
NAME MDI

DESCRIPTION 'MDI example'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 2048
STACKSIZE 8192

EXPORTS FrameWndProc
    MDIChildWndProc
```

资源描述文件:MDI. RC

```
#include "windows. h"
#include "mdi. h"

MDIMenu MENU
BEGIN
    POPUP "&Window"
    BEGIN
        MENUITEM "&New Window\tShift+F8",ID_NEW
        MENUITEM "New &Menu", ID_MENU1
    END

    POPUP "&All"
    BEGIN
        MENUITEM "&Tile\tCtrl+T", ID_TILE
        MENUITEM "&Cascade\tCtrl+C", ID_CASC
        MENUITEM "&Arrange Icons", ID_ICON
        MENUITEM "C&lose", ID_CLOSE
    END
    POPUP "&Single"
    BEGIN
        MENUITEM "&Maximize", ID_MAXI
        MENUITEM "&Restore", ID_REST
        MENUITEM "&Next", ID_NEXT
    END
END

MDI2Menu MENU
BEGIN
    POPUP "&Toggle"
    BEGIN
```

```
        MENUITEM "New &Menu", ID_MENU2
    END
END

MDIMenu ACCELERATORS
BEGIN
    VK_F8, ID_NEW,SHIFT,VIRTKEY
    "^T", ID_TILE
    "^C", ID_CASC
END
```

头文件:MDI.H

```
#define ID_FIRST 10

#define ID_LIST 100

#define ID_TILE 100
#define ID_CASC 101
#define ID_ICON 102
#define ID_CLOSE 103

#define ID_NEW 200
#define ID_MENU1 201
#define ID_MENU2 202

#define ID_MAXI 300
#define ID_REST 301
#define ID_NEXT 302
```

MAKE 文件:MDI.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = MDI.H

all: MDI.EXE

MDI.RES: MDI.RC $(RESFILES)

MDI.OBJ: MDI.C $(H)

MDI.EXE: MDI.OBJ MDI.RES MDI.DEF
    link /NOD/A:16 MDI,MDI,nul,libw slibcew,MDI.def
    rc MDI.res MDI.exe
```

如下运行 NMAKE 可创建 MDI 应用程序:

nmake mdi. mak

MDI. EXE 怎样工作

在 MDIInit 初始程序中登录了两个类：一个是框架窗口类，另一个是所有子窗口类。框架窗口有一个名为 MDIMenu 的菜单，cbWndExtra 字段赋值为 4，以为每个窗口保留 4 个附加字节。用户可为这两个类取任何名字，在本例中，框架窗口类使用 Frame，子窗口类则使用 Child。

用 WS_CLIPCHILDREN 风格创建框架窗口的 CreateWindow 函数产生一条 WM_CREATE 消息，这个消息被直接传送给窗口函数，在此消息的基础上获得实例、设置菜单和第二个菜单的句柄。

CIENTCREATESTRUCT 结构类型的 ccs 变量由弹出式菜单的句柄填充，子窗口的标题和第一个子窗口的 ID 值出现在此弹出式菜单上。

CreateWindow 函数在创建 MDI 客户窗口时需要 ccs 变量。由于框架窗口是父窗口，所以用户可以指定 WS_CHILD 风格参数和框架窗口的窗口句柄。

另外，为滚动条设置了 WS_CLIPCHILDREN 和 WS_VSCROLL 风格。由于客户窗口没有自己的客户区域，因此可用 ShowWindow 函数显示窗口。WM_CREATE 消息被处理后，控制就返回给 WinMain 函数。

如果两个窗口之中有一个不能被创建，则整个 MDI 应用程序就会终止。否则，框架窗口将会显示出来，并且它的加速键表也被装入。由于第一个子窗口应该立即在框架窗口中显示出来，所以用户必须在进入消息循环之前创建它。

要做到这一点，应给 MDICREATESTRUCT 结构类型的 mcs 变量赋值，初始位置和大小被置为 0。因为窗口是以 WS_MAXIMIZE 风格显示的，它将以最大尺寸显示出来。变量 mcs 的 WM_MDICREATE 消息被传送给客户窗口，所以客户窗口创建一个 MDI 子窗口。

客户窗口把 WM_CREATE 消息传送给子窗口的窗口函数，由于 MDI 子窗口与列表框完全重叠，故可用 CreateWindow 函数创建另一个窗口，这个窗口比 MDI 子窗口低一级并有预定义的列表框类，DlgDirList 函数用当前目录中的所有文件来填充列表框。

为了能存取指定选择串的列表框的句柄和索引，SetWindowWord 函数把这些值写入四个附加字节中，焦点也被传送给列表框窗口。直到接收到 WM_SIZE 消息，MoveWindow 函数才把列表框设置为与父窗口一样大。GetWindowWord 函数从附加字节中检索所需的列表框的句柄，然后 WM_SIZE 消息还必须传送给缺省例程 DefMDIChildProc。

第一个 MDI 子窗口创建后，WinMain 转移到消息循环，这时可以检验每条消息是否有框架窗口或子窗口的加速键关键字。如果消息包含加速键关键字，它就立即转移到正确的窗口函数。

框架窗口有一个包含弹出式菜单 Window，ALL 和 Single 的菜单，它们有不同的菜单项。活动的 MDI 子窗口可以通过 Maximize 和 Restore 菜单项来重新安排尺寸。为了达到这个目的，WM_MDIMAXIMIZE 和 WM_MDIRESTORE 消息都传送给客户窗口。

为了调用下一个 MDI 子窗口，可使用加速键 Ctrl＋F6 或 Next 菜单项，它们都传送 WM_MDINEXT 消息给父窗口。弹出式菜单中的这两个菜单项影响所有现存 MDI 子窗口。窗口既可以借助 WM_MDLCASCADE 消息重叠放置，也可借助 WM_MDITILE 消息并列放

置。第三项通过传送 WM_MDIICONARPANGE 消息给客户窗口，从而重新安排所有图标。

关闭所有窗口和图标

如果要从 All 菜单中选择 Close 项关闭所有子窗口，则可借助 ShowWindow 函数和 SW_HIDE 值使客户窗口消失，这避免了上面的子窗口遭到破坏时，所有现存子窗口都显示出来。

在没有子窗口时，客户窗口重新显示出来。删除过程在循环中进行。一旦 GetWindow 函数返回了一个合法的子窗口句柄，这个句柄就由 WM_MDIDESTROY 消息的 wParam 参数传给客户窗口。除正常子窗口外，Close 菜单项也检查作为图标显示的子窗口，然后由此例程关闭这些窗口。

用户可用 New Window 菜单项创建所需的所有 MDI 子窗口。为了做到这一点，MDICREATESTRUCT 结构类型的变量必须填以提供的数据，就如用户在开始时所做的一样。除名字 Child 外，标题还包含一个用来标志单个子窗口的数字。

除了上面提到的菜单项，Window 弹出式菜单还有一个名为 New Menu 的菜单项，当用户选择此项时，整个 MDIMenu 被 MDI2Menu 所代替，这个菜单也是由 .RC 文件定义的。

这个新菜单只由一个名为 Toggle 的弹出式菜单组成，菜单中的菜单项 New Menu 可以用来存取初始状态。用户可用 WM_MDISETMENU 消息交换框架窗口的菜单。新菜单句柄由低位字传送，而弹出式菜单句柄由高位字传送。

以上提到的弹出式菜单句柄是需要的，这样 Windows 才知道它应向哪个弹出式菜单添加 MDI 子窗口标题。如果句柄省略了，就不显示标题。每次交换菜单，用户必须用 DrawMenuBar 函数提取菜单条。

除 WM_CREATE 和 WM_SIZE 消息之外，MDI 子窗口的窗口函数还处理另外三条消息。一旦子窗口被激活，它就接收焦点。

在范例应用程序中，活动窗口的列表框应当有焦点，这样用户才能用光标键选择另外的字符串，而不必按鼠标按钮。

因此，在 WM_SETFOUCS 消息期间，焦点借助 SetFoucs 函数传送给列表框。用 GetWindowWord 从附加字节中读取所需的列表框句柄。

当用户移动列表框里的光标时，父窗口（子窗口的窗口函数）就接收到 WM_COMMAND 消息及 wParam 参数里的 ID 值。

被传送到列表框的 LB_GETCURSEL 消息存取光标的位置，这个位置放在附加字节里偏移量为 2 的位置处，这个值对于 WM_MDIACTIVATE 消息很必要，因为它被递增、存储并经过 LB_SETCURSEL 消息传送给列表框。这样，光标被放到下一个字符串上。

第二十章　剪贴板

剪贴板使得在应用程序之间交换信息成为可能。任何一个处理文件的应用程序都可以访问它所占用的内存。剪贴板被用来存储和传送格式化数据，并对其加以处理。

每次处理将返回文件信息，比如在 Windows 中申请了内存等。当选中了 Cut 或 Copy 菜单时，剪贴板就会接收到一个句柄。只要选中了 Paste 菜单，被存储的数据便可以拷贝到另一个文件中去。剪贴板中可见的部分是应用程序 CLIPBRD.EXE，而不是剪贴板本身。

几个可用于剪贴板的 Windows 函数如下：

函数	描述
ChangeClipboardChain	从观察窗链表中取出的观察窗
CloseClipboard	关闭剪贴板
EmptyClipboard	清除剪贴板的内容
EnumClipboardFormats	剪贴板格式计数
GetClipboardData	从剪贴板中得到数据
GetClipboardFormatName	得到当前剪贴板格式
GetClipboardOwner	得到剪贴板的拥有者的窗口句柄
GetClipboardViewer	得到观察窗链表的第一个窗口的句柄
GetPriorityClipboardFormat	从剪贴板中得到最高优先级的数据格式
IsClipboardFormatAvailable	若数据为指定格式，则返回 TRUE
OpenClipboard	打开剪贴板
SetClipboardData	向剪贴板中拷贝一个数据句柄
SetClipboardViewer	在观察窗链中设置句柄

剪贴板提供了几种传递数据的格式。下表列出了五种最常用的格式：

格式	内容
CF_TEXT	全程内存包含以“0”结尾的文本
CF_BITMAP	传递句柄的位图
CF_DIF	全程内存包含 DIF 格式的数据
CF_METAFILEPICT	全程内存包含元文件图片结构
CF_SYLK	全程内存包含符号链格式的数据

当应用程序用函数 GetClipboardData 从剪贴板中提取数据时，必须遵从剪贴板中数据的格式，否则将无法传送数据。

20.1 文本格式

文本格式(CF_TEXT)处理未格式化的文本,即不包含控制字符或格式信息的文本。未格式化的文本只包含 ASCII(或 Windows 中的 ANSI)字符。

20.1.1 向剪贴板中拷贝文本

向剪贴板中拷贝文本需要下面几个步骤:

(1)把文本拷贝到全程内存中

```
hClipData = GlobalAlloc(GMEM_MOVEABLE,GlobalSize(hMyData));
lpszClipData = GlobalLock(hClipData);
lpszMyData = GlobalLock(hMyData);
lstrcopy (lpszClipData,lpszMyData);
GlobalUnlock(hMyData);
GlobalUnlock(hClipData);
```

(2)打开剪贴板

```
OpenClipboard(hWnd);
```

(3)清除剪贴板的内容

```
EmptyClipboard ();
```

(4)向全程内存传送句柄

```
SetClipboardData (CF_TEXT,hClipData),
```

(5)关闭剪贴板

```
CloseClipboard ();
```

文本通过剪贴板传递时,必须被放入全程内存中。本例假设句柄 hMyData 指向一块全程内存块,当剪贴板需要此文本时可在这个块中找到。当应用程序不再需要已传给剪贴板的句柄时,文本被拷贝到另一块全程内存中,另一块的句柄则被写入剪贴板中。

因此,一个新的内存块被申请,并被函数 GlobalLock 锁定了。两个结构指针(32 位的地址指针)被函数 lstrCopy 用来把文本拷贝到新块中。

因此,我们可以打开剪贴板并清除旧的内容。当包含剪贴板的窗口被打开时,其中的数据便可以被传送。如果剪贴板是空的,hClipData 将传送 CF_TEXT 格式,然后剪贴板可被关闭掉。

函数 OpenClipboard 和函数 CloseClipboard 必须在同一消息循环中被调用。一旦打开了剪贴板,应用程序便不能再打开或使用剪贴板了。

剪贴板不应给一个被标识为已分配的内存块赋予句柄。一旦句柄被写入剪贴板,它就不能再从其他应用程序中获得信息,因为它的拥有者就是剪贴板本身。

当应用程序结束时,内存块并不被释放(此任务由 Windows 完成),直到函数 EmptyClipboard 调用它为止。

20.1.2 从剪贴板中获得文本

(1)打开剪贴板

```
OpenClipboard (nWnd);
```

(2)从剪贴板中获得全程内存块句柄

```
hClipBit = GetClipboardData (CF_TEXT);
```

(3)向全程内存块中拷贝数据

```
hMyData = GlobalAlloc(GMEM_MOVEABLE,GlobalSize(hClipData));
lpszClipData = GlobalLock(hClipData);
lpszMyData = GlobalLock(hMyData);
lstrcopy (lpszMyData,lpszClipData);
GlobalUnlock(hMyData);
GlobalUnlock(hClipData);
```

(4)关闭剪贴板

```
CloseClipboard ();
```

在向剪贴板传送数据之前,应用程序必须确保剪贴板可以接收文本格式的数据。使用函数 IsClipboardFomatAvalibale 或 GetClipboardData 便可完成此功能。

两个函数都将传送常量 CF_TEXT。如果剪贴板中有文本,则第一个函数返回 TRUE,此函数是少数几个不打开剪贴板的函数之一。

提取数据之前必须打开剪贴板。函数 GetClipboardData 给数据提供一个句柄。如果格式与当前剪贴板格式不匹配,则句柄为 NULL。这只适用于调用函数 GetlpboardData 和 CloseClipboard,因为它们属于剪贴板函数。

与向剪贴板中写数据类似,必须申请另一块全程内存,并把数据拷贝到此块中。值得注意是,返回句柄的内存块必须被释放掉,并且关闭剪贴板,然后其他应用程序才能使用剪贴板。

20.1.3 文本格式示例

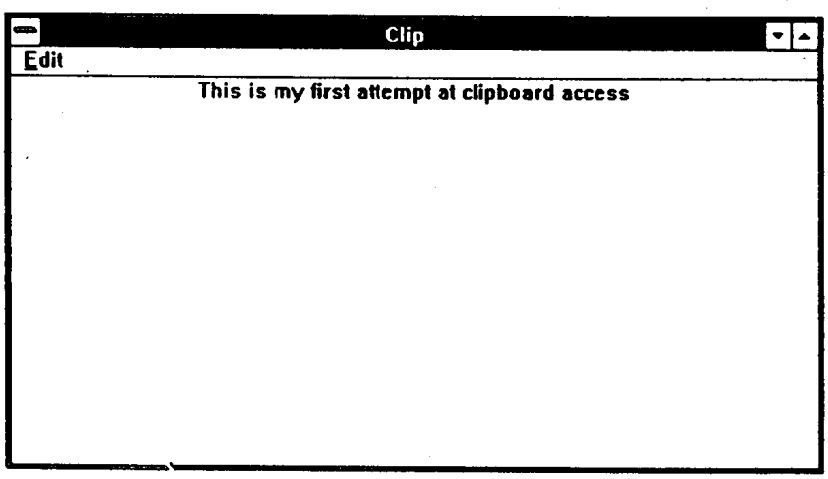


图 20.1

本例产生一个包含 Cut,Copy,Paste 菜单项的菜单。如果选中了前两个菜单项,客户区的当前文本便被传送到剪贴板中。Paste 命令把这些数据粘贴到客户窗口中。

粘贴到客户区的文本已经在剪贴板中存在。这个例子允许在不同的 Windows 应用程序中传送文本(例如 WRITE. EXE 或 WINWORD. EXE)。

新函数	描述
CloseClipboard	关闭剪贴板
EmptyClipboard	清除剪贴板中的内容
GetClipboardData	从剪贴板中获取数据
GlobalAlloc	获得全程内存块
GlobalLock	锁定全程内存块
GlobalReAlloc	重新申请全程内存块(新的大小)
GlobalSize	计算全程内存块的大小
GlobalUnlcok	解锁全程内存块
IsClipoboardFormatAvailable	如果当前格式可用,则返回 TRUE
lstrcpy	把一个串拷贝到另一个串中
OpenClipboard	打开剪贴板
SetClipboardData	向剪贴板中存入数据

源代码:CLIP. C

```
/* * CLIP. C * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Simple clipboard access * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows. h"                                    // Include windows. h header file
#include "Clip. h"                                       // Include clip. h header file

#if _MSC_VER >= 700                                      // Running Microsoft C 7. 00 or
#pragma warning (disable:4028)                           // higher? Disable warning 4028
#endif


BOOL ClipInit ( HANDLE );
long FAR PASCAL ClipWndProc( HWND,unsigned,WORD,LONG);
void WarningBox(void);

/* * WinMain (main window function for all Windows applications) * * * * * /

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow)
    HANDLE hInstance,hPrevInstance;                      // Current & previous instance
    LPSTR lpszCmdLine;                                   // Long ptr after prg. name
    int cmdShow;                                         // App. window's appearance
{
    MSG msg;                                             // Message handle
    HWND hWnd;                                           // Window handle

    if (! hPrevInstance)                                 // Initialize current instance
    {
        if (! ClipInit( hInstance ))
```

```
            return FALSE;
    }

    hWnd = CreateWindow("Clip","Clip",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,0,CW_USEDEFAULT,0,
            NULL,
            NULL,
            hInstance,
            NULL);

    ShowWindow( hWnd,cmdShow );                          // Make window visible
    UpdateWindow( hWnd );                                // Update window

    while (GetMessage(&msg,NULL,0,0))                    // Message reading
    {
        TranslateMessage(&msg);                          // Message translation
        DispatchMessage(&msg);                           // Send message to Windows
    }
    return (int)msg.wParam;                              // Return wParam of last message
}

/** Specify window class **********************/

BOOL ClipInit( hInstance )
HANDLE hInstance;
{
    WNDCLASS wcClipClass;

    wcClipClass.hCursor = LoadCursor( NULL,IDC_ARROW );
    wcClipClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcClipClass.lpszMenuName = (LPSTR)"Menu";
    wcClipClass.lpszClassName = (LPSTR)"Clip";
    wcClipClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcClipClass.hInstance = hInstance;
    wcClipClass.style = CS_VREDRAW | CS_HREDRAW;
    wcClipClass.lpfnWndProc = ClipWndProc;
    wcClipClass.cbClsExtra = 0 ;
    wcClipClass.cbWndExtra = 0 ;

    if (! RegisterClass( &wcClipClass ) )
        return FALSE;
    return TRUE;
}

/** ClipWndProc ***********************************/
/** Main window function: All messages are sent to this window **/
/**********************************************/

long FAR PASCAL ClipWndProc( hWnd,message,wParam,lParam )
HWND hWnd;
```

```
unsigned message;
WORD wParam;
LONG lParam;
{
    static HMENU hMenu;
    static HANDLE hClipData;
    static HANDLE hMyData;
    char chText[] = "This is my first attempt at clipboard access";
    LPSTR lpszMyData,lpszClipData = NULL;
    static BOOL bMyData;
    RECT rect;
    PAINTSTRUCT ps;
    HDC hDC;

    switch (message)
    {
        case WM_CREATE:                                          // Create window
            hMenu = GetMenu(hWnd);
            if (! (hMyData = GlobalAlloc( GMEM_MOVEABLE
                    |GMEM_NODISCARD,(DWORD)sizeof(chText))))
            {
                WarningBox();
                SendMessage(hWnd,WM_CLOSE,0,0L);
                return (FALSE);
            }
            lpszMyData = GlobalLock(hMyData);
            lstrcpy(lpszMyData,chText);
            GlobalUnlock(hMyData);
            bMyData = TRUE;
            break;
        case WM_INITMENU:                                        // Initialize menu
            if (IsClipboardFormatAvailable(CF_TEXT))
                EnableMenuItem(wParam,ID_PASTE,MF_ENABLED);
            else
                EnableMenuItem(wParam,ID_PASTE,MF_GRAYED);
            break;
        case WM_COMMAND:                                         // Menu items
            switch(wParam)
            {
                case ID_CUT:
                case ID_COPY:
                    if (! (hClipData = GlobalAlloc( GMEM_MOVEABLE
                            |GMEM_NODISCARD,GlobalSize(hMyData))))
                    {
                        WarningBox();
                        return (FALSE);
                    }
                    lpszClipData = GlobalLock(hClipData);
                    lpszMyData = GlobalLock(hMyData);

                    lstrcpy(lpszClipData,lpszMyData);
```

```
        GlobalUnlock(hClipData);
        GlobalUnlock(hMyData);

        if (OpenClipboard(hWnd))
        {
            EmptyClipboard();
            SetClipboardData(CF_TEXT,hClipData);
            CloseClipboard();
        }

        if (wParam == ID_CUT)
        {
            bMyData = FALSE;
            EnableMenuItem(hMenu,ID_CUT,MF_GRAYED);
            EnableMenuItem(hMenu,ID_COPY,MF_GRAYED);
        }

        InvalidateRect(hWnd,NULL,TRUE);
        break;
    case ID_PASTE:
        if (OpenClipboard(hWnd))
        {
            if (! (hClipData = GetClipboardData(CF_TEXT)))
            {
                CloseClipboard();
                return (FALSE);
            }

            if (! (hMyData = GlobalReAlloc( hMyData,
                    GlobalSize(hClipData),GMEM_MOVEABLE)))
            {
                if (! (hMyData = GlobalAlloc( GMEM_MOVEABLE|
                        GMEM_NODISCARD,GlobalSize(hClipData))))
                {
                    WarningBox();
                    CloseClipboard();
                    return (FALSE);
                }
            }

            lpszMyData = GlobalLock(hMyData);
            lpszClipData = GlobalLock(hClipData);
            lstrcpy(lpszMyData,lpszClipData);
            GlobalUnlock(hMyData);
            GlobalUnlock(hClipData);
            CloseClipboard();
            EnableMenuItem(hMenu,ID_CUT,MF_ENABLED);
            EnableMenuItem(hMenu,ID_COPY,MF_ENABLED);
            bMyData = TRUE;

            InvalidateRect(hWnd,NULL,TRUE);
```

```
                    }
                    break;
                default:
                    return (DefWindowProc(hWnd,message,wParam,lParam));
                    break;
            }
            break;
        case WM_PAINT:                                          // Client area redraw
            hDC = BeginPaint (hWnd,&ps);

            if (bMyData)
            {
                lpszMyData = GlobalLock (hMyData);
                GetClientRect (hWnd,&rect);
                DrawText (hDC,lpszMyData,-1,&rect,DT_CENTER | DT_WORDBREAK);
                GlobalUnlock (hMyData);
            }
            EndPaint (hWnd,&ps);
            break;
        case WM_DESTROY:                                        // Destroy window
            GlobalFree(hMyData);
            GlobalFree(hClipData);
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;
    }
    return(0L);
}

void WarningBox(void)
{
    MessageBox( GetFocus(),
            "Not enough memory","ATTENTION!",
            MB_ICONHAND | MB_SYSTEMMODAL | MB_OK);
    return;
}
```

模块定义文件:CLIP. DEF

```
NAME Clip

DESCRIPTION 'Clipboard demonstration'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
```

```
HEAPSIZE 4096
STACKSIZE 4096

EXPORTS ClipWndProc @1
```

资源文件:CLIP. RC

```
#include "clip. h"
#include "windows. h"

Menu MENU
BEGIN
    POPUP "&Edit"
    BEGIN
        MENUITEM "Cu&t", ID_CUT
        MENUITEM "&Copy", ID_COPY
        MENUITEM "&Paste", ID_PASTE
    END
END
```

头文件:CLIP. H

```
#define ID_CUT 11
#define ID_COPY 12
#define ID_PASTE 13
```

MAKE 文件:CLIP. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES =
H = CLIP. H

all: CLIP. EXE

CLIP. RES: CLIP. RC $ (RESFILES)

CLIP. OBJ: CLIP. C $ (H)

CLIP. EXE: CLIP. OBJ CLIP. RES CLIP. DEF
    link /NOD/A:16 CLIP,CLIP,nul,libw slibcew,CLIP. def
    rc CLIP. res CLIP. exe
```

像下面这样运行 NMAKE 便可产生应用程序 CLIP:

```
nmake clip. mak
```

CLIP. EXE 是怎样工作的

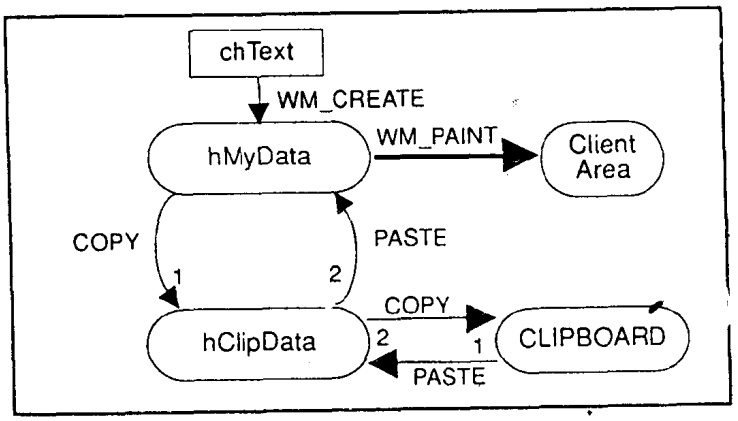


图 20.2

这个应用程序包含两个全程内存块。一个用来存储剪贴板数据，另一个用来存储写到客户区中的文本。

两个全程内存块必须是未用的，这样剪贴板传送和获得的句柄才不会属于某个应用程序，这还使得剪贴板尽可能少地被使用。

这两个内存块还可用于 Copy 菜单项，它在客户区中将显示一行由变量 chText 定义的初始化文本。有关句柄指示的数据流，可参阅上面的图示。

消息 WM_CREATE 通过函数 GlobalAlloc 给初始化的文本提供一个全程内存块。如果没有可用内存块，应用程序将显示一个消息框。

显示了这个消息框后，应用程序将发送消息 WM_CLOSE 并停止运行。若没有显示消息框，则函数 lstrcpy 将解除对内存块中文本的锁定。函数 lstrcpy 还必须能够使用各种指针，因为 chText 是近指针而 lpszMyData 是远指针。

在 Windows 中，锁定的内存块应该用函数 GlobalUnlock 尽快地解锁。还需要一个由 GetMenu 产生的菜单句柄来修改每个菜单项的状态。布尔变量 bMyData 为 TRUE 时，意味着已经发送消息 WM_PAINT 显示文本。

打开菜单前片刻发送消息 WM_INITMENU。如果剪贴板包含文本格式的数据，应用程序将激活 Paste 菜单项。可由函数 IsClipboardFormatAvaliable 判定每个菜单项的状态。

基于剪贴板的应用程序中常有 Cut 和 Copy 菜单项，它们是通过句柄 hMyData 进行标识的。选中 Cut 或 Copy 命令后都将通过句柄 hClipData 把文本本身(或文本的一个拷贝)放入一个新的全程块中。

函数 GlobalSize 通过句柄 hMyData 计算新内存块的大小。当函数 SetClipboardData 成功地打开了剪贴板并清除了原来的数据之后，就关闭这个句柄。如果用户选中了 Cut 菜单，将发生三件事情：

(1)Cut 和 Copy 菜单项无效。

(2)布尔变量 bMyData 改变状态。

(3)再一次出现消息 WM_PAINT 之前,客户区中不显示文本。

如果用户选中了 Paste 菜单项,剪贴板中的文本将被用于粘贴。函数 GetClipboardData 可完成此任务。可根据数据句柄取出数据并进行粘贴。虽然有可用内存块,但它的大小不一定合适,函数 GlobalReAlloc 将设置新的大小。

第一次调用 GlobalReAlloc 时,由于在调用前还未申请内存,所以将返回 NULL。函数 GlobalAlloc 得到一新内存块,然后像 Copy 菜单项一样拷贝文本(但是方向不同)。这些必须在关闭剪贴板和删除句柄之前进行。

最后,布尔变量 bMyData 被置为 TRUE,Paste 菜单在例程 MF_ENABLED 中修改其他入口。函数 InvalidateRect 发送消息 WM_PAINT 并显示新的文本。

完成上述任务的函数 DrawText 需要一个指向文本的指针,所以必须用函数 GlobalLock 锁定全程内存块。函数 DrawText 指明了整个客户区,而客户区的大小是由函数 GetClientRect 计算出来的。

应用程序运行结束时,函数 ClobalFree 释放掉所有的全程内存块,这时所有的句柄都无效了。

20.2 位图格式

标识符 CF_BITMAP 允许剪贴板交换位图,这个过程很像前面描述的文本传送过程,但是位图的管理要复杂得多,因为位图的结构比文本结构复杂。

20.2.1 向剪贴板中写一个位图

把位图放置到剪贴板上的过程几乎和把文本放到剪贴板上完全相同。需要一个将要放到剪贴板上的位图的句柄,但并不需要申请一块全程内存。函数 SetClipboardData 从 CF_BITMAP 得到格式信息。

20.2.2 从剪贴板中获得一个位图

为了从剪贴板中获得一个位图,首先要用函数 OpenClipboard 打开剪贴板。

(1)打开剪贴板

```
OpenClipboard (hWnd);
```

(2)从剪贴板得到全程内存块的句柄

```
hClipBit = GetClipboardData (CF_BITMAP);
```

(3)在屏幕上显示位图

```
hDC = GetDC(hWND);
hMemDC = CreateCompatibleDC(hDC)
hOldBit = SelectObject(hMemDC,hClipBit);
GetObject(hClipData,szieof(BITMAP),&bm);
BitBit(hDC,20,20,bm. bm Width,bm. bmHeight,hMemDC,0,0,SRCCOPY);
SelectObject(hMemDC,hOldBit);
DeleteDC(hMem,hDC);
ReleaseDC(hWnd,hDC);
```

(4)关闭剪贴板

CloseClipboard ()

函数 GetClipboardData 通过位图句柄获得位图数据，但是在显示位图(例如用函数 BitBlt)之前，还需要一个内存设备环境，因此还要调用函数 CreateCompatibleDC。

函数 SelectObject 得到剪贴板中原来位图的句柄。函数 GetObject 计算出正确显示位图的大小，并给 BITMAP 结构变量赋值。输出函数 BitBlt 包含位图的宽度和高度。

内存设备环境在这里是源设备环境，而显示环境为目标环境。位图被显示之后，旧的位图被放回内存设备环境，应用程序两把两个设备环境都释放掉。

函数 CloseClipboard 用来关闭位图句柄。如果第一个输出消息 WM_PAINT 处理完毕，则像关闭文本句柄一样关闭位图句柄。这个应用程序程序包含两个位图句柄，一个代表剪贴板，另一个代表屏幕。

位图不能用函数 lstrocpy 拷贝，但可以用两个内存设备环境进行传送(参看下面的例子 CLIPBIT)。

20.2.3 位图格式示例

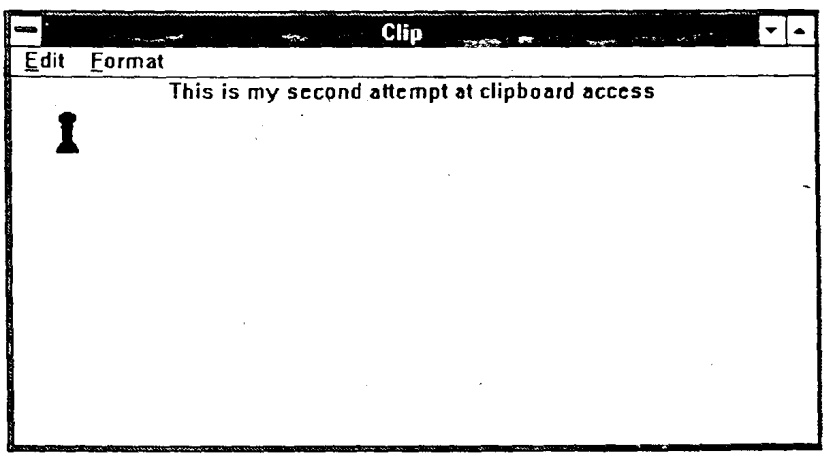


图 20.3

这个应用程序的基础是本章前面所列的应用程序 CLIP.EXE。CLIPBIT 允许在两个 Windows 应用程序之间交换文本和位图。应用程序的菜单项根据格式的要求进行了修改。

这里还定义了一个初始化位图，在应用程序被启动时显示在客户区中。

源代码:CLIPBIT.C

```
/* * CLIPBIT.C * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Clipboard access for text and bitmap data * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                          // Include windows.h header file
#include "Clipbit.h"                          // Include clipbit.h header file

#if _MSC_VER >= 700                           // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                // higher? Disable warning 4028
```

```
#endif

BOOL ClipInit ( HANDLE );
long FAR PASCAL ClipWndProc( HWND,unsigned,WORD,LONG);
void WarningBox(void);

/* * WinMain * * * * * * * * * * * * * * * * * * * * * * * * */

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow)
    HANDLE hInstance,hPrevInstance;                    // Current & previous instance
    LPSTR lpszCmdLine;                                 // Long ptr to after prg. name
    int cmdShow;                                       // App. window's appearance
{
    MSG msg;                                                    // Message handle
    HWND hWnd;                                                  // Window handle

    if (! hPrevInstance)                                // Initialize current instance
    {
        if (! ClipInit( hInstance ))
            return FALSE;
    }

    /* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("Clip","Clip",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,0,CW_USEDEFAULT,0,
            NULL,
            NULL,
            hInstance,
            NULL);

    ShowWindow( hWnd,cmdShow );                         // Make window visible
    UpdateWindow( hWnd );                                       // Update window

    while (GetMessage(&msg,NULL,0,0))                           // Message reading
    {
        TranslateMessage(&msg);                                 // Message translation
        DispatchMessage(&msg);                          // Send message to Windows
    }
    return (int)msg. wParam;                        // Return wParam of last message
}

BOOL ClipInit( hInstance )                                      // Clipboard initialization
HANDLE hInstance;
{

    /* * Specify window class * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClipClass;
```

```
    wcClipClass.hCursor = LoadCursor( NULL,IDC_ARROW );
    wcClipClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcClipClass.lpszMenuName = (LPSTR)"Menu";
    wcClipClass.lpszClassName = (LPSTR)"Clip";
    wcClipClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcClipClass.hInstance = hInstance;
    wcClipClass.style = CS_VREDRAW | CS_HREDRAW;
    wcClipClass.lpfnWndProc = ClipWndProc;
    wcClipClass.cbClsExtra = 0 ;
    wcClipClass.cbWndExtra = 0 ;

    if (! RegisterClass( &wcClipClass ) )
        return FALSE;
    return TRUE;
}

/* * ClipWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL ClipWndProc( hWnd,message,wParam,lParam )
    HWND hWnd;
    unsigned message;
    WORD wParam;
    LONG lParam;
{
    static HANDLE hInst;                                    // Instance handle
    static HMENU hMenu;                                      // Menu handle
    static HANDLE hClipData;                                   // Clipboard
    static HANDLE hMyData;                                      // handles
    static HBITMAP hBitmap,hMyBit,hClipBit,hBitOld,hBit2Old;

    char chText[] = "This is my second attempt at clipboard access";
    LPSTR lpszMyData,lpszClipData;
    LPSTR lpszMyBit,lpszClipBit;
    static BOOL bMyData,bMyBitmap;
    static BOOL bText = TRUE;

    RECT rect;
    PAINTSTRUCT ps;
    HDC hDC,hMemDC,hMem2DC;
    BITMAP bm;

    switch(message)
    {
        case WM_CREATE:                                      // Create window
            hInst = GetWindowWord(hWnd,GWW_HINSTANCE);
            hMenu = GetMenu(hWnd);
            if (! (hMyData = GlobalAlloc( GMEM_MOVEABLE
                    |GMEM_NODISCARD,(DWORD)sizeof(chText))))
            {
```

```
            WarningBox();
            SendMessage(hWnd,WM_CLOSE,0,0L);
            return (FALSE);
        }
        lpszMyData = GlobalLock(hMyData);
        lstrcpy(lpszMyData,chText);
        GlobalUnlock(hMyData);
                                                        // Load pawn bitmap
        hBitmap = LoadBitmap(hInst,"Pawn");

        hDC = GetDC(hWnd);
        hMemDC = CreateCompatibleDC( hDC );
        hBitOld = SelectObject(hMemDC,hBitmap);

        hMem2DC = CreateCompatibleDC( hDC );
        hMyBit = CreateCompatibleBitmap(hDC,32,32);
        hBit2Old = SelectObject(hMem2DC,hMyBit);

        BitBlt(hMem2DC,0,0,32,32,hMemDC,0,0,SRCCOPY);
        SelectObject(hMemDC,hBitOld);
        SelectObject(hMemDC,hBit2Old);

        DeleteDC(hMemDC);
        DeleteDC(hMem2DC);
        ReleaseDC(hWnd,hDC);

        bMyData = TRUE;
        bMyBitmap = TRUE;

        break;
    case WM_INITMENU:                                   // Initalize menu
        if (bText)
        {
            if (IsClipboardFormatAvailable(CF_TEXT))
                EnableMenuItem(wParam,ID_PASTE,MF_ENABLED);
            else
                EnableMenuItem(wParam,ID_PASTE,MF_GRAYED);
        }
        if (! bText)
        {
            if (IsClipboardFormatAvailable(CF_BITMAP))
                EnableMenuItem(wParam,ID_PASTE,MF_ENABLED);
            else
                EnableMenuItem(wParam,ID_PASTE,MF_GRAYED);
        }
        break;
    case WM_COMMAND:                                    // Get messages from menu bar
        switch(wParam)
        {
            case ID_CUT:
            case ID_COPY:
```

```
if (bText)
{
    if (! (hClipData = GlobalAlloc( GMEM_MOVEABLE
            |GMEM_NODISCARD,GlobalSize(hMyData))))
    {
        WarningBox();
        return (FALSE);
    }
    lpszClipData = GlobalLock(hClipData);
    lpszMyData = GlobalLock(hMyData);

    lstrcpy(lpszClipData,lpszMyData);
    GlobalUnlock(hClipData);
    GlobalUnlock(hMyData);

    if (OpenClipboard(hWnd))
    {
        EmptyClipboard();
        SetClipboardData(CF_TEXT,hClipData);
        CloseClipboard();
    }

    if (wParam == ID_CUT)
    {
        bMyData = FALSE;
        EnableMenuItem(hMenu,ID_CUT,MF_GRAYED);
        EnableMenuItem(hMenu,ID_COPY,MF_GRAYED);
    }
}                                                    // for text

if (! bText)
{
    hDC = GetDC(hWnd);
    hMemDC = CreateCompatibleDC( hDC );
    hBitOld = SelectObject(hMemDC,hMyBit);
    GetObject(hMyBit,sizeof(BITMAP),(LPSTR)&bm);

    hMem2DC = CreateCompatibleDC( hDC );
    hClipBit = CreateCompatibleBitmap( hDC,bm. bmWidth,
            bm. bmHeight);
    hBit2Old = SelectObject( hMem2DC,hClipBit);

    BitBlt(hMem2DC,0,0,bm. bmWidth,bm. bmHeight,
            hMemDC,0,0,SRCCOPY);
    SelectObject(hMemDC,hBitOld);
    SelectObject(hMemDC,hBit2Old);

    DeleteDC(hMemDC);
    DeleteDC(hMem2DC);
    ReleaseDC(hWnd,hDC);
```

```
            if (OpenClipboard(hWnd))
            {
                EmptyClipboard();
                SetClipboardData(CF_BITMAP,hClipBit);
                CloseClipboard();
            }
            if (wParam == ID_CUT)
            {
                bMyBitmap = FALSE;
                EnableMenuItem(hMenu,ID_CUT,MF_GRAYED);
                EnableMenuItem(hMenu,ID_COPY,MF_GRAYED);
            }
        }                                                          // Bitmap

        InvalidateRect(hWnd,NULL,TRUE);
        break;
    case ID_PASTE:
        if (bText)
        {
            if (OpenClipboard(hWnd))
            {
                if (! (hClipData = GetClipboardData(CF_TEXT)))
                {
                    CloseClipboard();
                    return (FALSE);
                }

                if (! (hMyData = GlobalReAlloc( hMyData,
                        GlobalSize(hClipData),GMEM_MOVEABLE)))
                {
                    if (! (hMyData = GlobalAlloc(GMEM_MOVEABLE
                            |GMEM_NODISCARD,GlobalSize(hClipData))))
                    {
                        WarningBox();
                        CloseClipboard();
                        return (FALSE);
                    }
                }

                lpszMyData = GlobalLock(hMyData);
                lpszClipData = GlobalLock(hClipData);
                lstrcpy(lpszMyData,lpszClipData);
                GlobalUnlock(hMyData);
                GlobalUnlock(hClipData);
                CloseClipboard();
                EnableMenuItem(hMenu,ID_CUT,MF_ENABLED);
                EnableMenuItem(hMenu,ID_COPY,MF_ENABLED);
                bMyData = TRUE;

                InvalidateRect(hWnd,NULL,TRUE);
            }
```

```
        }                                                    // Text

        if (! bText)
        {
            if (OpenClipboard(hWnd))
            {
                if (! (hClipBit = GetClipboardData(CF_BITMAP)))
                {
                    CloseClipboard();
                    return (FALSE);
                }

                hDC = GetDC(hWnd);
                hMemDC = CreateCompatibleDC( hDC );
                hBitOld = SelectObject(hMemDC,hClipBit);
                GetObject(hClipBit,sizeof(BITMAP),(LPSTR)&bm);

                hMem2DC = CreateCompatibleDC( hDC );
                hMyBit = CreateCompatibleBitmap( hDC,bm. bmWidth,
                        bm. bmHeight);
                hBit2Old = SelectObject(hMem2DC,hMyBit);

                BitBlt(hMem2DC,0,0,bm. bmWidth,bm. bmHeight,
                        hMemDC,0,0,SRCCOPY);
                SelectObject(hMemDC,hBitOld);
                SelectObject(hMemDC,hBit2Old);

                DeleteDC(hMemDC);
                DeleteDC(hMem2DC);
                ReleaseDC(hWnd,hDC);

                CloseClipboard();
                EnableMenuItem(hMenu,ID_CUT,MF_ENABLED);
                EnableMenuItem(hMenu,ID_COPY,MF_ENABLED);
            }
            bMyBitmap = TRUE;
            InvalidateRect(hWnd,NULL,TRUE);
        }
        break;
    case ID_TEXT:
        bText = TRUE;
        CheckMenuItem(hMenu,ID_TEXT,MF_CHECKED);
        CheckMenuItem(hMenu,ID_BIT,MF_UNCHECKED);
        break;
    case ID_BIT:
        bText = FALSE;
        CheckMenuItem(hMenu,ID_TEXT,MF_UNCHECKED);
        CheckMenuItem(hMenu,ID_BIT,MF_CHECKED);
        break;
    default:
        return (DefWindowProc(hWnd,message,wParam,lParam));
```

```
                    break;
                }
                break;
            case WM_PAINT:                                          // Client area redraw
                hDC = BeginPaint (hWnd,&ps);
                if (bMyData)
                {
                    lpszMyData = GlobalLock (hMyData);
                    GetClientRect (hWnd,&rect);
                    DrawText (hDC,lpszMyData,-1,&rect,DT_CENTER | DT_WORDBREAK);
                    GlobalUnlock (hMyData);
                }

                if (bMyBitmap)
                {
                    hMemDC = CreateCompatibleDC( hDC );
                    hBitOld = SelectObject(hMemDC,hMyBit);
                    GetObject(hMyBit,sizeof(BITMAP),(LPSTR)&bm);
                    BitBlt(hDC,20,20,bm.bmWidth,bm.bmHeight,hMemDC,0,0,SRCCOPY);
                    SelectObject(hMemDC,hBitOld);
                    DeleteDC(hMemDC);
                }

                EndPaint (hWnd,&ps);
                break;
            case WM_DESTROY:                                        // Destroy window
                GlobalFree(hMyData);
                GlobalFree(hClipData);
                PostQuitMessage(0);
                break;
            default:
                return (DefWindowProc( hWnd,message,wParam,lParam ));
                break;
        }
        return(0L);
}

void WarningBox(void)
{
    MessageBox( GetFocus(),
            "Not enough memory","ATTENTION!",
            MB_ICONHAND | MB_SYSTEMMODAL | MB_OK);
        return;
}
```

模块定义文件:CLIPBIT.DEF

```
NAME Clipbit

DESCRIPTION 'Clipboard demonstration with bitmap'

EXETYPE WINDOWS
```

```
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS ClipWndProc @1
```

资源文件:CLIPBIT. RC

```
#include "clipbit. h"
#include "windows. h"

Pawn BITMAP pawn. bmp

Menu MENU
BEGIN
    POPUP "&Edit"
    BEGIN
        MENUITEM "Cu&t", ID_CUT
        MENUITEM "&Copy", ID_COPY
        MENUITEM "&Paste", ID_PASTE
    END
    POPUP "&Format"
    BEGIN
        MENUITEM "&Text", ID_TEXT, CHECKED
        MENUITEM "&Bitmap", ID_BIT
    END
END
```

头文件:CLIPBIT. H

```
#define ID_CUT 11
#define ID_COPY 12
#define ID_PASTE 13
#define ID_ TEXT 21
#define ID_BIT 22
```

MAKE 文件:CLIPBIT. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES = pawn. bmp
H = CLIPBIT. H

all: CLIPBIT. EXE
```

```
CLIPBIT.RES: CLIPBIT.RC $(RESFILES)

CLIPBIT.OBJ: CLIPBIT.C $(H)

CLIPBIT.EXE: CLIPBIT.OBJ CLIPBIT.RES CLIPBIT.DEF
    link /NOD/A:16 CLIPBIT,CLIPBIT,nul,libw slibcew,CLIPBIT.def
    rc CLIPBIT.res CLIPBIT.exe
```

像下面这样运行 NMAKE 便可产生应用 CLIPBIT：

```
nmake clipbit.mak
```

CLIPBIT 是怎样工作的

布尔变量 bText 的状态决定了剪贴板按照文本格式还是位图格式处理数据，它的缺省值为 TRUE(即文本格式)。当 bText 的状态被修改时，确认标志便移到 Format 菜单的正确命令上。

对消息 WM_CREATE 所做的处理是从资源描述文件 CLIPBIT.RC 中调出一个 32×32 的位图。这个位图应该用已有的 hMyBit 句柄而不是它自身的句柄来显示。两个内存设备环境都将拷贝这个位图。如果允许在客户区显示位图，则布尔变量 bMyBitmap 为 TRUE。

打开 Edit 菜单之前，必须调用函数 IsClipboardFormatAvaliable 来确保 Paste 菜单项的状态是正确的。如果剪贴板中存在位图或文本，则这个菜单项有效；否则，这个菜单项将变灰。

菜单项 Cut 和 Copy 使用两个内存设备环境，一个用来选择现存的位图，另一个用来选择函数由 CreateCompatibleBitmap 建立的空位图，通过句柄 hClipBit 便可访问这个空位图。

因为函数 BitBlt 需要位图的宽度和亮度作参数，所以要用函数 GetObject 计算被传送的位图的大小。由于位图句柄保持有效，所以两个内存设备环境都被关闭。现在句柄 hClipBit 便可传送给剪贴板了。当参数 wParam 的值等于 ID_CUT 时，两个菜单项的状态都被认为是 NF_GRAYED。同时，当 bMyBitmap 被赋值为 FALSE 时，位图便不能被输出。

当用户选中了 Paste 菜单项时，处理方法与选中了 Cut 和 Copy 菜单项完全相同，只是拷贝的方向相反(即数据被粘贴)。

函数 BitBlt 输出位图，这时目标显示环境是为函数 BeginPaint 准备的显示环境。为了控制被显示的位图，只需要一个内存设备环境就够了。

20.3 关于格式的附加知识

很多应用程序使用剪贴板来交换文本和位图数据。但是，Windows 的剪贴板不太灵活，尤其是当用户想在它上面执行特殊命令时。幸运的是，程序员几乎可以控制剪贴板数据交换的各个方面。

20.3.1 剪贴板中的多种格式

一旦用户打开了剪贴板并准备接收数据，函数 EmptyClipboard 便清除剪贴板中的内容，这可以防止新数据被无意义地添加到旧数据后面。但是，这可能迫使剪贴板同时存储两种格

式的数据(例如文本和位图)。函数 SetClipboardData 可以组合使用 EmptyClipboard 和 CloseClipboard,使得用户可以在剪贴板中存放另一种格式的数据。

```
OpenClipboard(hWnd);
EmptyClipboard();
SetClipboardData (CF_TEXT,hClipData);
SetClipboardData (CF_SYLK,hSylk);
SetClipboardData (CF_BITMAP,hBit);
CloseClipboard();
```

函数 CountClipboardFormats 在用户选中 Paste 菜单项时给出剪贴板中不同格式的个数,函数 EnumClipboardFormats 可判断当前剪贴板中的数据是哪一种格式。

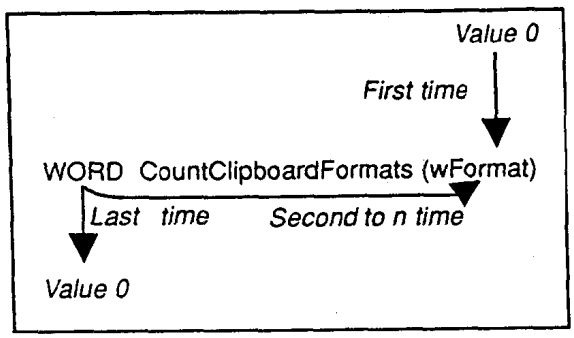


图 20.4 函数 EnumClipboardFormats

为了获取第一种格式,参数必须等于零。如果函数的返回值为零,则剪贴板包含所有可用的格式;否则,系统缺省地设置为最后一次返回的格式。

20.3.2 延迟提交

剪贴板可以交换大量的信息。数据被送入全程内存中,根据情况放置到不同位置。通常,数据不能立即从应用程序提取到剪贴板中。

数据会被放在全程内存块中一段时间而不能被使用。采用"延迟提交"技术可以避免这种情况。这种技术将一个全程块的句柄赋为 NULL,并把句柄传给 SetClipboardData。

当另一个应用程序调用函数 GetClipboardData 从剪贴板提取数据时,Windows 把句柄 NULL 传给它。然后 Windows 发送消息 WM_RENDERFORMAT 给剪贴板的拥有者,剪贴板的拥有者是拥有剪贴板中数据的最后一个窗口。

这条消息为剪贴板的拥有者申请一块全程内存,并把数据拷贝到这个内存块中。参数 wParam 指明了所期望的格式,于是函数 SetClipboardData 便传递了句柄而没有打开和清除剪贴板。

除了消息 WM_RENDERFORMAT 之外,另外还有两个用于延迟提交的消息。一旦调用了其他应用程序,函数 EmptyClipboard 便给剪贴板指定一个新的拥有者。前一个拥有者获得了消息 WM_DESTROY_CLIPBOARD,因此剪贴板不需要重新建立数据了。

20.3.3 自定义数据格式

剪贴板还可提供私有数据有格式。Microsoft Windows 应用程序 Write 就有自己的数据格

式，这种格式包含字模和格式化信息。私有数据格式的用途是在同一个应用程序的不同实例之间交换数据，或者在不同的应用程序中对一种格式取不同的名称。

为了使用私有数据格式，函数 RegisterClipboardFormat 必须指定一个可随意定义的格式名。此函数的返回值是新登录的格式，可在下一次调用剪贴板时作为预定义的格式名。

这个新格式的十六进制值的范围是 C000H 到 FFFFH。如果两个应用程序用同样的名字登录一种新格式，函数 RegisterClipboardFormat 将提供同样的格式码(即两个应用程序可以用这种私有格式交换数据)。

通过使用像 CLIPBRD. EXE 这样的剪贴板观察窗，私有数据格式便可以显示于客户区中了。这种显示可以采用任何一种预定义的格式，也可以用 CF_OWNERDISPLAY 格式。于是作为结果的消息可被处理了。

下面是显示格式：

- CF_DSPTEXT
- CF_DSPIFMAP
- CF_DSPMETAFILEPICT

如果剪贴板中有数据，则私有格式和某一种显示格式就会被拷贝到剪贴板中。剪贴板观察窗选择所需的显示格式，并把数据解释为文本、位图或中间文件。最后，它显示出私有数据格式的替换格式。

剪贴板观察窗口还可以完全地控制客户区的数据显示。在显示格式化的文本时就是如此。这时，函数 SetClipboardData 将选择参数 CF_OWNERDISPLAY，然后传送一个空句柄 NULL。

除了三个延迟提交消息之外，还有五个向剪贴板观察窗口和剪贴板拥有者发送信息的附加消息。

消息	含义
WM_ASKCBFORMATNAME	CB 观察窗发送的消息，用来询问格式名。
WM_PAINT CLIPBOARD	CB 观察窗发送的消息，用来声明客户区需要重画。
WM_SIZECLIPBOARD	CB 观察窗中客户区的大小。
WM_VSCROLLCLIPBOARD	垂直滚动 CB 观察窗。
WM_HSCROLLCLPBOARD	水平滚动 CB 观察窗。

20.4 剪贴板观察窗

剪贴板观察窗是一个当剪贴板内容改变时更新本身的应用程序。应用程序 CLIPBRD. EXE 是剪贴板观察窗的一个例子。虽然 Windows 系统中可以有多个观察窗，但 Windows 只考察一个观察窗(即当前剪贴板观察窗)。

Windows 只从这个观察窗提取窗口句柄，并且只向这个观察窗发送适当的剪贴板消息。如果其他某个观察窗发现剪贴改变了，一个观察窗链就被打开，并把消息传给其他观察窗。于是，这条链中的每个观察窗都收到了消息。

一旦函数 SetClipboardViewer 调用了某个应用程序，Windows 就把此应用程序看作当前观察窗，直到函数又调用了另一个应用程序为止。函数 SetClipboardViewer 提供最后一个观

察窗的句柄，如果此句柄的值为 NULL，则在观察窗链中不存在更早的观察窗。

观察窗链的祖先窗口句柄必须被保存起来，因为剪贴板观察窗消息（不管是从 Windows 还是从当前观察窗口发送来的）还要发给这个窗口。函数 SendMessage 用来发送这些消息。

WM_DRAWCLIPBOARD 是这样一类消息：如果接收者是观察窗链中的第一个观察窗，它将送回这条消息。当剪贴板的内容发生改变时，Windows 总是向当前剪贴板观察窗口发送这条消息。

观察窗主要是用来显示剪贴板内容的，因此，WM_DRAWCLIPBOARD 得到一个函数 InvalidateRect 产生的消息 WM_PAINT。处理 WM_PAINT 的结果是打开剪贴板并使得函数 GetClipboardData 能接收和显示新数据。

当一个观察窗应用程序执行完毕时，必须从观察窗链中删除掉，以免再有消息发送给这个应用程序，函数 ChangeClipboardChain 将完成此功能，它向当前观察窗发送消息 WM_CHANGECBCHAIN。

参数 wParam 包含将要从链中删除的窗口的句柄，参数 lParam 的低位字为链的祖先。因此，当函数 ChangeClipboardChain 正确地给出这两个参数时，它将用消息重新命名。同时还要注意，消息 WM_CHANGECBCHAIN 必须传遍整个观察窗链。

每个观察窗都必须验证存于 wParam 中的句柄是否等于保存的链祖先观察窗的句柄。如果相等，则保存的句柄就被建成一个新的观察窗，句柄被赋值给 lParam。

下面列出当从链中删除一个观察窗时发生的变化。此表假定链中包含六个观察窗，Viewer1 是第一个观察窗，Viewer2 是第二个观察窗，等等。

Viewer6 是当前观察窗，它直接从 Windows 接收观察消息。Viewer3 所代表的应用程序已运行完毕并从链中删除了。

· 旧状态

观察窗链表	存入的句柄祖先
Viewer1:hWnd1	NULL
Viewer2:hWnd2	hWnd1
Viewer3:hWnd3	hWnd2
Viewer4:hWnd4	hWnd3
Viewer5:hWnd5	hWnd4
当前观察窗：	
Viewer6:hWnd6	hWnd5

· 新状态

观察窗链表	存入的句柄祖先
Viewer1:hWnd1	NULL
Viewer2:hWnd2	hWnd1
Viewer3:hWnd3	hWnd2
Viewer4:hWnd4	hWnd3
Viewer5:hWnd5	hWnd4

当前观察窗：

Viewer6：hWnd6　　hWnd5

系统发送消息 WM_CHANGECBCHAIN 之后，Viewer2 变成了 Viewer4 的新祖先。

剪贴板观察窗示例

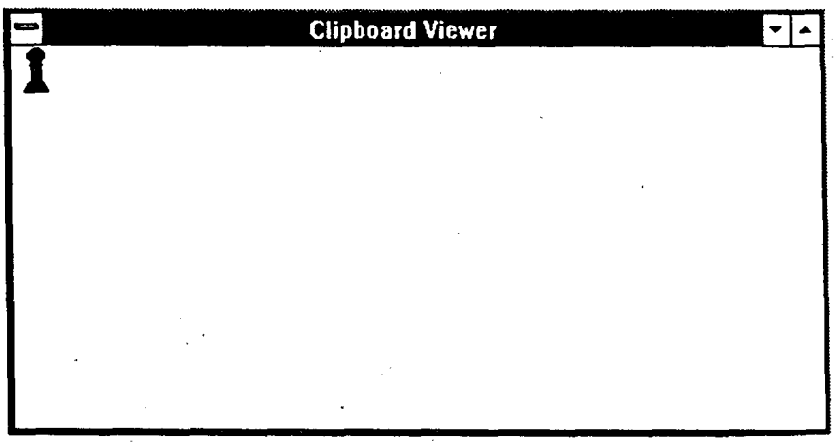


图 20.5

下面的应用程序在它的客户区内显示当前剪贴板的内容(假设剪贴板包含文本数据或位图数据)。

新消息	描述
WM_CHANGECBCHAIN	用来改变观察窗链表
WM_DRAWCLIPBOARD	用来改变剪贴板

新函数	描述
ChangeClipboardChain	改变观察窗链表
SetClipboardViewer	插入新的观察窗

源代码：VIEWER.C

```
/* * VIEWER.C * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Displays clipboard contents in either text or bitmap format * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows.h"                                    // Include windows.h header file

#if _MSC_VER >= 700                                     // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                          // higher? Disable warning 4028
#endif

long FAR PASCAL ViewerWndProc (HWND,unsigned,WORD,LONG) ;
BOOL ViewerInit (HANDLE);
```

```
/* * WinMain * * * * * * * * * * * * * * * * * * * * * * * * */

int PASCAL WinMain (hInstance,hPrevInstance,lpszCmdLine,nCmdShow)
    HANDLE hInstance,hPrevInstance;                    // Current & previous instance
    LPSTR lpszCmdLine ;                                // Ptr to string after prg. name
    int nCmdShow ;                                     // App. window's appearance
{
    HWND hWnd ;                                        // Window handle
    MSG msg ;                                          // Message handle

    if (! hPrevInstance)                               // Initialize current instance
    {
        if (! ViewerInit( hInstance ))
            return FALSE;
    }

    /* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow ("Viewer",
            "Clipboard Viewer",
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,0,
            CW_USEDEFAULT,0,
            NULL,
            NULL,
            hInstance,
            NULL);

    ShowWindow (hWnd,nCmdShow);                        // Make window visible
    UpdateWindow (hWnd);                               // Update window

    while (GetMessage (&msg,NULL,0,0))// Message reading
    {
        TranslateMessage (&msg);                       // Message translation
        DispatchMessage (&msg);                        // Send message to Windows
    }
    return msg. wParam;                                // Return wParam of last message
}

BOOL ViewerInit( hInstance )                           // Viewer initialization
HANDLE hInstance;                                      // Instance handle
{

    /* * Specify window class * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcViewerClass;

    wcViewerClass. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcViewerClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcViewerClass. lpszMenuName = NULL;
```

```
    wcViewerClass.lpszClassName = (LPSTR)"Viewer";
    wcViewerClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcViewerClass.hInstance = hInstance;
    wcViewerClass.style = CS_VREDRAW | CS_HREDRAW;
    wcViewerClass.lpfnWndProc = ViewerWndProc;
    wcViewerClass.cbClsExtra = 0;
    wcViewerClass.cbWndExtra = 0;

    if (! RegisterClass( &wcViewerClass ) )
        return FALSE;
    return TRUE;
}

/* * ViewerWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL ViewerWndProc (hWnd,message,wParam,lParam)
    HWND hWnd;
    unsigned message;
    WORD wParam;
    LONG lParam;
{

    static HWND hWndPrevViewer;                          // Handle for previous viewer
    HANDLE hText,hBitmap,hBitOld;
    HDC hDC,hMemDC;
    LPSTR lpszText;
    BITMAP bm;
    PAINTSTRUCT ps;
    RECT rect;

    switch (message)
    {
        case WM_CREATE:                                  // Create window
            hWndPrevViewer = SetClipboardViewer (hWnd) ;
            break ;
        case WM_CHANGECBCHAIN :                          // Change clipboard chain
            if (wParam == hWndPrevViewer)
                hWndPrevViewer = LOWORD (lParam) ;
            else
                SendMessage (hWndPrevViewer,message,wParam,lParam) ;
            break ;
        case WM_DRAWCLIPBOARD :
            if (hWndPrevViewer)
                SendMessage (hWndPrevViewer,message,wParam,lParam);
            InvalidateRect (hWnd,NULL,TRUE) ;
            break;
        case WM_PAINT:                                   // Client area redraw
            hDC = BeginPaint (hWnd,&ps) ;
            GetClientRect (hWnd,&rect) ;
```

```
            OpenClipboard (hWnd) ;
            if (hText = GetClipboardData (CF_TEXT))
            {
                lpszText = GlobalLock (hText) ;
                DrawText (hDC,lpszText,-1,&rect,DT_WORDBREAK) ;
                GlobalUnlock (hText) ;
            }
            if (hBitmap = GetClipboardData (CF_BITMAP))
            {
                hMemDC = CreateCompatibleDC( hDC );
                hBitOld = SelectObject(hMemDC,hBitmap);
                GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bm);
                BitBlt(hDC,0,0,bm. bmWidth,bm. bmHeight,hMemDC,0,0,SRCCOPY);
                SelectObject(hMemDC,hBitOld);
                DeleteDC(hMemDC);
            }
            CloseClipboard () ;
            EndPaint (hWnd,&ps) ;
            break ;
        case WM_DESTROY:                                        // Destroy window
            ChangeClipboardChain (hWnd,hWndPrevViewer) ;
            PostQuitMessage (0) ;
            break ;
        default:
            return DefWindowProc (hWnd,message,wParam,lParam);
    }
    return 0L ;
}
```

模块定义文件:VIEWER. DEF

```
NAME VIEWER
DESCRIPTION 'Clipboard Viewer Example'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 4096
EXPORTS ViewerWndProc
```

MAKE 文件:VIWER. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES =

all: VIEWER. EXE
```

```
VIEWER.OBJ: VIEWER.C

VIEWER.EXE: VIEWER.OBJ VIEWER.DEF
    link /NOD/A:16 VIEWER,VIEWER,nul,libw slibcew,VIEWER.def
    rc VIEWER.exe
```

像下面这样运行 NMAKE 便得到应用程序 VIEWER：

```
nmake viewer.mak
```

VIEWER.EXE 是怎样工作的

出现了消息 WM_CREATE 之后，函数 SetClipboardViewer 便把新观察窗添加到观察窗链中。一旦剪贴板的内容改变了，此观察窗的例程更接到消息 WM_DRAWCLIPBOARD，并根据这条消息判断出它的祖先观察窗。

消息 WM_PAINT 产生一块空间，以便显示出剪贴板中的内容。当消息 WM_PAINT 打开剪贴板之后，应用程序便检查剪贴板中的数据是否为文本格式或位图格式。如果是这两种格式，则准备输出。

如果某个观察从链中被删除了，则消息 WM_CHANGECBCHAIN 将改变观察窗的状态。应用程序运行结束时将调用函数 ChangeClipboardChain。

第二十一章　动态链接库

Windows 常使用下面几个动态链接库(DLL)：

- KERNEL. EXE
- USER. EXE
- GDI. EXE

Windows 的另几个动态链接库被用作设备驱动程序：DISPLAY. DRV(屏幕驱动程序)、MOUSE. DRV(鼠标驱动程序)和 SYSTEM. DRV(计时器)。

静态链接是通过链接程序把文件永久地组合到一起，因此不够灵活，动态连接则引导应用程序调用 DLL，而这些 DLL 是与应用程序分离的文件。动态链接库 DLL 被列在输入库中或由函数 IMPORTS 调用。

几个应用程序可同时从一个 DLL 中访问同一个对象。这个对象既可以是函数，也可以是资源。DLL 只能被调入为它申请的内存中一次。如果应用程序需要更新一个 DLL 定义的函数，开发者只需要更新、编译并链接 DLL，而不需要改动应用程序。

Windows DLL 与 Windows 应用程序的主要区别是 DLL 不需要对消息进行响应，也没有起始地址(即第一条指令的地址)，程序直接从 DLL 中定义的函数开始执行，DLL 不需要循环等待消息。通常，Windows 应用程序中的任务是主动的，而 DLL 模块则被动地等待调用。

DLL 编程将不断地促进 Windows 编程的发展。值得注意的是，DLL 必须放在当前目录或者环境变量 PATH 指定的目录下。

本章将详细地讨论 DLL。

21.1　远程函数

因为 Intel 处理器支持分段式内存体系，所以可以把函数调用划分为段内调用和段间调用。

如果函数与调用函数的代码位于同一段中，则是近程函数；如果位于不同的段中，则是远程函数。

21.1.1　基本知识

DLL 可以有自己的数据段，因此 DLL 必须为需要调用其函数的应用程序准备好数据段。因为所有的 Windows 函数都是在 DLL 中定义的(例如动态链接库 USER. EXE 中定义了函数 CreateWindow)，所以这个问题总是存在。使用编译开关-Gw 和函数 MakeProcInstance 可以解决这个问题。

21.1.2 入口和出口

用 C 编译器编译一个程序时，它在每个函数的开头和结尾放入一些代码，这些代码被称为入口和出口代码。只在 Windows 应用程序内使用的近程函数将使用入口和出口代码。

当用户在编译时加入了开关项-Gw 并把 Windows 应用程序定义为 FAR 时，系统将加上另一段入口代码：

```
Far functions and -GW switch
PUSH DS
POP AX
NOP
⋮
INC BP Prolog
PUSH BP
MOV BP,SP
PUSH DS
MOV DS,AX
SUB SP,Number
```

其中，Number 给出了未定义为 static 的局部变量所占的数据空间大小。当函数结束时，出口代码获得寄存器的值，并把它们通过入口放入堆栈中。

入口控制着数据段寄存器，通过栈堆把 DS 寄存器的值赋给 AX 寄存器，然后通过 BP 寄存器把 AX 寄存器的值赋给数据段寄存器。于是，在调用前后的 DS 值不变，这样就保证了调用函数后应用程序不会中断。

远程函数在文件中被定义为 Export 函数后，就可以从 Windows 中调用。其代码被调入内存时，Windows 将改变其入口的前两个字节。

```
NOP
NOP
NOP
⋮
INC BP Prolog
PUSH BP
MOV BP,SP
PUSH DS
MOV DX,AX
SUB SP,Number
```

在这里，前面的 PUSH DS 和 POP AX 指令被改为 NOP 指令。这样，入口把数据段记存器的值存入堆栈中，并把 AX 寄存器的值赋给了数据段寄存器。

这个改变保证了函数不会被直接调用，因为 DS 寄存器而不是应用程序的数据段寄存器包含了一个随机值。

在讨论为什么作这种改变之前，先看一下入口的另一个用途。DLL 中的所有函数都由远程函数、编译开关-Gw 和上面提到的入口及出口构成。在调入 DLL 时，Windows 按下面的方法修改前三个字节。

```
MOV AX,dlladdress
⋮
INC BP Prolog
PUSH BP
```

```
MOV BP,SP
PUSH DS
MOV DX,AX
SUB SP,Number
```

DLL 的数据段地址(程序中的 dlladress)被存入 AX 寄存器。一般每个 DLL 只有一个实例,因此也只有一个数据段。入口指令将数据段存入堆栈中,然后把 AX 寄存器的值赋给 DS 寄存器。

这保证了 DLL 中的所有函数将访问 DLL 的数据而不是应用程序的数据。

21.1.3 回调函数

回调函数是指在 Windows 应用程序中定义而被 Windows 调用的函数。这些函数一般为远程函数,出现在模块定义文件(.DEF 文件)的 Export 段中。对话框就是回调函数的一个例子。

虽然回调函数可从 Windows 之外激活,但它必须使用当前实例的数据段,而不是数据段寄存器中的 Windows DLL 数据段。

由于上面例子中包含了入口和三个 NOP 指令,所以从入口转移到函数时 AX 寄存器应包含当前实例的数据段。函数 MakeProcInstance 实现这个任务。

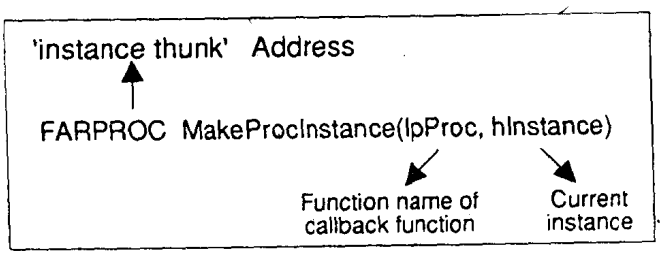


图 21.1

函数 MakeProInstance 传送回调函数的地址和当前实例的句柄,这个函数有一段代码 InstanceThunk,它由两条指令构成,用来读地址并把地址作为返回值。

```
'instance thunk'
MOV AX,dsaddress
JMP seqment:offset
```

值 dsadress 是当前实例的数据段地址。InstanceThunk 的地址被传给 Windows,以便 Windows 能正确地访问回调函数。

之后,寄存器 AX 将提供正确的数据段地址,程序转移到入口代码,并开始执行回调函数。

图 21.2 说明了这个过程。

对话框例程必须被定义为远程函数,并在 Windows 调用它之前输出到模块定义文件(.DEF 文件)中。为了建立一个 InstanceThunk,还需要函数 MakeProcInstance。

指向这两行代码的值 lpprocDia 被传送给函数 DialogBox。当对话框例程开始执行时,程序转移到 InstanceThunk,并从此转移到对话框函数的入口代码。

图 21.3 说明了这个例子的汇编语言形式:

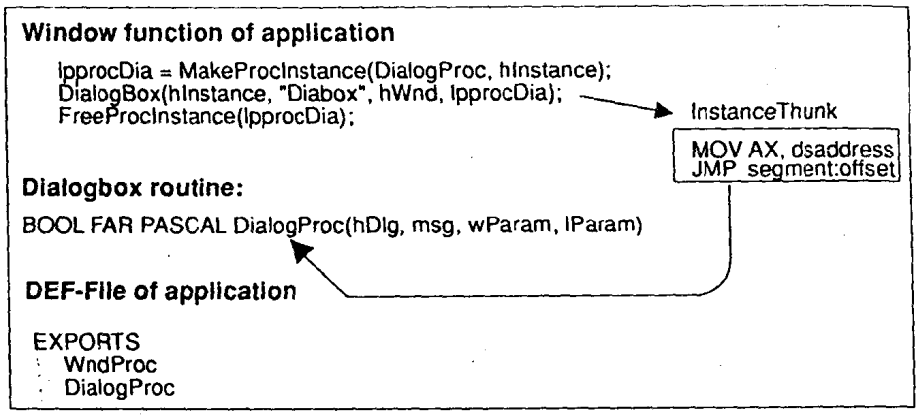


图 21.2 对话框例程

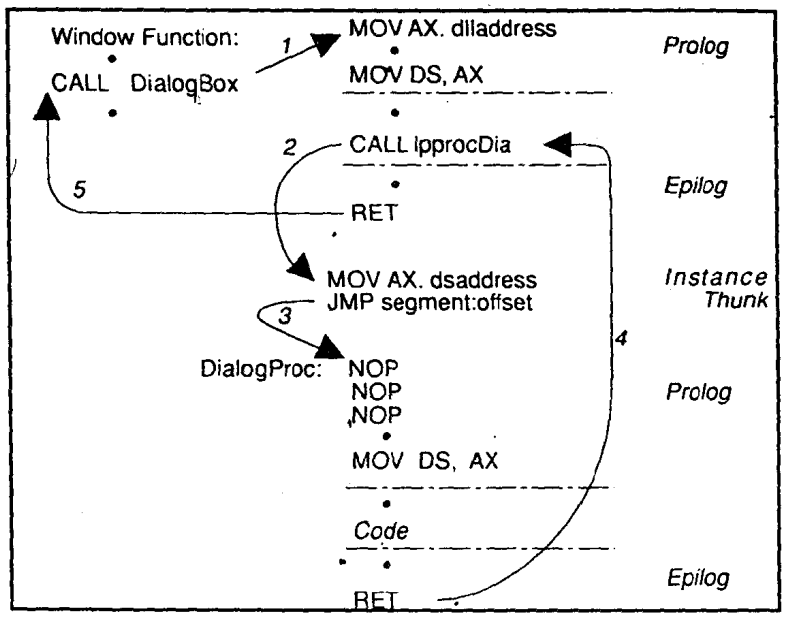


图 21.3 前例的汇编语言代码

一旦 Windows 应用程序调用了这个回调函数，程序便转移到 USER.EXE DLL 中的入口。入口把 DLL 数据段的地址调入数据段寄存器中。在调用对话框例程时，应用程序的数据段地址还要重新赋值到 DS 寄存器中。

InstanceThunk 和对话框例程的入口完成这些功能。Thunk 把当前实例的数据段地址赋给 AX 寄存器，然后入口又把 AX 寄存器的值赋给 DS 寄存器。于是，对话框函数便可在正确的数据段上进行操作了。

虽然几乎所有 Windows 应用程序所使用的窗口函数必须被说明为远程函数并放置在模块定义文件的 EXPORTS 段中，但是函数 MakeProcInstance 并不一定需要包含 InstanceThunk。

21.2 限 制

虽然DLL有自己的数据段，却没有自己的堆栈。DLL函数使用应用程序的堆栈。一旦运行到DLL时，数据段寄存器(DS)的值改为DLL数据段的地址，而堆栈段寄存器(SS)则不变。

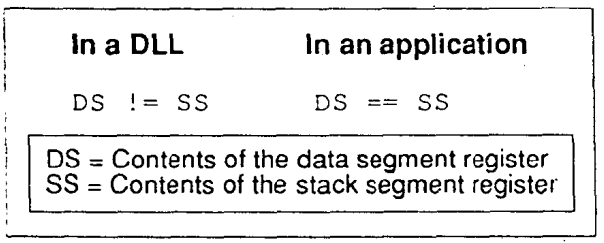


图 21.4

但是，C语言总是假定数据段寄存器(DS)与堆栈段寄存器相等，这是语言的变址方式决定的。

所有局部数据(被说明为static)和所有全程数据都放在应用程序的数据段中。编译器使用近程指针和数据寄存器(DS)进行数据寻址，堆栈则接收函数传送的未说明为static的局部变量。

在Windows中，编译器也使用近程指针，但访问数据时不是用DS寄存器而是用堆栈段寄存器(SS)。因此，近程指针既和DS寄存器有关又和SS寄存器有关。因为只能访问地址偏移量而不能访问段地址，所以编译器将无法判断近程指针使用的是哪个段寄存器。因此，C程序把数据段地址同时放到DS寄存器和SS寄存器之中。

现在让我们看一个使用C语言函数strcpy的例子，它清楚地说明了刚才的原理。这个例子将常量字符串"Hello"拷贝到变量Array中去，这个常量位于数据段中，而变量Array则位于堆栈中。

因为在Windows应用程序中数据段寄存器和堆栈段寄存器都包含着数据段地址，所以函数strcpy使用近程指针可以正确地访问这两个数据。

但是，当Windows DLL使用这个函数时，将得到意外的结果。

常量字符串位于DLL的数据段中，而变量Array则位于堆栈段中，但函数strcpy将通过数据段寄存器来访问这两个值，因为DLL的数据段地址存于数据段寄存器中。

于是，串"string"被拷贝到DLL的数据段中而不是变量Array中了。

因此，在开发DLL时，应避免在C函数中使用近程指针，除非这些近程指针所指的变量被说明为static或全程的。

应该使用Windows的lstrpy而不使用strcpy。这两个函数的参数类型都为LPSTR，这个类型包含了段地址和偏移量两部分，可以明确地说明数据的地址。

除了函数lstrcpy外，还有几个与通常C函数相对应的处理远指针的Windows函数：

- lstrcat
- lstrcmp

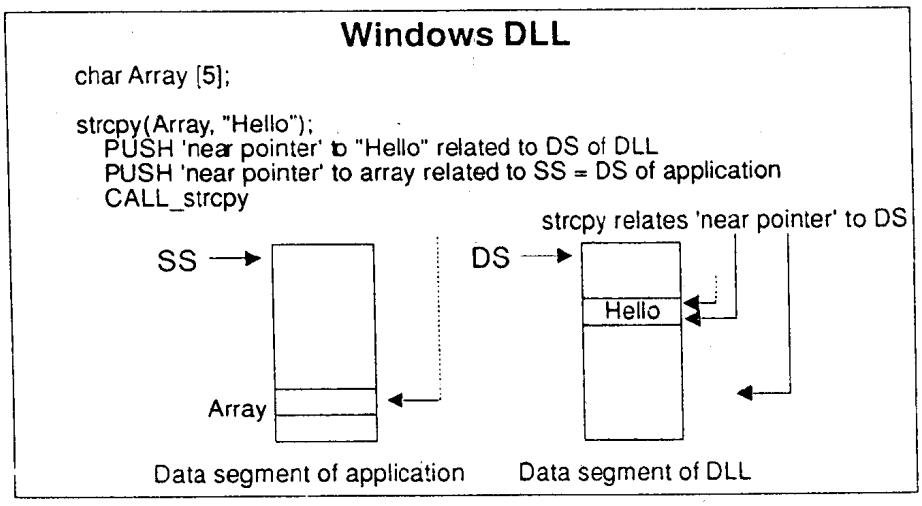


图 21.5 Windows DLL

- lstrcmpi
- lstrcpy
- lstrlen

DLL 编程还有其他一些限制，比如 DLL 函数不能调用参数个数不确定的 C 函数。编译之后，参数变量由堆栈的偏移量决定而不是由整个地址决定。这个问题也可由把普通 C 函数转换为特定的 Windows 函数来解决，例如把通常的 sprintf 函数改为 Windows 的 wsprintf 函数。

DLL 函数的指针参数也必须是远程的。在 Windows 中已经定义了很多数据类型，如 LPLOGPEN 和 LPRECT。

21.3 建立 DLL

DLL 可用来定义函数或资源。

21.3.1 源代码

DLL 可以包含一个初始化例程，此例程仅在应用程序首次调用 DLL 时被执行。像其他应用程序一样，它将增大一个内部指针的值。任何访问局部堆的程序都必须包含此初始化例程。

用这个例程可以在全程变量或寄存器中设置初始值。如果想访问某个寄存器，则需用汇编语言编写例程。SDK 中包含一个例子 LIBENTRY. ASM。

大多数 DLL 函数都必须加入选项 PASCAL，以保证代码更短，执行速度更快。

除了以上的限制外，DLL 函数与其他 Windows 应用程序没有什么区别。大多数 DLL 应包含头文件 WINDOWS. H。

21.3.2 资源

八种标准资源类型和自定义资源类型可作为资源 DLL。资源 DLL 对使用了外语的应用程序十分有用。资源 DLL 比标准的资源文件更为灵活。例如，更新 DLL 不需要修改应用程序。

资源 DLL 需要一个与普通 .RC 文件相似的 .RC 文件，这个文件由一个包含 WEP 函数的 C 文件和一个模块定义文件(.DEF 文件)构成。用正确的参数编译并链接这些文件，便得到一个资源 DLL。现在便可从应用程序中调用这个 DLL 了。

21.3.3 WEP

Windows 中的所有 DLL 均必须包含一个称为 WEP 的尾函数，当系统中止或 DLL 的内部计数器被置为零时将调用这个函数。

WEP 函数可运行许多结束例程，如关闭一个数据库，使得 DLL 不能再访问它。这种函数的最简形式如下：

```
VOID FAR FASCAL WEP( int bSystemExit )
{
    if( bSstemExit )
    {
        /* System Shutdown */
    }
    else
    {
        /* DLL counter = 0 */
    }
}
```

参数 bSystemExit 指示应关闭整个系统还是仅关闭 DLL。模块定义文件的 EXPORTS 段必须包含 WEP 函数(可参看下一节)。

21.3.4 模块定义文件

DLL 需要一个模块定义文件。下面是 DLL 模块文件的示例：

```
LIBRARY OwnDLL
DESCRIPTION 'Your own DLL'
EXETYPE WINDOWS
STUB WINSTUB.EXE
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE SINGLE
HEAPSIZE 0
EXPORTS WEP @10 RESIDENTNAME
```

语句 LIBRARY 指明了此模块是一个动态链接库，语句 DESCRIPTION，EXETYPE 和 STUB 与应用程序中语句的含义相同。

语句 DATA 可以包含关键字 SINGLE 或 NONE。这意味着 DLL 要么只有一个关于全程或 Static 数据的数据段，要么一个也没有。不能有多个数据段。

如果 DLL 使用自己的局部堆，参数 HEAPSIZE 将定义这个堆的大小。在第一次调用 DLL 时，必须用一个初始函数来初始化这个堆。通常，局部堆的大小被置为零。

语句 STACKSIZE 把堆大小赋给应用程序而不是 DLL，因为 DLL 不应有自己的堆栈。

EXPORTS 版本是必需的。WEP 函数必须可用于输出。WEP 函数通过一个序号和语句 RESIDENTNAME 使得系统能尽快地找到它。语句 RESIDENTNAME 保存内存中的输出信息。

由于有了 WEP 函数，DLL 必须包含一个可被指定为 MOVEABLE 或 DISCARDABLE 的数据段，同时，所有可被其他模块调用的函数必须被输出。

21.3.5 编译和链接 DLL

编译 DLL 时，C 编译器除了通常的编译开关外，还需要加上-Asnw 和-Alnw。这些编译开关指明了编译 DLL 时使用的内存模式。

开关-Asnw 和-Alnw 对访问数据的近程指针有用。字符“S”指示 Small 内存模式，字符“L”指示 Large 内存模式。

两个开关都可以通知编译程序，数据段和堆栈段是同一个段，因为编译程序不能假定这两个段是相同的，所以这两个开关是必需的。

当编译程序判断（通过一个赋值或一个标准 C 函数）有一个存于堆栈中的变量被近程指针访问时，将通过数据段进行远程寻址。

链接程序有一个附加的输出库 DLLCEW. LIB，这个库包含启动程序的动态链接库和动态链接库 KERNEL. EXE 中的例程所需要的信息。

如果 DLL 不包含任何资源，在编译和链接之后，也要使用资源编译器。

```
RC DLLFCT. DLL
```

运行资源编译器后，就可以访问这个动态链接库了。应用程序 IMPLIB 产生一个输出库，这个库用来指示 DLL 中的程序。把这个输出库链接到应用程序上便可以访问这些函数了。

有些 DLL 可以被编译并链接为可执行文件，像 GDI. EXE，KERNEL. EXE 和 USER. EXE 等。编译和链接应用程序的很多原则对可执行的 DLL 同样适用。

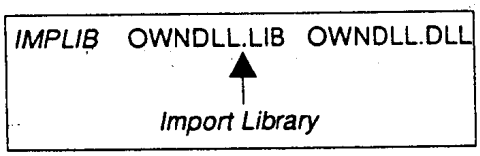


图 21.6 建立一个输出库

21.3.6 把 DLL 加到应用程序中

为了使用 DLL 中的函数，必须把函数输出到应用程序中，只需把适当的输出库链到应用程序上或在模块定义文件中插入 IMPORTS 语句即可。

如果函数 DLLFCT1 有序号（比如是 1），则可以将这个模块定义文件用于输出。

假如一个 DLL 需把其他的 DLL 作为资源进行访问，则这些 DLL 也需要被调入：

```
HANDLE LoadLibrary(LpLibFileName)
```

下面有两个例子，一个调入标准 DLL，另一个调入作为可执行文件的 DDL(GDI. EXE)：

```
HANDLE hLib;
```

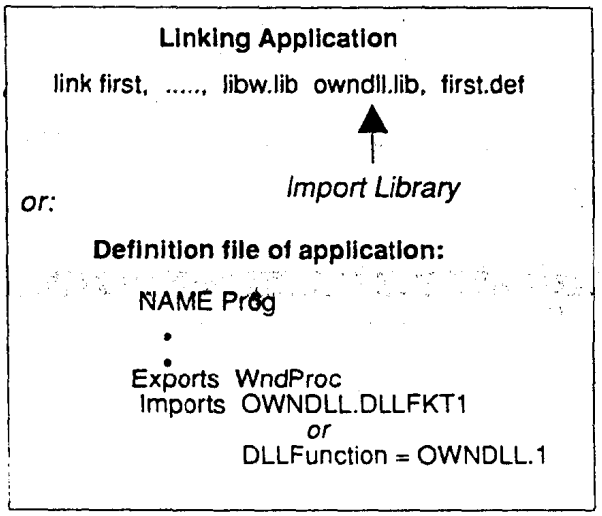


图 21.7 链接一个应用程序

```
hLib = LoadLibrary( "OWNRCDLL. DLL" );

HANDLE hLib;
    hLib = LoadLibrary( "OWNRCDLL. EXE" );
```

返回的句柄指明了被调入的模块实体。如果调入失败,返回值将小于 32。有效的句柄将被用于所有调入资源的子程序。

下列程序调用了资源描述文件中定义的资源,尽管应用程序不需要 DLL。

```
    hAccel = LoadAcceterator (hInstance,"AccelTable");
```

下面用 DLL 中定义的资源调用同一个资源:

```
    hAccel = LoadAccelerator (hLib,"AccelTable")
```

函数 FreeLibrary 在应用程序中止时释放库。

函数 LoadLibrary 也可在运行时把一个库模块链接到应用程序上。如果这样做,则不需要输出库的语句,也不需要模块定义文件中的 IMPORT 语句。

下例说明了这种情况。文件 OWNDLL. DLL 定义了函数 TreeOutput,这个函数以序号 5 输出到模块定义文件中。为了调用这个函数,应用程序中必须有下面几行代码:

```
HANDLE hLib;
FARPROC lpfnTreeO;

hLib = LoadLibrary( "OWNDLL. DLL" )

lpfnTreeO = GetProcAddress( hLib,MAKEINTRESOURCE( 5 ) );
```

或

```
lpfnTreeO = GetProcAddress( hLib,"TreeOutput" );

( * lpfnTreeO)();
FreeLibary( hLib );
```

首先,函数 GetProcAddress 用序号或所需函数的地址调入 DLL。执行完此函数后,函数

FreeLibrary 释放掉该库并递减内部 DLL 计数器。

21.4 DLL 示例

下面两个访问 DLL 的例子，一个显示了如果把 DLL 用作全程资源文件，另一个显示了如何用 DLL 来传送一组函数。

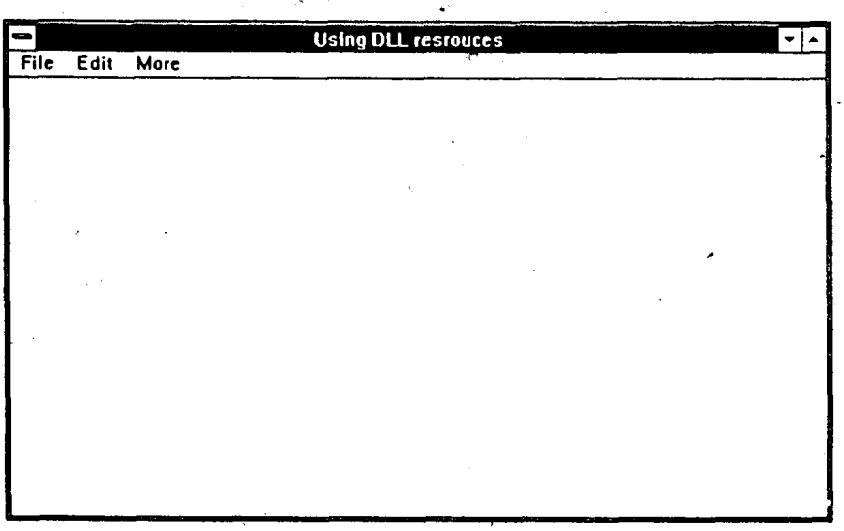


图 21.8

21.4.1 访问 DLL 资源的例子

这个例子包含有自己的光标和图标。使用 Windows 软件开发工具箱(SDK)中的 IMAGEDIT. EXE 便可建立图标。

菜单框包含三个菜单项。选中菜单 More 中的菜单项 Append 或键入 Ctrl+A 便可从一个串表中得到一个新的菜单项名，并把这个新菜单项加入到菜单 More 中。本例中的所有资源都存于类似 KERNEL. EXE 的可执行的资源 DLL 中。

菜单 File 和 Edit 及其各自的菜单项是虚设的。

我们首先列出 DLL 信息和编译链接信息，然后给出应用程序。

新函数	描述
FreeLibrary	递减 DLL 计数器，当计数器为 0 时，把 DLL 从内存中移去
LoadLibrary	把库模块调入内存

源代码：DLLRC. C

```
/* * DLLRC. C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Source code for DLLRC. DLL dynamic link library * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows. h"                          // Include windows. h header file
```

```
VOID FAR PASCAL WEP (int);                                    // Declare WEP

VOID FAR PASCAL WEP (int bSystemExit)
{
    if (bSystemExit)                                          // Error?
    {                                                         // Then
        MessageBeep(0);                                       // -BEEP-
        MessageBeep(0);                                       // twice
                                                              // System Shutdown
    }
    else                                                      // Everything OK?
    {                                                         // Then
        MessageBeep(0);                                       // beep once
                                                              // DLL counter = 0
    }
}
```

DLL 模块定义文件:DLLRC. DEF

```
LIBRARY DLLRC
DESCRIPTION 'DLL with resources'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA NONE
HEAPSIZE 0
EXPORTS WEP @10 RESIDENTNAME
```

DLL 资源描述文件:DLLRC. RC

```
#include "dll. h"

house ICON house. ico
pointer CURSOR pointer. cur

DLLMenu MENU
BEGIN
    POPUP "File"
    BEGIN
        MENUITEM "New", ID_NEW
        MENUITEM "Open", ID_OPEN
        MENUITEM "Exit", ID_EXIT
    END

    POPUP "Edit"
    BEGIN
        MENUITEM "Cut", ID_CUT
        MENUITEM "Copy", ID_COPY
        MENUITEM "Paste", ID_PASTE
        MENUITEM "Delete", ID_DEL
    END
```

```
    POPUP "More"
    BEGIN
        MENUITEM "Append\tCtrl+A", ID_APP
    END
END

DLLAccel ACCELERATORS
BEGIN
    "^A",ID_APP
END

STRINGTABLE
BEGIN
    STR1, "America"
    STR2, "Africa"
    STR3, "Australia"
    STR4, "Asia"
    STR5, "Europe"
END
```

DLL 头文件:DLL.H

```
#define ID_NEW 20
#define ID_OPEN 21
#define ID_EXIT 22
#define ID_CUT 23
#define ID_COPY 24
#define ID_PASTE 25
#define ID_DEL 26
#define ID_APP 27

#define STR1 10
#define STR2 11
#define STR3 12
#define STR4 13
#define STR5 14
#define STR6 15
```

DLL MAKE 文件:DLLRC.MAK

```
DEBUG =.0
PROGTYPE = 2
CFLAGS =-ASw -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -D_WINDLL -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = pointer.cur house.ico
H = DLL.H

all: DLLRC.DLL

DLLRC.RES: DLLRC.RC $(RESFILES)
```

```
DLLRC.OBJ: DLLRC.C $(H)

DLLRC.DLL: DLLRC.OBJ DLLRC.RES DLLRC.DEF
    link /NOD/NOE/A:16 DLLRC,DLLRC.DLL,nul,libw sdllcew,DLLRC.def
    rc DLLRC.RES DLLRC.DLL
```

像下面这样运行 NMAKE 便得到动态链接库 DLLRC：

```
    nmake dllrc.mak
```

源代码：FIRSTDRC.C

```
/** FIRSTDRC.C ***************************/
/** Improved version of FIRST.C application described in second chapter. **/
/** This application accesses external resources for information. **/
/*****************************************/

#include "windows.h"                          // Include windows.h header file
#include "dll.h"                              // Include dll.h header file

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

HANDLE hLib;                                  // Handle to library

BOOL FirstInit ( HANDLE );
long FAR PASCAL FirstWndProc( HWND,unsigned,WORD,LONG);

/** WinMain (main function for every Windows application) *******/

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;               // Current & previous instances
LPSTR lpszCmdLine;                            // Long ptr to string after
                                              // program name during execution
int cmdShow;                                  // Specifies the application
                                              // window's appearance
{
    MSG msg;                                  // Message variable
    HWND hWnd;                                // Window handle
    HANDLE hAccel;                            // Accelerator handle

    hLib = LoadLibrary("dllrc.dll");          // Load DLL
    if (hLib < 32)                            // If DLL cannot be loaded
        return FALSE;                         // return FALSE

    if (! hPrevInstance)                      // Initialize first instance
    {
        if (! FirstInit( hInstance ))         // If initialization fails
            return FALSE;                     // return FALSE
    }
    /** Specify appearance of application's main window *********/
```

```
    hWnd = CreateWindow("First",                          // Window class name
           "Using DLL resrouces",                         // Window caption
           WS_OVERLAPPEDWINDOW,                           // Overlapped window style
           CW_USEDEFAULT,                                 // Default upper-left x pos.
           0,                                             // Upper-left y pos.
           CW_USEDEFAULT,                                 // Default initial x size
           0,                                             // y size
           NULL,                                          // No parent window
           LoadMenu(hLib,"DLLMenu"),                      // Window menu used
           hInstance,                                     // Application instance
           NULL);                                         // No creation parameters

    ShowWindow( hWnd,cmdShow );                           // Make window visible
    UpdateWindow( hWnd );                                 // Update window

    hAccel = LoadAccelerators(hLib,"DLLAccel");           // Get DLL accelerators

    while (GetMessage(&msg,NULL,0,0))                     // Message reading
    {

        /* * Accelerator translation * * * * * * * * * * * * * * * * * */

        if (! TranslateAccelerator(hWnd,hAccel,&msg))
        {
            TranslateMessage(&msg);                       // Message translation
            DispatchMessage(&msg);                        // Send message to Windows
        }
    }

    return (int)msg. wParam;                              // Return wParam of last message
}

BOOL FirstInit( hInstance )                               // Initialize instance handle
HANDLE hInstance;                                         // Instance handle
{

    /* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcFirstClass;                                // Main window class

    wcFirstClass. hCursor = LoadCursor( hLib,"pointer" );
                                                          // Mouse cursor
    wcFirstClass. hIcon = LoadIcon( hLib,"house" );
                                                          // Default icon
    wcFirstClass. lpszMenuName = NULL;                    // No menu
    wcFirstClass. lpszClassName = "First";                // Window class
    wcFirstClass. hbrBackground = GetStockObject( WHITE_BRUSH );
                                                          // White background
    wcFirstClass. hInstance = hInstance;                  // Instance
    wcFirstClass. style = 0;                              // Horizontal & vertical redraw
                                                          // of client area
```

```
    wcFirstClass.lpfnWndProc = FirstWndProc;                    // Window function
    wcFirstClass.cbClsExtra = 0 ;                               // No extra bytes
    wcFirstClass.cbWndExtra = 0 ;                               // No extra bytes

    if (! RegisterClass( &wcFirstClass ) )                      // Register window class
        return FALSE;                                           // Return FALSE if registration fails
                                                                // If registration is successful,
    return TRUE;                                                // Return TRUE
}

/* * FirstWndProc * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL FirstWndProc( hWnd,message,wParam,lParam )
HWND hWnd;                                                      // Window handle
unsigned message;                                               // Message type
WORD wParam;                                                    // Message-dependent 16 bit value
LONG lParam;                                                    // Message-dependent 32 bit value
{
    static int i = STR1;
    static HMENU hSubMenu,hMenu;
    char szString[12];

    switch (message)                                            // Process messages
    {
        case WM_CREATE:                                         // Create window
            hMenu = GetMenu(hWnd);
            hSubMenu = GetSubMenu(hMenu,2);
            break;
        case WM_COMMAND:                                        // Add to More menu
            if (wParam == ID_APP)
            {
                LoadString(hLib,i,(LPSTR)szString,12);
                AppendMenu(hSubMenu,MF_ENABLED,i,szString);
                i++;                                            // Add a string to this menu
                if (i > STR5)                                   // i greater than 5?
                    i = STR1;                                   // Append STR1
            }
            break;
        case WM_DESTROY:                                        // Send WM_QUIT
            FreeLibrary(hLib);                                  // Release library
            PostQuitMessage(0);
            break;                                              // End of this message process
        default:                                                // Send other messages to
                                                                // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                              // End of this message process
    }
    return(0L);
}
```

DLL 模块定义文件:FIRSTDRC. DEF

```
NAME FirstDrc

DESCRIPTION '1st Windows Application with DLL - resource capability'

EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS FirstWndProc @1
```

DLL MAKE 文件:FIRSTDRC. MAK

```
DEBUG = 0
PROGTYPE = 2
CFLAGS =-ASw -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -D_WINDLL -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = DLL.H

all: FIRSTDRC.EXE

FIRSTDRC.OBJ: FIRSTDRC.C $(H)

FIRSTDRC.EXE: FIRSTDRC.OBJ FIRSTDRC.DEF
    link /NOD/NOE/A:16 FIRSTDRC,,nul,libw slibcew,FIRSTDRC.def
    rc FIRSTDRC.EXE
```

像下面这样运行 NMAKE 便可得到应用程序 FIRSTDRC:

```
nmake firstdrc.mak
```

FIRSTDRC. EXE 和 DLLRC. DLL 是怎样运行的

动态链接库的 .RC 文件由光标和图标信息、包含五个串的串表、包含键盘加速键 Ctrl+A 的键盘加速键(accelerator)表、包含三个子菜单的菜单构成。

源代码中需要 WEP 函数。函数 MessageBeep 在 DLL 计数器减小到零时发出蜂鸣声。当用户终止 Windows 系统时,WEP 函数转移到 IF 语句分支,并且计算机将蜂鸣两声。

在模块定义文件中,关键字 LIBRARY 通知链接程序链接一个 DLL。因为资源 DLL 不包含任何数据,所以 DATA 命令中包含单词 NONE。WEP 函数必须被输出,以便 Windows 能访问这个 DLL。

源代码和 .RC 文件都将使用 DLL 资源，它们也应包含头文件 DLL.H，这个头文件定义了菜单项和串的 ID 值。

我们把应用程序 FIRSTDRC.EXE 从前面讲述的应用程序 FIRST.EXE 改编为动态链接库，库句柄存储为全程变量，以便被函数 WinMain 和窗口数访问。函数 LoadLibrary 在开始运行应用程序时把文件 DLLRC.DLL 调入内存。

如果 DLL 既不在当前目录下，也不在 PATH 指定的路径中，则 Windows 将显示一个消息框来通知用户在 A：盘中插入装有 DLL 的软盘。用户可按下 Retry 或 Cancel 按钮来重试或取消这次对 DLL 的访问。如果用户按下了 Cancel，则 DLL 句柄将包含一个出错码（即一个小于 32 的值）并且终止应用程序。

WNDCLASS 结构将调入 DLL 句柄（而不是实例句柄）定义的光标和图标。菜单句柄 DLLMenu 的倒数第三个参数负责传送建立窗口所需的数据，这些数据是函数 LoadMenu 从 DLL 中获得的。为调入加速键表而使用的 DLL 句柄也将被调入。

窗口函数将发送三条消息：WM_CREATE、WM_COMMAND 和 WM_DESTROY。消息 WM_CREATE 给菜单 More 提供菜单句柄，这个句柄还将被消息 WM_COMMAND 使用。

然后便可得到菜单项 Append 的 ID 值，当此菜单项被选中的时候，一个变量将从 DLL 资源文件所定义的串表中得到一个字符串。选中 Append 菜单项可把这个串加入到菜单 More 中。到达表尾时，表又从头开始计数。

在应用程序结束之前，函数 FreeLibrary 将把内部计数器减一。当计数器被置为零时，就是说没有其他应用程序正在使用 DLL，于是 Windows 把这个 DLL 从内存中删去。

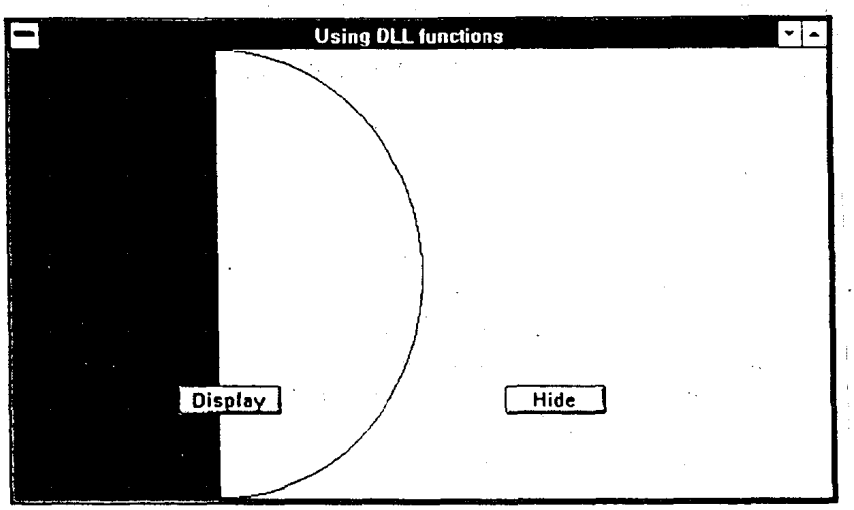


图 21.9

21.4.2 DLL 函数访问示例

应用程序 FIRST.EXE 用一个文件 DLLFUNC.DLL 中的函数来产生风格为 PUSHBUTTON 的子窗口，其中的按钮允许用户画出或擦去一个椭圆或一个矩形。

根据所选颜色的不同，矩形可以用栅格进行“粉刷”或者填充颜色。回调函数控制着这些图画的显示。

源代码：DLLFUNC. C

```
/* * DLLFUNC. C * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Source code for DLL, supplying functions to FIRSTDFC. EXE application * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include "windows. h"                                    // Include windows. h header file
#include "dllfunc. h"                                    // Include dllfunc. h header file

VOID FAR PASCAL WEP (int);                               // Declare WEP

VOID FAR PASCAL WEP (int bSystemExit)
{
    if (bSystemExit)                                     // Error?
    {
        // System Shutdown
    }
    else                                                 // Everything OK?
    {                                                    // Then exit
        // DLL counter = 0
    }
}

/* * Create push button * * * * * * * * * * * * * * * * * * * */

DWORD FAR PASCAL CreatePushB (HWND hWndParent, LPSTR lpButtonName)
{
    HANDLE hInst;
    HWND hButton;
    DWORD dButton;
    static WORD idButton = 0;

    idButton++;
    hInst = GetWindowWord(hWndParent, GWW_HINSTANCE);
    hButton = CreateWindow("button", lpButtonName,
            BS_PUSHBUTTON | WS_VISIBLE | WS_CHILD, 0, 0, 0, 0,
            hWndParent, idButton, hInst, NULL);
    dButton = MAKELONG(hButton, idButton);
    return (dButton);
}

/* * Specify screen * * * * * * * * * * * * * * * * * * * * * */

BOOL FAR PASCAL Screen(HDC hDC, POINT ptRec, POINT ptEll, FARPROC lpCallScreen)
{
    int iColorIndex;
    HBRUSH hBrush;
    RECT rect;
```

```
    Ellipse(hDC,0,0,ptEll.x,ptEll.y);

    iColorIndex = (*lpCallScreen)();

    switch (iColorIndex)
    {
        case 1:
            hBrush = CreateHatchBrush(HS_HORIZONTAL,0x00FF0000);
            break;
        case 2:
            hBrush = CreateHatchBrush(HS_CROSS,0x0000FF00);
            break;
        case 3:
            hBrush = CreateHatchBrush(HS_VERTICAL,0x000000FF);
            break;
        default:
            hBrush = CreateSolidBrush(0x00000000);
            break;
    }
    rect.left = 0;
    rect.top = 0;
    rect.right = ptRec.x;
    rect.bottom = ptRec.y;
    FillRect(hDC,&rect,hBrush);
    DeleteObject(hBrush);
    return (TRUE);
}
```

DLL 模块定义文件:DLLFUNC.DEF

```
LIBRARY DLLFUNC
DESCRIPTION 'DLL with functions'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE DISCARDABLE
DATA PRELOAD MOVEABLE SINGLE
HEAPSIZE 0
EXPORTS CreatePushB @1
    Screen @2
    WEP @3 RESIDENTNAME
```

DLL 头文件:DLLFUNC.H

```
DWORD FAR PASCAL CreatePushB (HWND,LPSTR);
BOOL FAR PASCAL Screen(HDC,POINT,POINT,FARPROC);
```

DLL MAKE 文件:DLLFUNC.MAK

```
DEBUG = 0
PROGTYPE = 2
CFLAGS =-ASw -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -D_WINDLL -nologo

.rc.res: ; rc -r $*.rc
```

```
RESFILES =
H = DLLFUNC.H

all: DLLFUNC.DLL

DLLFUNC.OBJ: DLLFUNC.C $(H)

DLLFUNC.DLL: DLLFUNC.OBJ DLLFUNC.DEF
    link /NOD/NOE/A:16 DLLFUNC,DLLFUNC.DLL,nul,libw sdllcew,DLLFUNC.def
    rc DLLFUNC.DLL
    implib DLLFUNC.LIB DLLFUNC.DLL
```

像下面这样运行 NMAKE 便可得到动态链接库 DLLFUNC：

```
nmake dllfunc.mak
```

源代码：FIRSTDFC.C

```
/** FIRSTDFC.C ************************/
/** Improved version of FIRST.C application described in second chapter. **/
/** This application accesses a DLL function for outside information. **/
/*************************************/

#include "windows.h"                          // Include windows.h header file
#include "dllfunc.h"                          // Include dllfunc.h header file

#if _MSC_VER >=700
#pragma warning (disable:4028)
#endif

BOOL FirstInit ( HANDLE );
int FAR PASCAL ScreenColor(VOID);
long FAR PASCAL FirstWndProc( HWND,unsigned,WORD,LONG);

/** WinMain (main function for every Windows application) *******/

int PASCAL WinMain( hInstance,hPrevInstance,lpszCmdLine,cmdShow )
HANDLE hInstance,hPrevInstance;               // Current & previous instances
LPSTR lpszCmdLine;                            // Long ptr to string after
                                              // program name during execution
int cmdShow;                                  // Specifies the application
                                              // window's appearance
{
    MSG msg;                                  // Message variable
    HWND hWnd;                                // Window handle

    if (! hPrevInstance)                      // Initialize first instance
    {
        if (! FirstInit( hInstance ))         // If initialization fails
            return FALSE;                     // return FALSE
    }
```

```
    /* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow("First",                                    // Window class name
           "Using DLL functions",                                   // Window caption
           WS_OVERLAPPEDWINDOW,                                     // Overlapped window style
           CW_USEDEFAULT,                                           // Default upper-left x pos.
           0,                                                       // Upper-left y pos.
           CW_USEDEFAULT,                                           // Default initial x size
           0,                                                       // y size
           NULL,                                                    // No parent window
           NULL,                                                    // Window menu used
           hInstance,                                               // Application instance
           NULL);                                                   // No creation parameters

    ShowWindow( hWnd,cmdShow );                                     // Make window visible
    UpdateWindow( hWnd );                                           // Update window

    while (GetMessage(&msg,NULL,0,0))                               // Message reading
    {
        TranslateMessage(&msg);                                     // Message translation
        DispatchMessage(&msg);                                      // Send message to Windows
    }

    return (int)msg. wParam;                                        // Return wParam of last message
}

BOOL FirstInit( hInstance )                                         // Initialize instance handle
HANDLE hInstance;                                                   // Instance handle
{

    /* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcFirstClass;                                          // Main window class

    wcFirstClass. hCursor = LoadCursor( NULL,IDC_ARROW );
                                                                    // Mouse cursor
    wcFirstClass. hIcon = LoadIcon( NULL,IDI_APPLICATION );
                                                                    // Default icon
    wcFirstClass. lpszMenuName = NULL;                              // No menu
    wcFirstClass. lpszClassName = "First";                          // Window class
    wcFirstClass. hbrBackground = GetStockObject( WHITE_BRUSH );
                                                                    // White background
    wcFirstClass. hInstance = hInstance;                            // Instance
    wcFirstClass. style = CS_HREDRAW | CS_VREDRAW;                  // Horizontal & vertical redraw
                                                                    // of client area
    wcFirstClass. lpfnWndProc = FirstWndProc;                       // Window function
    wcFirstClass. cbClsExtra = 0 ;                                  // No extra bytes
    wcFirstClass. cbWndExtra = 0 ;                                  // No extra bytes

    if (! RegisterClass( &wcFirstClass ) )                          // Register window class
        return FALSE;                                               // Return FALSE if registration fails
```

```
                                                    // If registration is successful,
    return TRUE;                                                    // Return TRUE
}

/* * FirstWndProc * * *
```

```
            }
            break;
        case WM_PAINT:
            hDC = BeginPaint(hWnd,&ps);
            if (bScreen)
            {
                pt1.x = xClient/4;
                pt1.y = yClient;
                pt2.x = xClient/2;
                pt2.y = yClient;

                lpCallScreenColor = MakeProcInstance(ScreenColor,hInst);
                Screen(hDC,pt1,pt2,lpCallScreenColor);
                FreeProcInstance(lpCallScreenColor);
            }
            EndPaint(hWnd,&ps);
            break;
        case WM_DESTROY:                                    // Send WM_QUIT
            PostQuitMessage(0);
            break;                                          // End of this message process
        default:                                            // Send other messages to
                                                            // default window function
            return (DefWindowProc( hWnd,message,wParam,lParam ));
            break;                                          // End of this message process
    }
    return(0L);
}

int FAR PASCAL ScreenColor()
{
    DWORD dColor;

    dColor = GetSysColor(COLOR_ACTIVECAPTION);
    switch(dColor)
    {
        case (0x00FF0000):
            return 1;
        case (0x0000FF00):
            return 2;
        case (0x000000FF):
            return 3;
        default:
            return 0;
    }
}
```

DLL 模块定义文件:FIRSTDFC. DEF

```
NAME FirstDfc

DESCRIPTION '1st Windows Application with DLL functions'
```

```
EXETYPE WINDOWS

STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 4096

EXPORTS FirstWndProc @1
ScreenColor @2
```

DLL MAKE 文件:FIRSTDFC. MAK

```
DEBUG = 0
PROGTYPE = 2
CFLAGS =-ASw -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -D_WINDLL -nologo

. rc. res: ; rc -r $ * . rc

RESFILES =
H = DLLFUNC. H

all: FIRSTDFC. EXE

FIRSTDFC. OBJ: FIRSTDFC. C $ (H)

FIRSTDFC. EXE: FIRSTDFC. OBJ FIRSTDFC. DEF
    link /NOD/NOE/A:16 FIRSTDFC, ,nul,libw slibcew DLLFUNC. lib,FIRSTDFC.def
    rc FIRSTDFC. EXE
```

像下面这样运行 NMAKE 便可建立应用程序 FIRSTDFC:

```
nmake firstdfc. mak
```

FIRSTDFC. EXE 和 DLLFUNC. DLL 是怎样工作的

当应用程序调用 DLL 函数 CreatPushB 和 Screen 时,将有以下动作:

消息 WM_CREATE 将产生那些按钮。函数 CreatePushB 将把窗口句柄和文本传送给主窗口,初始时文本为空串。因为每个子窗口都有自己的 ID 值,所以每次调用函数时,计数器 idButtonf 都将递增。在传送了实例的句柄后,函数 CreateWindow 将产生一个按钮。ID 值和窗口句柄被返回给 DWORD 类型的一个变量。

窗口句柄通过函数 MoveWindow 发送消息 WM_SIZE,以获得窗口大小并使得窗口可见。

一旦用户按下了某个按钮,消息 WM_COMMAND 就被发送给父窗口,参数 wParam 记录下这个按纽的 ID 值。如果用户按下了 Hide 按钮,布尔变量 bScreen 将被置为 FALSE(在其他情况下,此变量被置为 TRUE)。函数 InvalidateRect 将产生消息 WM_PAINT。

这个消息读取变量 bScreen 的设置情况,如果置为 TRUE,则在客户区左上角显示图像。DLL 函数 Screen 有四个参数:一个显示环境句柄和两个 POINT 类型的变量(决定图的大

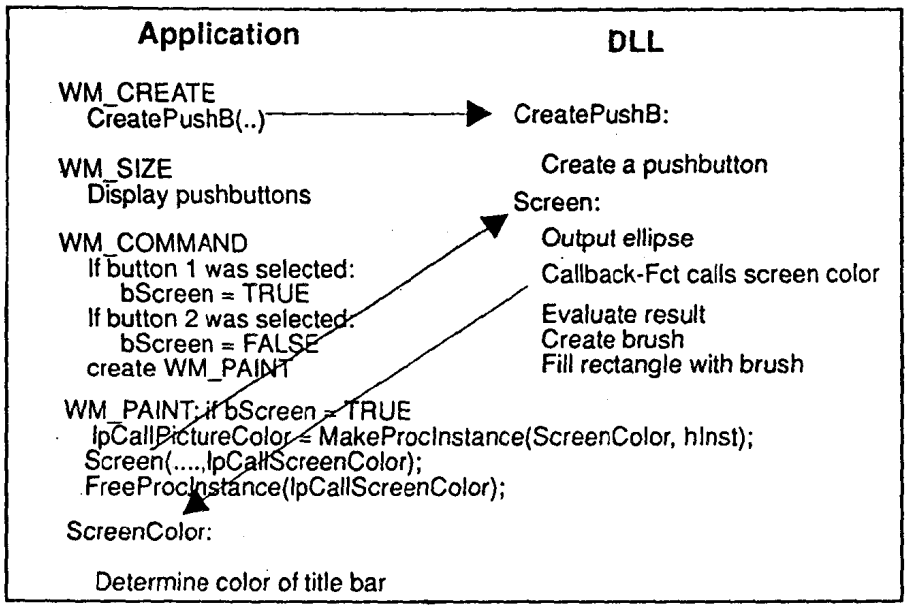


图 21.10　消息的处理过程

小)。客户区的原点即是图像的起点。最后一个参数是回调函数 ScreeColor 传送的 InstanceThunk 的地址,这个地址是由函数 MakeProcInstance 得到的。

DLL 函数 Screen 在指定的坐标处显示一个椭圆。根据回调函数的返回值,将建立一到四个刷子,这些刷子用函数 FillRect 来填充矩形图案。

回调函数 ScreenColor 通过 GetSysColor 来判断当前标题条的颜色,在 Windows 会话中使用控制面板可改变该颜色。根据标题条的颜色是绿、红、还是蓝色将返回 1 到 3 的值,其他情况下返回零值。

若用户按下了 Hide 按钮,布尔变量 bScreen 将被置为 False,函数 BeginPaint 和 EndPaint 可隐藏图像。

DLL 的模块定义文件不仅需要在 EXPORTS 段中提供 WEP 函数,还要提供函数 CreateBrushB 和函数 Screen。

这三个函数在 DLL 中都要被定义为远程的。若 DLL 有数据段,则 DATA 语句中必须包含关键字 SINGLE。

在应用程序的模块定义文件中,必须输出窗口函数和一个远程回调函数。因为已有了输出库,所以 DLL 函数不需要 IMPORTS 语句。

第二十二章　OLE

Windows 环境允许几个程序同时工作，许多事件可以并行处理。但是要使多任务机制真正发挥作用，必须有一个有效的数据交换形式。

22.1　Windows 下的通信

在 Windows 中，程序能够作为一个更大的联合系统的一部分运行，此联合系统包含局部的和远程的两种应用程序。一个应用程序可能要用到来自不同程序的数据，Windows 运用不同的方式获得这一协同操作的能力。

这些方式包括剪贴板、共享内存、DDE 和 DLE。借助剪贴板，用户可使系统数据被所有的应用程序访问。通常，用户手工地向剪贴板拷贝数据，再将数据重定向到另一位置。一般说来，剪贴板是一个需要用户介入的用于传送数据的临时存储单元。当我们提及 DLE 时，都要用到剪贴板。

另一方面，公共内存区只在特别适合于交互的应用程序中才能发挥最好的作用。Windows 内有一种方式就是通过动态链接库(DLL)实现的。不幸的是，使用公共内存区通信的应用程序被限制在一个程序单元中进行数据交换，因为数据交换只有通过标准协议才能进行。

通信问题

Windows 正是在遇到一个又一个问题的情况下“成长”起来的，基于这个原因，我们会碰到越来越多的由不同人员开发的应用程序，Microsoft 向我们提供了动态数据交换(DDE)标准和对象链接及嵌入(DLE)标准。

DDE

DDE(动态数据交换)允许 Windows 应用程序之间通过某种通用语言或某种协议交换文本数据和图形数据。

DDE 标准

DDE 是一个官方的正式通信协议，它基于内部的数据交换。DDE API 函数允许应用程序间的通信。由于 DDE 是一个固定的协议，所以每个应用程序都能在遵守这一协议的情况下参与通信。

DLE 使通信更加方便

DLE 应用程序的原理在于永久性对象的链接和嵌入。一旦某个已嵌入的对象被重新打开，就在客户和服务器之间重新建立一个链接，并完成这种链接和嵌入过程。DDE 不能执行这种任务，因为这种任务需要一个由用户确定的固定链接或由应用程序产生的临时链接。

22.1.1 DDE 的背景

在应用程序间的通信完全独立于用户时，DDE 工作得最好。以下给出一些可能的应用：

- 向电子表格应用程序请求数据，例如 Microsoft Excel。
- 向面向公式的应用程序请求数据，例如 Borland ObjectVision。
- 向字处理器输入图像或文本，如 Microsoft Word for Windows。
- 集成应用程序访问用于作复杂运算的标准应用程序。
- 向不同计算机请求数据，例如 SQL 服务器和主机仿真器。

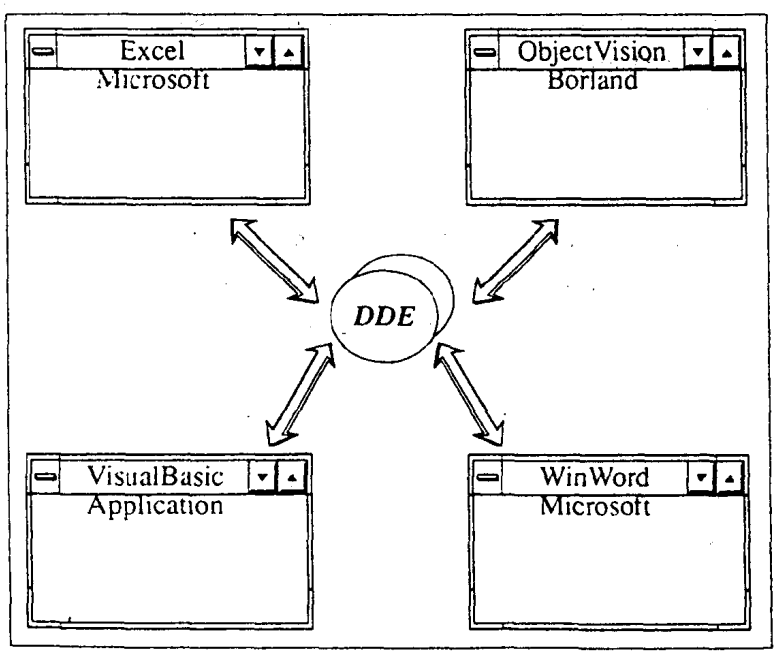


图 22.1 DDE 的几种可能的使用方式

对 DDE 的编程自从 Windows 3.1 出现以来变得更加容易，这主要归功于一系列新的 API 调用。在 Windows 的早期版本中，DDE 要求直接的数据交换。

DDE 访问：API 和宏

宏语言方便了 DDE 的实现。用于 Windows 的 Microsoft Execl 和 Microsoft Word 都体现了 DDE 命令的特色。如果读者拿 API 调用与这些宏指令相比较的话，将会看出两者之间只有少许不同。

· 初始化交换值

```
DDE API:DdeConnect(idInst,hszAppName,hszTopic,lpRes)
WinWord:Channel = DDEInitiate(szAppName,szTopic)
Excel: INITIATE(szAppName,szTopic)
```

· 数据传送

```
DDE API:DdeClientTransaction(lpvData,cbDataLen,
        hConv,hszItem,uFmt,uType,uTimeout,IpuResult)
WinWord:DDEPoke(Conv,DataElement)
Excel: SEND(Conv,DataElement,Data)
```

· 读数据

```
DDE API:DdeClientTransaction(lpvData,cbDataLen,
        hConv,hszItem,uFmt,uType,uTimeout,IpuEResult)
WinWord:Data = DDDERequest $ (Conv,DataElement)
Excel: QUERY(Conv,DataElement)
```

· 执行指令

```
DDE API:DdeClientTransaction(lpvData,cbDataLen,
        hConv,hszItem,uFmt,uType,uTimeout,IpuEResult)
WinWord:Data = DDDERequest $ (Conv,DataElement)
Excel: EXECUTE(Conv,Command)
```

· 结束通信

```
DDE API:DdeDisconnect(hConv)
WinWord:DDETerminate(Conv)
Excel: CHANNEL. CLOSE(Conv)
```

DDE 放置两个应用程序到“说明项”上，一旦一个链接被接通，就自动产生这个安放动作。用户能在两个应用程序间执行连续的数据传送操作。

这种通信类似于人之间的会话，首先是从其中一个应用程序开始的。

用于初始化通信的应用程序称为客户。正如商业领域内一样，客户应用程序希望得到一种服务，所以客户请求服务器应用程序为其服务。应用程序每次只能有一个链接，但是可以在单一的链接上建立多重链接。

而且，一个应用程序可同时作为客户和服务器使用，此程序可以请求服务器为其服务器（履行客户的职责），同时执行对另一个客户应用程序的服务。

由应用程序的不同任务指定哪一个应用程序作为客户，哪一个应用程序作为服务器。

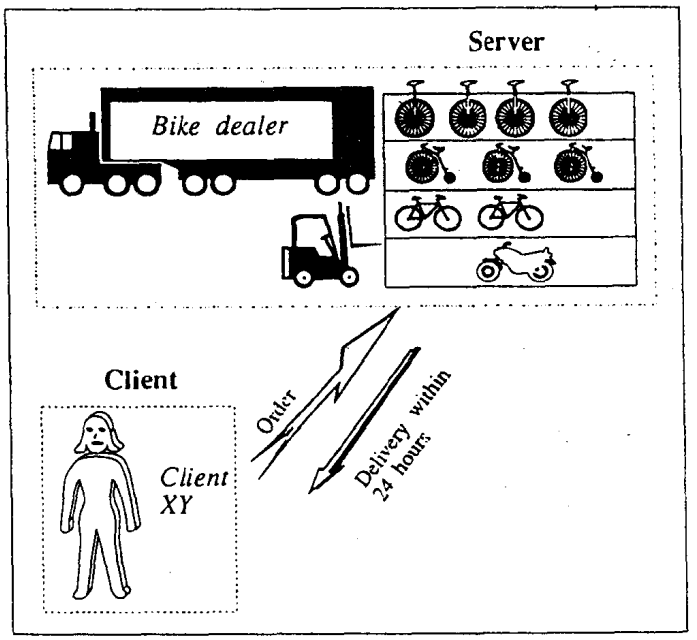


图 22.2　客户－服务器间的关系

22.1.2　生成一个数据链接

在两个应用程序进行通信之前，必须有一个 DDE 链接存在。客户应用程序一般对链接进行初始化，而服务器靠发出所需要的有效数据来应答这个初始化动作。

在产生链接的程序中，必须确定两件事。首先，客户必须声明所希望与之通信的服务器的名称。其次，客户必须表示出会话的主题。

一个 DDE 会话开始时，每当一个服务器应用程序接收到一个查询信息，程序就检查主题和项，以决定服务器是否支持这一信息。如果服务器支持的话，就发出应答，于是就开始会话。

一旦产生一个链接以后，服务器的名称和主题名称就不可以改变。服务器/名称组合唯一地标识相应的链接，并且作强制性的保留，直到通信结束。

但是，如果客户或者服务器在传输过程中改变这些项之一，那么通信就会突然终止。

22.1.3　主动与被动数据链接

一旦初始化 DDE 通信后，客户应用程序就能够指示服务器产生一个固定的数据传送，这就是说，数据元素必须连续更新。这种类型的链接产生一个连续的数据流，这个流直到通信结束都一直存在。这种主动型链接或热链接之所以称为主动的，就在于数据稍有改变则立即激活服务器应用程序。

与主动链接不同的被动链接或温链接(有时称作冷链接)只有当客户特定请求一个更新时才会更新数据。这样，如果用户编辑一个用被动链接链向客户的服务器信息时，链向客户应用程序的信息就保持不变，这就使得客户负责向服务器访问更新信息。

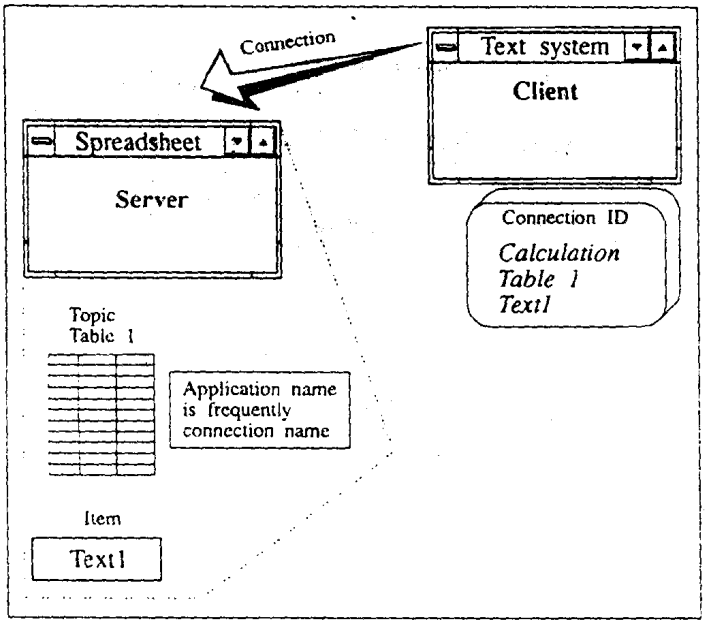


图 22.3　定义名称产生一个 DDE 链接

22.2　对象嵌入和链接:OLE

现在我们已经看到应用程序间的通信是多么重要和有用了。在前面的例子中,应用程序是工具,DDE 是通信的形式。要将 DDE 作为端点(通信的端点)使用,我们需要知道所希望交换的数据的确切类型。

为进一步地进行通信,我们想要另外一个工具。用户可能已了解了剪贴板。剪贴板很好地处理了图像应用程序和字处理应用程序间的数据交换。

剪贴板允许用户粘贴一幅图像到一个文本文档中。如果图像中产生错误,就意味着用户不得不打开 Paintbrush 编辑图像,向剪贴板复制图像,再次打开文本文档,且将已编辑的图像粘贴到文本文档中。发现的错误越多,这一操作重复的次数也就越多。

如果用户能向剪贴板复制图像,并且一次粘贴图像于文本文档后就可以立即自动地用 DDE 更新图像,那么情况就比较好了,这即是 OLE 能为用户所做的事情。

22.2.1　OLE 如何工作

正如 DDE 一样,OLE 也是用于数据和指令交换的标准协议,只有 DDE 不需要操作系统的支持。OLE 以通过 DDE 发送指令的方式建立在 DDE 协议之上,不需要来自用户或程序员的输入。

Windows API 调用有助于 OLE 和 DDE 应用程序的编写。这些函数用 Windows 内的 DLL 处理,处理 OLE 的 DLL 分为用于客户的 OLECLI. DLL 程序库和用于服务器的 OLESVR. DLL 程序库。DDEML. DLL 文件执行所有的 DDE 函数调用。

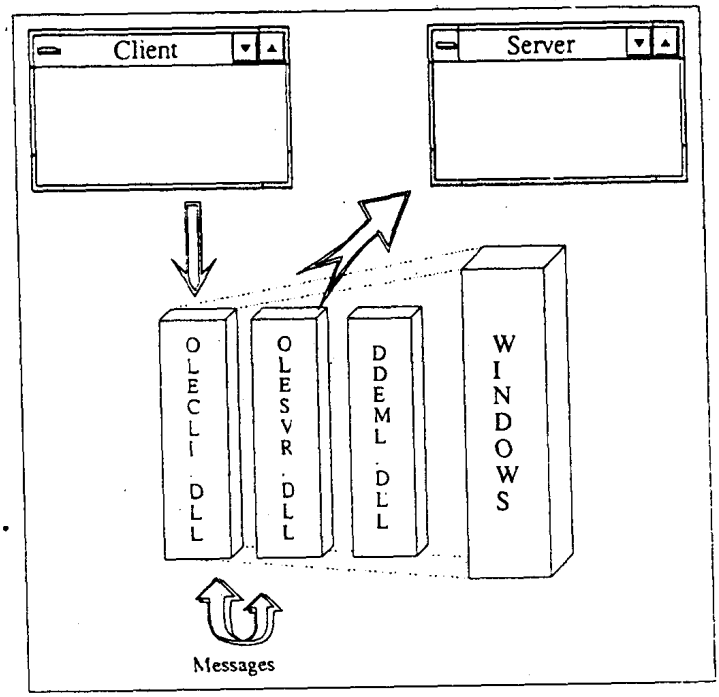


图 22.4 OLE 和 DDE 技术

接管 DDE 函数

因为 OLE 是 DDE 的一个超集，所以 OLE 几乎能够做 DDE 所能做的每一件事，支持 OLE 的程序因而也就能通过一个主题产生一个链接。除此之外，服务器通过项来请求数据。数据在客户和服务器都承认的一定格式下传送。

服务器向客户传送数据的方法与客户向服务器请求数据的方法是一致的。

客户可以以字符串的形式向服务器传送一个命令行，正如 DDE 能够做的一样。与 DDE 不同的是，OLE 还能够处理大量常规数据格式，诸如位图、独立于设备的位图(DIB)和由 OLE 取代的元文件。

DDE 的最佳使用

如前所述，DDE 用于在同一点上需要及时处理多个链接的应用程序，特别是在有许多不断更新的项的地方。一个实时信息系统的实现是一种典型的例子，这个例子与先前一样，证实 DDE 的存在。

22.2.2 重要的 OLE 概念

在 OLE 中不断重复出现的一系列重要概念将在下面作出简要的解释。

OLE 对象

OLE 关心的焦点是对象。OLE 的任务如今不再只是交换数据，而是交换完整的对象。这

些 OLE 对象是可被压缩的各种不同数据，这些不同数据的成分，举例来说是由文本、数字、图像甚至声音组成的。数据封装而成的 OLE 对象能够由每一个 OLE 应用程序使用，而不必知道这些对象的当前结构。

复合文档

用复合文档来存储有很大区别的 OLE 对象。在复合文档中可以将对象收集在一起，并且用其自身的数据装填这些对象。这就好比一本杂志中一页，每页中印有带有图像的文本。

客户

一个客户应用程序是一种能够接收、指向和存储 OLE 对象的程序。

服务器

对象由服务器应用程序按顺序产生，这个应用程序中的数据是可用的并且是可修改的。服务器是唯一需要知道对象中详细数据结构的应用程序。

嵌入

使一个对象成为一个文档的固定成分称为嵌入，它是在文档内部完成的。

链接

仅仅靠指向设置于一文件中的源数据的指针方式来访问对象，对象可以由 OLE 客户说明。即使对 OLE 客户，文件的更新也可能导致其后的自动重新引进。这个更新过程与附属选项无关。

激活对象

激活对象用于准许一个 OLE 客户处理一个已嵌入或已链接的对象，而不必知道数据的结构。

对象包

除了对象嵌入和对象链接以外，也可能以图标形式代表一个对象。

22.2.3 客户和服务器间的协作

嵌入和链接对象

OLE 的主要应用就是固定地嵌入和链接对象。一旦 OLE 对象被激活以后，此处理依靠客户和服务器二者之间重新建立起某种关联来进行。这一任务不能全由 DDE 完成，因为一个固定链接也许不得不由用户产生，或者，一个非固定链接不得不由应用程序产生。

如果需要应用 DDE，那么可轻而易举地根据 DDE 的附加用法来应用它。

服务器的任务

每个服务器都有一些基本属性。如果服务器由 OLE DLL 调用的话，那么服务器程序启

动时就加载入一个对象，这样才可以处理这个对象。服务器还在对象改变时通知 OLE DLL，从而实现对象的更新。如果剪贴板用于 OLE 内的数据传送，那么服务器就必须根据已建立的常量将对象数据保存在剪贴板中。用于数据传送的不只一种格式，还有几种不同的数据格式。

客户的任务

从另一角度上看，客户的任务是接受 OLE 对象，根据协议在文档中说明对象并且在必要时存储这些对象。

22.3 OLE 中的客户

以下我们将着眼于客户应用程序。OLE 由 60 多个函数组成，因而应该是我们在编制一个客户程序时应了解的主要对象。就一个较小的应用程序而言，这不应该像看上去那么困难，但我们有必要对 OLE 应用程序有深刻的理解。

OLE 客户

称为 OLE 客户的是指向一个 OLE 对象并且通常还能存储此对象的应用程序。在这种情况下，OLE 对象是数据的集合，OLE 客户不必了解这些数据的结构，这样的对象一般在其自身的窗口里显示。

举例来说，OLE 对象可以是由 Parintbrush 产生的一幅图，在这幅图中，客户可能并不知道图形的结构。当然，这个结构信息并不重要，因为这是 Paintbrush 程序的任务。在指向一个对象、更新一个对象或者在运行时与服务器通信时，客户都由 OLE API 支持。此时需用的资源是动态链接库 OLECLIEN. DLL。

OLE CLIENT API

客户应用程序借助 OLE'CLIENT 库调用建立与服务器程序的通信链接。客户的任务包括从服务器取得一个 OLE 对象，并且在内存中或在磁盘上单独地存储这个对象。

22.3.1 客户应用程序的能力

现代软件应该具有友好的用户接口，因此，向 OLE 客户应用程序推荐一种确定的结构，它类似于附属于所处理的子菜单的辅助菜单项。

自然地，客户应用程序让每一开发者决定是否遵循它，但是为将来的用户着想，最好使用一个标准化程序。我们已经了解 OLE 对象是通过剪贴板频繁地在服务器和客户之间交换的。

这样，用户就有机会保留菜单选项 Copy、Cut 和 Paste，并且通过菜单项 Paste Link 来扩展。这四个菜单项提供一个用户接口，使用户能够向剪贴板复制和插入 OLE 对象。

有了这四个菜单项，我们自然还没有得到一个 OLE 应用程序，但是这些项提供了一个良好的开端。

Paste

如果我们想开发一个客户应用程序，那么包含 Paste 和 Paste Link 这两个菜单选项是比较重要的。用户激活的菜单项 Paste 不断地将剪贴板内的 OLE 对象插入到文档中的当前位置，这种方法就称作嵌入。

使用 Paste，用户就会理解 OLE 对象是完全处于客户程序内的，这样就不能与服务器有直接的链接，那么很自然，有可能再次激话服务器程序来处理已嵌入的 DLE 对象。

Past Link

我们可以使用 Pate Link 在客户程序中插入已链接的 DLE 对象，因此有了和服务器之间的直接链接。

示例：显示重要的 OLE 菜单选项

以下是一个典型的例子，它用于激活 OLE 的特定菜单项。

```
void FAR MenuActual( HMENU hMenu)                          // (De)activate menu items
{
    int MenuStatus = MF_GRAYED;                            // Menu item status (GRAYED

    MenuStatus = (hWndChild ? MF_ENABLED : MF_GRAYED);     // Cut,Copy and Clear
    EnableMenuItem(hMenu,IDM_CUT,MenuStatus);              // available only when
    EnableMenuItem(hMenu,IDM_COPY,MenuStatus);             // OLE client window
    EnableMenuItem(hMenu,IDM_CLEAR,MenuStatus);            // is available
    EnableMenuItem(hMenu,IDM_OBJECT,MenuStatus);

    if (! hClientDoc)                                      // Links··· item available only
    {
        EnableMenuItem(hMenu,IDM_LINKS,MF_GRAYED);
    }
    else
    {
        EnableMenuItem(hMenu,IDM_LINKS,otObject ==
            OT_LINK ? MF_ENABLED : MF_GRAYED);
    }                                                      // Check for OLE object in Clpbrd
    if ((OleQueryCreateFromClip(PROTCOL,olerender_draw,0) == OLE_OK
            || OleQueryCreateFormClip(SPROTOCOL,olerender_draw,0) == OLE_OK))
        EnableMenuItem(hMenu,IDM_PASTE,MF_GRAYED);
    else
        EnableMenuItem(hMenu,IDM_PASTE,MF_GRAYED);

    /* * Check for linked OLE object in Clipboard * */
    if (OleQueryLinkFromClip(PROTOCOL,olerender_draw,0) == OLE_OK)
        EnableMenuItem(hMenu,IDM_PASTELINK,MF_ENABLED);
    else
        EnableMenuItem(hMenu,IDM_PASTELINK,MF_GRAYED);
}
```

除了请求设置在剪贴板上的 OLE 对象以外，还需要特定的 OLE API 函数查询已嵌入的

和已链接的对象。

OleQueryCreateFromClip 函数检查剪贴板内是否有一个对象可被嵌入，如果有的话，则 OLE 返回 OLE_OK。专用于链接的函数是 OleQueryLinkFromClip。

OleQueryCreateFromClip 和 OleQueryLinkFromClip

```
OLESTATUS OleQueryCreateFromClip(lpszprotocol,renderopt,cfFormat);
OLESTATUS OleQueryLinkFromClip(lpszProtocol,renderopt,cfFormat);
```

参数	数据类型	描述
lpszProtocol	LPSTR	指向协议名称的指针
renderopt	OLEOP_RENDER	延迟选项
ofFormat	OLECLIPFORMAT	剪贴板数据格式

裁剪

因为可以产生新的 OLE 对象，所以也应提供删除对象的选项。Cut 菜单项只有在一个对象实际存在时才是可见的。

链接的特征

对于一个已嵌入的对象，产生的特殊属性没有多大意义。一旦一个嵌入的对象存在以后，最好让用户检测甚至人工地操作链接，对象数据的更新正在此刻发挥作用。通常显示一个对话框，用户可在框里中断链接，或从自动更新切换到人工更新。

对象激活(动词)

进一步地，可为服务器产生的每一个有效动词动态得增添一个菜单选项。同样地，也要为 OLE 文档内的每一个对象动态地加进一个菜单项。

22.3.2 生成 OLE 复合文档

OLE 的一个基本概念就是将有很大差别的不同种类的数据收集在一个文档中，此文档称作复合文档。这个文档可由任何对象组成，例如 Paintbrush 图像和文本。

正是由于复合文档的存在，才不必再分别处理文本和图像，因为文档都将其作为自身的对象处理。哪个数据可作为某种对象存储在复合文档中取决于 OLE 对象是已链接的还是已嵌入的。

链接只包括指向文件和服务器的指针，这样就免去了对原始数据的重新存储。嵌入的对象必须存储在客户程序。显然，客户不必知道数据结构(这是服务器的职责)。复合文档所需的空间由于存储原始数据而自然地增加。

生成一个客户文档

首先，文档必须与 OLE 程序库一起登录，这可使用 OleRegisterClientDoc 函数(参看下面内容)实现。第一个参数为类名，已在登录接收数据的窗口时提供。

通常，第一个参数是子窗口的类名。文档通过第二个参数(lpszDoc 参数)接受另一个名称。第三个参数为 0。lplhDoc 参数将某一句柄赋予新的 OLE 文档。

OLESTATUS OleRegisterClientDoc (lpszClass,lpszDoc,reserved,lplhDoc)

参数	数据类型	描述
lpszClass	LPSTR	指向类名的指针
lpszDoc	LPSTR	指向文档名的指针
reserved	LONG	保留
lplhDoc	LHCLIENTDOCFAR *	指向确定文档句柄的变量的指针

例子

```
LHCLIENTDOC LHCLIENTDOC = null;
OleRegisterClientDoc ("OleCl","OLECL_PBrush_0",OL,&lhCLientDOC);
```

结果便是登录 OLE 文档 OLECL_PBrush_0。已登录的窗口的名称是类名的两倍。本例中类名是 OleCl。第三个参数为内部数据保留并且隐含为零。作为登录结果，函数 OleRegisterClientDoc 返回 OLE 文档包柄(放到 lhClientDoc 变量中)，此句柄将被后面的 OLE 函数所调用。

22.3.3 登录剪贴板格式

数据交换在许多情况下是通过剪贴板进行的。实际上，不仅是数据，还有所有的对象也被交换。虽然剪贴板已能识别多种数据类型，但是 OLE 还需要附加的数据格式，这些格式通过登录记录下来。以数据格式代表数据是源于对象交换的需要。

剪贴板在对象传输过程中通常要接收四种类型的数据，这些类型称作 Native Data、OwnerLink Data、Presentation Format 和 ObjectLink Data。Native Data 仅仅用于 OLE 服务器。OLE 客户程序不必处理这些数据，但是在嵌入情况下必须存储这些数据。

Windows 3.1 Clipboard Viewer 使用户看到剪贴板内实际上存放的是哪些格式，因为每一格式有一个独立的菜单选项。我们建议用户生成并且拷贝一个 Paintbrush 图像，再用 Clipboard Viewer 检查它的格式。

显示格式用于显示，通过显示格式，OLE 程序库能够显示数据。OwnerLink Data 区分数据的所有者和服务器所服务的客户。产生 ObjectLind Data 需要对象的源和对对象的精确描述。

登录剪贴板格式对于以后访问某一对象的数据是不可缺少的。为执行登录，可使用 RegisterClipboardFormat 函数。实际上它并不是一个专用的 OLE 函数，只是一个典型的剪贴函数。格式名称已经赋值为 ObjectLink，Native 和 OwnerLink。

下面是一个嵌入过程的示例。

剪贴板格式登录示例

```
OLECLIPFORMAT vcfLink = 0;                    /* ObjectLink format */
OLECLIPFORMAT vcfNative = 0;                  /* Native format */
```

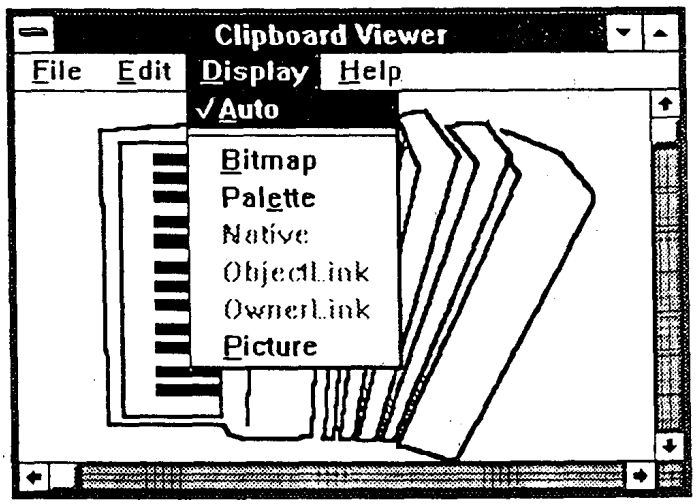


图 22.5

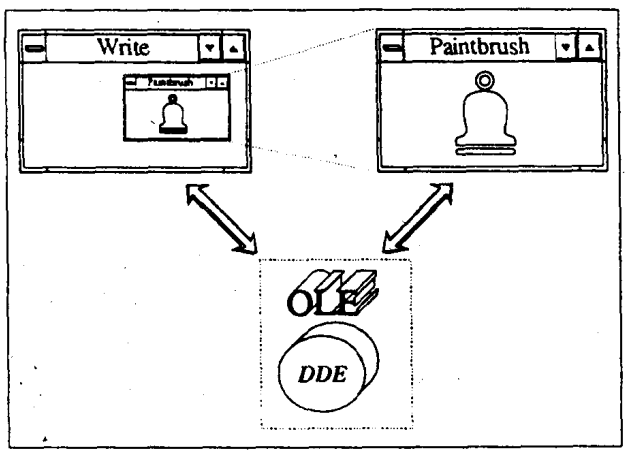


图 22.6 OLE 的使用情况

```
OLECLIPFORMAT vcfOwnerLink = 0;                          /* OwnerLink format */

                                                  // Registration fo the Clipboard format for OLE

vcfLink = RegisterClipboardFormat("ObjectLink");
vcfNative = RegisterClipboardFormat("Native");
vcfOwnerLink = RegisterClipboardFormat("OwnerLink");
```

本例中登录的三种剪贴板格式用于获取后来传送给 OLE 函数的指针。在窗口类登录过程中，格式登录最好在应用程序中进行。在客户应用程序内，特定的格式通常用于查询当前剪贴板中的数据，这种格式允许访问已链接的对象的文件名。

OLE 剪贴板格式细节

如果用户对恰到好处的格式结构感兴趣的话，我们在此给出一些补充的信息。Native Data 的结构只由服务器决定。更有意思的是，OwnerLink 和 ObjectLink 这两种格式包含链接数据。

OwnerLink 和 ObjectLink 都有同样的结构：前者处理嵌入对象，后者处理链接对象。格式描述由三个字符串组成的全程内存块，它们一个紧挨着一个放置，由零分开。

位置	意义
字符串 1	对象类名
字符串 2	文档或路径名
字符串 3	服务器的独立对象名

一旦知道了 OwnerLink 和 ObjectLink 的结构之后，用户就能够决定剪贴板的类名、文档名和对象名。下面的例子访问剪贴板上的 OwnerLink 数据并返回链接对象的名称。

```
/ * Determining the name of the linked OLE object * /
OleGetData( lpObject,vcfLink,&hData);
lpdata = GlobalLock(hdata);
while ( * lpdata++);                                    / * position after the server name * /
lstrcpy( cLinkName,lpdata);
GlobalUnlock(hdata);
```

通过 OleGetData 函数读出数据，提供数据块句柄。vcfLink 参数确定由 RegisterClipboardFormat("ObjectLink")预先登录的格式。

22.3.4 嵌入 OLE 对象

嵌入一个对象有三种方法。对象可能存在于剪贴中，也可能存在于文件中，还可以使用一个服务器程序产生。让我们检查一下如何用 OLE 将一幅图像粘贴到一个 Microsoft Windows Write 文档中，以生成一个 Paintbrush 图像，并且在 Write 内嵌入它。

到此为止，我们还未涉及 OLE。在用户粘贴到 Write 的图像上双击鼠标按钮，运行 Paintbrush 的下一个实例，这一实例包含一个对 Write 的 OLE 链接。注意每一个运行着的 Paintbrush 实例的标题条。

第一个实例包含源图像文件的名称，第二个实例的标题条包含名称"Paintbrush Picture in(name_of_Write_file)"。

每一个实例的 File 菜单也各不相同。第一个实例有一个 Save 项，这和任何普通的 Paintbrush 实例相同。但是，由 OLE 启动的 Paintbrush 应用程序有 Update 菜单项而没有 Save 项。

使用剪贴板嵌入对象

OleCreateFromClip 函数允许嵌入剪贴板对象。让我们检验一下 OleCreateFromClip 如何生成一个来自剪贴板的嵌入对象。从下列定义中我们发现有必要确定一些参数的地址。

```
OLESTATUS  OleCreateFromClip ( lpszProtocol, lpClient, lhcChientDoc, lpszObjname,
                                lplpObject,renderopt,cfFormat)
```

参数	数据类型	描述
lpsz	LPSTR	指向协议名的指针
lpClient	LPOLECLIENT	指向 OLECLIENT 结构的指针
lhClientDoc	LHCCLIENTDOC	OLE 文档句柄
lpszObjectname	LPSTR	指向 OLE 对象名的指针
lplpObject	LPOLEOBJECT. FAR *	指向接受新对象地址的变量的指针
renderopt	OLEOPT_RENDER	OLE 显示选项
cfFormat	OLECLIPFORMAT	剪贴板格式

下例带有参数,它更便于读者理解:

```
OpenClipboard(hWndParent);
OleCreateFromClip("StdFileEditing", lpClient, lhClientDoc, "Client_001", &lpObject,
                  olerender_draw, 0);
CloseClipboard();
```

1. lpszProtocol

第一个参数 lpszProtocol 通常指向字符串"StdFileEditing",它带有 OLE 配置其通信的协议。它产生一个对象,用于后面的处理过程。另外,lpszProtocol 还可用来指向"static"(静态)字符串,但是以后不再处理对象,因而没有设置其内容。

2. lpClient

lpClient 产生指向 OLECLIENT 结构的指针,这一结构通过 OLECLIENTVTBL 定义 OLE 回调函数。

3. lhClientDoc

lhClientDoc 参数生成 OLE 对象链。

4. lpszObjname

在给新 OLE 对象取名时,我们可以自由发挥自己的想象力,越抽象越好,因为名称必须不是二义性的,或至少在 OLE 文档中是可识别的。用户不会希望让不同的图像具有相同的名称。

5. lplpObject

OleCreateFromClip 需要一个变量来允许 OLE 程序定义指向新对象的指针。

6. renderopt,cfFormat

最后,用户可将 renderopt 和 cfFormat 看作为一个独立单元。一般情况下,renderopt 具有常量值 olerender_draw,这时 cfFormat 就无意义了。这种情况下,OLE 程序库用 OleDraw 在 WM_PAINT 消息期间接管绘图工作。但是为避免冒险,请参看下表。

格式	内容
olerender_none	OLE 程序库不返回数据并且无显示
olerender_draw	客户调用 Oledraw 函数，OLE 程序库显示数据
olerender_format	客户必须使用 OleGetData 自己取数据，cfFormat 必须被输入

由于选择返回的选项 olerender_format，客户程序库将不得不自己从剪贴板取数据。cfFormat 参数报告剪贴板中当前数据所使用的格式，这种格式可以是任何一种剪贴板格式，但是 OLE 程序库只能显示 CF_METAFILEPICT、CF_DIB 和 CF_BITMAP 格式，所在其他格式必须自己显示出来。

解决要点

让我们看另一个图示，然后讨论在此图中显示的不同结构。

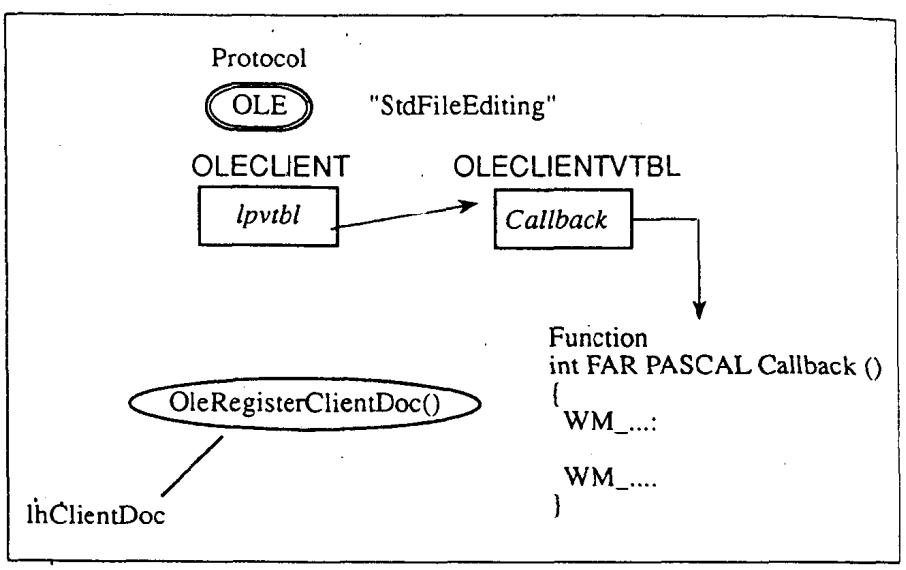


图 22.7　OLE 和 OleCreateFromClip

OLECLIENT 结构

OLECLIENT 结构包含指向一个函数地址表的指针，这一指针指向 OLECLIENTTLUB 结构，这个结构包含实际的 OLE 专用入口。lpvtbl 指针可以指向客户专用的数据。OLECLIENT 结构不需要除 OLECLINETVTBL 结构外的更多的入口，与此相关的数据集中由客户使用，并且禁止其他程序占用，例如，不能由服务器占用。OLECLIENT 结构布局如下：

```
typedef struct _OLECLIENT
{
    LPOECLIENTVTBL lpvtbl;

    /* Client-specific data can be here */
} OLECLIENT;
```

字段	数据类型	描述
lpvtbl	LPOLECLIENTVTBL	指向 OLE 为客户请求的函数表的指针

OLE 客户结构的分配示例

```
LPOLECLIENT lpclient;

                                                    // allocates the OLECLIENT sturcture
hobjClient = GlobaalAlloc(GMEM_FIXED | GMEM_ZEROPOINT,sizeof(OLECLIENT));
lpclient = (LPOLECLIENT)GlobalLock(jobjClient);
```

通常，OLECLIENT 结构作为一个全程内存块分配。GlobalAlloc 生成一个内存句柄 hobjClient，它由 GlobalLock 转换为一个 lpClient 指针。

OLECLIENTVTBL 结构

与 OLECLIENT 结构不同，对于 OLECLIENTVTBL 结构，每个客户应用程序只能占用一次。它仅由一个入口组成，这个入口在 OLE 中非常重要，因为在这里输入响应 OLE 客户的消息函数，没有这个入口将不能进行 OLE 通信。

OLECLIENTVTBL 结构布局如下：

```
typedef sturct _OLECLIENTVTBL
{
    int (FAR PASCAL * CallBack) (LPOLECLIENT pclient,
            OLE_NOTIFICATION notification,
            LPOLEOBJECT lpobject);
} OLECLIENTVTBL
```

域	数据类型	描述
CallBack	int(FAR PASCAL *)(..)	指向 OLE 客户回调函数的指针

OLE 客户回调函数的地址输入 OLECLIENTVTBL 结构的 CallBack 域。每当一条 OLE 消息送到应用程序中时，OLE 就访问这一函数。虽然函数地址必须作为一个过程实例地址来传输，但此处我们通过 MakeProcInstarce 函数取得这个地址。

示例：分配 OLECLIENT 和 OLECLIENTVTBL

OLECLIENT 结构必须至少填以对 OLECLIENTVTBL 结构的引用和对 OLE 回调函数的引用。

```
LPOLECLIENT lpclient;
OLECLIENTVTBL clientTbl;

                                                    // allocates the OLECLIENT structure
hobjClient = GlobalAlloc(GMEM_FIXED | GMEM_ZEROPOINT,sizeof)OLECLIENT));
lpclient = (LPOLECLIENT)GolbalLock(hobjClient);
                                        // allocates the OLECLIENT + OLECLIENTVTBL structure
/* Entry of the OLE Callback message function */
lpclient->lpvtbl = (LPOLECLIENTVTBL)&clientTbl;
lpclient->lpvtbl->CallBack = MakeProcInstance(CallBack,hInst);
```

附接私有数据给 OLECLIENT 结构

按标准，OLECLIENT 结构只由一个入口组成(对 OLECLIENTVTBL 结构的引用)。

因为每个新的 OLE 对象需要一个新的 OLECLIENT 结构，它是存储专用对象数据的好地方。例如，每当我们希望插入一个 Paintbrush 图像时，作为对象的图像必须要求一个 OLECLIENT 结构。

通常通过 GlobalAlloc 函数分配内存。由于从一定程度上看，块是动态分配的，所以我们可以立即向结构附加一些专用数据项。以下示例表明这种动作是如何进行的。

```
OLECLIENTVTBL clientTbl;
typedef struct _oleclientprivat
{
    LPOLECLIENTVTBL lpvtbl
    int icount;
    HWND hWndp;
} OLECLIENTPRIVAT;
typedef OLECLIENTPRIVAT FAR *LPOLECLIENTPRIVAT;
LPOLECLIENTPRIVAT lpclient;

                                        // allocates the OLECLIENT structure + private data
hobjClient = GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT,
            sizeof(OLECLIENTPRIVAT));
lpclient = (LPOLECLIENTPRIVAT)GlobalLock(hlobalLock(hobjClient);
                                        // Entry of the OLECLIENTVTBO structure
lpclient->lpvtbl = (LPOLECLIENTVTBL)&clientTbl;
                                        // filling in the private data
lpclient->iCount = 1;
```

iCount 和 hWndP 域作为专用对象附接到 OLECLIENT 结构上。OLE 只需要 lpvtbl 域，所以这两个保留的数据项对 OLE 来说没有意义，因此只用于客户应用程序。

嵌入一个新的对象

如果要嵌入的对象还不存在，那么就可以首先由客户生成它。通常客户有一个名为 Insert Object 的菜单项用于完成此目的。

我们能在 Write 中生成一个文本文档，并且在文档中嵌入一个新的 Paintbrush 图像而不使用剪贴板。为嵌入 Paintbrush 图像，移动插入点到所希望安放的位置，再选择 Edit/Insert Object…。Insert Object 对话框列出可用的对象，从 Object Type 中选择 Paintbrush 列表框，然后单击 OK 按钮。

用更新来嵌入

借助 OLE，客户启动一个 Paintbrush 服务器实例。要生成图像，可选择 File/Update。在有一个嵌入对象时，由 Update 项代替 Save。这个 Update 项在用户从 File Manager 或 Program Manager 中启动 Paintbrush 时不会出现。服务器负责修改菜单项。我们现在准备看一看一些技术上的细节。

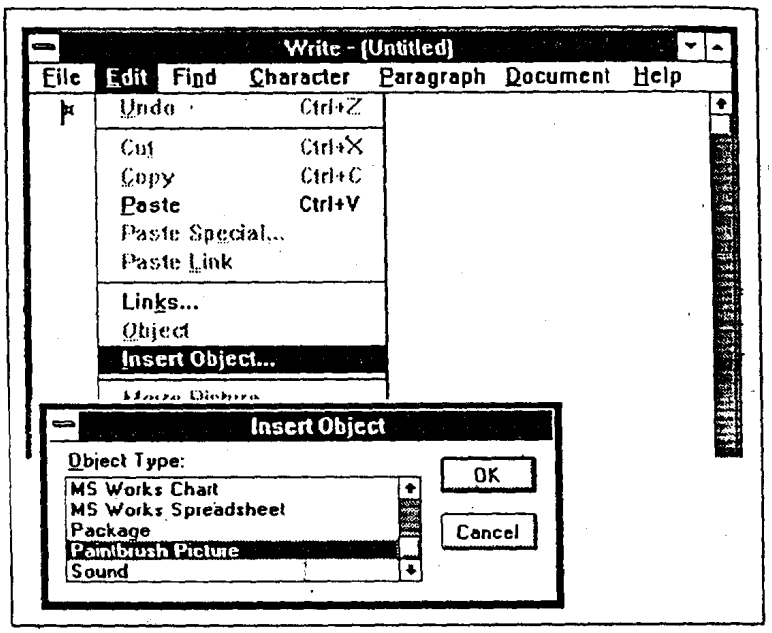


图 22.8　在 Write 中嵌入对象

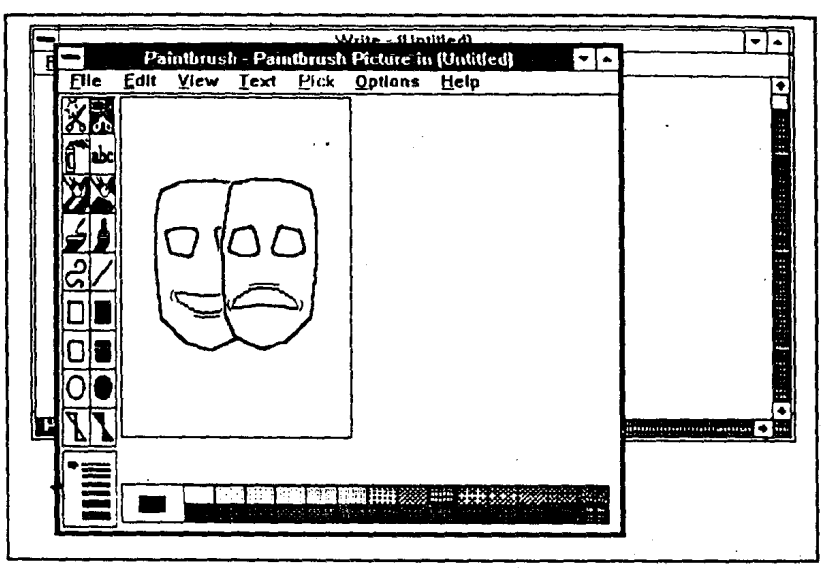


图 22.9　Write 和 OLE 启动 Paintbursh

OLE 技术

用户选择 Edit/Insert Object…时，出现一个对话框，在其中列出有效的服务器名。客户（此例中为 Write）在登录数据库中查找这个信息。客户找遍所有留在登录数据库中的服务器

名，并且将所选择的信息放在对话框内。当用户选择 Paintbrush Picture 并单击 OK 按钮时，将产生 OLE 与 Paintbrush 链接的信息。Paintbrush 运行，服务器(Paintbrush)上的 File 菜单提供 Update 选项而不是 Save 选项。选择 File/Update 向 Write 传送图片。

对象

图像不能由 Write 作为一个 Paintbrush 图片立即读出。Write 通过 OLE 从 Paintbrush 接收一个 OLE 对象，该对象由两个信息包组成，一个能由 Write 读出和显示，另一个包含只能由 Paintbrush 处理的数据。Write 窗口内可见的图像来自显示包，这一显示包由 Windows 位图或元文件格式的数据组成。更通用的显示格式是元文件。

摘要：在文本中嵌入一幅新图片

(1)用 Write 产生文本。
(2)在文档中放置插入点。
(3)选择 Edit/Insert Object…。
(4)选择 Paintbrush Picture 并按下 OK 按钮。
(5)自动启动 Paintbrush，生成图像。
(6)选择 File/Update。
(7)退出 Paintbrush，继续在 Write 下工作。

检查 OLE

用户也许会认为，OLE 数据交换与 Windows 3.1 下拷贝和粘贴数据并没有什么区别，我们将证明 OLE 的确是有用的。从 File Manager 或 Program Manager 中运行 Paintbrush，生成一幅图片。选择图片并且向剪贴板拷贝它。不存储图片文件而退出 Paintbrush。

图 22.10

图片仍然在剪贴板中。从 File Manager 或 Program Manager 中运行 Write。选择 Edit/Paste。由于还未保存 Write 文档，所以标题条内带有名称“Untitled”。选择用户刚粘贴的图片，并且在其上双击鼠标按钮，运行 Paintbrush。

现在缩小 Paintbrush，观察缩小了的图标。图标的标签将为“Paintbrush-Paintbrush Picturein [Untitled]”。

在 Write 窗口上按下鼠标按钮，确信已见到 Write 窗口和 Paintbrush 图标。选择 File/Save As…并且以 TEXT.WRI 存储文档。一旦文件被保存起来，Write 就借助 OLE 向 Paintbrush 传送文件名。再看 Paintbrush 图标，图标的标签将为“Paintbrush-Paintbrush Picture in [TEXT.WRI]”。

Paintbrush -
Paintbrush Picture in
TEST.WRI

图 22.11

Paintbrush -
Paintbrush Picture
in (Untitled)

图 22.12

摘要:在文本中嵌入一幅图片

(1)运行 Paintbrush 产生一幅图片。
(2)选择图片,再选择 Edit/Copy。
(3)保存图片并且退出 Paintbrush。
(4)运行 Write 产生文本文档。
(5)将插入点移到用户希望在当前剪贴板上放置 Paintbrush 图片的位置。
(6)选择 Edit/Paste 或 Edit/Paste Special…。

摘要:编辑一个嵌入的图片

(1)在 Write 内,用鼠标双击图片。
(2)Paintbrush 启动且装入图片。
(3)对图片作必要的修改。
(4)选择 File/Update。
(5)退出 Paintbrush。

OleCreate 函数

OleCreate 执行生成一个新的 OLE 对象所必需的大部分任务。已经知道服务器的类名以后,实现这些任务是很简单的(例如用于 Paintbrush 的 PBrush)。

```
OLESTATUS OleCreate(lpszProtocol,lpClient,lpszClassname,lhClientDoc,lpszObjname,
                    lplObject,renderopt,cfFormat);
```

除了 lpszClassname 中的类名以外,OleCreate 与 OleCreateFromClip 函数(参看 OleCreateFromClip 描述)的参数相同。

下例不适合于普通的使用,但是它显示出如何通过 OLE 用 Paintbrush 产生和嵌入一个

新的对象。

```
char szClassName[255];                                    /* OleCreat() */

lstrcpy( szClassName,"PBrush" );
OleCreate( "StdFileEditing",lpclientm (LPSTR)szClassName,lhClientDos,
        szarObjectName,&lpObject,olerender_draw,0);
```

客户应用程序决定来自登录数据的类名，类名允许客户应用程序和所有有效的 OLE 服务器交互。

22.3.5 链接 OLE 对象

OLE 也允许在应用程序间链接对象(例如在 Paintbrush 和 Write 之间)。在这种情况下，Paintbrush 函数作为服务器，Write 作为客户，结果是带有一链接图像的文本文档。

嵌入对象与链接对象之间的不同点在于被动数据的位置。如果插入了一个对象(即文本中的一幅图像)，完整的对象图片将在文档中与文本融为一体，因而所有数据将存储在一起。

在对象链接过程中，加入对象的被动数据处于不同的文件中。如果一个 Paintbrush 图片已链接到一个 Write 文本中，则两个文作为整体出现。在对象能被链接之前，它必须只存在于一个文件中。

一旦建立了一个链接，已链接的元素就能够自动更新。在采用这种技术之前，先让我们看一个例子。

链接图形和文本

让我们看看怎样才能在一个 Write 文本中链接一幅 Paintbrush 图片。首先用 Paintbrush 生成并且存储图片，其次选择图片并且使用 Edit/Copy 将其拷贝到剪贴板中。

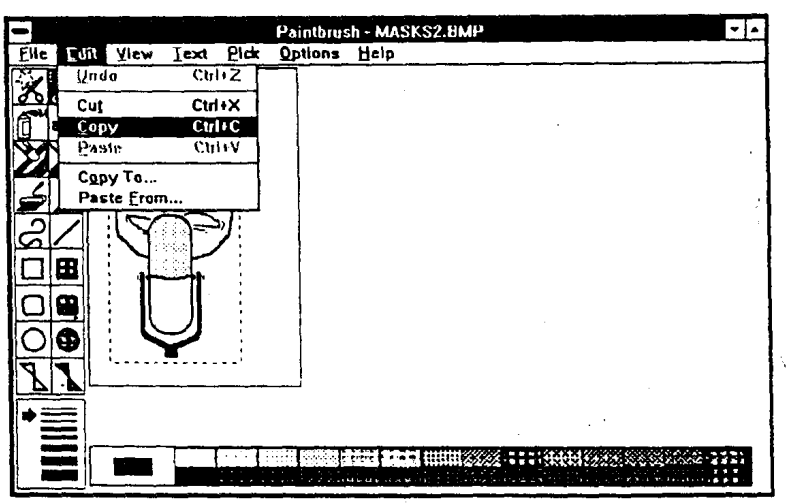


图 22.13 链接一幅已保存的 Paintbrush 图片

链接一幅已保存的 Paintbrush 图片

运行 Write，在用户想要安放图片的地方放置插入点，并且选择 Edit/Paste Link，这样可产生 Paintbrush 图片与 Write 文档间的 OLE 链接。

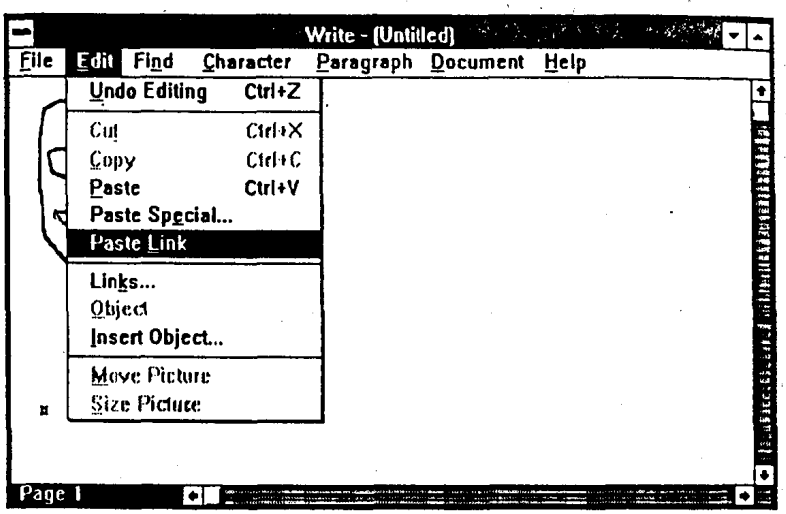


图 22.14　在 Write 内链接一幅图片

链接控制

链接便于自动更新信息。例如，如果用户改变了已链接到 Write 文档中的 Paintbrush 图片，就在 Write 文档中更新链接。用户可以选择 Write 中的 Edit/Links…来改变链接状态。对话框将列出现存的链接(包括 Paintbrush 图片在内)。

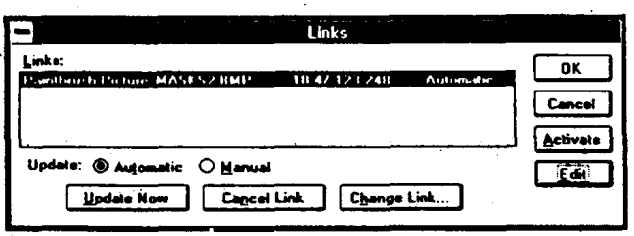


图 22.15　检验链接

摘要：链接一个对象到文档中

(1)在服务器应用程序中生成一个对象。
(2)存储对象。
(3)使用 Edit/Copy 向剪贴板拷贝服务器对象。
(4)切换到客户应用程序，并且选择 Edit/Paste Link。

图形的处理

编辑链接图片的过程与编辑嵌入图形的过程是一致的。在 Write 内的图片上双击鼠标按钮，以自动启动 Paintbrush。因为图形作为一个独立的对象设置在其自身的 Paintbrush 文件中，所以 Paintbrush 标题条就显示图片的文件名。在存储文件的过程中，似乎不存在 OLE 链接。选择 File/Save 菜单。由于与 Write 的 OLE 链接执行一种自动更新操作，所以 Write 文档也可自动更新。

使用剪贴板链接一个对象

OleCreateLinkFromClip 函数通过剪贴板执行对象链接过程。从以上的描述可以看到，OleCreateLinkFromClip 的许多部分可用于 OleCreateFromClip(参看 OleCreateLinkFromClip 描述)。

```
OLESTASTUS OleCreateLinkFromClip (lpszProtocol, lpClint, lhClintDoc, lpszObjnane,
                                  lplpObject, renderopt, ofFormat);
```

以下示例根据剪贴板中保存的合适的对象生成一个链接对象。

```
                                  // Create OLE linkage #define PROTOCOL ((LPSTR)"StdFileEditing")

OpenClipboard(hWndParent);
OleCreateLinkFromClip(PROTOCOL, lpclient, lhClientDoc,
        szarobjectName, &lpObject, olerender_draw, 0);
CloseClipboard();
```

链接操作

一旦存在 DLE 链接，就可以操纵这个链接了。如果用户希望避免自动更新，可将链接改变为人工更新。选择 Write 内的 Edit/Links…，显示一个对话框，从对话框中可以改变链接状态。列表框列出了所有链向当前文档的链接(此时为 Paintbrush 图片)。

为了设置人工更新的 Paintbrush 图片，要确信已选择了图片名(若不只存在一个链接)，并且按下了 Manual 选项按钮。

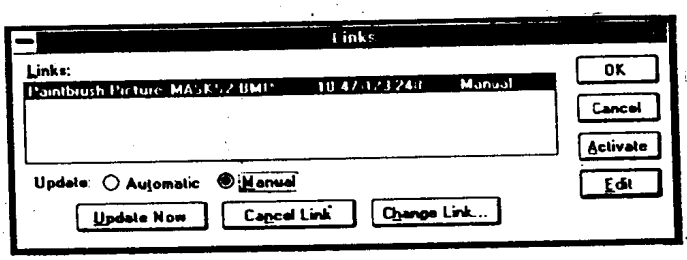


图 22.16　人工链接

人工链接更新

当切换为人工更新时，链接不会在用户每次编辑 Paintbrush 图片时自动更新。反之，用

户必须编辑 Paintbrush 图片，切换到 Write，选择 Edit/Links…，并且单击 Update Now 按钮，这样才能更新。

摘要：人工更新过程

(1)在 Write 内选择 Edit/Links…。
(2)在列表框中选择链接项。
(3)按击 Update Now 按钮。

中断链接

当用户不再需要一个链接时，可以选择 Write 内的 Edit/Links…中断链接，再对已选择的链接按下鼠标按钮 Cancel Link。Paintbrush 图片保留在 Write 中，但是却作为一个静态元素(不存在 DLE 链接)。

在用户按下 Cancel Link 按钮时，链接对象名从 Links 对话框中消失。当用户退出这个对话框并且在 Write 内双击图片按钮时，就会出现一个消息框，说明对象既不是嵌入的也不是链接的。

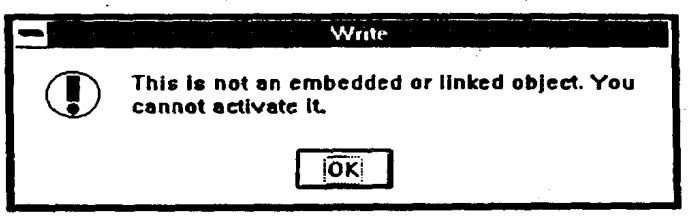


图 22.17　中断链接

在用户已确定要中断某个链接的情况下，只能中断一个链接：一旦链接被中断，就不能再生。

修复一个已中断的 OLE 链接

用户可能在拷贝文件时偶然地改变了某个图片的名称，或者有意更换了文件名。如果图片是通过 OLE 链向 Write 或另一个应用程序的话，那么这种名称的改变会导致一个链接被中断。

选择 Edit/Links…，显示 Links 对话框。按下 Change Link…按钮，出现 Chang Link 对话框。在对话框中，用户可以选择已换名的文件。按下 OK 按钮，用新选择的链接代替失效的链接。

摘要：修复链接过程

(1)在客户程序中选择对象。
(2)选择 Edit/Links…。
(3)选择失效的链接。
(4)按下 Chang Link…按钮。

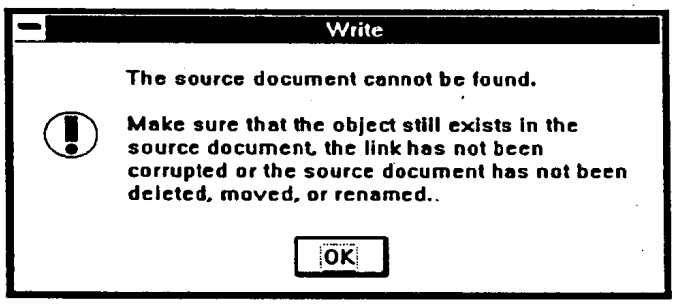


图 22.18　中断 OLE 链接

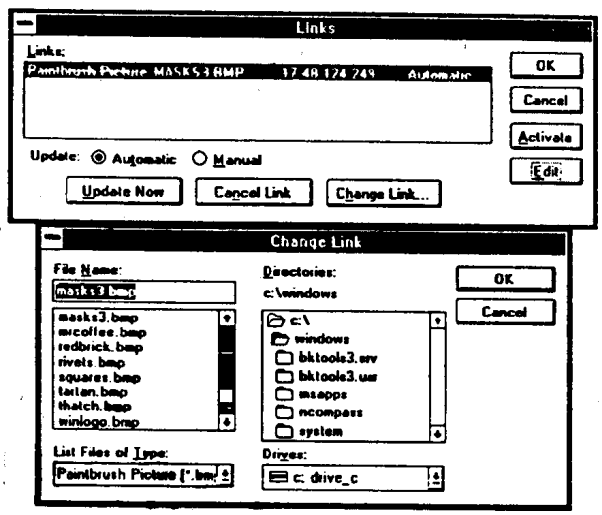


图 22.19　修复一个中断的链接

(5)从 FileName:列表框中选择新的文件名。

(6)按下 OK 按钮。

链接数据

从 Write 示例可以看出,链接可被浏览或通过一个对话框进行人工操作。我们还不准备尝试人工操作过程,但是获取链接数据应该是没有问题的。

WM_INITDIALOG 消息可确定链接类型。OLE 链接可以要求一个自动的或人工的更新。OleGetLinkUpdateOptions 函数为一个自动或人工链接返回 oleupdatealways 或 oleupdate_oncall,返回不同于这两个值的值说明一个链接被中断。

```
OLEOPT_UPDATE fActual;

/* Determine automatic or manual connection */

OleGetLinkUpdateOptions( lpObject,&fActual);
swith ( fActual)
```

```
{
    case oleupdate_always:
        lstrcpy( szType,"Automatic");
        break;
    case oleupdate_oncall:
        lstrcpy( szType,"Manual");
        break;
    default:
        lstrcpy( szTypem,"Broken");
        break;
}
```

22.3.6 绘制 OLE 对象

在客户应用程序内绘制一个嵌入对象或链接对象是一件简单的事情。WM_PAINT 消息接收一个来自某一子窗口的应答，而 OleDraw 则执行实际的任务。

```
OLESTATUS OleDraw (lpObject,hDC,lpBounds,lpWBounds,hFormatDC);
```

参数	数据类型	描述
lpObject	LPOLEOBJECT	指向所要绘制的对象的指针
hDC	HDC	设备环境句柄
lpBounds	LPRECT	显示对象所要绘制的帧
lpWBounds	LPRECT	显示元文件 hDC 帧
hFormatDC	hFormatDC	对象格式化设备环境句柄

在示例中，子窗口显示 OLE 对象。为了简单起见，对象应该与子窗口有同样的大小。GetClintRect 函数以窗口的大小填充 RECT 结构。

lpObject 指针识别将要显示的对象。除了子窗口的设备环境外，不需要更多的项。最后两个函数参数可置为零。

WM_PAINT 消息

```
case WM_PAINT:
    BeginPaint(hWnd,(LPPAINTINSTRUCT)&ps);

                                                    // Draw the child window
    GetClientRect(hWnd,&rc);
    OleDraw( lpObject,ps. hdc,&rc,NULL,NULL);
    EndPaint(hWnd,(LPAINTINSTRUCT)&ps);
    break;
```

一旦决定以源大小显示对象，就必须考虑映像模式。OleQueryBounds 函数决定对象的大小，这就导致了返回值的问题，因为 OleQueryBounds 在 RECT 结构中保存大小参数，而这种结构的参数指的是 MM_HIMETRIC 映像模式。正是由于这个原因，我们就必须重新计算大小参数以适合当前模式。例子中窗口大小是通过 SetWindowPos 来设置的，这个函数需要屏幕坐标，因此必须进行下列转换：

```
OleQueryBounds(lpObjectm &rc);
{
```

```
    /* Conversion of logical HTMETRIC coordinates in
       OleQueryBounds into screen coorinates */
    hDC = GetDC(NULL);
    mapm = SetMapMode(hDC,MM_HIMERRIC);
    LPtoDP(hDC,(LPOINT)&rc,2);
    SetMapMode(hDC,mapm);
    ReleaseDC(NULL,hDC);

    rc.right/2,rc.bottom/2,SWP_NOZORDER | SWP_NOMOVE | SWP_DRAWFRAME);
    ShowWindow( hwndChild,SW_SHOW);
}
```

22.3.7 OLE 回调函数

OLE 回调函数与 Window 函数类似，每当 OLE 客户程序库必须向一个客户发送一条消息时，就使用这个回调函数。OLECLIENTVTBL 通过 MakeProcInstance 接收回调函数的地址。

OLE 回调函数必须在模块定义文件中输出。客户通常要查询下列消息：

消息	描述
OLE_CLOSED	用户期望保留对象
OLE_SAVED	用户已终止服务器程序
OLE_CHANGED	对象已改变
OLE_RELEASE	释放
OLE_QUERY_PAINT	绘制对象

OLE_QUERY_PAINT

当 OleDraw 绘制一个 OLE 对象时，OLE 客户 DLL 周期性地发送 OLE_QUERY_PAINT 的更新信息。必须返回 TRUE 值，以确信实际上绘制了对象。返回值为 FALSE 时停止绘制过程。

OLE_SAVED

用户已存储了服务器对象。只有嵌入对象产生一条 OLE_SAVED 消息，才说明对象已被存储。链接对象不会产生 OLE_SAVED 消息。

OLE_CLOSED

用户关闭服务器程序，所以应该显示客户应用程序。常用 BringWindowToTop 函数将窗口显示出来。当然，OLE_CLOSE 消息仅在链接对象下产生。

OLE_CHANGE

OLE_CHANGE 消息帮助客户应用程序判断一个链接对象是否应该更新。OLE_CHANGE 在对象的处理过程中不断地更新。

22.3.8 客户应用程序示例

客户应用程序包含的 OLE 函数越多，应用程序就越大。这里仅局限于一个较小的 OLE

应用程序，此程序允许嵌入和链接一个 Paintbrush 图片。

应用程序包含可接受 Paintbrush 图片格式的某个 OLE 对象的子窗口。文件存储和装入将是不可见的。由于我们已将应用程序严格限于 Paintbrush，所以程序将请求来自登录数据库的信息。

Edit 菜单和 OLE Object 菜单便于与 OLE 程序库进行通信。用户选择 OLE Object/Paste Picture 可生成一个 OLE 对象。这个选项运行 Paintbrush，产生一幅可在 OLE 客户程序中嵌入的图片。

Edit 菜单提供了使用剪贴板的 Cut，Copy，Paste 选项，并可访问 OLE 服务器，以进行链接维护（此例中指 Paintbrush）。

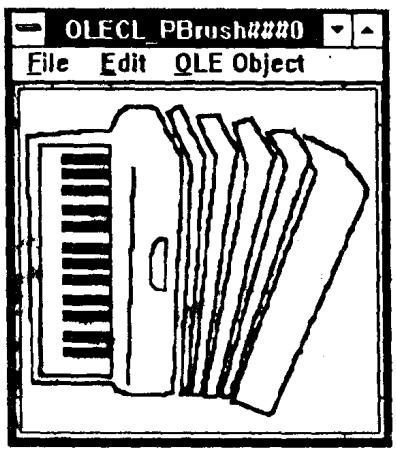


图 22.20

新函数	描述
OleActivate	激活对象
OleCopyToClipboard	拷贝对象
OleCreateFromClip	生成嵌入对象
OleCreateLinkFromClip	生成链接对象
OleDraw	绘制对象
OleDelete	删除对象
OleGetData	返回数据
OleGetLinkUpdateOptions	返回更新选项
OleQueryCreateFromClip	为协议查询剪贴板
OleQueryLinkFromClip	为链接对象请求剪贴板
OleQueryName	返回对象名
OleQueryRleaseError	在异步操作过程中询问错误值
OleQueryType	询问是否链接对象或者是否为静态对象
OleQueryReleaseStatus	询问是否释放对象操作过程
OlRenaneClientDoc	向客户程序库报告新文档名
OleRegisterClientDoc	登录客户文档

OleRevokeClientDoc	关闭文档
OlSetHostName	指定服务器标题条
OleUpdate	更新对象显示

新结构	描述
OLECLIENT	包含指向 OLECLIENTVTBC 的指针
OLECLENTVTBL	包含回调函数

新状态消息	描述
OLE_CLOSED	对象关闭
OLE_SAVED	对象存储
OBJECT_CHANGED	对象修改
OLE_RELEASE	对象释放
OLE_WAIT_FOR_RELEASE	等待释放
OLE_OK	结束异步操作

源代码:OLECL. C

```
/* * OLECL. C * * *
```

```
            MessageBox(NULL,szString,NULL,MB_ICONEXCLAMATION);
            return nError;
        }
    }

    /* * Specify appearance of application's main window * * * * * * * * * */

    hWndParent = CreateWindow(szAppName,szAppName,
        WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,0,
        GetSystemMetrics(SM_CXSCREEN)/3,
        GetSystemMetrics(SM_CYSCREEN)/2,
        NULL,NULL,hInstance,NULL);

    if(hWndParent == NULL)
    {
        LoadString(hInstance,IDS_ERR_WINDOW,szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWndParent,nCmdShow );
    UpdateWindow( hWndParent );
    if ( ! ClientOleInit())
        return FALSE;

    while (GetMessage(&msg,NULL,0,0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int)msg. wParam;
}

int nCwRegisterClasses(HANDLE hInstance)
{
    WNDCLASS wcClass;

    wcClass. hCursor = LoadCursor( NULL,IDC_ARROW );
    wcClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcClass. lpszMenuName = (LPSTR)"OleClMenu";
    wcClass. lpszClassName = szAppName;
    wcClass. hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcClass. hInstance = hInstance;
    wcClass. style = CS_VREDRAW | CS_HREDRAW;
    wcClass. lpfnWndProc = MainWndProc;
    wcClass. cbClsExtra = 0 ;
    wcClass. cbWndExtra = 0 ;

    if (! RegisterClass( &wcClass ) )
        return FALSE;

    wcClass. hIcon = NULL;
```

```
    wcClass.lpszMenuName = NULL;
    wcClass.lpszClassName = szChildName;
    wcClass.style = CS_DBLCLKS | CS_VREDRAW | CS_HREDRAW;
    wcClass.lpfnWndProc = ClientOleWndProc;

    if (! RegisterClass( &wcClass ) )
        return FALSE;

    return TRUE;
}

/* * MainWndProc: All messages are sent to this window * * * * * * * * * */

long FAR PASCAL MainWndProc(HWND hWnd,unsigned message,
            WORD wParam,LONG lParam )
{
    switch (message)
    {
        case WM_CREATE:
            break;
        case WM_INITMENU:
            MenuActual( (HMENU)wParam);
            break;
        case WM_COMMAND:
            switch(wParam)
            {
                case IDM_CUT:
                    ClientOleObjectBusyNow();
                    if (! ClientOleObjectCopyThis())
                        ErrorMessages(F_CLPBRD_EDITCUT);
                    ClientOleObjectBusyNow();
                    ClientOleObjectDeleteThis();
                    break;
                case IDM_COPY:
                    if (! ClientOleObjectCopyThis())
                        ErrorMessages(F_CLPBRD_EDCOPY);
                    break;
                case IDM_CLEAR:
                    ClientOleObjectBusyNow();
                    ClientOleObjectDeleteThis();
                    break;
                case IDM_PASTE:
                    if (hwndChild)
                        ClientOleObjectBusyNow();
                    ClientOleObjectDeleteThis();
                    if (! ClientOleObjectPasteThis(TRUE))
                    {
                        ErrorMessages(F_CLPBRD_EDPASTE);
                        break;
                    }
                    break;
```

```
            case IDM_PASTELINK:
                if (hwndChild)
                    ClientOleObjectBusyNow();
                ClientOleObjectDeleteThis();
                if (! ClientOleObjectPasteThis(FALSE))
                {
                    ErrorMessages(F_CLPBRD_EDPASTE);
                    break;
                }
                break;
            case IDM_LINKS:
                DialogBox(hInst,MAKEINTRESOURCE(DTPROP),
                        hWndParent,lpClientOleLinking);
                break;
            case IDM_INSERT:
                if (hwndChild)
                    ClientOleObjectBusyNow();
                ClientOleObjectDeleteThis();                    // Create embedded OLE object
                if (! ClientOleObjectEmbedded())
                {
                    ErrorMessages(F_OBJECT_CREATE);
                    return FALSE;
                }
                break;
            case IDM_EXIT:
                ClientOleObjectBusyNow();
                SendMessage(hWnd,WM_SYSCOMMAND,SC_CLOSE,0L);
                break;
            case IDM_OBJECT:                                    // Check for activity
                ClientOleActivity( OLEVERB_PRIMARY);
                break;
            default:
                break;
        }
        break;
    case WM_CLOSE:
        ClientOleObjectBusyNow();
        ClientOleObjectDeleteThis();                            // Remove OLE client document
        ClientOleDocumentRemoval();
        DestroyWindow( hWnd);
        break;
    case WM_DESTROY:
        ClientOleEndtheApp();
        PostQuitMessage(0);
        break;
    case WM_QUERYENDSESSION:
        ClientOleObjectBusyNow();
        break;
    default:
        return (DefWindowProc( hWnd,message,wParam,lParam ));
        break;
```

```
    }
    return(0L);
}

long FAR PASCAL ClientOleWndProc(HWND hwnd,unsigned msg,
            WORD wParam,LONG lParam)
{
    PAINTSTRUCT ps;
    RECT rc;

    switch (msg)
    {
        case WM_INVALID_LINK:                                    // Private message
            if ((ErrorSOle(OLE_ERROR_OPEN) == OLEATTEMPTS) &&
                    (wParam == OLE_UPDATE))
                PostMessage (hwnd,WM_COMMAND,
                    IDD_UPDATE,0L);
            break;
        case WM_PAINT:
            BeginPaint(hwnd,(LPPAINTSTRUCT)&ps);                 // Draw child window
            GetClientRect(hwnd,&rc);
            OleDraw( lpObject,ps. hdc,&rc,NULL,NULL);
            EndPaint(hwnd,(LPPAINTSTRUCT)&ps);
            break;
        case WM_COMMAND:
            switch (wParam)
            {
                case IDD_EDIT:
                    PostMessage(hWndParent,WM_COMMAND,
                            IDM_OBJECT,0L);
                    break;
                case IDD_UPDATE:                                 // Update linked object
                    if ( otObject == OT_LINK)
                    {
                        if (ErrorSOle(OleUpdate( lpObject)))
                        {
                            if(OleQueryReleaseError( lpObject) == OLE_ERROR_LAUNCH )
                                ErrorMessages(F_SERVER_START);
                        }
                    }
                    break;
                case IDD_DESTROY:
                    DestroyWindow(hwnd);
                    return 0L;
                case IDD_LINKDONE:                               // End link updates
                    PostMessage(hwndChild,WM_UPDATE,0,0L);
                default:
                    break;
            }
            break;
        case WM_LBUTTONDBLCLK:                                   // Edit OLE object
```

```
            PostMessage(hWndParent,WM_COMMAND,IDM_OBJECT,0L);
            break;
        default:
            return (DefWindowProc(hwnd,msg,wParam,lParam));
            break;
    }
}

/* * Assigns OLE document name and registers object as client OLE document * */

void FAR ClientOleTitle( LPSTR lpTitle)
{
    if (lhClientDoc)
    {
        if (ErrorSOle(OleRenameClientDoc( lhClientDoc,lpTitle)))
            ErrorMessages(W_OLE_COMCHANNEL);
    }
    else
    {
        if (ErrorSOle(OleRegisterClientDoc(szAppName,
                lpTitle,0L,&lhClientDoc)))
        {
            ErrorMessages(W_OLE_COMCHANNEL);
            lhClientDoc = NULL;
        }
    }
    SetWindowText(hWndParent,lpTitle);
}

/* * Displays message box containing error messages taken from OLECL. RC * */

void ErrorMessages(WORD iIndex)
{
    char sz[80];

    /* * If no OLE document exists,no error text appears * * * * * * * * */

    if (lhClientDoc == NULL)
        return;

    LoadString(hInst,iIndex,sz,80);
    MessageBox(hWndParent,sz,szAppName,MB_OK | MB_ICONEXCLAMATION);
}

/* * ClientOleAllWaitforIt: Waits until all asynchronous OLE operations are * */
/* * completed. When the counter equals NULL,the program continues * */

void ClientOleAllWaitforIt(void)
{
    MSG msgWait;
```

```
    if (cOleWaitforTrans)
    {
        while (cOleWaitforTrans)
        {
            if (GetMessage(&msgWait,NULL,NULL,NULL))
                DispatchMessage(&msgWait);
        }
    }
}

/** ClientOleDocumentRemoval: Removes OLE object and OLE client document **/

void ClientOleDocumentRemoval(void)
{
    if (hwndChild)                                          // Destroy child window
    {
        DestroyWindow(hwndChild);
        hwndChild = NULL;
    }

    ClientOleAllWaitforIt();                                // Wait until OLEs are released

    if (lhClientDoc)
    {
        if (ErrorSOle(OleRevokeClientDoc(lhClientDoc)))
            ErrorMessages(W_OLE_COMCHANNEL);
        lhClientDoc = NULL;
    }
}

/** ClientOleObjectBusyNow: Checks whether OLE object is busy and returns **/
/** FALSE if object is free,and TRUE if object is busy **/

BOOL ClientOleObjectBusyNow (void)
{
    BOOL fOleAllocated = FALSE;
    LONG objType;
    HANDLE hLink;
    LPSTR lpLink;
    char str[200];

    if ((OleQueryType (lpObject,&objType) == OLE_OK)
            && (objType != OT_STATIC)
            && (OleQueryReleaseStatus(lpObject) == OLE_BUSY))
    {
        fOleAllocated = TRUE;
        if (OleGetData(lpObject,objType == OT_LINK ?
                vcfLink : vcfOwnerLink,&hLink) == OLE_OK)
        {
            if (lpLink = (LPSTR) GlobalLock (hLink))
            {
```

```
                wsprintf ((LPSTR)str,"OLE server %s is busy. Please release.",lpLink);
                MessageBox(hWndParent,str,szAppName,MB_OK | MB_ICONEXCLAMATION);
                GlobalUnlock (hLink);
            }
        }
    }
    return( fOleAllocated);
}

/* * Clear the OLE client document * */

void FAR ClientOleObjectDeleteThis( void)
{
    if (hwndChild == 0)
        return;

    if (ErrorSOle(OleDelete( lpObject)))
        ErrorMessages(F_OBJECT_CLEAR);

    DestroyWindow(hwndChild);
    hwndChild = 0;
}

/* * Copies an OLE object to the Clipboard * */

BOOL FAR ClientOleObjectCopyThis( void)
{
    BOOL fRevert = FALSE;

    if ( ! lpObject)                                    // OLE object doesn't exist?
        return( FALSE);                                 // Then you cannot copy it

    if (! OpenClipboard(hWndParent))                    // Ensure Clipboard can be opened
        return( FALSE);

    EmptyClipboard();

    fRevert = ! ErrorSOle(OleCopyToClipboard(lpObject));

    CloseClipboard();
    return( fRevert);
}

/* * Pastes an OLE object from Clipboard; fTyp specifies embedded or linked * */

HWND FAR ClientOleObjectPasteThis(BOOL fTyp)
{
    LONG objType;
    WORD wBuffSize = 50;

    if (! OpenClipboard(hWndParent))                    // Clipboard could not be opened
```

```
        return( NULL);

    if (fTyp)                                                    // Paste OLE object as embedded
    {
        if (ErrorSOle(OleCreateFromClip(PROTOCOL,lpclient,lhClientDoc,
                szObjectName,&lpObject,olerender_draw,0)))
        {
            if (ErrorSOle(OleCreateFromClip(SPROTOCOL,lpclient,lhClientDoc,
                    szObjectName,&lpObject,olerender_draw,0)))
                lpObject = NULL;
            else
                otObject = OT_STATIC;
        }
        else
        {                                                        // Determine OLE object type
            OleQueryType(lpObject,&objType);
            switch (objType)
            {
                case OT_EMBEDDED:
                    otObject = OT_EMBEDDED;
                    break;
                case OT_LINK:
                    otObject = OT_LINK;
                    break;
                default:
                    otObject = OT_STATIC;
                    break;
            }
        }
    }
    else                                                         // Create OLE link
    {
        if (ErrorSOle(OleCreateLinkFromClip(PROTOCOL,lpclient,lhClientDoc,
                szObjectName,&lpObject,olerender_draw,0)))
            lpObject = NULL;
        else
            otObject = OT_LINK;
    }
    CloseClipboard();

    if (! lpObject)
        return( NULL);

    return( ClientOleWindowCreation( TRUE));
}

/* * ErrorSOle: Central error routine for displaying OLE errors * */
BOOL FAR ErrorSOle(OLESTATUS olestat)
{
    switch (olestat)
    {
```

```
        case OLE_WAIT_FOR_RELEASE:
            cOleWaitforTrans++;                                    // Increment counter
        case OLE_OK:
            return( FALSE);
        case OLE_ERROR_STATIC:
            ErrorMessages(W_OBJECT_STATIC);
            break;
        case OLE_ERROR_REQUEST_PICT:
        case OLE_ERROR_ADVISE_RENAME:
        case OLE_ERROR_DOVERB:
        case OLE_ERROR_SHOW:
        case OLE_ERROR_OPEN:
        case OLE_ERROR_NETWORK:
        case OLE_ERROR_ADVISE_PICT:
        case OLE_ERROR_COMM:
            ErrorMessages( F_BROKENLINK );
            break;
        case OLE_BUSY:
            ErrorMessages( F_SERVER_ISBUSY);
        default:
            break;
    }
    return( TRUE);
}

/** OLE client callback function **/
int FAR PASCAL CallBack(LPOLECLIENT lpclient,
    OLE_NOTIFICATION flags,LPOLEOBJECT lpObject)
{
    RECT rc;
    WORD method;
    HDC hDC;
    WORD mapm;

    switch (flags)
    {
        case OLE_CLOSED:
        case OLE_SAVED:
        case OLE_CHANGED:
            if ( hwndChild)
            {
                /** Check whether an OleCreate() exists,or if size is updated **/
                if ((! fVisible) || (fTrackSize))
                {
                    if (! ErrorSOle(OleQueryBounds(lpObject,&rc)))
                    {
                        hDC = GetDC(NULL);                         // Calculate logical coordinates
                        mapm = SetMapMode(hDC,MM_HIMETRIC);
                                                                   // from OleQueryBounds screen
                        LPtoDP(hDC,(LPPOINT)&rc,2);                // coordinates as HIMETRIC
                        SetMapMode(hDC,mapm);
```

```
                    ReleaseDC(NULL,hDC);

                    SetWindowPos(hwndChild,NULL,0,0,rc. right/2,rc. bottom/2,
                        SWP_NOZORDER | SWP_NOMOVE | SWP_DRAWFRAME);

                    fVisible = TRUE;
                    ShowWindow( hwndChild,SW_SHOW);
                }
                else
                {
                    if( flags == OLE_CLOSED )
                    {
                        PostMessage(hwndChild,WM_COMMAND,      // Contains no object data
                            IDD_DESTROY,0L);
                        break;
                    }
                }
            }
            InvalidateRect(hwndChild,NULL,TRUE);
        }
        break;
    case OLE_RELEASE:
        if (cOleWaitforTrans)
        {
            --cOleWaitforTrans; se OLE_RELEASE:
            {
            if (cOleWaitforTrans)
            {
                --cOleWaitforTrans;                                    // Decrement counter
            }
            else
                ErrorMessages(F_RELEASER);
            switch ( OleQueryReleaseError(lpObject))
            {
                case OLE_OK:
                    switch (OleQueryReleaseMethod(lpObject))
                    {
                        case OLE_SETUPDATEOPTIONS:
                            PostMessage(hwndChild,WM_COMMAND,
                                IDD_LINKDONE,0L);
                            break;
                        default:
                            break;
                    }
                    break;
                case OLE_ERROR_STATIC:
                    ErrorMessages(W_OBJECT_STATIC);
                    break;
                case OLE_ERROR_REQUEST_PICT:
                case OLE_ERROR_ADVISE_RENAME:
                case OLE_ERROR_DOVERB:
```

```
                    case OLE_ERROR_SHOW:
                    case OLE_ERROR_OPEN:
                    case OLE_ERROR_NETWORK:
                    case OLE_ERROR_ADVISE_PICT:
                    case OLE_ERROR_COMM:
                        if ((method = OleQueryReleaseMethod
                                (lpObject)) ! = OLE_UPDATE)
                            method = NULL;
                        PostMessage(hwndChild, WM_INVALID_LINK, method, 0L);
                        break;
                    default:
                        switch (OleQueryReleaseMethod(lpObject))
                        {
                            case OLE_DELETE:
                                ErrorMessages(F_OBJECT_CLEAR);
                                break;
                            case OLE_LNKPASTE:
                                ErrorMessages(F_CLPBRD_EDPASTE);
                                break;
                            case OLE_ACTIVATE:
                                ErrorMessages(F_SERVER_START);
                                break;
                            case OLE_UPDATE:
                                ErrorMessages(F_UPDATEDATA);
                                break;
                            case OLE_CREATE:
                                ErrorMessages(F_OBJECT_CREATE);
                                break;
                            default:
                                break;
                        }
                        break;
                }
            }
            break;
        case OLE_QUERY_RETRY:                                   // Retry or
        case OLE_ QUERY_PAINT:                                  // Repaint
            return( TRUE);
        default:
            break;
    }
    return( FALSE);
}

HWND FAR ClientOleObjectEmbedded(void)                          // Create an embedded object
{
    char szClassName[255];                                      // Class name for OleCreate()
    lstrcpy( szClassName,"PBrush" );
    if (ErrorSOle(OleCreate(PROTOCOL, lpclient,
            (LPSTR)szClassName, lhClientDoc,
            szObjectName, &lpObject, olerender_draw, 0)))
```

```
        return( NULL);

    otObject = OT_EMBEDDED;                                  // Object type is embedded
    return( ClientOleWindowCreation( FALSE));
}

/* * Initializes OLE client structures,registering Clipboard format first * */

BOOL FAR ClientOleInit(void)
{
    vcfLink =RegisterClipboardFormat("ObjectLink");          // Clipboard format
    vcfNative =RegisterClipboardFormat("Native");            // regisration for
    vcfOwnerLink=RegisterClipboardFormat("OwnerLink");       // OLE

    hobjClient = GlobalAlloc(GMEM_FIXED |                    // Allocates OLECLIENT structure
            GMEM_ZEROINIT,sizeof(OLECLIENT));
    lpclient = (LPOLECLIENT)GlobalLock(hobjClient);
    if ( ! lpclient || ! hobjClient)
    {
        GlobalUnlock(hobjClient);
        GlobalFree(hobjClient);
        return( FALSE);
    }
    lpclient->lpvtbl = (LPOLECLIENTVTBL)&clientTbl;          // Allocate OLECLIENT struct
    (FARPROC)lpclient->lpvtbl->CallBack =                    // Start OLE callback
        MakeProcInstance(CallBack,hInst);                    // function

                                                             // Start dialog function
    lpClientOleLinking = MakeProcInstance( ClientOleLinking,hInst);

    ClientOleTitle(szObjectName);                            // Allocate title bar,register
                                                             // as client OLE document
    return( TRUE);
}

void FAR MenuActual( HMENU hMenu)                            // (De)activate menu items
{
    int MenuStatus = MF_GRAYED;                              // Menu item status (GRAYED)

    MenuStatus = (hwndChild ? MF_ENABLED : MF_GRAYED);       // Cut,Copy and Clear
    EnableMenuItem(hMenu,IDM_CUT,MenuStatus);                // available only when
    EnableMenuItem(hMenu,IDM_COPY,MenuStatus);               // OLE client window
    EnableMenuItem(hMenu,IDM_CLEAR,MenuStatus);              // is available
    EnableMenuItem(hMenu,IDM_OBJECT,MenuStatus);

    if (! lhClientDoc)                                       // Links··· item available only
    {                                                        // if OLE client doc registered
        EnableMenuItem(hMenu,IDM_LINKS,MF_GRAYED);
    }
    else
    {
```

```
        EnableMenuItem(hMenu,IDM_LINKS,otObject ==
                OT_LINK ? MF_ENABLED : MF_GRAYED);
    }                                                    // Check for OLE object in Clpbrd
    if ((OleQueryCreateFromClip(PROTOCOL,olerender_draw,0) == OLE_OK
            || OleQueryCreateFromClip(SPROTOCOL,olerender_draw,0) == OLE_OK))
        EnableMenuItem(hMenu,IDM_PASTE,MF_ENABLED);
    else
        EnableMenuItem(hMenu,IDM_PASTE,MF_GRAYED);

    /* * Check for linked OLE object in Clipboard * */

    if (OleQueryLinkFromClip(PROTOCOL,olerender_draw,0) == OLE_OK)
        EnableMenuItem(hMenu,IDM_PASTELINK,MF_ENABLED);
    else
        EnableMenuItem(hMenu,IDM_PASTELINK,MF_GRAYED);
}

void FAR ClientOleActivity(int iVerb)                    // Executes activity (Verb)
{
    RECT rc;
    if( hwndChild == NULL )
        return;
    if ( otObject == OT_STATIC)                          // If the object is static,
        return;                                          // no activity exists
    GetClientRect(hwndChild,(LPRECT)&rc);
    if (ErrorSOle(OleActivate( lpObject,iVerb,TRUE,      // Execute activity words
            TRUE,hwndChild,&rc)))
    {
        if(OleQueryReleaseError( lpObject) == OLE_ERROR_LAUNCH )
            ErrorMessages( F_SERVER_START );
    }
}

/* * Creates the OLE client window for OLE document display * */

HWND ClientOleWindowCreation( BOOL fShow)
{
    HANDLE hdata;
    LPSTR lpdata;
    RECT rc;
    int cb = 14;
    char sz[14];

    GetClientRect( hWndParent,&rc);

    if ( hwndChild == 0)                                 // Create child window
    {
        hwndChild = CreateWindow( szChildName,"",
            WS_CHILD | WS_CLIPSIBLINGS |
            WS_THICKFRAME | WS_BORDER,
            rc.left,rc.top,
```

```
            rc.right - rc.left,
            rc.bottom - rc.top,
            hWndParent,NULL,hInst,NULL);
    }
    ClientOleWaitforIt(lpObject);

    fVisible = fShow;
    fTrackSize = TRUE;

    if( otObject == OT_EMBEDDED )                    // Handle as embedded OLE object
    {
        OleQueryName(lpObject,(LPSTR)sz,&cb );       // Get embedded OLE object name
        ErrorSOle(OleSetHostNames(lpObject,
            (LPSTR)szAppName,(LPSTR)sz ));
        ClientOleTitle( szObjectName);
    }

    if (otObject == OT_LINK)                         // Handle as linked OLE object
    {
        if ((hdata = ClientOleObjectGetData( vcfLink))   // Get linked object name
                && (lpdata = GlobalLock(hdata)))
        {
            while ( *lpdata++);                      // Position after server name
            lstrcpy( cLinkName,lpdata);
            GlobalUnlock(hdata);
        }
        else
            lstrcpy( cLinkName,"");
        ClientOleTitle( cLinkName);
    }

    if( fShow )                                      // Display window if fShow=TRUE
        ShowWindow(hwndChild,SW_SHOW);
    return( hwndChild);
}

/** Dialog box routine for linking,updating and ending link **/

BOOL FAR PASCAL ClientOleLinking(HWND hDlg,unsigned msg,
        WORD wParam,LONG lParam)
{
    HWND hwndList;                                   // List box handle
    HANDLE hData = NULL;
    LPSTR lpstrData = NULL;
    char szType[30];
    char szFull[255];

    switch (msg)
    {                                                // Determine list box handle for
        case WM_INITDIALOG:                          // link display
            hwndList = GetDlgItem(hDlg,IDD_LINKNAME);
```

```
    if ( otObject == OT_LINK)                          // Should only be linked object
    {
        if (! (hData = ClientOleObjectGetData(vcfLink)))
        {
            ErrorMessages(F_CLPBRD_EDPASTE);
            return(TRUE);
        }
        switch (ClientOleObjectGetUpdateOpt())         // Determine auto or manual link
        {
        case oleupdate_always:
            lstrcpy( szType,"Automatic");
            break;
        case oleupdate_oncall:
            lstrcpy( szType,"Manual");
            break;
        default:
            lstrcpy( szType,"Broken");
        }
        lpstrData = GlobalLock(hData);

        lstrcpy(szFull,"PaintBrush Picture");          // Server class name
        lstrcat(szFull," ");

        while ( *lpstrData++);                         // Get document & element names
        lstrcpy( cLinkName,lpstrData);

        lstrcat(szFull,lpstrData);                     // Add file name
        lstrcat(szFull," ");

        while ( *lpstrData++);                         // Add element name
        lstrcat(szFull,lpstrData);
        lstrcat(szFull," ");

        lstrcat(szFull,szType);                        // Start of link type

        SendMessage(hwndList,LB_ADDSTRING,0,           // Send list box message
                (LONG)(LPSTR)szFull);

        GlobalUnlock(hData);
    return( TRUE);
case WM_COMMAND:
    hwndList = GetDlgItem(hDlg,IDD_LINKNAME);
    switch (wParam)
    {
        case IDCANCEL:
            EndDialog(hDlg,FALSE);
            return TRUE;
        case IDD_EDIT:
            SendMessage(hwndChild,WM_COMMAND,wParam,0L);
            EndDialog(hDlg,TRUE);
```

```
                return TRUE;
            case IDD_UPDATE:
                SendMessage(hwndChild,WM_COMMAND,wParam,0L);
                return TRUE;
            default:
                break;
        }
    default:
        return( FALSE);
    }
}

/** Returns OLE link information from Clipformat parameter,returning **/
/** handle information or NULL (error state) **/

HANDLE FAR ClientOleObjectGetData( OLECLIPFORMAT ClipFormat)
{
    BOOL fSuccess = FALSE;
    HANDLE hData;                                                    // Data handle

    fSuccess = ! ErrorSOle(OleGetData( lpObject,ClipFormat,&hData));

    return (fSuccess ? hData : NULL);
}

/** Reads update status of a linked object,then returns either **/
/** oleupdate_always,oleupdate_oncall or oleupdate_onsave **/

OLEOPT_UPDATE ClientOleObjectGetUpdateOpt(void)
{
    BOOL fSuccess = FALSE;
    OLEOPT_UPDATE fActual;                                  // Option to update object ap-)
    if ( otObject == OT_LINK)                                // plies to linked object only)
        fSuccess = ! ErrorSOle(OleGetLinkUpdateOptions(
            lpObject,&fActual));

    return (fSuccess ? fActual : oleupdate_onsave);


void FAR ClientOleEndtheApp(void)                          // Free OLECLIENT struct memory
{
    if (lpclient)
    {
        GlobalUnlock(hobjClient);
        GlobalFree(hobjClient);
        lpclient = NULL;
    }
}

void ClientOleWaitforIt(LPOLEOBJECT lpObject)// Wait loop while object is busy
{
```

```
    MSG msg;

    while (OleQueryReleaseStatus(lpObject) ==
            OLE_BUSY)
    {
        if (GetMessage(&msg,NULL,0,0))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

模块定义文件:OLECL. DEF

```
NAME OleClient
DESCRIPTION 'OLE Client Application'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS MainWndProc @1
    ClientOleWndProc @2
    CallBack @3
    ClientOleLinking @4
```

资源描述文件:OLECL. RC

```
#include "OleCl. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"

    F_OBJECT_CLEAR, "OLE object could not be cleared"
    F_CLPBRD_EDITCUT,"Cut Clipboard failed"
    F_CLPBRD_EDCOPY, "Copy Clipboard failed"
    F_CLPBRD_EDPASTE,"Paste Clipboard failed"
    F_OBJECT_CREATE, "No new object could be created"
    E_OBJECT_BUSY, "OLE object is busy"
    F_RELEASER, "DDE support invalid"
    F_SERVER_START, "Server application could not be started"
    F_UPDATEDATA, "OLE object could not be updated"
    F_SERVER_ISBUSY, "Server busy"
    F_BROKENLINK, "Broken link"
```

```
    W_OBJECT_STATIC, "Warning: Not the expected static object"
    W_OLE_COMCHANNEL,"Warning: Communication with OLE library failed"

END

OleClMenu MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_EXIT
    END

    POPUP "&Edit"
    BEGIN
        MENUITEM "Cu&t", IDM_CUT
        MENUITEM "&Copy", IDM_COPY
        MENUITEM "&Paste", IDM_PASTE
        MENUITEM "Paste &Link", IDM_PASTELINK
        MENUITEM SEPARATOR
        MENUITEM "Cl&ear", IDM_CLEAR
        MENUITEM SEPARATOR
        MENUITEM "&Links...", IDM_LINKS
        MENUITEM "Edit &Object", IDM_OBJECT
    END

    POPUP "&OLE Object"
    BEGIN
        MENUITEM "Paste Picture",IDM_INSERT
    END
END

/* Links dialog box */

DTPROP DIALOG 15,27,292,61
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Links"
BEGIN
    LISTBOX IDD_LINKNAME,2,14,240,17,WS_VSCROLL | WS_TABSTOP
    LTEXT "Update:",-1,3,35,44,8,NOT WS_GROUP
    PUSHBUTTON "&Now",IDD_UPDATE,19,43,48,14
    DEFPUSHBUTTON "&Edit",IDD_EDIT,254,43,32,14,WS_GROUP
    PUSHBUTTON "Cancel",IDCANCEL,254,3,32,14
    LTEXT "Links:",5,3,4,42,8
END
```

头文件:OLECL. H

```
#include <windows. h>
#include <string. h>
#include <ole. h>
```

```
char szString[128];                                  /* Variable for loading resource text */
char szAppName[20];                                  /* Window class name */
char szChildName[20];                                /* Child window class name */
char szObjectName[] = {"OLECL_PBrush### 0"};         /* OLE doc name */
HANDLE hInst;
HWND hWndParent;                                     /* Main window */
HWND hwndChild = 0;                                  /* Window containing the OLE object */

LPOLECLIENT lpclient;
LPOLEOBJECT lpObject;                                /* Pointer to OLE object */
WORD otObject;                                       /* Type of OLE object */
BOOL fVisible;                                       /* If true,display follows */
BOOL fTrackSize;                                     /* If true,autosizing occurs */
char cLinkName[255];                                 /* OLE link name of document */
OLECLIENTVTBL clientTbl;
OLECLIPFORMAT vcfLink = 0;                           /* Clipboard format "ObjectLink" */
OLECLIPFORMAT vcfNative = 0;                         /* Clipboard format "Native" */
OLECLIPFORMAT vcfOwnerLink = 0;                      /* Clipboard format "OwnerLink" */
FARPROC lpClientOleLinking;                          /* Dialog function for linking */
HANDLE hobjClient;                                   /* Handle for Client vtbl */
LHCLIENTDOC lhClientDoc = NULL;                      /* Handle for OLE client document */
unsigned int cOleWaitforTrans = 0;                   /* Counter for asynchronous
                                                        OLE transactions */

int nCwRegisterClasses(HANDLE);
long FAR PASCAL MainWndProc( HWND,unsigned,WORD,LONG);
long FAR PASCAL ClientOleWndProc( HWND,unsigned,WORD,LONG);
void ErrorMessages( WORD);
BOOL FAR ErrorSOle(OLESTATUS);
void FAR MenuActual( HMENU);

HANDLE FAR ClientOleObjectGetData( OLECLIPFORMAT);
BOOL FAR ClientOleObjectCopyThis(void);
HWND FAR ClientOleObjectPasteThis(BOOL);
void FAR ClientOleObjectDeleteThis(void);
BOOL ClientOleObjectBusyNow(void);
OLEOPT_UPDATE ClientOleObjectGetUpdateOpt(void);
HWND FAR ClientOleObjectEmbedded(void);

BOOL FAR PASCAL ClientOleLinking(HWND,unsigned,WORD,LONG);
HWND ClientOleWindowCreation( BOOL);
void FAR ClientOleEndtheApp(void);
void ClientOleDocumentRemoval(void);
BOOL FAR ClientOleInit(void);
void FAR ClientOleTitle(LPSTR);
void ClientOleAllWaitforIt(void);
void ClientOleWaitforIt(LPOLEOBJECT);
void FAR ClientOleActivity(int);

/* Menu */
#define IDM_EXIT 0x100
```

```
#define IDM_CUT 0x101
#define IDM_COPY 0x102
#define IDM_PASTE 0x103
#define IDM_PASTELINK 0x104
#define IDM_CLEAR 0x105
#define IDM_LINKS 0x106
#define IDM_OBJECT 0x107
#define IDM_INSERT 0x108

/* Dialog box */
#define DTPROP 1
#define IDD_LINKNAME 0x300
#define IDD_EDIT 0x301                                    // Button: Edit
#define IDD_UPDATE 0x302                                  // Button: Update
#define IDD_LINKDONE 0x303
#define IDD_LISTBOX 0x304                                 // Listbox
#define IDD_DESTROY 0x305

/* Error messages */
#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21
#define F_OBJECT_CLEAR 0x200
#define F_CLPBRD_EDITCUT 0x201
#define F_CLPBRD_EDCOPY 0x202
#define F_CLPBRD_EDPASTE 0x203
#define F_OBJECT_CREATE 0x204
#define E_OBJECT_BUSY 0x205
#define F_RELEASER 0x206
#define F_SERVER_START 0x207
#define F_UPDATEDATA 0x208
#define F_SERVER_ISBUSY 0x209
#define F_BROKENLINK 0x20A
#define W_OBJECT_STATIC 0x20B
#define W_OLE_COMCHANNEL 0x20C

/* Private messages */
#define WM_UPDATE (WM_USER + 0x100)
#define WM_INVALID_LINK (WM_USER + 0x101)

/* OLE protocol */
#define PROTOCOL ((LPSTR)"StdFileEditing")
#define SPROTOCOL ((LPSTR)"Static")

#define OLEATTEMPTS 3                                     /* Maximum number of OLE repeated attempts */
```

MAKE 文件:OLECL.MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc
```

```
RESFILES =
H = OLECL.H
LIBS_EXT = olecli.lib

all: OLECL.EXE

OLECL.RES: OLECL.RC $(RESFILES)

OLECL.OBJ: OLECL.C $(H)

OLECL.EXE: OLECL.OBJ OLECL.RES OLECL.DEF
    link /NOD/A:16 OLECL,OLECL,nul,libw slibcew $(LIBS_EXT),OLECL.def
    rc OLECL.res OLECL.exe
```

按如下方式运行 NMAKE，产生 OLECL 应用程序：

```
nmake olecl.mak
```

22.4 对象封装器

如果用户曾经打印过一封信，就很可能知道，在旧文档里也可能有一些有用的信息。用户很可能又启动了一个字处理器，打开那个文档，再将数据拷贝到正在处理的文件中。

用文档实现这些动作是容易的，但是查找多份文档可能要花费很多时间。一个方便的解决办法是从 File Manager 中选择文档，并且将文档拖放到当前文件中。

对象封装和拖放

让我们通过一个例子来实现这一工作。假定用户在自己的 D:\LETTERS 目录中存放着一个名叫 MTOLDDOC.WRI 的文档文件，这一文件包含需要用于 MYNEWDOC.WRI 新文档文件的重要信息。运行 Write，并且打入新文档名，在 MYNEWDOC.WRI 中保存它。运行 File Manager，定位到 D:\LETTERS 目录中。

将 MYOLDDOC.WRI 图标拖到 Write-MYNEWDOC.WRI 窗口，释放鼠标按钮，MYOLDDOC.WRI 文件的图标此时出现在 Write-MYNEWDOC.WRI 窗口中。这就是一个拖放操作的例子。

图 22.21

如果用户将图标移出 File Manager 窗口，则图标的形状改变成一个“NO…”符号。将图标移进 Write 窗口，它又改变成“sheet of papar”，正如出现在 File Manager 中的一样。

图 22.22

图标的形状显示了当前图标所在的应用程序是否支持 OLE。如果 File Manager 连续以“NO…”符号显示图标的话，则应用程序不支持 OLE。

图 22.23

MYOLDDOC. WRI 文档的图标此时出现在 Write-MYNENDOC. WRI 窗口中。

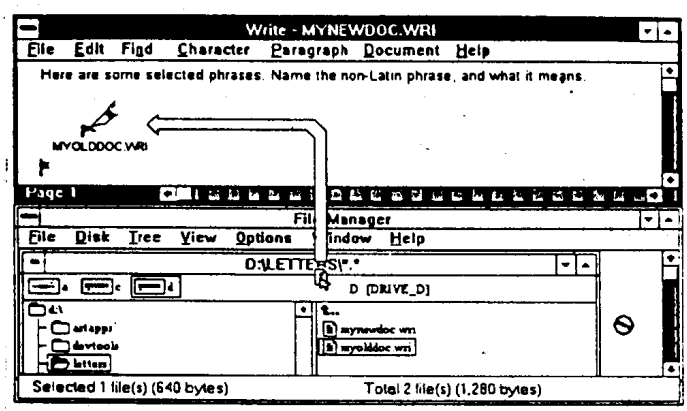


图 22.24　File Manager 支持 OLE

在 Write 内双击　YOLDDOC. WRI 图标，以访问这个文档。OLE 启动 Write 的另一个副本，以便显示老文件的内容。MYOLDDOC. WRI 图标是一个对象包。

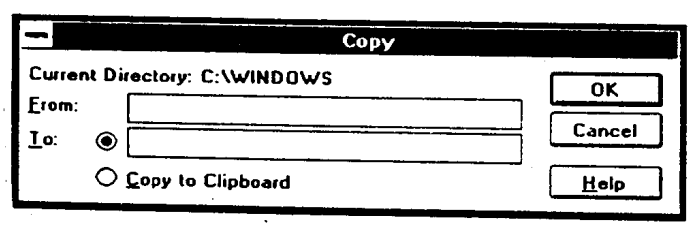


图 22.25　对象包和剪贴板

通过将文件拷贝到剪贴板上也可以创建对象包。从 File Manager 中选择文件，再选择 File/Copy。按下 Copy to Clipboard 按钮和 OK 按钮。

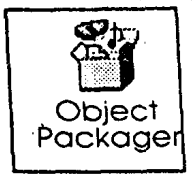


图 22.26

选择目标文档，选择 Edit/Paste 将对象包粘贴到目标文档中。

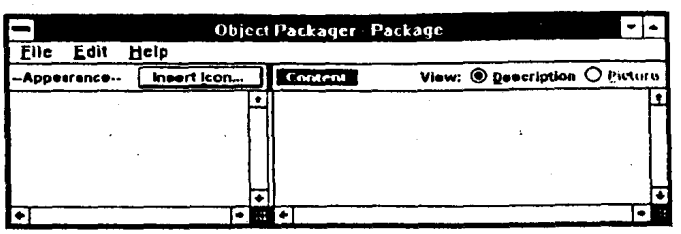


图 22.27　对象包

用对象包修改对象链

Windows 3.1 提供了独立的可用它传输、修改、更新对象包。

嵌入文档变成对象包

对象包可由嵌入文档或链接文档组成。Object Packager 可让用户交互地创建对象包。开始时，Object Packager 以窗口的形式出现，此窗口由许多不同的工作区组成。

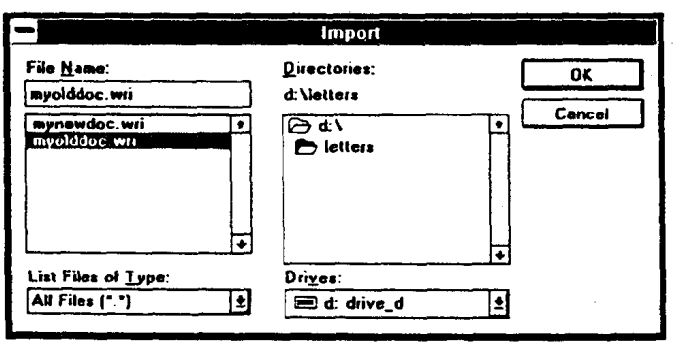


图 22.28　Object Packager

--Appearnce--窗口显示对象包的外观，--Content--窗口显示文档名和内容。

欲创建窗口新的对象包，可选择--Content--，再选择 File/Import…。

从 Import 对话框内选择 MYOLDDOC. WRIT 文件并且按下按钮 OK。--Appearance--窗口显示 MYOLDDOC. WRI 图标，同时--Contents--窗口列出名为 Copy of MYOLDDOC. WRI 的

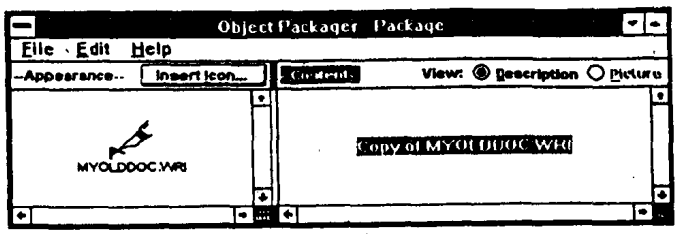


图 22.29 新对象包

描述。在用户选择 File/Import…的时候，Object Packager 将专用文件作为一个 Write 文档识别，再显示 Write 图标。对象包被插入到一个目录文档(如 Write 文档)中。

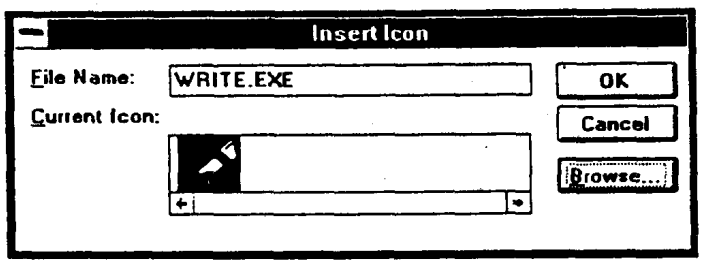


图 22.30 使用一标准 Windows 程序图标

Edit/Copy Package 命令向剪贴板拷贝对象包。在相应位置上选择 Edit/Paste 可在一个 Write 文档内插入这个对象包。以这种方式生成对象包与用 File Manager 生成对象包的结果是一样的。

摘要：使用 Object Packager 产生对象包

(1)运行 Object Packager 并且选择--Content--窗口。
(2)选择 File/Import…。
(3)选择源文档。
(4)按下按钮 OK。
(5)文件名出现在--Content--窗口，而图标出现在-Appearance--窗口。
(6)选择 Edit/Copy Package。
(7)切换到目标文档。
(8)选择 Edit/Paste，将对象包拷贝到目标文档。

改变对象包图标

一旦 Object Packager 已被装入，并且一个对象包也被装入以后，代表对象包的图标就会在--Appearance--窗口出现。由 Object Packager 使用的缺省图标是应用程序用于产生文档的图标，用户可以按下 Insert 按钮来改变这一图标。还可选择一个 Windows 缺省图标，或选择一个用户自定义图标。为了选择缺省 Windows 图标，可在列表框内找寻 Current Icon。

如果用户更希望使用另一个图标，可按下 Browse 按钮。这样就打开了一个对话框，此框允许用户选择一个专用的应用程序，再从中选择一个图标。

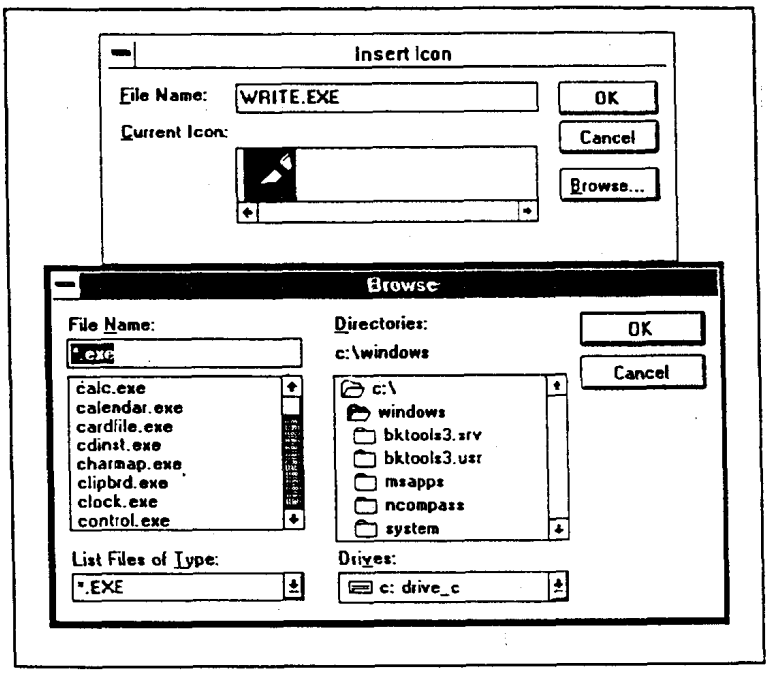


图 22.31　从一个文件中获取一个图标

如果用户希望使用自己本身的图标，那么可使用 Paintbrush 设计用户图标，并且向剪贴板拷贝此图标，再切换到 Object Packager，选择--Appearance--窗口。选择 Edit/Paste 在对象包中插入新图标。

22.5　OLE　流

OLE 的面向对象特性是先前使用的文件编辑方法所不能满足的。在这些旧方法中，文件中仅仅存储纯数据，对数据不能进行封装。一旦想要加工 OLE 对象，且仍用文件方式的话，就会出现代码与数据封装问题。

流允许我们不分隔出数据元素，而直接对文件中的数据进行操作与写入，流还允许任选 I/O 设备直接向其发送对象。

要了解流是如何工作的，请考虑一个 C 的源代码结构。流和 C 代码都具有如下性质：

- 都是可以在两种方向上顺次传输数据的输入/输出通道。
- 顺序排序入口，就文件而言，数据被写入文件以便以后被读出。流也提供同样的功能，而且还可以存储不同的对象类型。
- 使用相似的文件操作。

下列基本事件需使用流：

(1)分配 OLESTREAM 结构。

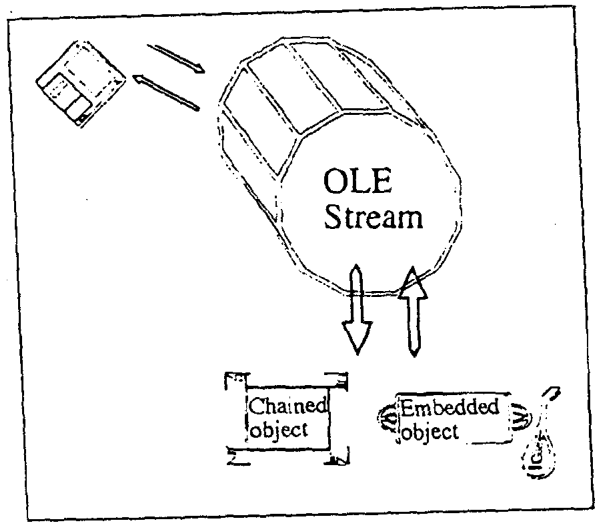


图 22.32　流操作类似于顺序文件

(2)在 OLESTREAMVTBL 结构中添加 Put 和 Get 方法,以便对流进行读出与写入。

(3)调用函数 OleSaveStream 向流中写入对象。

(4)调用函数 OleSavedClientDoc 通知 OLE 客户库。

(5)调用函数 OleLoadFromStream 从流中读数据。

(6)在程序结束时释放 OLESTREAM 结构。

下面我们将对 OLE 对象操作与流进行详细考察。

22.5.1　OLESTREAM 和 OLESTREAMVTBL 结构

第一步是分配 OLESTREAM 结构,分配方式与 OLECLIENT 结构类似,调用入口参数是指向虚拟方法的指针。

这个称为 OLESTREAMVTBL 的指针定义了两种负责用于对输入/输出设备读和写操作的方法。有必要定义接口类型(例如文件或内存),执行物理的读写操作。

实例登录处理内存和结构分配,因为这些处理只需要执行一次。OLESTREAM 结构包含以下元素:

```
typedef struct _OLESTREAM
{
    LPOLESTREAMVTBL lpstbl;
} OLESTREAM;
```

域	数据类型	描述
lpstbl	LPOLESTREAMVTBL	指向用于输入/输出的方法表的指针

OLESTREAM 结构分配示例

```
LPOLESTREAM lpstream;
```

```
HANDLE hobjStream;
                                                    // Allocates the OLESTREAM structure
hobjStream = GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT,sizeof(OLESTREAM);
lpstream = (lpolestream)GlobalLock(hobjStream);
```

OLESTREAM 结构作为一个全程内存块来分配。调用 GlobalAlloc 函数产生 hobjStream 内存句柄，GlobalLock 将其转换成一个 lpstream 指针。

OLESTREAMVTBL 结构

OLESTREAMVBTL 结构由两个入口组成，这两个入口包含对输入/输出媒体的读写方法。OLESTREAMVTBL 结构包含下列结构：

```
typedef struct _OLESTREAMVTBL
{
    DOWRD Get (lpstream,lpszBuf,cbbuf);
    DOWRD Pat (lpstream,lpszBuf,cbbuf);
} OLESTREAMVTBL;
```

域	描述
Get	读数据
Put	写数据

Get 和 Put 域中必须填以两种方法用以处理读写任务。对于每一个 OLE 流，OLE 客户 DLL 调用相应的方法。函数地址必须作为过程实例的地址传送，此地址可通过调用函数 MakeInstance 来获得。

我们使用 MakeProcInstance 函数获得这个地址。

例子：OLESTREAM/OLESTREAMVTBL 分配

对 OLESTREAMVTBL 结构的引用，包含两种 Get 和 Put 方式，这个引用必须在 OLESTREAMVTBL 结构中输入。

```
LPOLESTREAM lpstream;
OLESTREAMVTBL sTbl;
HANDLE hobjStream;

                                                    // allocates the OLESTREAM structure
hobjStream = GolbalAlloc(GMEM_FIXED | GMEM_ZEROINIT,sizeof(OLESTREAM);
lpstream = (LPOLESTREAM)GlobalLock(hobjStream);

/* allocates the OLESTREAM + OLESTREAMVTBL structure and entry of the methods */
lpstream->lpstbl = (LPOLESTREAMVTBL)&StB1;
lpstream->lpstbl->Get= MakeProcInstance(Get,hInst);
lpstream->lpstbl->put= MakeProceInstance(put,hInst);
```

供给 OLESTREAM 数据句柄

OLESTREAM 结构只由一个入口组成，即对 OLESTREAMVTBL 结构的引用。为了将对象传送到文件中，我们需要一个文件句柄。

面向对象编程原理在于数据封装。在这种情况下，我们可以在 OLESTREAM 结构中插入

文件句柄。GlobalAlloc 函数一般处理分配，同时也可能为生成 OLESTREAM 结构而打开文件。

```
typedef struct _olestreamprivat
{
    LPOLESTREAMVTBL lpstbl;
    int fh;
} OLESTREAMPRIVAT;
typedef OLESTREAMPRIVAT FAR *LPOLESTREAMPRIVAT;
LPOLESTREAMPRIVAT lpstream;
OLESTREAMVTBL StBL;
HANDLE hobjStream;

                                        // Allocates the OLESTREAM structure including file handle
hobjStream = GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT,sizeof(OLESTREAMPRIVAT));
lpstream = (LPOLESTREAMPRIVAT)GlobalLock(hobjStream);

                                    // Allocates the OLESTREAM and OLESTREAMVTBL structures
                                                                        // Entry of metyods
lpstream->lpstbl = (LPOLESTREAMVTBL)&sTbl;
lpstream->lpstbl->Get= MakeprocInstance(Get,hLinst);
lpstream->lpstbl->put = MakeProcInstance(put,hInst);

                                                                        // Open file
lpstream->fn = _lnpen();
```

22.5.2 读/写一个 OLE 流

Put 和 Get 方法必须对 OLE 客户程序有效，这两种方法可以和 C 读写函数相互对应。同样地，也可与输入函数对应。记住，这些方法只用于 OLE 程序，并且在应用程序中不会直接被调用，在 Windows 函数中也不直接调用它们。

Put 写流

Put 方法可写流的独立字节。Put 有以下基本结构：

```
LONG Put (lpstream,lpszBuf,obbuf)
```

域	数据类型	含义
lpstream	LPOLESTREAM	指向 OLESTREAM 结构的指针
lpszBuf	LPSTR	指向数据缓冲区的指针
obbuf	DWORD	被写的字节数

Get 读流

一旦对象已被送到运行流中，OLE 程序库让用户使用 Get 方法从运行流中取得一个对象。

```
LONG Get (lpstream,lpszBuf,cbbuf)
```

域	数据类型	含义
lpstream	LPOLESTREAM	指向 OLESTREAM 结构的指针
lpszBuf	LPSTR	指向数据缓冲区的指针
cbbuf	DWORD	用于读的字节数

例子:Put 和 Get 方法

下例从(向)一个文件中读(写)流。在这种情况下的文件句柄归属于 OLESTREAM 结构。

```
DWORD FAR Get(LPSTREAMPRIVAT lpstream,LPSTR lpszBufm DWORD cbuf)
{
    /* If the quantity of data is asmaller than 64K */
    if (cbbuf( 0x00010000)
    {
        /* use the Standard _lread() function */
        ruturn( _lread(lpstream->fn,lpszBuf,
                (WORD) cbbuf));
    }

    /* use own lread() function */
    return( lread(lpstream->fn,lpszBuf,cbbuf));
}

DWORD FAR Put(LPSTREAMPRIVAT lpstream,LPSTR lpszBuf,DWORD cbbuf)
{
  /* If the data quantity is less than 64K */
    if (cbbuf,0x00010000)
    {
        /* use standard _lwrite() function */
        ruturn( _lwrite(lpstream->fn,lpszBuf,
                (WORD) cbbuf));
    }
    /* Use own lwrite() function */
    return( lwrite(lpstream->fn,lpszBuf,cbbuf));
}
```

OLE 对象可超出 64K

既然存在 OLE,客户应用程序就必须假定单一的数据量能超过预先强制的 64K 的限制。由于某些标准 C 函数仍不能一次处理超过 64K 的数据,所以必须为溢出的数据块开辟溢出区。以下演示如何从(向)一个文件中读出(写入)超过 64K 的数据量。

```
DWORD PASCAL Alread(int hFile,LPSTR lpszBuf,DWORD cbbuf)
{
    /* Pointer to data buffer */
    BYTE huge * hpBuffer = lpszBuf;
    /* Number of data to be read */
    DWORD dwByteCount = cbbuf;

    while (dwByteCount > MAXREAD)
    {
        if (_read(hFile,hpBuffer,(WORD)MAXREAD)
```

```
            ! - MAXREAD
        return( 0);
        dwByteCount -= MAXREAD;
        hpBuffer += MAXREAD;
    }
    if ((DWORD)_lread(hFile,hpBuffer,
            (WORD)dwByteCount ! = dwByteCount)
        retrun dwBytes;
}

DWORD PASCAL lwrite(int hFile,LPSTR lpszBuf,DWORD cbbuf)
{
    /* Pointer to data buffer */
    BYTE hug *hpBuffer = lpszBuf;
    /* Number of data to be written */
    DWORD dwBytecount = cbbuf;

    while (dwByteCount > MAXREAD)
    {
        if (_lwrite(hFile,(LPSTR)hpBuffer,
                (WORD)MAXREAD ! =MAXREAD)
            return( 0);
        dwBytecount -= MAXREAD;
        hpBuffer += MAXREAD;
    }

    if (_lwrite(hFile,(LPSTR)hpBuffer,
            (WORD)dwByteCount) ! = (WORD)dwByteCount)
        return( 0);
    return( dwBytes);
}
```

两个函数都以 PASCAL 说明，以防止与标准函数发生冲突。在第三个传送的参数内，函数区别于 C 标准，这是因为数据块现在能使用 DWORD 而不使用 WORD。第一个被传送的参数是文件句柄，第二个是数据缓冲区。

22.5.3 存储和装入一个对象

到现在为止，我们介绍了基本的流操作。OleSaveToStream 存储一个对象，且有下列结构。

OleSaveToStream 函数

OLESTAEAM OleSaveToStream (lpObject,lpStream)

域	数据类型	含义
lpObject	LPOLEOBJECT	指向将要存储的对象的指针
lpStream	LPOLESTREAM	指向 OLESTREAM 结构的指针

存储比较简单，这归功于在生成 OLE 流时所做的准备工作。面向对象的结构使得浏览

一个客户应用程序更为容易。OleSaveToStream 的典型应用是通过一个应用程序的菜单项存储对象。记住，对每个对象，都必须调用 OleSaveToStream。

OleSavedClientDoc 函数

一旦对象完全处于流中，就可由 OleSavedClientDoc 函数通知 OLE 客户程序库。

```
OLESTATUS OleSavedClientDoc (lhClientDoc)
```

只需要复合文档句柄即可。

OleLoadFromStream 函数

OleLoadFromStream 函数从流中选择一个对象。不管对象是链接的还是嵌入的，这种对象选择都没有区别。在任何情况下，函数的工作也是相同的。

```
OLESTATUS OleLoadFromStream (lpStream,lpszProtocol,lpClient,
        lpClientDoc,lpszObjname,lplpObject)
```

参数	数据类型	描述
lpStream	LPSTREAM	指向 OLESTREAM 结构的指针
lpszProtocl	LPSTR	指向协议名称的指针
lpClient	LPOLECLIENT	指向 OLECLIENT 结构的指针
lhClientDoc	LHCLIENTDOC	OLE 文档句柄
lpszObjname	LPSTR	指向 OLE 对象名的指针
lplpObject	LPOLEOBJECT FAR	指向接受新对象地址的变量的指针

参数与 OleCreateFromClip 及 OleCreateLinkFromClip 函数很类似，下例产生一个来自流的对象：

```
                                        // load an OLE object from the stream
#define PROTOCOL ((LPSTR)"StdFileEditing")

OleLoadFromStream(lpStream,PROTOCOL,
    lpclient,lhClientDoc,szarObjectName,
    &lpObject);
```

22.6 服务器

在使用 OLE 的过程中，服务器的访问很关键。在前面所列出的 OLECL 应用程序中，将 Paintbrush 应用程序作为服务器。如果用户程序能够使用现存的服务器，这当然是很好的。但是，用户一般需要编写其自身的服务器应用程序。因此，我们将给出对于开发一个服务器程序的一些提示。

22.6.1 一般的提示

OLE 服务程序库 OLESVR.DLL 包含十个函数，使用这些函数，服务器自己可在程序库中登录，并且这十个函数向程序库报告对象和文档的变化。

客户和服务器之间的通信

在本章开头我们提到了客户/服务器的通信。既然用户对 OLE 的认识有所增加，那么就让我们通过看下列图示来研究一下两者是如何互相影响的，同时研究一下所产生的通路。

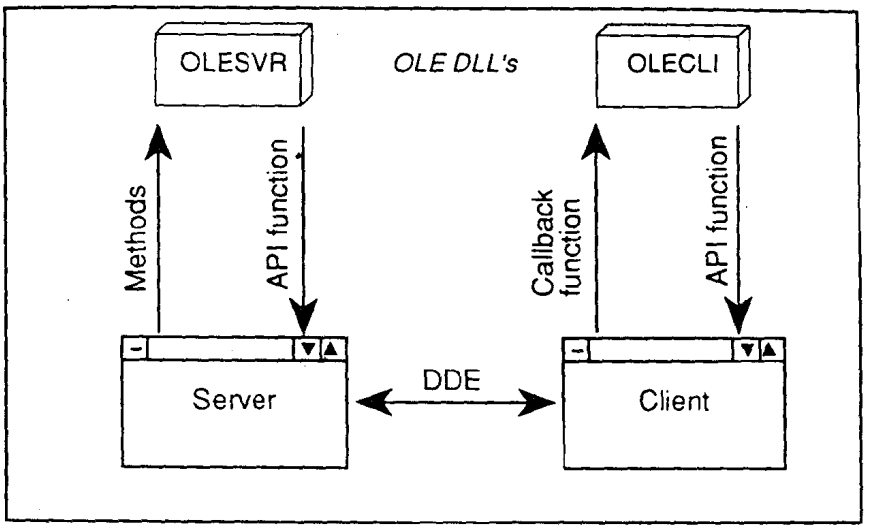


图 22.33　客户和服务器间的通信

客户和服务器间的通信在不同层次上发生。两个应用程序都能根据 OLE 的要求调用 API 函数，这个调用由相应的 DLL(OLECLI 用于客户，OLESVR 用于服务器)完成。

我们也提到过客户回调函数，现在我们再讨论一下服务器方法，这类方法实现对象和文档(在特定的结构中定义)的不同动作。两个 OLE 程序库通过 DDE 相互通信(这种通信在后面的版本中可能会有所变化)。

假设我们的客户应用程序示例包含一个链接对象，这个对象是用户双击鼠标按钮，且运行服务器 Paintbrush。这样，用户就可以编辑图像，切去图像的一部分等。服务器必须向客户报告每种变化，以更新客户区。

服务器包含 OLECLIENT 和 OLECLIENTVTBL 结构(指向 OLECLIENTVTBL 回调函数的指针)。客户函数调用 OLECLI 库中的某个函数来检索新的数据，这个函数向服务器程序发送一个查询，再依次激活相应的服务器方法。

对服务器有益的 OLE 结构

除了已经提到的两种结构以外，还有处于头文件中的另外六种结构与服务器协同工作。这八种结构可被分成四组，每组中有一个表，这个表包含对下一种结构的引用；并且每下一种结构都包含由客户使用的指向服务器、对象或文档方法的指针。

以下列出四对结构：

数据结构	含义
OLECLIENT	对 OLECLIENTVTBL 结构的引用
OLECLIENTVTBL	指向客户回调函数的指针

OLEOBJECT	对 OLEOBJECTBL 结构的引用
OLEOBJECTTVTBL	指向对象方法的虚拟方法表的指针
OLESERVER	对 OLESERVERVTBL 结构的引用
OLESERVERTBL	指向服务器方法的虚拟方法表的指针
OLESERVERDOC	对 OLESERVERDOCTBL 结构的引用
OLESERVERDOCTBL	指向文档方法的虚拟方法表的指针

用户也许已注意，这些结构使用三种不同的方法类型，每个 OLE 方法作为一个回调函数定义，并且必须输出到定义文件中。在一个来自服务器 DLL 的查询事件中，这些方法处理大部分工作。

服务器类型

服务器可能是一个微型的或一个完整的执行过程。微型服务器不能在没有客户应用程序的情况下独立地运行，这就是说，微型服务器只能通过一个客户才能被激活。

以绘图程序 MS Draw 为例，此程序比如说是由 Windows 2.0 微型服务器 Word 来支持的，且在对象不超过 50K 的情况下使用。但是，如果希望使用独立于其他程序的服务器，那么譬如 Excel 或 Paintbrush 之类的一个完整执行过程就是必要的了。

进一步地说，服务器应用程序既可能是单一的文档服务器(SDI)，也可能是一个多文档服务器(MDI)。Excel 应用程序就是作为一个 MDI 服务器生成的，即在多个对象要修改时只打开一个附加窗口。另一方面，只要有了一个 SDI 服务器，就可启动程序的下一个实例。

使用现有应用程序

协同使用一个现有的 Windows 应用程序与 OLE 的过程中，有一些因素需要考虑。

数据结构和文件总应该包含一个版本号，然而，服务器可能会混淆来自同一源文件的不同数据元素，所以，要使用 cbWndExtra 为每一窗口分配几个恰当的字节，以便更好地完成对象和文档的定位。用户程序将不得不与剪贴板交互作用，并且应能支持元文件格式或位图格式。

22.6.2 设立一个服务器的必要步骤

这些独立步骤应按给定的顺序执行。

步骤 1：定义用户自身的 OLE 结构

除了以上描述的数据结构以外，用户还能定义自己本身的数据结构，这些数据结构的第一个域应包含对三个虚拟方式表之一的引用，任何需要的数据可存放在这些结构中。

```
typedef struct
{
    LPOLESERVERDOCVTBL lpvtbl;
    BOOL fFlags
    HANDLE hMem;
} MYOWNDOC
```

步骤 2:在登录数据库中安置服务器

每个服务器必须在登录数据库中登录,并且说明所能访问的对象。这些操作通过 RegCreateKey,RegSetValue 和 RegCloseKey 函数完成。客户应用程序于是就能决定服务器是否有效地建立在数据库入口之上。

所有引用 OLE 的键从分层树结构的最高层上开始(键直接处于根目录的下方,根目录由 HKEY_CLASSES_ROOT 常数定位)。

每个服务器必须至少建立这样两个键,一个定义对象的类名,另一个定义文件的扩展名。键被赋予一定的值,这些值用于启动相关的被请求的服务器(例如,用户在一个客户应用程序内双击对象时)。

因此,有必要传输类名给键,并且要带扩展名。下表显示这样的例子。

关键字名称	关键字内容
HKEY_CLASSES_ROOT\PBRUSH	Paintbrush 图像
HKEY_CLASSES_ROOT\.PCX	PBrush
HKEY_CLASSES_ROOT\	服务器文本
HKEY_CLASSES_ROOT\.RC	MyServer Key

HKEY_CLASSES_ROOT\Classname 键至少占有分层结构的两层,这个键的子键必须具有固定的预定义名。

HKEY_CLASSES_ROOT\CLassname\Protocol\StdFileEditing

这个层次上的键描述由服务器支持的 OLE 协议的专用属性。

另外,带有 server 名的键必须包含服务器应用程序的文件名,包括完整的路径名。

```
HKEY_CLASSES_ROOT\CLassname\protocol\StdFileEditing\server
HKEY_CLASSES_ROOT\CLassname\protocol\StdFileEditing\verb0
HKEY_CLASSES_ROOT\CLassname\protocol\StdFileEditing\verb1
HKEY_CLASSES_ROOT\CLassname\protocol\StdFileEditing\verb2
```

因为服务器一般不只产生一个有效动词,所以,所有动词必须由一个唯一的子键来定址。在这种情况下,子键 0 总是标记主要的动词。

步骤 3:传送数据到剪贴板

每种剪贴板格式必须有自己的句柄,以便在剪贴板上顺序排放这些句柄。绝对有必要遵循下列函数调用次序:

. OpenClipboard
. EmptyClipboard
. SetClipboardDataH ;H 用于每种格式
. CloseClipboard

对链接对象来说,必须使用 ObjectLink 格式。也有必要在保存文档之前阻止用户向剪贴板拷贝文档。

步骤 4:实现方法结构

这一步,用户必须用每一方法的过程实例地址来填充服务器、文档和对象三个虚拟方法

表(VTBL),这些地址须通过 MakeProcInstance 函数决定,因为这些方法是回调函数。所有方法必须包含在模块定义文件的 EXPORTS 段。这些方法的出现次序是不重要的。方法本身也可作为哑函数存于当前文件中。

步骤 5:初始化服务器应用程序

一旦服务器应用程序的第一个实例被启动之后,就必须执行下列初始化过程:

- 登录 Native Data、OwnerLink 和 ObjectLink 格式。
- 使用过程实例地址初始化 VTBL。
- 初始化 OLESERVER 结构。
- 用 OleRegisterServer 函数登录服务器。
- 根据命令行决定服务器是用于嵌入对象还是链接对象。
- 初始化 OLESERVERPOC 结构。
- 用 OleRegisterServerDoc 函数登录初始文档。

OleRegisterServer 函数定义如下:

```
OLESTAUS OlesRegisterServer(lpszClass,lpsrvr,lplhserver,hInst,srvruse)

LPSTR lpszClass;                                    /* Pointer to class name */
LPOLESERVER lpsrvr;          /* Pointer to previously provided OLESERVER structure */
LHSERVER FAR * lplhserver;                  /* Pointer to the server handle */
LPLONG hInst;                                       /* Instance hanle */
OLE_SERVER_USE srvruse;                  /* One or more instances possible */
```

类名作为根目录的一个子键,必须在登录数据库中具有相同的定义。如果成功地执行了调用,那么第三个参数将指向服务器句柄,这个句柄必须传输给所有后续的 OLESVR 调用。

为了决定服务器是用于链接对象还是用于嵌入对象,有必要在命令行中查找单词"embedding"。这种查找不区分大小写(亦即,大写与小写同样对待)。如果单词包含一个相关的文件名,那么指定的文件就包含一个链接对象。如果只有单词"embedding"而没有文件名,那么对象就是一个嵌入对象。在这两种情况下,还不应该显示出服务器窗口(SW_HIDE 常量被传送给 ShowWindow 函数,以取代 CmdShow 参数)。如果服务器 DLL 认为有必要显示,那么 DLL 会激活所需要的方法。如果服务器应用程序作为一个独立应用程序启动的话,"embedding"就不会出现在命令行中。

在初始化步骤的执行过程中会产生不同的错误,这些错误严重到足以导致整个服务器程序终止。我们建议用户对所有函数调用均检查其返回值。例如,OleRegisterServerDoc 函数在执行成功时返回值为 OLE_OK。

步骤 6:改变窗口标题和菜单项

由于用户知道哪个对象正处于服务器窗口中,所以,对于链接对象,就应将文件名加到标题条中。在处理一个嵌入对象时,标题条的结构在某种程度上应该如下所示:

Name of the Server applicatior - Objecttype - client document

另外,ModifyMenu 函数应将 Save 菜单项改为 Update,并且将 Exit 菜单项改为 Exit and Return to…,以回到客户文档。

步骤 7：实现基本方法

为了使服务器应用程序能像一个真正的服务器一样工作，必须写出专用的基本方法。下表显示哪些方法是必要的。

服务器方法	含义
Create	在客户通过菜单项 Insert Object 生成了一个新的菜单对象的情况下由服务器 DLL 调用。
CreateFromTemplate	在客户通过菜单项 Insert Object 生成了一个新的复制对象的情况下由服务器 DDL 调用。
Edit	如果客户已激活了一个嵌入对象，并且服务器必须生成一个新文档，则由服务器 DLL 调用。
Exit	当出现一个严重错误强行终止服务器时，由服务器 DLL 调用。
Open	如果用户在客户程序中激活一个链接对象，则由服务器 DLL 调用。
Release	由服务器 DLL 调用，告诉服务器没有文档或对象被处理。

文档方法	含义
Close	在文档要被关闭时，由服务器 DLL 调用。
GetObject	是否客户激活了函数 OleGetObject。
Release	在所有链向对象的 DDL 链接关闭时，由服务器 DLL 调用。
Save	在关闭客户的过程中，若用户想存储一个包含链接对象的文档，则由服务器 DLL 调用，以存储一个文件。
SetHostNames	由服务器 DLL 调用，以返回客户文档名和对象名。

对象方法	含义
DoVerb	由服务器 DLL 调用，以执行一个特殊动作。
Enumformats	在客户 DLL 的查询过程中，由服务器 DLL 调用几次，以确定服务器返回的数据格式。
GetData	一旦客户想显示对象或在文件中存储对象，就由服务器 DLL 调用。
QueryProtocol	由服务器 DLL 调用，使其能够确定服务器支持的是哪一种协议 。
Realease	如果一个对象不再与一个客户相连，则由服务器 DLL 调用。
SetData	在 DLL 启动服务器以编辑一个嵌入对象时，由服务器 DLL 调用。
Show	在必须显示一个对象时，由服务器 DLL 调用。

步骤 8：设计一个简单的停机

为了退出一个服务器应用程序，有必要执行一个停机操作，释放三个虚拟方式表（使用 FreeProcInstance 函数）。也要调用 OleRevokeServer 函数（最好在处理 WM_CLOSE 消息的过程中），以释放所有文档和所有对象。

一些附加要点

此处的描述对初步验证服务器应用程序来说已足够了。以下几点也是值得注意的。

如果用户选择 File/Open…，File/Save AS…或 File/Save | Update，那么就有必要要求响应。对一个服务器来说，每次只能包含一个文档(SDI)，File/New 和 File/Open…可能引起一个链接被中断。

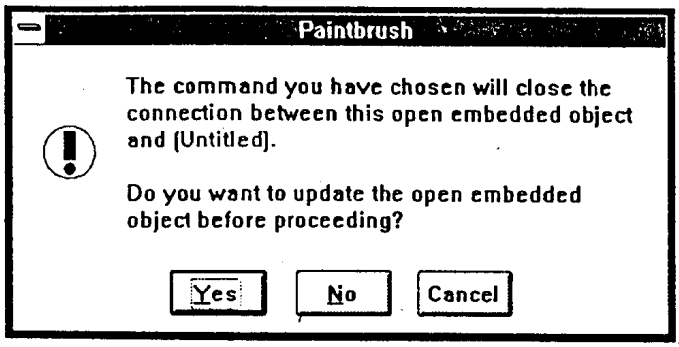


图 22.34

服务器应用程序在关闭前显示上图所示的消息框(在处理嵌入对象的过程中)。嵌入对象的变化在客户中还没有被更新。如果消息框被确认，则客户应用程序的对象在 Paintbrush 退出前被更新。

利用三个虚拟方法表还可实现其他 OLE 方法。

第二十三章　系统登录数据库

Windows 3.1 允许通过应用程序进行系统范围内的数据处理，这与 Windows 初始化文件 WINI.INI 和 SYSTEM.INI 有明显的相似性。

但是，登录数据库具有一个标准化的结构，并且它的速度较高，使得它更适合于在程序运行时读取数据。登录数据库不是应用程序的专用数据，它不代替初始化文件。

当然，应当注意到，这可通过使用 API 函数实现。

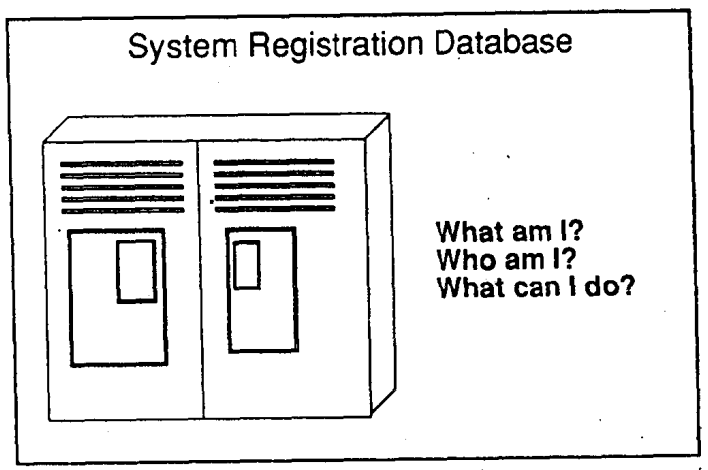


图 23.1　登录数据库的含义

23.1　数据库的结构

数据库的主要工作是使文件管理器和打印管理器相结合，同时，它对运行 OLE 应用程序也是很重要的。

用户可以用两种方法访问数据库：

- 一种方法是用 API 直接从程序内调用 Dynamic Link Library SHELL.DLL。另一种方法是运用 Software Development Kit (SDK)提供的特殊工具 REGEDIT.EXE。
- REGEDIT.EXE 编辑程序对程序开发非常有用，终端用户不必直接在数据库里进行编辑。

数据库由 REG.DAT 文件构成，这些文件可以在 Windows 目录中找到。数据被定义为层次树，并且以二进制形式存储在 REG.DAT 文件中。只能在有限的程度内用十六进制编辑程序检索目录。

因为结构被严格地配置为树状，所以它由以“键”标识的节点所构成，最高一级是根目录，键从这里开始分枝，每个键能定位一个值，这个键由 ASCII 字符串构成。

因此，容许范围被限制在 ASCII 码 32 到 127 的字符之间，并且包括反斜杠或空格之类的控制符。然而，由键定位的值可由任务所期望的 ASCII 字符组成，而这只在它们保持为字符串的情况下才能成立。

下列函数可用来访问数据库：

RegCloseKey	关闭一个键
RegCreateKey	创建一个键
RegDeleteKey	删除一个键
RegEnumKey	返回一个键的子键
RegOpenKey	打开一个键
RegQueryValue	读取键值
RegSetValue	使串与键值关联

RegOpenKey

打开一个由应用程序支持的键就开始了数据库会话过程。RegOpenKey 函数可打开一个键，以便允许随后的读值操作。

LONG RegOpenKey (hKey,lpszSubKey,phkResult)

参数	数据类型	含义
hKey	HKEY	更高层打开键的句柄
lpszSubKey	LPSTR	键定义
phkResult	PHKEY	接收被打开键的句柄的地址

通常，键是从根目录指定的，如果是这样的话，RegOpenKey 就接收常量 HKEY_CLASSES_ROOT。

参数 lpszSubKey 表明指定键名称的字符串，第三个参数(return)必须指向 HKEY 类型的变量。在这个函数结束后，这个参数包含一个打开的键的的句柄，在以后使用 RegCloseKey 函数关闭此键时需要这个句柄。

RegCreateKey

如果在数据库空白区不存在键，则可用 RegCreateKey 函数产生一个新的键。RegCreateKey 和 RegOpenKey 有相同的参数，但名字不同。

RegQueryValue

RegSetValue

在打开或者产生一个新键之后，可以用 RegQueryValue 函数读出值，或者用 RegSetValue 函数设置值。

LONG RegSetValue (hKey,lpszSubKey,lpszValue,lpcb)

参数	数据类型	含义
hKey	HKEY	键句柄
lpszSubKey	LPSTR	键名
lpszValue	LPSTR	返回串的缓冲区
lpcb	LONG FAR *	包含返回长度的变量的地址

示例:RegQueryValue

```
dw = KEYNAMESIZE;
if (! RegQueryValue (hKeyClasses,szClass,szName,&dwSize))
```

当使用 RegQueryVale 时,应当注意在调用前重置期望的字符串长度。字符串长度由一个变量指定,变量的地址作为第四个参数输入。

即使在随后的调用中,也应重置长度,因为 RegQueryValue 函数把实际长度作为返回值写入变量中。

RegSetValue

```
LONG RegSetValue (hKey,lpszSubKey,dwType,lpszValue,cb)
```

参数	数据类型	含义
hKey	HKEY	键句柄
lpszSubKey	LPSTR	键名
dwType	DWORD	REG_SZ 常量
lpszValue	LPSTR	将设置为字符串的值
cb	DWORD	忽略

示例:RegSetValue

```
RegSetValue(HKEY_CLASSES_ROOT,
    ".C",                                    /* key */
    REG_SZ,
    "PROG",                                  /* value */
    4);
```

RegCloseKey

很重要的一点是要保证运用 RegCloseKey 函数再次关闭已经输入值的键。只有这样,值才被真正传送到数据库中,并且可供其他应用程序读出。

RegEnumKey

RegEnumKey 函数可在键范围内查找所有子键,例如由 HEKY_CLASSES_ROOT 指定的句柄。

```
LONG RegEnumKey (hKey,dwIndex,lpszBuffer,cbBuffer)
```

参数	数据类型	含义
hKey	HKEY	键句柄
lpszIndex	DWORD	键索引
lpszBuffer	LPSTR	指向存放键的缓冲区的指针
cbBuffer	DWORD	键缓冲区的大小

为了逐一确定所有子键，必须从零开始增大计数器 dwIndex，这可以通过使用循环语句 FOR 来实现。

示例：RegEnumKey

```
for (i = 0;RegEnumKey(HKEY_CLASSES_ROOT,i++,szClass,KEYNAMESIZE);
```

每调用一次，该函数就返回一个相同级别的子键，并且将键名放入缓冲区中。

23.2 登录数据库和 OLE

OLE 应用程序要使用登录数据库中的信息。客户一般要读取登录数据库，以获得有关系统服务器的数据。Microsoft Windows Write 访问对客户来说是很少见的。

另一方面，服务器把它的值存储在登录数据库中。为了允许所有客户和服务器都与登录数据库相配，必须以与所有客户和服务器一致的结构存储数据。

图 23.2 说明表明了 Paintbrush 应用程序是如何完成这一点的。

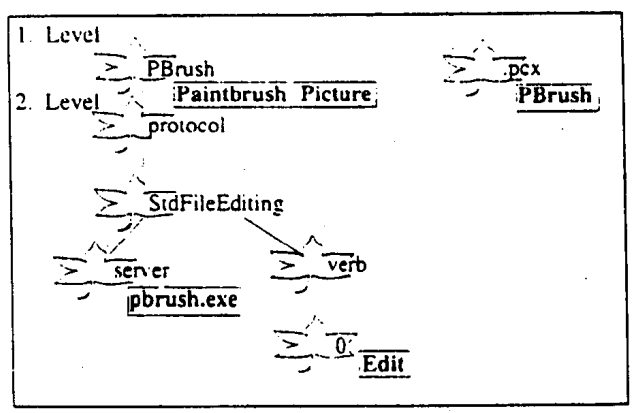


图 23.2 Paintbrush 服务器的登录数据库结构

客户应用程序在程序开始时读取登录数据库以获得所有服务器名。对于 Paintbrush 而言，即是从 PBrush 键下得到的名字 PBRUSH. EXE。

检索名字是按如下步骤进行的：先检查根目录下所有不以句点开头的键，然后在\protocol\StdFileEditing\server 中检索键名。所需要的是这些检索出来的键的值。

对于服务器所支持的每个类都必须输入类名和服务器路径名，所以，类名应当与 OLE

函数 OleRegisterServer 使用的一样。

如果 OLE 函数有对象处理器 DLL，则后者必须与处理器键一起输入。

23.3 登录数据库和 OLE 示例

例如，有一个有效的列表框，其中填有系统中输入的服务器名。

列表框的大小必须总是适合于主窗口的大小，WM_SIZE 和 MoveWindow 函数可做到这一点。

这个列表框已由 WM_CREATE 建立和填充。要使程序规模较小且便于管理，所有 OLE 功能都未被使用。

用户可以轻易地把这个示范程序加入到书后面出现的 OLE 应用程序中。

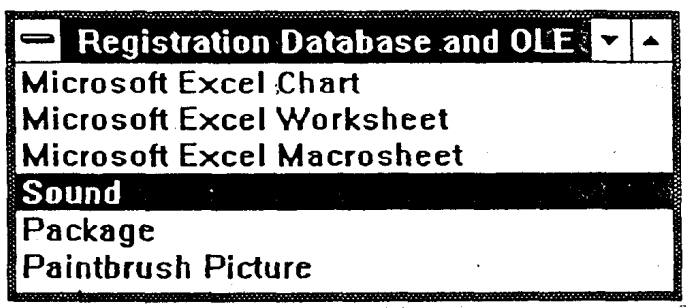


图 23.3

新函数	描述
RegEnumKey	返回键的子键
RegOpenKey	打开一个键
RegQueryValue	读取键值

源代码：REGOLE. C

```
/* * REGOLE. C * * *
```

```
    MSG msg;                                                    // Message handle
    HWND hWnd;                                                  // Window handle
    int nError;                                                 // Error integer

    strcpy(szAppName,"regole");                                 // Application name
    hInst = hInstance;                                          // Instance handle

    if (! hPrevInstance)
    {                                                           //Register first instance

        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                                                // If registration fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;                                      // display error
        }
    }

    /* * Specify appearance of application's main window * * * * * * * * * */

    hWnd = CreateWindow(szAppName,
        "Registration Database and OLE",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,0,
        GetSystemMetrics(SM_CXSCREEN)/3,
        GetSystemMetrics(SM_CYSCREEN)/2,
        NULL,NULL,hInstance,NULL);

    if(hWnd == NULL)                                            // If window handle = NULL
    {
        LoadString(hInstance,IDS_ERR_WINDOW,
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;                                  // display error
    }
    ShowWindow( hWnd,nCmdShow );                                // Make window visible
    UpdateWindow( hWnd );                                       // Update window

    while (GetMessage(&msg,NULL,0,0))                           // Message reading
    {
        TranslateMessage(&msg);                                 // Message translation
        DispatchMessage(&msg);                                  // Send message to Windows
    }
    return (int)msg. wParam;
}

int nCwRegisterClasses(HANDLE hInstance)
```

```
{
    /* * Specify window class * * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;

    wcClass.hCursor = LoadCursor( NULL,IDC_ARROW );
    wcClass.hIcon = LoadIcon( NULL,IDI_APPLICATION);
    wcClass.lpszMenuName = (LPSTR)"DlgMenu";
    wcClass.lpszClassName = szAppName;
    wcClass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wcClass.hInstance = hInstance;
    wcClass.style = CS_VREDRAW | CS_HREDRAW;
    wcClass.lpfnWndProc = StdDlgWndProc;
    wcClass.cbClsExtra = 0 ;
    wcClass.cbWndExtra = 0 ;

    if (! RegisterClass( &wcClass ) )
        return FALSE;
    return TRUE;
}

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function: All messages are sent to this window * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,unsigned
        message,WORD wParam,LONG lParam )
{
    #define KEYNAMESIZE 50
    DWORD dwSize = 0L;
    HKEY hkeyTemp;
    char szClass[KEYNAMESIZE];
    char szExec[KEYNAMESIZE];
    char szName[KEYNAMESIZE];
    int i;                                          // Index for reading subkey
    HKEY hkeyClasses = HKEY_CLASSES_ROOT;
    static HWND hWndList;

    switch (message)
    {
        case WM_CREATE:
            hWndList = CreateWindow("LISTBOX",
            "OLE Server",
            WS_CHILD | WS_VISIBLE,
            0,0,0,0,
            hWnd,NULL,GetWindowWord(hWnd,GWW_HINSTANCE),NULL);
            SendMessage(hWndList,LB_RESETCONTENT,0,0L);

            for (i=0;! RegEnumKey(hkeyClasses,i++,szClass,KEYNAMESIZE);)
                if ( *szClass ! = '.')
                {
```

```
                    /* * Check if this class really accesses a server * * * * * * * * * * */

                    dwSize = 0;
                    hkeyTemp = NULL;
                    lstrcpy(szExec,szClass);
                    lstrcat(szExec,"\\protocol\\StdFileEditing\\server");

                    if (! RegOpenKey(hkeyClasses,szExec,&hkeyTemp))
                    {

                      /* * Get class name's string * * * * * * * * * * * * * * * * * * */

                      dwSize = KEYNAMESIZE;
                      if (! RegQueryValue(hkeyClasses,szClass,szName,&dwSize))
                      {
                        SendMessage(hWndList,LB_ADDSTRING,0,(DWORD)(LPSTR)szName);
                      }
                      RegCloseKey(hkeyTemp);
                    }
                  }

            break;
        case WM_SIZE:
            MoveWindow( hWndList,0,0,LOWORD(lParam),HIWORD(lParam),TRUE);
            break;
        case WM_COMMAND:
            switch(wParam)
            {
                default:
                    return (DefWindowProc(hWnd,message,
                        wParam,lParam));
                    break;
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return (DefWindowProc( hWnd,message,wParam,
                    lParam ));
            break;
    }
    return(0L);
}
```

模块定义文件:REGOLE. DEF

```
NAME Regole
DESCRIPTION 'Registration database and OLE'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
```

```
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:REGOLE. RC

```
#include "regole. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END
```

头文件:REGOLE. H

```
#include <windows. h>
#include <string. h>
#include <shellapi. h>

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21

char szString[128];                          /* Variable for loading resource text */
char szAppName[20];                          /* Window class name */
HANDLE hInst;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:REGOLE. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES =
H = REGOLE. H
LIBS_EXT = shell. lib

all: REGOLE. EXE

REGOLE. RES: REGOLE. RC $ (RESFILES)

REGOLE. OBJ: REGOLE. C $ (H)
```

```
REGOLE.EXE: REGOLE.OBJ REGOLE.RES REGOLE.DEF
    link /NOD/A:16 REGOLE,REGOLE,nul,libw slibcew shell.lib,REGOLE.def
    rc REGOLE.res REGOLE.exe
```

运行 NMAKE 以生成 REGOLE 应用程序：

```
nmake regole.mak
```

所有在登录数据中根目录下的键都被使用 ReyEnumKey 函数的循环语句读出。

所以，我们真正获取了根目录下的关键字。必须把常量 HEKY_CLASSES_ROOT 作为句柄输入 RegÉnumKey 中。变量 i 起键记数器的作用，并且每经过一次循环，它的值都递增。在这种情况下，所寻找的是前面不带句点的键(即结构为\protocol\StdFileEditing\Server 的键)。

```
for (i=0;! RegEnumKey(hkeyClasses,i++,szClass,KEYNAMESIZE);)
    if (*szClass ! = '.')
    {
```

在 szClass 中检索出的类名成为构成关键字的基础。为此，字符串"\\protocol\\StdFileEditing\\Server"被附加到类名上，注意在 C 语言中使用两个反斜杠(\\)，因为单反斜杠(\)代表控制字符。

后面的字符串将只包含单反斜杠(\)。RegOpenKey 函数打开键表达式。如果 RegOpenKey 返回一个 0 值，则键已被找到，因而能够用 RegQueryValue 读出它了。

```
/** Check if this class really accesses a server ********/

dwSize = 0;
hkeyTemp = NULL;
lstrcpy(szExec,szClass);
lstrcat(szExec,"\\protocol\\StdFileEditing\\server");

if (! RegOpenKey(hKeyClasses,szExec,&hKeyTemp))
{

    /** Get class name's string ****************/
    dwSize = KEYNAMESIZE;
    if (! RegQueryValue(hKeyClasses,szClass,szName,&dwSize)
    {
        SendMessage(hWndList,LB_ADDSTRING,0,(DWORD)(LPSTR)szName);
    }
    RegCloseKey(hKeyTemp);
}
```

服务器名因而附属于使用 LB_ADDSTRING 的列表框，所以我们获得了系统中的所有可用的 OLE 服务器。

23.4 文件管理器支持示例

Windows 3.1 推出后，登录数据库更好地将文件与程序而不是 WIN.INI 联系起来。现在，Windows 文件管理器检索 WIN.INI 中的[extensions]段，以便找到文件和处理文件的应用程序之间的联系。

```
[Extensions]
cal=calendar.exe ^.cal
crd=cardile.exe ^.crd
trm=terminal.exe ^.trm
ini=notepad.exe ^.ini
pcx=pbrush.exe ^.pcx
bmp=paintbrush.exe ^.bmp
wri=write.exe ^.wri
rec=recorder.exe ^.rec
```

从 Windows 3.1 开始，登录数据库提供了一个预定义键，文件关联由生成带有所要求的扩展名的键来实现。对于 Paintbrush 而言，键为“.pcx”，并且指向类名 PBrush。如果类名在根目录下与“\\shell\open\\command”子键一起定义，那么把程序名与子键联系起来就有可能了。

以菜单选择为基础，此程序将在数据库中生成几个新的入口，这些新入口将实现“.c”，“.h”，“.def”和“.rc”文件扩展名与 Notepad 应用程序间的联系。

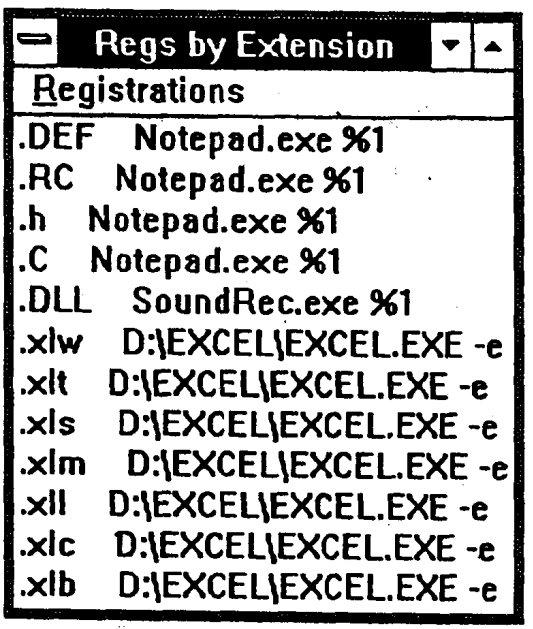


图 23.4

新函数	描述
RegCloseKey	关闭键
RegCreateKey	创建键
RegSetKey	给键赋一个串

源代码：REGPROG.C

```
/* * REGPROG.C * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * Application demonstrating registration database * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
#include "regprog.h"                          // Include REGPROG.H header file

#if _MSC_VER >= 700                           // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                // higher? Disable warning 4028
#endif

/** WinMain (main function for every Windows application ********/

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,
LPSTR lpszCmdLine,int nCmdShow)
{
    MSG msg;                                  // Message variable
    HWND hWnd;                                // Window handle
    int nError;                               // Error integer

    strcpy(szAppName,"regprog");              // Application name
    hInst = hInstance;                        // Instance handle

    if (! hPrevInstance)                      // Initialize first instance
    {
        if ((nError=nCwRegisterClasses(hInstance)) ==-1)
        {
                                              // If initialization fails
            LoadString(hInstance,IDS_ERR_CLASS,
                szString,sizeof(szString));
            MessageBox(NULL,szString,NULL,
                MB_ICONEXCLAMATION);
            return nError;                    // display error
        }
    }

    /** Specify appearance of application's main window *********/

    hWnd = CreateWindow(szAppName,            // Window class name
        "Regs by Extension",                  // Window caption
        WS_OVERLAPPEDWINDOW,                  // Overlapped window style
        CW_USEDEFAULT,0,                      // Upper-left x/y pos.
        GetSystemMetrics(SM_CXSCREEN)/3,      // Initial x/y
        GetSystemMetrics(SM_CYSCREEN)/2,      // default sizes
        NULL,NULL,hInstance,NULL);            // Other parameters

    if(hWnd == NULL)                          // Window handle = NULL?
    {
        LoadString(hInstance,IDS_ERR_WINDOW,  // Display error
            szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,
            MB_ICONEXCLAMATION);
        return IDS_ERR_WINDOW;
    }
    ShowWindow( hWnd,nCmdShow );              // Make window visible
```

```
    UpdateWindow( hWnd );                                    // Update window

    while (GetMessage(&msg,NULL,0,0))                        // Message reading
    {
        TranslateMessage(&msg);                              // Message translation
        DispatchMessage(&msg);                               // Send message to Windows
    }
    return (int)msg. wParam;                                 // Return wParam of last message
}

int nCwRegisterClasses(HANDLE hInstance)
{
    /* * Specify window class * * * * * * * * * * * * * * * * * * * */

    WNDCLASS wcClass;                                        // Main window class

    wcClass. hCursor = LoadCursor( NULL,IDC_ARROW );                    // Cursor
    wcClass. hIcon = LoadIcon( NULL,IDI_APPLICATION);                   // Icon
    wcClass. lpszMenuName = (LPSTR)"RegProgMenu";                       // Menu
    wcClass. lpszClassName = szAppName;                                 // Window class
    wcClass. hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);       // White bkgnd
    wcClass. hInstance = hInstance;                                     // Instance
    wcClass. style = CS_VREDRAW | CS_HREDRAW;                // Horiz. & Vert. redraw
    wcClass. lpfnWndProc = StdDlgWndProc;                               // Window function
    wcClass. cbClsExtra = 0 ;                                           // No extra bytes
    wcClass. cbWndExtra = 0 ;                                           // No extra bytes

    if (! RegisterClass( &wcClass ) )                        // Register window class
        return FALSE;                                        // Return FALSE if registration fails
    return TRUE;                                             // If registration is successful,return TRUE
}

/* * StdDlgWndProc * * * * * * * * * * * * * * * * * * * * * * * */
/* * Main window function * * /
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

long FAR PASCAL StdDlgWndProc(HWND hWnd,unsigned message,
        WORD wParam,LONG lParam )
{
    #define KEYNAMESIZE 50
    DWORD dwSize = 0L;
    HKEY hkeyTemp;
    char szClass[KEYNAMESIZE];
    char szExec[KEYNAMESIZE * 2];
    char szName[KEYNAMESIZE * 2];
    int i;
    HKEY hkeyClasses = HKEY_CLASSES_ROOT;                    // Key assignments
    static HWND hWndList;

    switch (message)
    {
```

```
case WM_CREATE:                                                    // Create listbox
    hWndList = CreateWindow("LISTBOX",
    "Standard Dialog Boxes",
    WS_CHILD | WS_VISIBLE,
    0,0,0,0,
    hWnd,NULL,
    GetWindowWord(hWnd,GWW_HINSTANCE),NULL);
    SendMessage(hWndList,LB_RESETCONTENT,0,0L);
    break;
case WM_SIZE:
    MoveWindow( hWndList,0,0,LOWORD(lParam),HIWORD(lParam),TRUE);
    break;
case WM_COMMAND:
    switch(wParam)
    {
        case IDM_SHOW:
            SendMessage(hWndList,LB_RESETCONTENT,0,0L);
            for (i = 0; ! RegEnumKey(hkeyClasses,i++,szClass,KEYNAMESIZE); )
                if ( * szClass == '.')
                {
                    dwSize = 0;
                    hkeyTemp = NULL;
                    lstrcpy(szExec,szClass);
                    lstrcat(szExec,"\\shell\\open\\command");

                    if (! RegOpenKey(hkeyClasses,szExec,&hkeyTemp))
                    {
                        dwSize = KEYNAMESIZE;
                        if (! RegQueryValue(hkeyClasses,
                                szExec,szName,&dwSize))
                        {
                            lstrcat( szClass," ");                 // Spacing
                            lstrcat( szClass,szName);
                            SendMessage(hWndList,LB_ADDSTRING,0,
                                (DWORD)(LPSTR)szClass);
                        }
                        RegCloseKey(hkeyTemp);
                    }
                    else
                    {
                        if (! RegOpenKey(hkeyClasses,szClass,&hkeyTemp))
                        {
                            dwSize = KEYNAMESIZE;
                            if (! RegQueryValue(hkeyClasses,szClass,szName,&dwSize))
                            {
                                RegCloseKey(hkeyTemp);
                                lstrcpy(szExec,szName);
                                lstrcat(szExec,"\\shell\\open\\command");
                                if (! RegOpenKey(hkeyClasses,szExec,&hkeyTemp))
                                {
                                    dwSize = KEYNAMESIZE;
```

```
                    if (! RegQueryValue(hkeyClasses,szExec,
                            szName,&dwSize))
                    {
                        lstrcat( szClass," ");                    // Spacing
                        lstrcat( szClass,szName);
                        SendMessage(hWndList,LB_ADDSTRING,0,
                                (DWORD)(LPSTR)szClass);
                    }
                }
            }
            RegCloseKey(hkeyTemp);
        }
    }
}
    break;
case IDM_EXT:                                    // Create key . C . h . RC . DEF
                                                 // and assign to PROG
    RegSetValue(HKEY_CLASSES_ROOT,
            ".C",                                                  // Key
            REG_SZ,
            "PROG",                                                // Value
            4);
    RegSetValue(HKEY_CLASSES_ROOT,
            ".h",                                                  // Key
            REG_SZ,
            "PROG",                                                // Value
            4);
    RegSetValue(HKEY_CLASSES_ROOT,
            ".RC",                                                 // Key
            REG_SZ,
            "PROG",                                                // Value
            4);
    RegSetValue(HKEY_CLASSES_ROOT,
            ".DEF",                                                // Key
            REG_SZ,
            "PROG",                                                // Value
            4);
    RegCreateKey(HKEY_CLASSES_ROOT,      // Create PROG\…\COMMAND key
            "PROG\\shell\\open\\command",                          // Key handle
            &hkeyTemp);
    RegSetValue(HKEY_CLASSES_ROOT,                    // Assign key value
            "PROG\\shell\\open\\command",                          // Key
            REG_SZ,
            "Notepad. exe %1",                                     // Value
            14);                                                   // Length of text
    RegCloseKey(hkeyTemp);                            // Write value on closing
    break;
case IDM_EXIT:
    SendMessage(hWnd,WM_SYSCOMMAND,SC_CLOSE,0L);
    break;
default:
```

```
                    return (DefWindowProc(hWnd,message,wParam,lParam));
                    break;
                }
                break;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            default:
                return (DefWindowProc( hWnd,message,wParam,lParam ));
                break;
        }
        return(0L);
}
```

模块定义文件:REGPROG. DEF

```
NAME Regprog
DESCRIPTION 'Registration and applications'
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'

CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 4096
STACKSIZE 5012

EXPORTS StdDlgWndProc @1
```

资源描述文件:REGPROG. RC

```
#include "regprog. h"
#include "windows. h"

STRINGTABLE
BEGIN
    IDS_ERR_WINDOW, "Window could not be created"
    IDS_ERR_CLASS, "Class could not be registered"
END

RegProgMenu MENU
BEGIN
    POPUP "&Registrations"
    BEGIN
        MENUITEM "&Show", IDM_SHOW
        MENUITEM "by &Extension",IDM_EXT
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_EXIT
    END

END
```

头文件:REGPROG. H

```
#include (windows. h)
#include (string. h)
#include (shellapi. h)

#define IDS_ERR_CLASS 20
#define IDS_ERR_WINDOW 21

#define IDM_EXT 100
#define IDM_SHOW 101
#define IDM_EXIT 102

char szString[128];                                    /* Variable for loading resource text */
char szAppName[20];                                    /* Window class name */
HANDLE hInst;

int nCwRegisterClasses(HANDLE);
long FAR PASCAL StdDlgWndProc( HWND,unsigned,WORD,LONG);
```

MAKE 文件:REGPROG. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

. rc. res: ; rc -r $ * . rc

RESFILES =
H = REGPROG. H
LIBS_EXT = shell. lib

all: REGPROG. EXE

REGPROG. RES: REGPROG. RC $ (RESFILES)

REGPROG. OBJ: REGPROG. C $ (H)

REGPROG. EXE: REGPROG. OBJ REGPROG. RES REGPROG. DEF
    link /NOD/A:16 REGPROG,REGPROG,nul,libw slibcew $ (LIBS_EXT),REGPROG. def
    rc REGPROG. res REGPROG. exe
```

运行 NMAKE 生成 REGPROG 应用程序:

```
nmake regprog. mak
```

23.5 生成新入口

在 WM_CREATE 消息期间生成的列表框填充了主窗口。选择 Extension 菜单项可启动带有参数 IDM_EXT 的 WM_COMMAND 消息,这将生成一个新的数据库入口。

```
case IDM_EXT:                                          // Create key .c .h .RC .DEF
                                                       // and assign to PROG
    RegSetValue(HKEY_CLASSES_ROOT,
            ".C",                                      // Key
            REG_SZ,
            "PROG",                                    // Value
            4);
    RegSetValue(HKEY_CLASSES_ROOT,
            ".h",                                      // Key
            REG_SZ,
            "PROG",                                    // Value
            4);
    RegSetValue(HKEY_CLASSES_ROOT,
            ".RC",                                     // Key
            REG_SZ,
            "PROG",                                    // Value
            4);
    RegSetValue(HKEY_CLASSES_ROOT,
            ".DEF",                                    // Key
            REG_SZ,
            "PROG",                                    // Value
            4);
    RegCreateKey(HKEY_CLASSES_ROOT,                    // Create PROG\…\COMMAND key
            "PROG\\shell\\open\\command",              // key handle
            &hkeyTemp) ;
    RegSetValue(HKEY_CLASSES_ROOT,                     // Assign key value
            "PROG\\shell\\open\\command",              // key
            REG_SZ,
            "Notepad.exe %1",                          // Value
            14);                                       // Length of text
    RegCloseKey(hKeyTemp);                             // Write value on closeing
    break;
```

RegSetValue 函数规定了键".c"等，并且指定了类名"PROG"，随之借助 RegCreateKey 生成了"Prog\\shell\\open\\command"键。

一旦生成了键，赋值就通过 RegSetValue 进行，并且在 Notepad 中拥有了程序名入口。附属参数%1 作为文件名的存放地址在程序开始运行时被传送给 Notepad。

查询赋值

为了获得所有扩展名和程序间的联系，考虑到下面两种情况是很重要的。

一种方法是使用如下格式：

"PROG\\SHELL\\OPEN\\COMMAND" = "PBrush.EXE %1"

和键入

".ext" = "PROG"

第二种方法是直接通过".ext\\shell\\open\\command"="PBrush.EXE %"的方式来实现联系。

第二十四章　QuickCase：W

这一章简单介绍各种开发工具，然后详细地讨论 Microsoft 软件包 QuickCase：W。

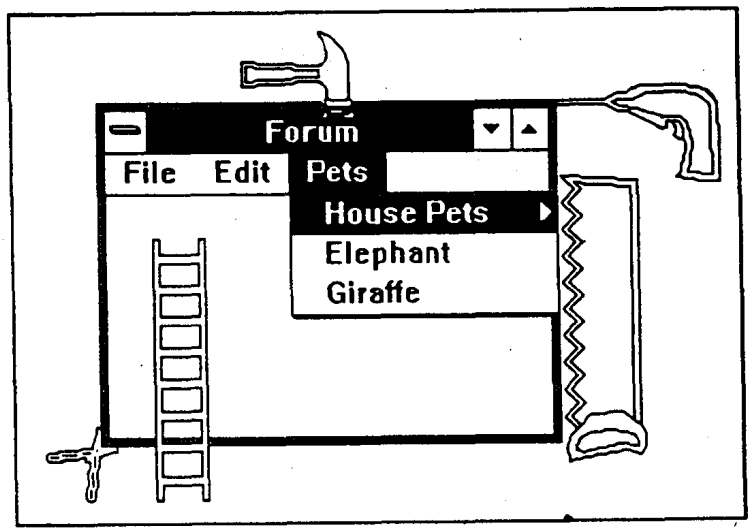


图 24.1

24.1　各种开发工具及其用途

欲创建 Windows 应用程序，不但要熟习 Windows 应用程序编程，还要精通 C 语言。考虑到这些前提，不久前，一位初学者感到编写第一个应用程序非常困难。很多初学者不久便感到厌烦并停止了编程。

如今，这种情况得以改善。程序员除了 C 语言外，还可以使用 Tubo Pascal（或 Visual Basic。另外还出现了高级的开发工具，如 ObjectVision，ToolBook 和 QucikCase：W，它们从不同角度来实现 Windows 应用程序。

ObjectVision

Borland International 的产品 ObjectVision 所使用的方法与其他应用程序完全不同。它首先建立好窗体的各个部分，然后再提供应用程序特有的属性。每一部分都可看成是一个有名字的对象。窗体使开发者不通过编程便可设置方便的入口屏蔽位（ObjectVision 通过使用判定树方法）。这棵树建立于由计算机求值的一系列条件的基础上。

ToolBook

Asymetrix ToolBook 也不需要开发者具备 C 语言编程知识，通过 TooBook 可访问或建立像按钮这样的预定义对象。由此工具开发的应用程序包含一页或多页，它们构成“一本书”。在只读模式下，这本书只能被读取；在授权模式下，可以修改它的对象。每一页包含多个对象，分配给这些对象的函数是用 ToolBook 特有的语言 OpenScript 编写的。

24.2 使用 QuickCase:W

QuickCase:W 在通用 Windows 编程方面做得更好，因为它的开发者不需要 C 语言知识。本书的其他章节都使用 C 语言，所以本章仍使用它。这一章对 QuickCase:W 作一个简略的描述。

每个较大的 Windows 应用程序都可分成两部分：一部分与 Windows 的图形用户界面(GUI)有关，另一部分则针对应用中的专门问题。QucikCase:W 也是用这种方法编写的。

24.2.1 概述

QucikCase:W 解决了 Windows 中的一些专门问题，从而减少了开发 GUI 信息的时间，给开发者以更多的时间来用 C 语言编写程序的主要函数。

QucikCase:W 有以下优点：

- QuickCase:W 有一个可从中调用源编辑器或其他工具的接口。
- Quickcase:W 中的 WYSIWYG 编辑器可用来建立窗口和菜单，也可用来把对话框链接到菜单项上。
- QuickCase:W 可以在菜单中插入代码，比如插入位图码等。
- 可以把自己的代码结合到 QuickCase:W 中。
- QuickCase:W 的代码生成器可建立所需转化的文件。

如果用 C 语言之外的语言来开发 Windows 程序，则需首先手工输入应用程序的 Windows 框架。在本书前面的章节中已遇到过这种情况。QuickCase:W 可以链接其他资源，还可绘制应用程序的窗口和菜单。

可以指定滚动条和颜色，以控制应用程序的外观。QuickCase:W 界面是作为嵌套窗口而建立的。外围窗口代表 QuickCase:W，里面的窗口代表应用程序。

一旦为应用程序绘制好了主窗口并赋予了恰当的属性，便可以用任意的名字保存这个窗口。QuickCase:W 以扩展名 .WIN 建立一个文件。每当用 QucikCase:W 生成代码时，结果文件的名字与 WIN 文件相同，但扩展名不同。

QuickCase:W 在生成的代码中插入注释，这些注释的程度由 Options/Comments…命令的设置来决定。在本章中，对于 LOW 设置和 High 设置，我们都要作一尝试。

开发工具 QuickCase:W 也有自身的不足，它不能减轻程序员必须读懂整个代码的负担(程序员只有如此，才能把 QuickCase:W 的框架与应用程序代码联成一体)。程序员只有通过编程才能处理窗口的内容，读对话框信息，向打印机发送信息，等等。

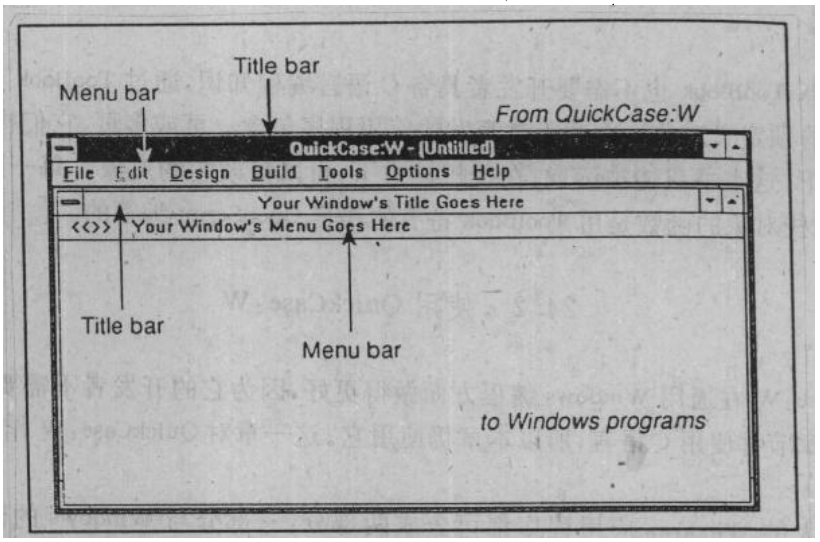


图 24.2 QuickCase:W

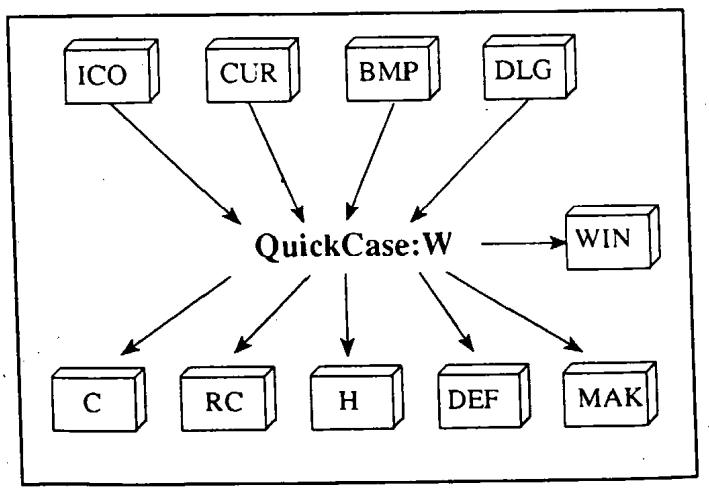


图 24.3 QuickCase:W 生成的文件的扩展名

24.2.2 建立原型

让我们来利用 QuickCase:W 建立一个 Windows 应用程序的原型，这个原型对于开始进行一项较大的软件工程很有帮助。因为有了它，就可减小用户与开发者之间的分歧和误解。

在工程开发时通常要建立一个约定规范的备忘录，但由于其内容可以用不同的方法解释，因此经常会导致严重的错误。

在 QuickCase:W 的帮助下建立一个原型，通常需要按下面的顺序执行这些步骤：

(1)与用户讨论应用程序的界面，然后再构思应用程序的界面。

(2)用相应的编辑器(例如 DLGEDIT 和 IMAGEDIT)建立对话框、光标和图标。

(3)用 QuickCase:W 画出窗口及其组件。

(4)用 QuickCase:W 生成程序代码。

(5)把应用程序中特有的代码加入到已产生的代码中去。

(6)如果需要则更新文件 QuickCase:W。

我们可以写一个很短的应用程序来实现上面的 1 到 4 步和第 6 步,我们将忽略掉第 5 步,因为本书的其他章节讨论了这个问题。

第 1 步:讨论并构思应用程序界面

在设计过程中,必须与用户协商,以决定显示哪一种界面。假设我们与用户在一个决定休息时间的应用程序 LEISURE 上达成协议,这个应用程序将包含一个带有两个菜单项的菜单条,其中一个菜单由位图菜单项构成,另一个菜单由文本菜单项构成,每个菜单项允许用户调用一个对话框(另一个对话框将在晚些时候加入到代码中)。应用程序还要有自己的图标、自己的光标和标准滚动条。

第 2 步:产生对话框和其他资源

资源开发工作(即对话框、位图等)应该在产生应用程序的主界面之前完成。一旦资源都建立好了,所需做的就只是用 QuickCase 把它们链接到应用程序中。

第 3 步:用 QuickCase:W 建立应用程序窗口

在这里才真正地开始了使用 QuickCase:W 的计算机软件工程。菜单 Design 对我们极其重要,如果想按顺序看一遍菜单项,使用它是最合适的。

• Style…

窗口风格的类型是指放置于窗口边界的元素,如标题条、系统菜单和框架本身等。所有可用的风格参数都列在对话框 Style 中,用户可在这里选择所需的参数。如果某个窗口没有标题条,则相应的复选框中的"X"将被删去:Minimize Button,Maximize Button,Menu Bar 和 System Menu。

图 24.4 说明了为激活复选框所设置的标志位。以字母"WS"开头的标志指的是 CreateWindows,而以字母"CS"开头的标志指的是 WNDCLASS 结构的风格。如果复选框 Initially Visible 没有被选中,则进入消息循环之前函数 ShowWindow 不会被调用。如果复选框 Menu Bar 被禁用,则 QuickCase:W 中窗口的菜单条都会消失。

• Title…

每个应用程序都应该有一个标题,由应用程序的名字或简短的描述构成。这个标题将显示在标题条和 TaskManager 中。

如果对话框 Style 中的复选框 Title Bar 保持被禁用状态,则即使输入了标题,也不会有标题显示出来。反之,复选框 Title Bar 为活动状态时,在文本被输入并被验证之后,将显示在标题的位置。在本应用程序中标题为"QuickCase:W Example"。

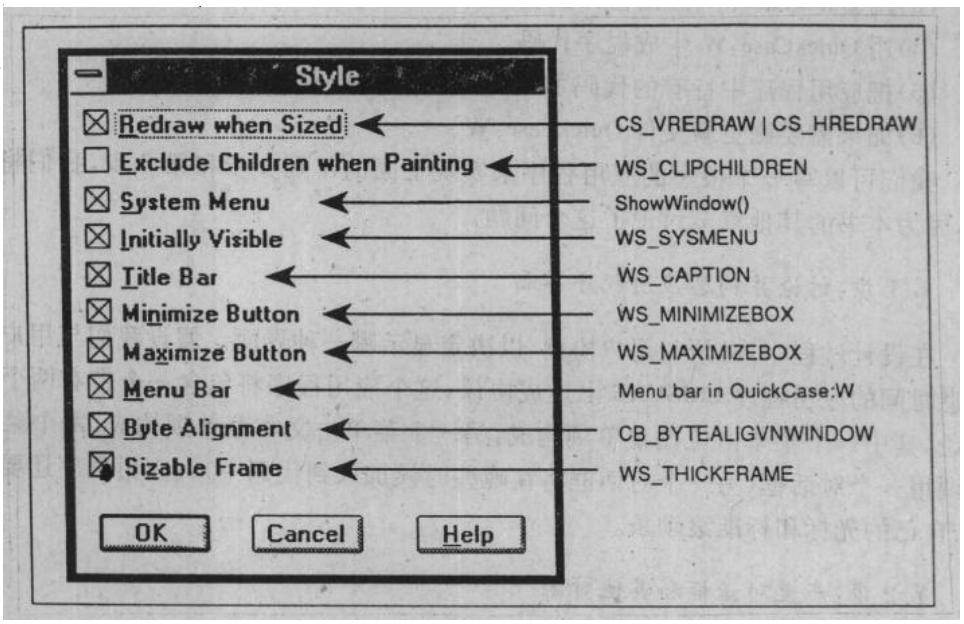


图 24.4　对话框风格

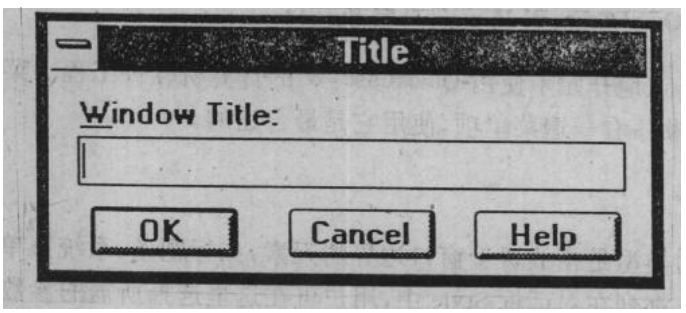


图 24.5　对话框的标题

• Icon…和 Cursor…

这两项允许把自己的鼠标光标和图标加入到 QuickCase:W 的代码中。如果用户没有定义自己的光标和图标，WNDCLASS 将指定缺省光标(IDC_ARROW)和缺省图标(IDI_APPLICATION)。

每个应用程序都应该有一个图标，而这些图标应尽可能地不同，以便用户更容易区分两个最小化的应用程序。

光标和图标分别被存放在以 .CUR 和 .ICO 为扩展名的文件中。一旦选中了菜单项 Icon 或 Cursor，对话框中将显示一个文件名，从这个文件中可选择所需的资源。

本应用程序既有用户自定义的光标(文件 PENCIL.CUR)，又有用户自定义的图标(文件 ABOOK.ICO)。根据这些设置，资源描述文件 LEISURE.RC 将包含下面两行：

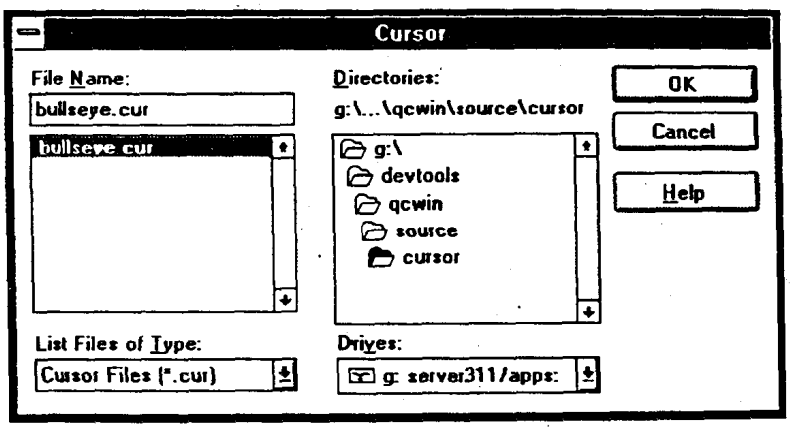


图 24.6　对话框 Cursor

LEISURE ICON ABOOK. ICON

PENCIL CURSOR PENCIL. CUR

QuickCase：W 自动地把项目中的图标名赋为资源中的图标名。因为 WIN 文件是以名字 LEISURE. WIN 保存的，图标相应地被称为 LEISURE。在登录类时，可以发现下面这样一个入口：

```
wndclass. hIcon = LoadIcon(hInst,"LEISURE");
```

这个登录过程的过程名为 nCwRegisterClass。

因为很多 Windows 应用程序中鼠标有多种图案，所以光标资源名与相应的资源文件名（而不是项目名）相似。与 WNDCLASS 中的光标不同，由 QuickCase：W 访问的鼠标被处理为运行时设置的光标，而不是在代码段中产生的光标。下面几行中，全程变量 hCur 被定义在头文件中：

```
int nCwRegisterClasses(void)
{
    WNDCLASS wndclass;
    ⋮
    wndclass. hcursor = NULL;
    ⋮
}

⋮
LONG FAR PASCAL wndProc(HWND hWnd,WORD Message,WORD wParam,LONG lParam)
{
    ⋮
    switch (Message)
    {
        ⋮
        case WM_CREATE:
            ⋮
            hCur = LoadCursor(hLinst,"PENCIL");
            break;
```

```
        case WM_MOUSEMOVE:
            SetCursor(hCur);
            break;
    }
```

• Colors…

大多数应用程序在显示黑色文本或彩色图形时，使用的是白底色。菜单项 Colors 可以改变这种设置。它的对话框有三个滚动条，用以组合三种基本 RGB(红、绿、蓝)颜色。对话框上部的单选按钮指定所选的颜色为前景色还是背景色。

要把窗口的底色设置为黄色，只需把蓝色成分设置为 0，而把另两种颜色成分设置为最大值。

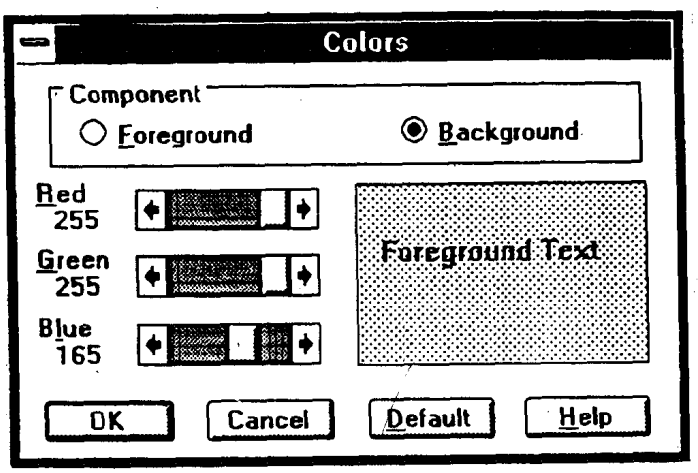


图 24.7　对话框的颜色

在源代码中，所期望的背景颜色被赋值给 WNDCLASS 中的 hbrBackground 域。

```
int nCwRegisterClasses(void)
{
    WNDCLASS wndclass;
    wndclass.hbrBackground = CreateSolidBrush(
            RGN(255,255,0));
    ⋮
}
```

• Scroll Bars…

滚动条使得应用程序可以显示窗口所不能完全容纳的大量文本。可以给定垂直滚动条或水平滚动条，也可以同时指定两种滚动条，并且可以设置滚动范围。QucikCase:W 把这些范围的值传给函数 SetScrollRange。

在这里同时指定了水平和垂直滚动条。垂直滚动条的范围为 10，水平滚动条的范围为 20。QuickCase:W 在函数 CreateWindow 中有两种窗口风格类型:WS_HSCROLL 和 WM_VSCROLL。在消息 WM_CREATE 中设置了滚动区域，这样便产生了用来控制滚动条的滚动条框架。

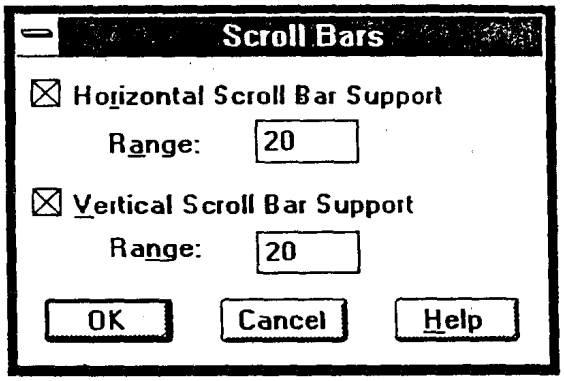


图 24.8 对话框 Scroll Bars

```
case WM_CREATE:
    SetScrollRange(hWnd,SB_HORZ,0,20,FALSE);
    sHPos = 0;                                    /* scroll bar is initially set to 0 */
    SetScrollRange(hWnd,SB_VERT,0,10,PALSE);
    sVPos - 0;                                    /* scroll bar is initially 0 */
     :
    break;
case WM_HSCROLL:
    swith (wParam)
    {
        case SB_LINEDOWN:
            sHPos = min(20,sHPos + 1);
            break;
        case SB_LINEUP:
            sHPos = max(0,sHPos -1);
            break;
        case SB_PAGEUP:
            sHPos -=-20;
            break;
        case SB_PAGEDOWN:
            sHPos += 20;
            break;
        default:
            return FALSE;
    }
    SetScrollPos(hWnd,SB_HORZ,sHPos,TRUE);
    break;
```

与消息 WM_VSCROLL 相应的变量是 sVPos,它每次增减的值为 10。

• Size and Location…

大多数应用程序可在 Windows 中设置其在屏幕上的尺寸和位置。在这里,函数 CreateWindow 和常量 CW_USEDEFAULT 实现此功能,但是用户也可以自己设置这些值。如果想把窗口设置为最大尺寸(Maximized 模式),只需选择单选按钮 Maximized 即可。若想把

窗口按最小尺寸显示(Minimized 模式),则选择单选按钮 Minimized。

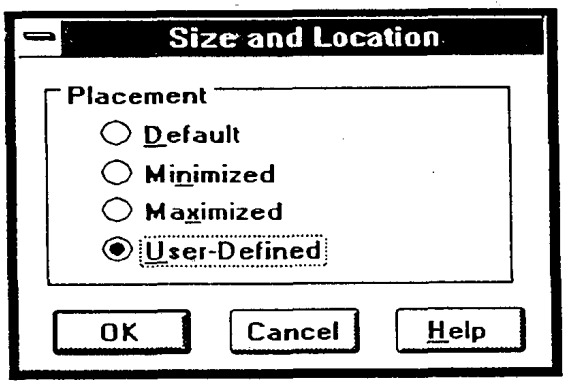


图 24.9 对话框的尺寸和位置

若选中了单选按钮 UserDefined 并按下了按钮 OK,将显示一个消息框,让用户输入窗口的尺寸和位置。从菜单 System 中选择 Close 将取消这个设置。

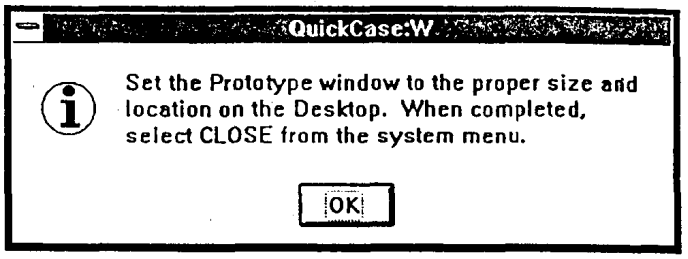


图 24.10 改变窗口尺寸和位置的消息框

本例把应用程序的窗口放在屏幕的左上角。为保证每次运行此程序时窗口都在选定的位置上,需进行下面的计算:

```
nWndunits = GetDialogBaseUnits ();
nWndx = LOWORD(nWndunits);
nWndy = HIWORD(nWndunits);
nX = ((0 * nWndx) / 4);
nY = ((0 * nWndy) / 8);
nWidth = ((185 * nWndx) / 4);
nHeight = ((125 * nWndy) / 8);
```

然后,nX、nY、nWidth 和 nHeight 被传送到函数 hCreateWindow 中。

• Unlinked Dialogs…

未链接的对话框是通过程序而非选择菜单项来调用的。在本应用程序中,对话框允许通过一定的计算而让用户输入 age 和 sport 的位置。

菜单项 Unlinked Dialogs 允许通过选中所需的对话框来声明对话框,也可通过按下按钮 Add 分别把它们移到对话框 Selected Dialogs 中。

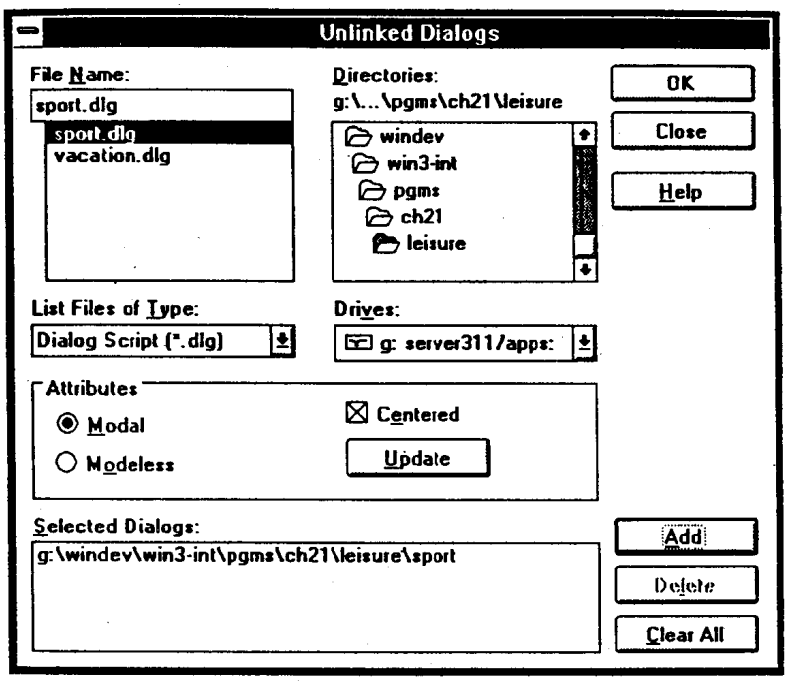


图 24.11　对话框 Unlinked Dialogs

由于对话框 Unlinked Dialogs 是非模态对话框，而这是一种不能任意定位的对话框，因此在这里要在 Attributes 组中选择 Modeless 单选按钮，并且屏蔽掉复选框 Centered。按下按钮 Add，然后按下按钮 OK。当生成了程序代码之后，QuickCase:W 为每个未链接的对话框建立一个对话框例程，并在过程 WinMain 中加入激活这些对话框所必需的代码，但这些代码并不被执行。

```
#if 0
    lpfnSPORTMsgProc = MakeProcInstance((FARPROC)SPORTMsgProc,hInst);
    hDlgSPORT = CreateDialoghInst,"SPORT",hWnd,lpfnSPORTMsgProc);
#endif
```

由于使用的是非模态对话框，QuickCase:W 还将自动修改消息循环。对话框 hDlgSPORT 在头文件中被定义为全程变量。

```
while(GetMessage(&msg,NULL,0,0)
{
    if(hDlgSPORT && IsDialogMessage(hDlgSPORT,&MSG))
        continue;

    TranslateMessage(& %msg);
    DispatchMessage(&msg);
}
```

• 菜单

下一步任务是产生菜单。在 QuickCase:W 中，只要在窗口菜单条中的〈〉字符上双击鼠标按钮便可建立菜单和菜单项。

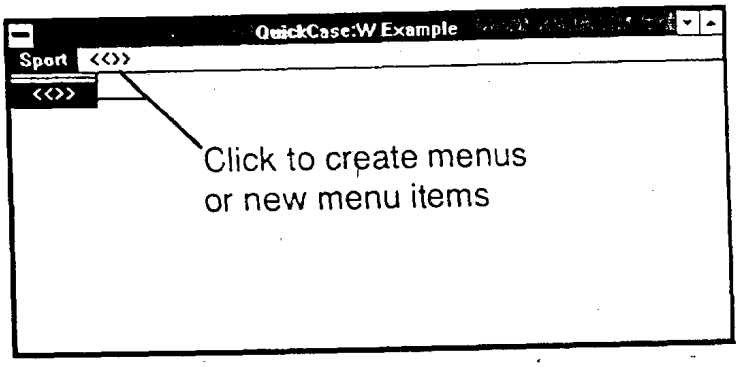


图 24.12　定义菜单

双击鼠标按钮后，将出现一个对话框，提示用户输入菜单标题或菜单项。因为在菜单条中不可能把位图也当作菜单项对待，所以在开始定义菜单时，单选按钮被屏蔽掉了。进一步，概念 Dropdown Menu 被当作 Link to…选项中的一个选择，以保证在菜单条中只能建立菜单标题，这与前面讲过的 CUA 相似。

一旦在文本框“Name：”中输入了所需的文本（例如“Sport”）并按下了按钮 OK，在应用程序中将出现新入口。然后可以先在菜单条中加入下一个菜单标题，或在标题 Sport 下加入一个菜单项。两个区域中都将显示字符〈〉。

如果想在菜单中放置多个入口，可以插入下一个入口（一个字符串），也可以插入一个用来分隔菜单项的横线，还可以不插入文本菜单项而插入一个位图。选中 Bitmap 单选按钮将显示另一个对话框，从这里可以选择一个位图。QuickCase 不能在其开发环境中显示位图。位图的名字被当作菜单项的名字（例如，SPORT1 代表访问位图 SPORT1. BMP 的菜单项）。

可以给每个菜单项指定下面的四个操作之一：

- 空操作：None
- 下拉式菜单：Dropdown Menu
- 对话框：Dialog Box
- 自定义代码：User Defined Code

选择 None 后，还要在以后把菜单项的代码加到程序中去。选择 Dropdown Menu 将产生一个菜单，在菜单中可加入其他的入口。选择 Dialog Box 并按下 Configue Link 按钮将显示另一个对话框，此对话框与未链接对话框相似。只有这里产生的对话框才永久地依附于指定的菜单项。最后一个选项可以建立起现有的菜单代码与菜单项的联系，这些代码存于扩展名为 .INC 的 ASCII 文件中。当几个开发者共同设计一个大的项目时，这个选项特别有用。

- 组合键

对每个菜单项还可以指定一个组合键，存储在文本框“Acelerator”中。组合键可以用字符〈+〉联起来（例如 Ctrl+W）。当按下按钮 OK 时，QuickCase：W 将检查是否允许这个项。

- 位图用作菜单项

在本例中，菜单 Sport 包含四个菜单项，它们都被指定为位图。BMP 文件（SPORT1. MBP，

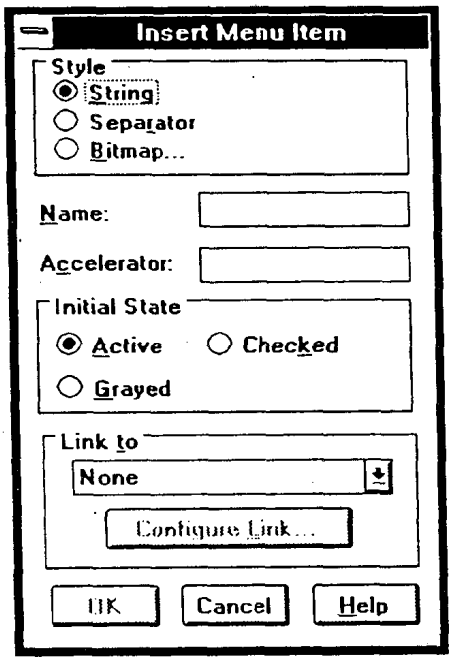


图 24.13　未链接对话框

SPORT2.BMP,SPORTS3.BMP 和 SPORT4.BMP)必须和其他的项目文件放在同一个目录中。

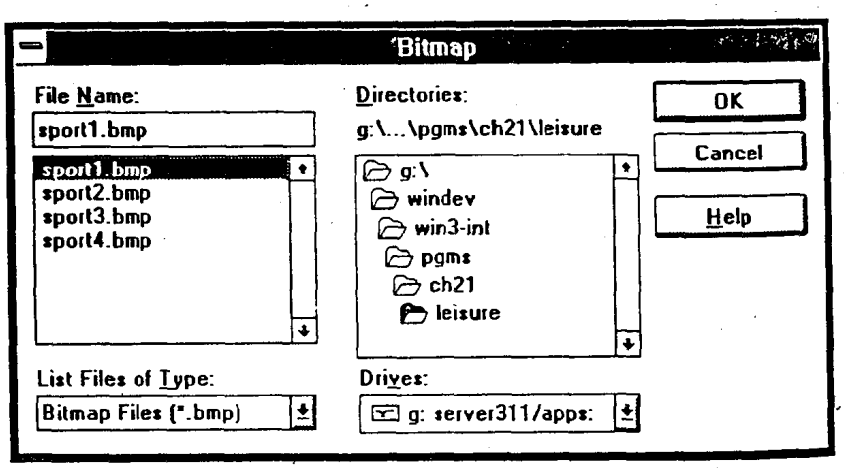


图 24.14　用位图作菜单项

菜单 Vacation 上有菜单项 Sport Type…和 Location…。应用程序可以访问对话框 Location,但不能访问对话框 Sport Type(访问此对话框需要附加的程序,这将留在以后讨论)。运行应用程序 LEISURE.EXE 可以看到这些是怎样执行的。

现在看一下刚才的设置将怎样影响源代码。每个菜单项的 ID 值在头文件中被赋以有意义的名字,这些名字总是以字母 IDM 开头,后跟一个下划线和菜单标题的首字母,然后又是

一个下划线，最后是菜单项的文本。

```
POPUP "&Vacation"
BEGIN
    MENUITEM "&Sport Type…\tCtrl+S",IDM_V_SPORTTYPE
    MENUITEM "&lOCATION…\tCtrl+L",IDM_V_LOCATION
END
```

位图通过消息 WM_CREATE 被指派给子菜单。在用 QuickCase:W 开发程序时，位图与光标一样接受同名的相应文件，但没有扩展名 .BMP。

```
case WM_CREATE:
    ⋮
    hBitmap = LoadBitmap(hInst,"STORT1);
    ModifyMenu(hMenu,1050,MF_BYCOMMAND|MF_BITMAP,
        1050,(LPSTR)MAKELONG(hBitmap,0));
    hBitmap = LoadBitmap(hInst,"STORT2);
    ModifyMenu(hMenu,1100,MF_BYCOMMAND|MF_BITMAP,
        1100,(LPSTR)MAKELONG(hBitmap,0));
    hBitmap = LoadBitmap(hInst,"STORT3);
    ModifyMenu(hMenu,1150,MF_BYCOMMAND|MF_BITMAP,
        1150,(LPSTR)MAKELONG(hBitmap,0));
    hBitmap = LoadBitmap(hInst,"STORT4);
    ModifyMenu(hMenu,1200,MF_BYCOMMAND|MF_BITMAP,
        1200,(LPSTR)MAKELONG(hBitmap,0));
```

因为菜单项 Location…将激活一个对话框，所以过程实例的地址用相应的 ID 值确定下来，对话框于是可以被调用了，然后释放掉这个地址。

```
case WM_COMMAND:
    switch (wparam)
    {
        case IDM_V_LOCATION:
        {
            FARPROC lpfnVACATIONMsgProc;

            lpfnVACATIONMsgProc = MakeProcInstance((FARPROC) VACATIONMsgProc,hInst);
            nRc = DialogBox(hInst,MAKEINTRESOURCE(100),nWnd,lpfnVACATIONMsgProc);
        }
        break;
```

不管是未链接的对话框还是已链到例程上的对话框，所有对话框的例程名都是以不带扩展名的对话框文件名命名的，后面以“MsgProc”结尾。

当我们给菜单项 Location…分配对话框 VACATION. DLG 时，便指定了这个对话框放在屏幕中央。在这种情况下，对话例程 LOCATIONMsgProc 通过消息 WM_INITDIALOG 调用子例程 cwCenter。

```
    case WM_INITDIALOG:
        cwCenter(hWndDlg),0);
        break;
```

这个子例程首先计算对话框(hWnd)的尺寸和主窗口(hWndMain)的尺寸。因为函数 GetWindowRect 返回屏幕坐标，所以传来的客户区坐标必须被重新计算。只有这样才能根据函数 cwCenter 的第二个参数计算起始点的坐标，并调整 y 坐标的值。最后，函数 MoveWindow 将对话框移到计算好的位置上。

```
void cwCenter(hWnd,top)
HWND hWnd;
int top;
{
    POINT pc;
    RECT swp;
    RECT rParent;
    int iwidth;
    int iheight;

    GetWindowRect(hWnd,&swp);
    GetClientRct(hWndMain,&rParent);
    iwidth = swp.right - swp.left;
    iheight = swp.bottom - swp.top;
    pt.x = (rParent.right - rParent.left) / 2;
    pt.y = (rParent.bottom - rParent.top) /2;
    ClientToScreen(hWndMain,&pt);
    pt.x = pt.x - (iwidth / 2);
    pt.y = pt.y - (iheight / 2);
    if(top)
        pt.y = pt.y + top;
    MoveWindow(hWnd,pt.x,pt.y,iwidth,iheight,FALSE);
}
```

因为两个 Vacation 菜单项将被组合键激活，所以在文本框“Accelerator”中放入了组合键。QuickCase:W 在资源描述文件 LEISURE.RC 中建立了一个加速键表，并在函数 WinMain 中扩展了消息循环。

// LEISURE.RC:

```
LEISURE ACCELERATORS
BEGIN
    "^S",IDM_V_SPORTTYPE
    "^1",IDM_V_LOCATION
END
```

// LEISURE.C:

```
hAccel = loadAccelerators(hInst,szAppName);
while(GetMessage(&msg,NULL,0,0)
{
    ⋮
    if(TranslateAccelerator(hWndMain,hAccel,&msg))
        continue;
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

第 4 步：产生程序代码

现在已建立好了接口，我们需要用 QuickCase:W 来产生可编译的 C 代码。以 LEISURE 为名保存好原型。QuickCase:W 将以扩展名 WIN 建立一个文件，然后选中菜单项 Build/Generate。建立的所有文件都将以原型中的名字命名。

在建立代码的过程中，窗口 Generate 将以最小模式显示(即显示为图标)。在这个图标上

双击鼠标按钮以观察生成的代码，如果愿意，还可以让 QuickCase:W 在机器上运行，自己去做别的事情。当 QuickCase:W 建立好源代码之后，计算机将发出蜂鸣声，并且显示一个消息框。

也可以从 Microsoft Quick 的编辑器观察这些源程序代码。

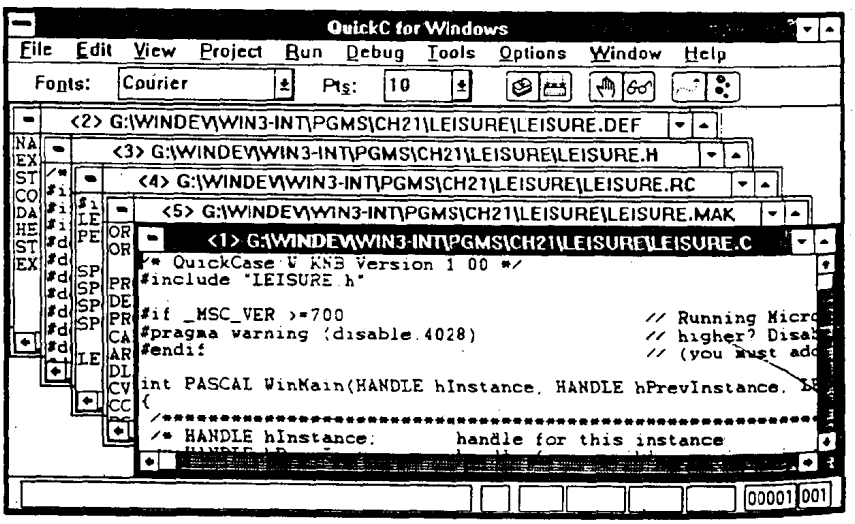


图 24.15 LEISURE 所生成的文件

在这个例子中，我们选择 Low 注释级。这些注释将使应用程序比所需的长，但可给用户和程序员提供更清楚的注释。本章的其余部分列举了这些程序。

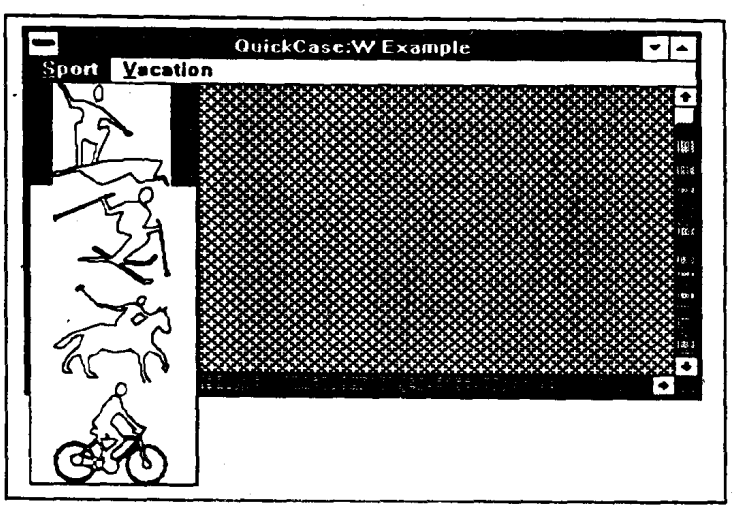


图 24.16 原型的显示

源代码:LEISURE. C

```
/* QuickCase:W KNB Version 1.00 */
#include "LEISURE.h"

#if _MSC_VER >=700                                    // Running Microsoft C 7.00 or
#pragma warning (disable:4028)                        // higher? Disable warning 4028
#endif                                                   // (you must add this code)

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpszCmdLine,int nCmdShow)
{
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* HANDLE hInstance; handle for this instance */
/* HANDLE hPrevInstance; handle for possible previous instances */
/* LPSTR lpszCmdLine; long pointer to exec command line */
/* int nCmdShow; Show code for main window display */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

MSG msg;                                          /* MSG structure to store your messages */
int nRc;                                            /* return value from Register Classes */
long nWndunits;                                     /* window units for size and location */
int nWndx;                                                        /* the x axis multiplier */
int nWndy;                                                        /* the y axis multiplier */
int nX;                                           /* the resulting starting point (x,y) */
int nY;
int nWidth;                                       /* the resulting width and height for this */
int nHeight;                                                                  /* window */

strcpy(szAppName,"LEISURE");
hInst = hInstance;
if(! hPrevInstance)
{
    /* register window classes if first instance of application */
    if ((nRc = nCwRegisterClasses()) == -1)
    {
        /* registering one of the windows failed */
        LoadString(hInst,IDS_ERR_REGISTER_CLASS,szString,sizeof(szString));
        MessageBox(NULL,szString,NULL,MB_ICONEXCLAMATION);
        return nRc;
    }
}

/* Create a device independant size and location */
nWndunits = GetDialogBaseUnits();
nWndx = LOWORD(nWndunits);
nWndy = HIWORD(nWndunits);
nX = ((36 * nWndx) / 4);
nY = ((55 * nWndy) / 8);
nWidth = ((212 * nWndx) / 4);
nHeight = ((134 * nWndy) / 8);
```

```
/* create application's Main window */
hWndMain = CreateWindow(
        szAppName,                                          /* Window class name */
        "QuickCase:W Example",                              /* Window's title */
        WS_CAPTION |                                        /* Title and Min/Max */
        WS_SYSMENU |                                        /* Add system menu box */
        WS_MINIMIZEBOX |                                    /* Add minimize box */
        WS_MAXIMIZEBOX |                                    /* Add maximize box */
        WS_THICKFRAME |                                     /* thick sizeable frame */
        WS_VSCROLL |                                        /* add vertical scroll bar */
        WS_HSCROLL |                                        /* add horizontal scroll bar */
        WS_CLIPCHILDREN |                                   /* don't draw in child windows areas */
        WS_OVERLAPPED,
        nX,nY,                                              /* X,Y */
        nWidth,nHeight,                                     /* Width,Height of window */
        NULL,                                               /* Parent window's handle */
        NULL,                                               /* Default to Class Menu */
        hInst,                                              /* Instance of window */
        NULL);                                              /* Create struct for WM_CREATE */

if(hWndMain == NULL)
{
    LoadString(hInst,IDS_ERR_CREATE_WINDOW,szString,sizeof(szString));
    MessageBox(NULL,szString,NULL,MB_ICONEXCLAMATION);
    return IDS_ERR_CREATE_WINDOW;
}

ShowWindow(hWndMain,nCmdShow);                              /* display main window */

hAccel = LoadAccelerators(hInst,szAppName);

while(GetMessage(&msg,NULL,0,0))                            /* Until WM_QUIT message */
{
    if(hDlgSPORT && IsDialogMessage(hDlgSPORT,&msg))
        continue;

    if(TranslateAccelerator(hWndMain,hAccel,&msg))
        continue;
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

/* Do clean up before exiting from the application */
CwUnRegisterClasses();
return msg.wParam;
}                                                           /* End of WinMain */


/* The following Dialog box calls are generated for the dialog boxes */
/* that have been selected as Unlinked dialog boxes. To use these */
/* calls,simply copy them to the appropriate location in the */
```

```
/* application. */

#if 0

    /* Create procedure handle,create nonmodal dialog */
    lpfnSPORTMsgProc = MakeProcInstance((FARPROC)SPORTMsgProc,hInst);
    hDlgSPORT = CreateDialog(hInst,"SPORT",hWnd,lpfnSPORTMsgProc);
#endif

/* * * * * * * * * * * * * * * * * * * * * * * * */
/*                                               */
/* Main Window Procedure                         */
/*                                               */
/* This procedure provides service routines for the Windows events */
/* (messages) that Windows sends to the window,as well as the user */
/* initiated events (messages) that are generated when the user selects */
/* the action bar and pulldown menu controls or the corresponding */
/* keyboard accelerators.                        */
/*                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * */

LONG FAR PASCAL WndProc(HWND hWnd,WORD Message,WORD wParam,LONG lParam)
{
HMENU hMenu=0;                                  /* handle for the menu */
HBITMAP hBitmap=0;                              /* handle for bitmaps */
HDC hDC;                                        /* handle for the display device */
PAINTSTRUCT ps;                                 /* holds PAINT information */
int nRc=0;                                      /* return code */
static WORD sVPos;                              /* Position of Vertical scrollbar slider */
static WORD sHPos;                              /* Position of Horizontal scrollbar slider */

switch (Message)
{
  case WM_COMMAND:
    /* The Windows messages for action bar and pulldown menu items */
    /* are processed here.                       */
    switch (wParam)
    {
    case IDM_S_SPORT1:
      /* Place User Code to respond to the */
      /* Menu Item Named "SPORT1" here.   */
      break;
    case IDM_S_SPORT2:
      /* Place User Code to respond to the */
      /* Menu Item Named "SPORT2" here.   */
      break;
    case IDM_S_SPORT3:
      /* Place User Code to respond to the */
      /* Menu Item Named "SPORT3" here.   */
      break;
    case IDM_S_SPORT4:
```

```
        /* Place User Code to respond to the */
        /* Menu Item Named "SPORT4" here. */
        break;
      case IDM_V_SPORTTYPE:
        /* Place User Code to respond to the */
        /* Menu Item Named "&Sport Type..." here. */
        break;
      case IDM_V_LOCATION:
        /* Place User Code to respond to the */
        /* Menu Item Named "&Location..." here. */
        {
          FARPROC lpfnVACATIONMsgProc;
          lpfnVACATIONMsgProc = MakeProcInstance((FARPROC)VACATIONMsgProc,hInst);
          nRc = DialogBox(hInst,MAKEINTRESOURCE(100),hWnd,lpfnVACATIONMsgProc);
          FreeProcInstance(lpfnVACATIONMsgProc);
        }
        break;
      default:
        return DefWindowProc(hWnd,Message,wParam,lParam);
      }
      break;                                          /* End of WM_COMMAND */
    case WM_CREATE:
      SetScrollRange(hWnd,SB_HORZ,0,20,FALSE);
      sHPos = 0;                                      /* scroll bar is initially set to 0 */
      SetScrollRange(hWnd,SB_VERT,0,10,FALSE);
      sVPos = 0;                                      /* scroll bar is initially 0 */
      hCur = LoadCursor(hInst,"PENCIL");
      hMenu = GetMenu(hWnd);
      hBitmap = LoadBitmap(hInst,"SPORT1");
      ModifyMenu(hMenu,1050,MF_BYCOMMAND | MF_BITMAP,
              1050,(LPSTR)MAKELONG(hBitmap,0));
      hBitmap = LoadBitmap(hInst,"SPORT2");
      ModifyMenu(hMenu,1100,MF_BYCOMMAND | MF_BITMAP,
              1100,(LPSTR)MAKELONG(hBitmap,0));
      hBitmap = LoadBitmap(hInst,"SPORT3");
      ModifyMenu(hMenu,1150,MF_BYCOMMAND | MF_BITMAP,
              1150,(LPSTR)MAKELONG(hBitmap,0));
      hBitmap = LoadBitmap(hInst,"SPORT4");
      ModifyMenu(hMenu,1200,MF_BYCOMMAND | MF_BITMAP,
              1200,(LPSTR)MAKELONG(hBitmap,0));
      break;                                          /* End of WM_CREATE */
    case WM_MOUSEMOVE:
      SetCursor(hCur);
      break;
    case WM_MOVE:                                     /* code for moving the window */
      break;
    case WM_SIZE:                                     /* code for sizing client area */
      break;                                          /* End of WM_SIZE */
    case WM_PAINT:                                    /* code for the window's client area */
      /* Obtain a handle to the device context */
      /* BeginPaint will sends WM_ERASEBKGND if appropriate */
```

```
    memset(&ps,0x00,sizeof(PAINTSTRUCT));
    hDC = BeginPaint(hWnd,&ps);

    /* Included in case the background is not a pure color */
    SetBkMode(hDC,TRANSPARENT);

    /* Inform Windows painting is complete */
    EndPaint(hWnd,&ps);
    break;                                          /* End of WM_PAINT */
  case WM_VSCROLL:
    switch(wParam)
    {
      case SB_LINEDOWN:
           sVPos = min(10,sVPos + 1);
           break;
      case SB_LINEUP:
           sVPos = max(0,sVPos - 1);
           break;
      case SB_THUMBPOSITION:
           sVPos = LOWORD(lParam);
           break;
      case SB_PAGEUP:
           sVPos -= 10;
           break;
      case SB_PAGEDOWN:
           sVPos += 10;
           break;
      default:
           return FALSE;
    }
    SetScrollPos(hWnd,SB_VERT,sVPos,TRUE);
    break;
  case WM_HSCROLL:
    switch (wParam)
    {
      case SB_LINEDOWN:
           sHPos = min(20,sHPos + 1);
           break;
      case SB_LINEUP:
           sHPos = max(0,sHPos - 1);
           break;
      case SB_THUMBPOSITION:
           sHPos = LOWORD(lParam);
           break;
      case SB_PAGEUP:
           sHPos -= 20;
           break;
      case SB_PAGEDOWN:
           sHPos += 20;
           break;
      default:
```

```
            return FALSE;
        }
        SetScrollPos(hWnd,SB_HORZ,sHPos,TRUE);
        break;
    case WM_CLOSE:                                          /* close the window */
        /* Destroy child windows,modeless dialogs,then,this window */
        DestroyWindow(hWnd);
        if (hWnd == hWndMain)
            PostQuitMessage(0);                             /* Quit the application */
        break;
    default:
        /* For any message for which you don't specifically provide a */
        /* service routine,you should return the message to Windows */
        /* for default message processing. */
        return DefWindowProc(hWnd,Message,wParam,lParam);
}
return 0L;
}                                                           /* End of WndProc */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                        */
/* Dialog Window Procedure                                */
/*                                                        */
/* This procedure is associated with the dialog box that is included in */
/* the function name of the procedure. It provides the service routines */
/* for the events (messages) that occur because the end user operates */
/* one of the dialog box's buttons,entry fields,or controls. */
/*                                                        */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */

BOOL FAR PASCAL VACATIONMsgProc(HWND hWndDlg,WORD Message,WORD wParam,
                                LONG lParam)
{

switch(Message)
{
    case WM_INITDIALOG:
        cwCenter(hWndDlg,0);
        /* initialize working variables */
        break;                                              /* End of WM_INITDIALOG */
    case WM_CLOSE:
        /* Closing the Dialog behaves the same as Cancel */
        PostMessage(hWndDlg,WM_COMMAND,IDCANCEL,0L);
        break;                                              /* End of WM_CLOSE */
    case WM_COMMAND:
        switch(wParam)
        {
            case 102:                                       /* Edit Control */
                break;
            case IDOK:
                EndDialog(hWndDlg,TRUE);
```

```
                break;
            case IDCANCEL:
                /* Ignore data values entered into the controls */
                /* and dismiss the dialog window returning FALSE */
                EndDialog(hWndDlg,FALSE);
                break;
        }
        break;                                          /* End of WM_COMMAND */
    default:
        return FALSE;
    }
    return TRUE;
}                                                       /* End of VACATIONMsgProc */

/* * * * * * * * * * * * * * * * * * * * * * * * * * */
/*  */
/* Dialog Window Procedure */
/*  */
/* This procedure is associated with the dialog box that is included in */
/* the function name of the procedure. It provides the service routines */
/* for the events (messages) that occur because the end user operates */
/* one of the dialog box's buttons,entry fields,or controls. */
/*  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * */

BOOL FAR PASCAL SPORTMsgProc(HWND hWndDlg,WORD Message,WORD wParam,LONG lParam)
{

switch(Message)
{
    case WM_INITDIALOG:
        cwCenter(hWndDlg,0);
        /* initialize working variables */
        break;                                          /* End of WM_INITDIALOG */
    case WM_CLOSE:
        /* Closing the Dialog behaves the same as Cancel */
        PostMessage(hWndDlg,WM_COMMAND,IDCANCEL,0L);
        break;                                          /* End of WM_CLOSE */
    case WM_COMMAND:
        switch(wParam)
        {
            case 102:                                   /* Radiobutton text: "20-30" */
                break;
            case 103:                                   /* Radiobutton text: "31-40" */
                break;
            case 104:                                   /* Radiobutton text: "41-50" */
                break;
            case 105:                                   /* Radiobutton text: "51-99" */
                break;
            case 107:                                   /* Checkbox text: "Water" */
                break;
```

```
            case 108:                                          /* Checkbox text: "Air" */
                break;
            case 109:                                          /* Checkbox text: "Land" */
                break;
            case IDOK:
                DestroyWindow(hWndDlg);
                hDlgSPORT = 0;
                FreeProcInstance(lpfnSPORTMsgProc);
                break;
            case IDCANCEL:
                /* Ignore data values entered into the controls */
                /* and dismiss the dialog window returning FALSE */
                DestroyWindow(hWndDlg);
                hDlgSPORT = 0;
                FreeProcInstance(lpfnSPORTMsgProc);
                break;
        }
        break;                                                 /* End of WM_COMMAND */
    default:
        return FALSE;
}
return TRUE;
}                                                              /* End of SPORTMsgProc */


/*************************************/
/*                                   */
/* nCwRegisterClasses Function       */
/*                                   */
/* The following function registers all the classes of all the windows */
/* associated with this application. The function returns an error code */
/* if unsuccessful, otherwise it returns 0.                          */
/*                                   */
/*************************************/

int nCwRegisterClasses(void)
{
WNDCLASS wndclass;                                             /* struct to define a window class */
memset(&wndclass, 0x00, sizeof(WNDCLASS));


/* load WNDCLASS with window's characteristics */
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_BYTEALIGNWINDOW;
wndclass.lpfnWndProc = WndProc;
/* Extra storage for Class and Window objects */
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInst;
wndclass.hIcon = LoadIcon(hInst,"LEISURE");
wndclass.hCursor = NULL;                                       /* so cursor may be changed */
/* Create brush for erasing background */
```

```
wndclass.hbrBackground = CreateSolidBrush(RGB(255,255,0));
wndclass.lpszMenuName = szAppName;                    /* Menu Name is App Name */
wndclass.lpszClassName = szAppName;                   /* Class Name is App Name */
if(! RegisterClass(&wndclass))
    return -1;

return(0);
}                                                     /* End of nCwRegisterClasses */

/* * * * * * * * * * * * * * * * * * * * * * * * * */
/* cwCenter Function */
/*  */
/* centers a window based on the client area of its parent */
/*  */
/* * * * * * * * * * * * * * * * * * * * * * * * * */

void cwCenter(hWnd,top)
HWND hWnd;
int top;
{
POINT pt;
RECT swp;
RECT rParent;
int iwidth;
int iheight;

/* get the rectangles for the parent and the child */
GetWindowRect(hWnd,&swp);
GetClientRect(hWndMain,&rParent);

/* calculate the height and width for MoveWindow */
iwidth = swp.right - swp.left;
iheight = swp.bottom - swp.top;

/* find the center point and convert to screen coordinates */
pt.x = (rParent.right - rParent.left) / 2;
pt.y = (rParent.bottom - rParent.top) / 2;
ClientToScreen(hWndMain,&pt);

/* calculate the new x,y starting point */
pt.x = pt.x - (iwidth / 2);
pt.y = pt.y - (iheight / 2);

/* top will adjust the window position,up or down */
if(top)
    pt.y = pt.y + top;

/* move the window */
MoveWindow(hWnd,pt.x,pt.y,iwidth,iheight,FALSE);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * */
/* CwUnRegisterClasses Function */
/*  */
/* Deletes any refrences to windows resources created for this */
/* application,frees memory,deletes instance,handles and does */
/* clean up prior to exiting the window */
/*  */
/* * * * * * * * * * * * * * * * * * * * * * * * * */

void CwUnRegisterClasses(void)
{
WNDCLASS wndclass;                                    /* struct to define a window class */
memset(&wndclass,0x00,sizeof(WNDCLASS));

GetClassInfo(hInst,szAppName,&wndclass);
DeleteObject(wndclass. hbrBackground);
UnregisterClass(szAppName,hInst);
}                                                     /* End of CwUnRegisterClasses */
```

模块定义文件:LEISURE. DEF

```
NAME LEISURE
EXETYPE WINDOWS
STUB 'WINSTUB. EXE'
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 5110
EXPORTS WndProc @1
    VACATIONMsgProc @2
    SPORTMsgProc @3
```

资源描述文件:LEISURE. RC

```
#include "LEISURE. h"
LEISURE ICON ABOOK. ICO
PENCIL CURSOR PENCIL. CUR

SPORT1 BITMAP SPORT1. BMP
SPORT2 BITMAP SPORT2. BMP
SPORT3 BITMAP SPORT3. BMP
SPORT4 BITMAP SPORT4. BMP

LEISURE MENU
BEGIN
    POPUP "&Sport"
    BEGIN
        MENUITEM "SPORT1",IDM_S_SPORT1
        MENUITEM "SPORT2",IDM_S_SPORT2
        MENUITEM "SPORT3",IDM_S_SPORT3
        MENUITEM "SPORT4",IDM_S_SPORT4
    END
```

```
    POPUP "&Vacation"
    BEGIN
        MENUITEM "&Sport Type…\tCtrl+S",IDM_V_SPORTTYPE
        MENUITEM "&Location…\tCtrl+L",IDM_V_LOCATION
    END
END

#include "VACATION.DLG"

#include "SPORT.DLG"


LEISURE ACCELERATORS
BEGIN
    "^S",IDM_V_SPORTTYPE
    "^l",IDM_V_LOCATION
END

STRINGTABLE
BEGIN
    IDS_ERR_CREATE_WINDOW, "Window creation failed!"
    IDS_ERR_REGISTER_CLASS, "Error registering window class"
END
```

头文件:LEISURE.H

```
/* QuickCase:W KNB Version 1.00 */
#include <windows.h>
#include <string.h>
#include <stdlib.h>
#define IDM_SPORT 1000
#define IDM_S_SPORT1 1050
#define IDM_S_SPORT2 1100
#define IDM_S_SPORT3 1150
#define IDM_S_SPORT4 1200
#define IDM_VACATION 2000
#define IDM_V_SPORTTYPE 2050
#define IDM_V_LOCATION 2100

#define IDS_ERR_REGISTER_CLASS 1
#define IDS_ERR_CREATE_WINDOW 2

char szString[128];                          /* variable to load resource strings */

HCURSOR hCur;
char szAppName[20];                          /* class name for the window */
HWND hInst;
HWND hWndMain;
HANDLE hAccel;
FARPROC lpfnSPORTMsgProc;
HWND hDlgSPORT;
```

```
void cwCenter(HWND,int);

LONG FAR PASCAL WndProc(HWND,WORD,WORD,LONG);
BOOL FAR PASCAL VACATIONMsgProc(HWND,WORD,WORD,LONG);
BOOL FAR PASCAL SPORTMsgProc(HWND,WORD,WORD,LONG);
int nCwRegisterClasses(void);
void CwUnRegisterClasses(void);
```

对话框文件:SPORT. DLG

```
100 DIALOG 41,20,202,100
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Sport Select"
FONT 8,"Helv"
BEGIN
    GROUPBOX "Age",101,4,7,50,85
    CONTROL "20-30 ",102,"Button",BS_AUTORADIOBUTTON,11,22,39,10
    CONTROL "31-40 ",103,"Button",BS_AUTORADIOBUTTON,11,41,39,10
    CONTROL "41-50 ",104,"Button",BS_AUTORADIOBUTTON,11,57,39,10
    CONTROL "51-99 ",105,"Button",BS_AUTORADIOBUTTON,11,74,39,10
    GROUPBOX "Sport Type",106,74,7,61,85
    CONTROL "Water",107,"Button",BS_AUTOCHECKBOX | WS_TABSTOP,79,26,48,10
    CONTROL "Air",108,"Button",BS_AUTOCHECKBOX |
            WS_TABSTOP,79,52,48,10
    CONTROL "Land",109,"Button",BS_AUTOCHECKBOX | WS_TABSTOP,
            79,75,40,10
    PUSHBUTTON "OK",IDOK,156,6,40,14
    PUSHBUTTON "Cancel",IDCANCEL,156,23,40,14
END
```

对话框文件:VACATION. DLG

```
100 DIALOG 6,18,127,82
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Location"
FONT 8,"Helv"
BEGIN
    LTEXT "Country: ",101,15,14,47,11
    EDITTEXT 102,14,28,78,12,ES_AUTOHSCROLL
    PUSHBUTTON "OK",IDOK,14,57,40,14
    PUSHBUTTON "Cancel",IDCANCEL,70,57,40,14
END
```

对话框文件:LEISURE. MAK

```
ORIGIN = QCWIN
ORIGIN_VER = 1.00

PROJ =LEISURE
DEBUG =1
PROGTYPE =1
CALLER =
ARGS =
```

```
DLLS =
CVPACK =1
CC =cl -qc
RC =rc
CFLAGS_G_WEXE =/AS /G2w /Zp /W3 /D_WINDOWS
CFLAGS_D_WEXE =/Gi /Od /Zi
CFLAGS_R_WEXE =/O /Os /DNDEBUG
CFLAGS_G_WDLL =/AS /G2w /Zp /Aw /W3 /D_WINDOWS /D_WINDLL
CFLAGS_D_WDLL =/Gi /Od /Zi
CFLAGS_R_WDLL =/O /Os /DNDEBUG
CFLAGS_G_WTTY =/AS /G2w /W3 /D_WINDOWS
CFLAGS_D_WTTY =/Gi /Od /Zi
CFLAGS_R_WTTY =/O /Os /DNDEBUG
CFLAGS_G_DEXE =/AS /W2
CFLAGS_D_DEXE =/Gi /Od /Zi
CFLAGS_R_DEXE =/O /Ot /DNDEBUG
CFLAGS = $ (CFLAGS_G_WEXE) $ (CFLAGS_D_WEXE)
LFLAGS_G_WEXE =/ST:5120 /A:16
LFLAGS_D_WEXE =/CO
LFLAGS_R_WEXE =
LFLAGS_G_WDLL =/ST:5120 /A:16
LFLAGS_D_WDLL =/CO
LFLAGS_R_WDLL =
LFLAGS_G_WTTY =/ST:5120 /A:16
LFLAGS_D_WTTY =/CO
LFLAGS_R_WTTY =
LFLAGS_G_DEXE =/NOI /ST:2048
LFLAGS_D_DEXE =/CO
LFLAGS_R_DEXE =
LFLAGS = $ (LFLAGS_G_WEXE) $ (LFLAGS_D_WEXE)
RCFLAGS =
RESFLAGS =-t
RUNFLAGS =
H = LEISURE. H
RESFILES = VACATION. DLG
DEFFILE = LEISURE. DEF
OBJS_EXT =
LIBS_EXT =

. rc. res: ; $ (RC) $ (RCFLAGS) -r $ * . rc

all: $ (PROJ). EXE

LEISURE. OBJ: LEISURE. C $ (H)

LEISURE. RES: LEISURE. RC $ (RESFILES) $ (H)

$ (PROJ). EXE: LEISURE. OBJ $ (OBJS_EXT) $ (DEFFILE)
    echo )NUL @<< $ (PROJ). CRF
LEISURE. OBJ +
$ (OBJS_EXT)
```

```
$(PROJ).EXE

G:\DEVTOOLS\QCWIN\LIB\+
G:\WINDEV\WIN3-INT\PGMS\CH21\+
/NOD slibcew oldnames libw
$(DEFFILE);
((
    link $(LFLAGS) @$(PROJ).CRF
    rc $(RESFLAGS) LEISURE.RES $(PROJ).EXE

$(PROJ).EXE: LEISURE.RES
    rc $(RESFLAGS) LEISURE.RES $(PROJ).EXE

run: $(PROJ).EXE
    $(PROJ) $(RUNFLAGS)
```

我们需要对一些应用程序中未实现的或用特殊方法实现的点进行寻址。如果这两类都不能完成，或不能建立主窗口，应用程序将终止运行并通知用户。

这些出错信息在资源描述文件的 STRINGTABLE 段中。对每个 Windows 应用程序，只要是由 QuickCase:W 建立的，都自动地指派一个资源描述文件。项目名 LEISURE 可在 szAppName 串中找到，类名和菜单将用到这个项目名。

当程序在退出消息循环后终止运行时，将调用子例程 CwUnRegisterClasses。这个例程释放掉类，不管这个类是否需要释放。由 Windows 本身完成这种释放操作，但是删除掉画背景的刷子常常是非常重要的。

在 QuickCase:W 产生的窗口例程中都有下列消息：

消息	含义
WM_CREATE	建立窗口
WM_SIZE	重新计算窗口大小
WM_MOVE	移动窗口
WM_PAINT	更新客户区
WM_CLOSE	关闭窗口

这里画的对话框相当粗糙，因此个别控件不是由常量、而是用数字寻址的，这些数据使得程序代码模糊不清。幸运的是，QuickCase:W 把与按钮相关的文本作为在消息 WM_COMMAND 中查询参数 wParam 时的注释：

```
case WM_COMMAND:
    switch(wParam)
    {
        case 102:                                   /* Radiobutton text: "20-30" */
            break;
        case 103:                                   /* Radiobutton text: "31-40" */
            break;
        case 104:                                   /* Radiobutton text: "41-50" */
            break;
        case 105:                                   /* Radiobutton text: "51-99" */
            break;
```

```
case 107:                                   /* Checkbox text: "Water" */
    break;
case 108:                                   /* Checkbox text: "Air" */
    break;
case 109:                                   /* Checkbox text: "Land" */
    break;
```

IDOK 值用于按钮 OK，IDCANCEL 值用于按钮 Cancel，在对话框文件中可找到这两个值。否则，只能在得到消息 WM_CLOSE 后，从 System 菜单中关闭对话框。

第 5 步：完成源代码

建立 Windows 应用程序的下一步是在 QuickCase：W 产生的代码中填入所需的内容，这将留给开发者完成。

第 6 步：更新 QuickCase：W 代码

一旦完成了上述五个步骤，并确信非常满意之后，仍很有可能需要做些改动。像增加一个菜单项这样的小问题可以用 QuickCase：W 直接解决。用 Build/Update 来更新文件，然后再编译一次。

24.2.3 注释级别

结束本章前，我们用 QuickCase：W 可达到的最高注释级来建立一个小例子。在菜单项 Options/Comments 中可改变注释级。

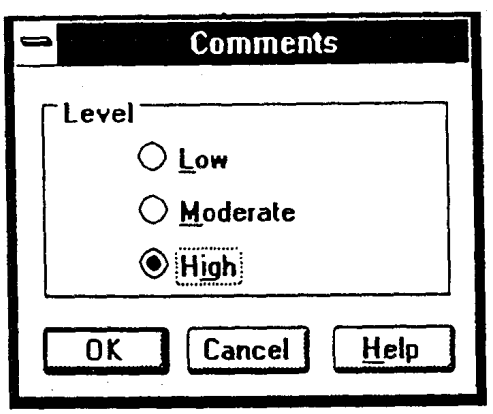


图 24.17 设置最高注释级

这个例子中的 C 代码只有几行，但是用最高注释级产生的注释却有几页。尽管有了详尽的注释，我们仍将发现这样的源程序很难读懂。

我们把这个项目命名为 COMMENTS，窗口设置为缺省的风格属性，标题设置为“Maximum Commentary”，菜单条包一个菜单标题和两个菜单项。选中 Menu 1/Dialog Box… 项将显示一个消息框。

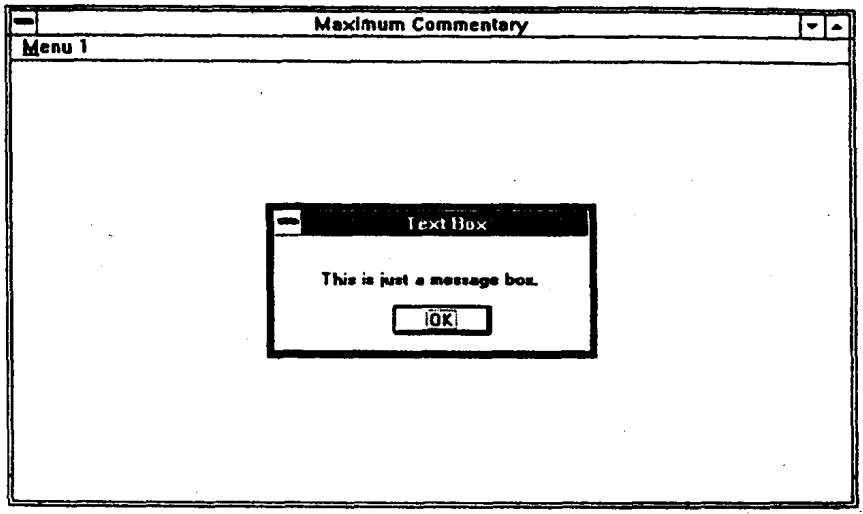


图 24.18　设置为最高注释级的例子

源代码:COMMENTS. C

```
/* QuickCase:W KNB Version 1.00 */

#include "windows.h"
#include "COMMENTS.h"

#if _MSC_VER >= 700
#pragma warning (disable:4028)
#endif

/* * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                     */
/*  Windows 3.0 Main Program Body                      */
/*                                                     */
/*  The following routine is the Windows Main Program. The Main Program */
/*  is executed when a program is selected from the Windows Control */
/*  Panel or File Manager. The WinMain routine registers and creates */
/*  the program's main window and initializes global objects. The */
/*  WinMain routine also includes the applications message dispatch */
/*  loop. Every window message destined for the main window or any */
/*  subordinate windows is obtained,possibly translated,and */
/*  dispatched to a window or dialog processing function. The dispatch */
/*  loop is exited when a WM_QUIT message is obtained. Before exiting */
/*  the WinMain routine should destroy any objects created and free */
/*  memory and other resources.                        */
/*                                                     */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * */

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpszCmdLine,int nCmdShow)
{
```

```
/* * * *
```

```
while(GetMessage(&msg,NULL,0,0))                    /* Until WM_QUIT message */
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

/* Do clean up before exiting from the application */
CwUnRegisterClasses();
return msg.wParam;
}                                                   /* End of WinMain */

/*************************/
/*                       */
/* Main Window Procedure */
/*                       */
/* This procedure provides service routines for the Windows events */
/* (messages) that Windows sends to the window, as well as the user */
/* initiated events (messages) that are generated when the user selects */
/* the action bar and pulldown menu controls or the corresponding */
/* keyboard accelerators. */
/*                       */
/* The SWITCH statement shown below distributes the window messages to */
/* the respective message service routines, which are set apart by the */
/* CASE statements. The window procedures must provide an appropriate */
/* service routine for its end user initiated messages, as well as the */
/* general Windows messages (ie. WM_CLOSE message). If a message is */
/* sent to this procedure for which there is no programmed CASE clause */
/* (i.e., no service routine), the message is defaulted to the */
/* DefWindowProc function, where it is handled by Windows */
/*                       */
/* For the end-user initiated messages, this procedure is concerned */
/* principally with the WM_COMMAND message. The menu control ID (or the */
/* corresponding accelerator ID) is communicated to this procedure in */
/* the first message parameter (wParam). The window procedure provides */
/* a major CASE statement for the WM_COMMAND message and a subordinate */
/* SWITCH statement to provide CASE clauses for the message service */
/* routines for the various menu item's, identified by their ID values. */
/*                       */
/* The message service routines for the individual menu items are the */
/* main work points in the program. These service routines contain the */
/* units of work performed when the end user select one of the menu */
/* controls. The required application response to a menu control is */
/* programmed in its associated CASE clause. The service routines may */
/* contain subroutine calls to separately compiled and libraried */
/* routines, in-line calls to subroutines to be embodied in this source */
/* code module, or program statements entered directly in the CASE */
/* clauses. Program control is switched to the appropriate service */
/* routine when Windows recognizes the end user event and sends a WM_COMMAND */
/* message to the window procedure. The service routine provides the */
/* appropriate application-specific response to the end user initiated */
/* event, then breaks to return control to the WinMain() routine which */
```

```
/* continues to service the message queue of the window(s).  */
/*  */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

LONG FAR PASCAL WndProc(HWND hWnd,WORD Message,WORD wParam,LONG lParam)
{
HMENU hMenu=0;                                          /* handle for the menu */
HBITMAP hBitmap=0;                                      /* handle for bitmaps */
HDC hDC;                                                /* handle for the display device */
PAINTSTRUCT ps;                                         /* holds PAINT information */
int nRc=0;                                              /* return code */

switch (Message)
{
    case WM_COMMAND:
        /* The Windows messages for action bar and pulldown menu items */
        /* are processed here.  */
        /* The WM_COMMAND message contains the message ID in its first */
        /* parameter (wParam). This routine is programmed to SWITCH on */
        /* the #define values generated by CASE:W for the menu items */
        /* in the application's header (*.H) file. The ID values have */
        /* the format,IDM_itemname. The service routines for the */
        /* various menu items follow the CASE statements up to the */
        /* generated BREAK statements.  */
        switch (wParam)
        {
            case IDM_M_MENUITEM1:
                /* Place User Code to respond to the */
                /* Menu Item Named "Menu Item &1" here.  */
                break;
            case IDM_M_DIALOGBOX:
                /* Place User Code to respond to the */
                /* Menu Item Named "&Dialog Box..." here.  */
                {
                    FARPROC lpfnTEXTMsgProc;

                    lpfnTEXTMsgProc = MakeProcInstance((FARPROC)TEXTMsgProc,hInst);
                    nRc = DialogBox(hInst,MAKEINTRESOURCE(100),hWnd,lpfnTEXTMsgProc);
                    FreeProcInstance(lpfnTEXTMsgProc);
                }
                break;
            default:
                return DefWindowProc(hWnd,Message,wParam,lParam);
        }
        break;                                          /* End of WM_COMMAND */
    case WM_CREATE:
        /* The WM_CREATE message is sent once to a window when the */
        /* window is created. The window procedure for the new window */
        /* receives this message after the window is created,but */
        /* before the window becomes visible.  */
        /*  */
```

```
        /* Parameters: */
        /*  */
        /* lParam - Points to a CREATESTRUCT structure with */
        /* the following form: */
        /*  */
        /* typedef struct */
        /* { */
        /* LPSTR lpCreateParams; */
        /* HANDLE hInst; */
        /* HANDLE hMenu; */
        /* HWND hwndParent; */
        /* int cy; */
        /* int cx; */
        /* int y; */
        /* int x; */
        /* LONG style; */
        /* LPSTR lpszName; */
        /* LPSTR lpszClass; */
        /* DWORD dwExStyle; */
        /* } CREATESTRUCT; */
        break;                                          /* End of WM_CREATE */
    case WM_MOVE:                                       /* code for moving the window */
        break;
    case WM_SIZE:                                       /* code for sizing client area */
        /* wParam contains a code indicating the requested sizing */
        /* lParam contains the new height and width of the client area */
        break;                                          /* End of WM_SIZE */
    case WM_PAINT:                                      /* code for the window's client area */
        /* Obtain a handle to the device context */
        /* BeginPaint will sends WM_ERASEBKGND if appropriate */
        memset(&ps,0x00,sizeof(PAINTSTRUCT));
        hDC = BeginPaint(hWnd,&ps);

        /* Included in case the background is not a pure color */
        SetBkMode(hDC,TRANSPARENT);

        /* Application should draw on the client window using */
        /* the GDI graphics and text functions. 'ps' the PAINTSTRUCT */
        /* returned by BeginPaint contains a rectangle to the */
        /* area that must be repainted. */

        /* Inform Windows painting is complete */
        EndPaint(hWnd,&ps);
        break;                                          /* End of WM_PAINT */
    case WM_CLOSE:                                      /* close the window */
        /* Destroy child windows, modeless dialogs, then, this window */
        DestroyWindow(hWnd);
        if (hWnd == hWndMain)
            PostQuitMessage(0);                         /* Quit the application */
        break;
    default:
```

```
        /* For any message for which you don't specifically provide a */
        /* service routine,you should return the message to Windows */
        /* for default message processing. */
        return DefWindowProc(hWnd,Message,wParam,lParam);
    }
    return 0L;
}                                                          /* End of WndProc */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                        */
/* Dialog Window Procedure                                */
/*                                                        */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */

BOOL FAR PASCAL TEXTMsgProc(HWND hWndDlg,WORD Message,WORD wParam,LONG lParam)
{

switch(Message)
{
    case WM_INITDIALOG:
        cwCenter(hWndDlg,0);
        /* initialize working variables */
        break;                                    /* End of WM_INITDIALOG */
    case WM_CLOSE:
        /* Closing the Dialog behaves the same as Cancel */
        PostMessage(hWndDlg,WM_COMMAND,IDCANCEL,0L);
        break;                                        /* End of WM_CLOSE */
    case WM_COMMAND:
        switch(wParam)
        {
            case IDOK:
                EndDialog(hWndDlg,TRUE);
                break;
            case IDCANCEL:
                /* Ignore data values entered into the controls */
                /* and dismiss the dialog window returning FALSE */
                EndDialog(hWndDlg,FALSE);
                break;
        }
        break;                                    /* End of WM_COMMAND */
    default:
        return FALSE;
}
return TRUE;
}                                                      /* End of TEXTMsgProc */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                        */
/* nCwRegisterClasses Function                            */
/*                                                        */
/* The following function registers all the classes of all the windows */
```

```
/* associated with this application. The function returns an error code */
/* if unsuccessful, otherwise it returns 0. */
/* */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */

int nCwRegisterClasses(void)
{
WNDCLASS wndclass;                                    /* struct to define a window class */
memset(&wndclass, 0x00, sizeof(WNDCLASS));


/* load WNDCLASS with window's characteristics */
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_BYTEALIGNWINDOW;
wndclass.lpfnWndProc = WndProc;
/* Extra storage for Class and Window objects */
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInst;
wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
/* Create brush for erasing background */
wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wndclass.lpszMenuName = szAppName;                    /* Menu Name is App Name */
wndclass.lpszClassName = szAppName;                   /* Class Name is App Name */
if(! RegisterClass(&wndclass))
    return -1;

return(0);
}                                                     /* End of nCwRegisterClasses */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* cwCenter Function */
/* */
/* centers a window based on the client area of its parent */
/* */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void cwCenter(hWnd, top)
HWND hWnd;
int top;
{
POINT pt;
RECT swp;
RECT rParent;
int iwidth;
int iheight;

/* get the rectangles for the parent and the child */
GetWindowRect(hWnd, &swp);
GetClientRect(hWndMain, &rParent);
```

```
/* calculate the height and width for MoveWindow */
iwidth = swp.right - swp.left;
iheight = swp.bottom - swp.top;

/* find the center point and convert to screen coordinates */
pt.x = (rParent.right - rParent.left) / 2;
pt.y = (rParent.bottom - rParent.top) / 2;
ClientToScreen(hWndMain,&pt);

/* calculate the new x,y starting point */
pt.x = pt.x - (iwidth / 2);
pt.y = pt.y - (iheight / 2);

/* top will adjust the window position,up or down */
if(top)
    pt.y = pt.y + top;

/* move the window */
MoveWindow(hWnd,pt.x,pt.y,iwidth,iheight,FALSE);
}

/*************************/
/* CwUnRegisterClasses Function */
/*  */
/* Deletes any refrences to windows resources created for this */
/* application,frees memory,deletes instance,handles and does */
/* clean up prior to exiting the window */
/*  */
/*************************/

void CwUnRegisterClasses(void)
{
WNDCLASS wndclass;                          /* struct to define a window class */
memset(&wndclass,0x00,sizeof(WNDCLASS));

UnregisterClass(szAppName,hInst);
}                                           /* End of CwUnRegisterClasses */
```

这些注释本身很有趣，值得一读。它们在解释个别的消息时，比我们所能加入的注解更为准确。但随着项目的增大，这些注释将变得过于庞大。

模块定义文件：COMMENTS.DEF

```
NAME COMMENTS
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE 4096
STACKSIZE 5110
EXPORTS WndProc @1
    TEXTMsgProc @2
```

资源描述文件：COMMENTS. RC

```
#include "COMMENTS. h"

COMMENTS MENU
BEGIN
    POPUP "&Menu 1"
    BEGIN
        MENUITEM "Menu Item &1",IDM_M_MENUITEM1
        MENUITEM "&Dialog Box…",IDM_M_DIALOGBOX
    END
END

#include "G:\WINDEV\WIN3-INT\PGMS\CH21\TEXT. DLG"

STRINGTABLE
BEGIN
    IDS_ERR_CREATE_WINDOW, "Window creation failed!"
    IDS_ERR_REGISTER_CLASS, "Error registering window class"
END
```

头文件：COMMENTS. H

```
/* QuickCase:W KNB Version 1. 00 */
#include <windows. h>
#include <string. h>
#define IDM_MENU1 1000
#define IDM_M_MENUITEM1 1050
#define IDM_M_DIALOGBOX 1100

#define IDS_ERR_REGISTER_CLASS 1
#define IDS_ERR_CREATE_WINDOW 2

char szString[128];                          /* variable to load resource strings */

char szAppName[20];                          /* class name for the window */
HWND hInst;
HWND hWndMain;

void cwCenter(HWND,int);

LONG FAR PASCAL WndProc(HWND,WORD,WORD,LONG);
BOOL FAR PASCAL TEXTMsgProc(HWND,WORD,WORD,LONG);
int nCwRegisterClasses(void);
void CwUnRegisterClasses(void);
```

对话框文件：TEXT. DLG

```
100 DIALOG 91,31,127,54
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Text Box"
```

```
FONT 8,"Helv"
BEGIN
    LTEXT "This is just a message box.",-1,18,16,91,26
    PUSHBUTTON "OK",IDOK,47,32,40,14
END
```

MAKE 文件：COMMENTS. MAK

```
DEBUG = 0
PROGTYPE = 1
CFLAGS =-AS -Osw -Gsw -Zpe -W3 -DLINT_ARGS -D_WINDOWS -nologo

.rc.res: ; rc -r $ *.rc

RESFILES = text.dlg
H = COMMENTS.H

all: COMMENTS.EXE

COMMENTS.RES: COMMENTS.RC $(RESFILES)

COMMENTS.OBJ: COMMENTS.C $(H)

COMMENTS.EXE: COMMENTS.OBJ COMMENTS.RES COMMENTS.DEF
    link /NOD/A:16 COMMENTS,COMMENTS,nul,libw slibcew,COMMENTS.def
    rc COMMENTS.res COMMENTS.exe
```

第二十五章　DOS 保护模式接口

近年来处理效率增长很快，但 MS-DOS 仍然没有跟上步伐。MS-DOS 可以处理扩展地址空间、多任务和虚拟内存管理，但这仅仅是间接的。更多的应用程序调用基于这些特征，而 MS-DOS 仍然在实模式（一种不再被 Windows 支持的模式）下工作。

大规模的应用程序因为需要再装入程序的覆盖块，且需要在常规内存、扩充内存和硬盘驱动器间交换大量的数据，因而使得执行速度很缓慢。

为了解决这个问题，“DOS 扩展”在推出 80286 处理器前就已出现在市场上。DOS 扩展允许保护模式下的程序在 MS-DOS 下运行，并且可以获取 DOS 服务。

这种扩展必须在每个 DOS 或 BIOS 函数调用前切换到实方式，然后在调用返回后再回到保护模式。一旦应用程序在保护模式下运行，就可直接寻址扩展内存。这种切换消除了中间 DOS 功能（如 BIOS 中断 15H）。这个扩充内存除了存取数据之外，对处理代码也是有效的。

同时，也出现了 32 位扩展，可以在 80386 或更高级的处理器上运行。它克服了实模式下最大段长为 64K 的限制，每段现在可以有 4 千兆字节的容量。

这称为“平台内存模式”，在此模式下，程序只处理线性地址。这加速了应用程序的运行，且消除了分段内存体系结构的不方便性。平台内存模式也简化了其数据源在没有使用分段内存结构的其他计算机系统里的大型程序的调度（如 UNIX）。

25.1　EMS 仿真

传统的 DOS 程序一直在寻找使用比 640K 更多的内存的办法。扩展内存规范（EMS）是在 8086 时代设计的，其操作遵循“执行一切换”规则。

有一个内存窗口，通常位于“适配器区域”A000H 到 F000H 中的 64K，该区域也是扩充内存段集中的地方。内存一般在处理器的地址空间里，并且可用常规方法访问。

代码可以在 EMS 内存中执行，内存的物理位置处在一个辅助插件板上（通过系统总线连接起来）。这个解决办法的不利之处是硬件成本较高，并且当 EMS 内存只是被偶尔需要时，这个问题将尤其严重。

80286 系统允许地址达到 16 兆字节，且 80286 和更高级的插件板可支持多于 1 兆字节。

EMS 因 80386 计算机的出现而真正引起了人们的兴趣，80386 处理器通过提供虚拟内存管理能力，从而更有效地使用 EMS。扩展内存转换为常规内存区域，仅仅重定义页表上的入口即可。EMS 仿真程序在保护模式下运行，只有那样才能获取对处理器进行扩充的能力，

并且，如果要继续应用所有常规 DOS 程序，EMS 仿真程序和操作系统可在虚拟 8086 方式下执行 DOS 程序。当一个类似 EMS 仿真程序的保护模式程序在后台运行时，前台程序以一兆地址空间加 HMA 段：偏移量寻址代替选择器、它自身的中断向量表和它自身的 I/O 端口。后者是重要的，因为 80836 能够增加不只一个这样的虚拟机(VM)，而是可以同时存在几个虚拟机，在这些虚机系统之间切换是非常简单的。Windows 就是如此(当有几个 DOS 外壳在同时运行时)。

当用户想在同一时间运行几个这样的程序时就出现问题了。EMS 仿真程序必须在保护模式下运行(当系统引导时仿真程序通常切换为此模式)。一般地，它通过系统文件 CONFIG.SYS 结合成一体。

25.2 VCPI

许多用户既需要一个高效的内存管理程序，以装入驱动程序和 TSR 到高端内存区(UMA)，从而节省常规内存，又需要根据 DOS 扩展运行大型程序。VCPI(虚拟控制编程接口)使得一些应用程序可以同时使用同一处理器资源。

这种技术包含两种应用程序类型：VCPI 服务器和 VCPI 客户。服务器提供 VCPI 的接口服务，而客户使用这些服务来开发扩展内存和保护模式。通常，VCPI 是一个 EMS 仿真程序。

EMS 仿真程序通常在系统引导时被装入，并且驻留在内存中，直到会话结束为止。这不是与应用程序相关的 DOS 扩展的情况。这种任务分配是实现如中断 67H(控制 LIM/EMS 的中断服务)的 VCPI 服务器的服务的判定理由。

这些功能被保留下来，并且辅助功能允许扩展内存管理，在 386 模式和保护模式之间切换，重编程中断控制器，并且访问 80386 处理器的特殊寄存器。

在切换到保护模式后，VCPI 客户可以自由访问由此模式所提供的所有功能，它们像 VCPI 服务器那样在同一优先级上运行。但这从一开始就是未定的，因为 VCPI 使得几个程序可以并行。

25.3 DPMI

为了实现 DOS 应用程序的多任务，Microsoft 公司开发了一个用于程序交互的可扩充单元，叫做 DPMI(DOS 保护模式接口)。虽然最初只是一个封闭的概念，但此标准仍被发表了，并且随着与其他公司的协作更进一步地发展着。

DPMI 也是以客户/服务器的方式工作的。服务器(或主机)可以提供下列服务：

- 扩展内存管理
- 描述符表管理(PM)
- 中断和异常管理
- 保护模式与实模式应用程序的通信。
- 80386 特殊寄存器的访问
- DMA 虚拟

VCPI 处理扩展内存，但是将其他任务留给客户，这些客户可以像保护模式下的程序那

样处理这些任务。

DPMI 使得它的客户在 VMS(即虚拟 8086 环境)中运行。因此,它们不能随意切换到保护模式,而必须调用一个 DPMI 主机功能来完成此任务。DPMI 保证客户程序在保护模式下运行,但其优先级低于主机。

客户将大多数保护模式的任务转交给主机,主机从总体上对系统进行高级控制。一个 DPMI 客户可以完全被另一个所控制,以便两个客户在不同的 VM 中运行。这其实是不会发生的,多个客户可以在同一 VM(虚拟系统)中交互,但是不会从另一个那里得到保护。

DPMI 还是比较新的,并且目前 Windows 3.1 是唯一的 DPMI 主机。不管怎么样,DPMI 将很快被定为标准(许多由 DPMI 支持的 DOS 扩展已经出现了)。DPMI 可以在 80286 或更高级的处理器中实现,这一点不像 VCPI。

DPMI 有 16 位和 32 位版本,甚至可以用一个 32 位主机操作一个 16 位客户,但不能反过来用。

25.4 DPMI 功能和 31H 中断

用户可能从未直接遇到过 DPMI 功能,除非用户打算开发自己的 DOS 扩展或内存管理程序。下面的概述可以说明,在一个古老的操作系统(如 MS-DOS)中实现现代软件的概念是多么困难。

DPMI 提供两个接口。2FH 多重中断初始化客户和主机之间的链接。一旦 DPMI 主机被安装,功能 1687H 就返回状态信息,以及用来切换系统到保护模式下的函数的地址。这个函数允许切换,即使处理器已经在 V86 方式下亦可。

与 DPMI 的更强的联系是由中断 31H 引起的,这只发生在保护模式下,在用上面提到过的函数切换之后才适用。

切换到保护模式下后,程序发现自己处于一个结构非常"单纯"的环境中:段寄存器 CS、DS 和 SS 分别对应于 64K 段的选择器(对应于程序在实模式下运行时的物理内存区域),这个段允许代码在从实模式切换到保护模式后作更进一步的处理。ES 包括一个描绘应用程序的 PSP 的选择器。

25.5 DPMI 和 Windows 应用程序

Windows 应用程序也可以访问 DPMI 服务,但这不是对所有的函数都适用。部分原因是,许多函数已被正规的 Windows API 所覆盖。在这种情况下,直接调用 DMPI 的做法是应当避免的,以防止在 Windos 核心中可能发生的冲突。另一方面,Windows 的标准模式只实现了 DPMI 标准的一个子集。

Windows 为管理描述符提供了一些函数,又为 DOS 内存分配和释放提供了两个函数,它们覆盖了 DPMI 接口的一个主要部分。Windows 未提供任何可从保护模式转换到实方式(或反之)的函数,但这些函数恰恰是 Windows 程序可访问的 DPMI 函数。

我们将涉及到下列函数。

• 取得实模式中断向量

返回一个实模式的中断向量。在保护模式下,中断处理程序通过一个分离表被调用。

• 设置实模式中断向量

为实模式设置一个新的中断向量。

• 模拟实模式中断

触发一个实模式中断。这个函数使得软件和硬件中断模拟成为可能。

• 调用带远返回帧(frame)的实模式过程

允许调用一个在实模式下操作的例程,也允许 Windows 应用程序调用一般不能在保护模式下运行的 TSR 函数。

• 调用带 IRET 帧的实模式过程

对应于前一函数,这里假定被调用的例程以 IRET 而不是 RETF 结束。

• 分配实模式回调地址

分配一个回调函数,允许程序在实模式下调用 Windows 应用程序的保护模式函数。

• 释放实模式回调地址

释放一个回调函数。

25.6 汇 总

本节汇总了 DPMI 功能,它们基于 0.9 版本。以下只能提供 DPMI 功能的概貌,有关更详细的说明,请向 Intel 公司索取有关文档。

25.6.1 中断 2FH 和实模式

1680H

空(无效)中断。一个在某时刻空闲的客户可以用这个功能请求空中断,且允许一个多任务处理器过早地切换任务。

输入:无

返回:AL = 0:被支持

AL = 80H:不被支持

1686H

查询 DPMI 模式,可确定中断 31H 是否有效。

输入:无
返回:AL = 0:中断 31H 有效
AL = 80H:DPMI 无效

1687H

DPMI 初始化。
输入:无
返回:AX 〈〉 0:无 DPMI 被安装
AX = 0:DPMI 被安装
BX = 1:32 位版本被支持
BX = 0:16 位版本被支持
CL = 处理器类型
CL = 2:80286
CL = 3:80386
CL = 4:80486
DH = DPMI 版本号
DL = DPMI 次版本号
SI = DPMI 段中的数据区大小
SI:DI = 切换到保护模式的函数地址
切换功能在下列情况下被调用:
输入:AX = 1:32 位应用程序
AX = 0:16 位应用程序
ES = DPMI 数据区的段地址:如果功能 1687H 在 SI 中返回空值则被忽略
返回:进位置位 = 出错,程序仍在实模式下
进位清除 = 切换成功
CS = 16 位代码段选择器
CS = 栈段选择器
DS = 数据段选择器
ES = PSP 选择器
FS = GS = 0:80286/80386

25.6.2 中断 31H 和保护模式

所有 DPMI 功能在返回成功时,进位标志被清除。在出错的情况下,进位标志被置位,但不会返回特定的出错代码。返回寄存器的内容在这种情况下是未定义的。下面的项仅当函数返回时进位标志被清除时才有效。

0000H

在局部描述符表内分配描述符。
输入:CX = 已分配的描述符的个数

返回:AX = 1:已分配序列选择器

0001H

释放 LDT 描述符。
输入:BX = 被释放的选择器
返回:无

0002H

把段地址转换为描述符。
输入:BX = 实模式段地址
返回:AX = 相应的选择器

0003H

返回下一个选择器的偏移量。
输入:无
返回:AX = 必须被加到选择器上的值,由 0000H 决定,以获得序列中下一个选择器

0004H:预留

0005H:预留

0006H

返回线性选择器基地址。
输入:BX = 选择器
返回:CX:DX = 线性段基地址

0007H

定义线性选择器基地址。
输入:BX = 选择器
CX:DX = 线性段基地址
返回:无
警告:只有由功能 0000H 分配的选择器受控

0008H

定义段长度。
输入:BX = 选择器
CX:DX = 段长度
返回:无

0009H

定义段存取权限。
输入:BX = 选择器
　　CL = 存取权限
　　CH = 80386 的扩充类型字节
返回:无
CH 在 16 位实现中被忽略。

000AH

为一个代码段建立别名描述符。
输入:BX = 选择器
返回:AX = 新选择器
一个有效的代码段选择器必须被传输到 BX 中。

000BH

返回标识一个选择器的描述符。
输入:ES:(E)DI = 包含 8 字节描述符拷贝的缓冲区
返回:在缓冲区中

000CH

定义标识一个选择器的描述符。
输入:BX = 选择器
　　ES:(E)DI = 包含新的 8 字节描述符的缓冲区
返回:无

000DH

请求特殊描述符。
输入:BX = 期望描述符的选择器
返回:无
不保证所期望的描述符是自由的。

0100H

在 DOS 区域中请求内容块。
输入:BX = 已被分配的段数
返回:AX = 实模式的内存块段地址
　　DX = 内存块的首选择器
多个选择器被定义,每当一个 16 位客户请求一个比 64K 大的块时。

0101H

在 DOS 区释放内存块。
输入:DX = 内存块的首选择器
返回:无
所有属于该块的描述符均被自动释放。

0102H

在 DOS 区中重分配内存块。
输入:BX = 段中新块的长度
　　DX = 内存块的首选择器
返回:无

0200H

返回实模式下的中断向量。
输入:DL = 中断号
返回:CX:DX = 实模式下中断向量的段:偏移量

0201H

定义实模式中断向量。
输入:BL = 中断号
　　CX:DX = 实模式中断向量的段:偏移量
返回:CX:(E)DX = 选择器:中断处理程序的偏移量

0202H

返回处理器异常处理程序。
输入:BL = 异常号(00H－1FH)
返回:CX:(E)DX = 选择器:异常处理程序的偏移量

0203H

为处理器异常定义处理程序。
输入:BL = 异常号(00H－1FH)
　　CX:(E)DX = 选择器:异常处理程序的偏移量
返回:无

0204H

返回保护模式中断处理程序。
输入:BL = 中断号
返回:CX:(E)DX = 选择器:中断处理程序的偏移量

0205H

定义保护模式中断向量。

输入:BL = 中断号(00H—1FH)

CX:(E)DX = 选择器:中断处理程序的偏移量

返回:无

0300H

模拟实模式中断。

输入:BL = 中断号

BH = 1:触发中断控制器/A20(数据、程序)行复位

BH = 0:忽略复位

CX = 从保护模式栈拷贝到实模式栈的字的个数

ES:(E)DI = 选择器:实调用结构的偏移量

下面列出实模式调用结构:

偏移量	长度	内容(寄存器)
00H	DWORD	EDI
04H	DWORD	ESI
08H	DWORD	EBP
0CH	DWORD	保留(0)
10H	DWORD	EBX
14H	DWORD	EDX
18H	DWORD	ECX
1CH	DWORD	EAX
20H	WORD	处理器标志
22H	WORD	ES
24H	WORD	DS
26H	WORD	FS
28H	WORD	GS
2AH	WORD	IP
2CH	WORD	CS
2EH	WORD	SP
30H	WORD	SS

0301H

调入以 RETF 结束的实模式例程。

输入:BH = 1:触发中断控制器/A20 行复位

BH = 0:忽略复位

CX = 从保护模式栈拷贝到实模式栈的字的个数

ES:(E)DI = 选择器:实模式调用结构的偏移量

返回:可能改变实模式结构的内容

0302H

调用以 IRET 结束的实模式例程。

输入:BH = 1:触发中断控制器/A20 行复位

BH = 0:忽略复位

CX = 从保护模式栈拷贝到实模式栈的实的个数

ES:(E)DI=选择器:实模式调用结构的偏移量

返回:可能改变实模式结构的内容

0303H

定义实模式回调函数。

输入:DS:(E)SI = 选择器:回调函数的偏移量

ES:(E)DI = 选择器:实模式调用结构的偏移量

返回:CX:DX = 段:实模式回调结构地址的偏移量

0304H

释放实模式回调函数。

输入:CX:DX = 段:实模式回调函数地址的偏移量

返回:无

0305H

在模式切换过程中返回状态的内存地址。

输入:无

返回:AX = 状态缓冲区长度

BX:CX = 用来保存或恢复状态的函数的实模式地址

SI:(E)DI = 用来保存或恢复状态的函数的保护模式地址

如果保存状态不是必须的,AX 将包含 0 值。

0306H

返回模式切换函数的地址。

输入:无

返回:BX:CX = 完成从实模式切换到保护模式的函数的实模式地址

SI:(E)DI = 完成从保护模式切换到实模式的函数的保护模式地址

寄存器分配如下:

AX = DS 的新内容

CS = ES 的新内容

DX = SS 的新内容

(E)BX = (E)SP 的新内容
SI = CS 的新内容
(E)DI = (E)IP 的新内容

0400H

返回 DPMI 版本号。
输入:无
返回:AH = 主版本号
AL = 次版本号
BX = 标志
位 0:32 位实现
位 1:切换到实模式而不是 V86 模式
位 2:虚拟内存管理被支持
CL = 处理器类型
CL = 2:80286
CL = 3:80386
CL = 4:80486
DH = 主中断控制器的当前基中断
DL = 从中断控制器的当前基中断

0500H

返回自由扩展内存信息。
输入:ES:(E)DI = 选择器:数据缓冲区的偏移量
返回:包含数据的缓冲区
内存信息块结构如下:

偏移量	长度	内容
00H	DWORD	以字节为单位的最大可用内存块
04H	DWORD	未锁定页的最高编号
08H	DWORD	已锁定页的最高编号
0CH	DWORD	以页为单位的总线性地址空间
10H	DWORD	未锁定页的总数目
14H	DWORD	自由页的数目
18H	DWORD	物理有效页的总数目
1CH	DWORD	以页为单位的自由线性地址空间
20H	DWORD	以页为单位的交换文件的长度
24H	DWORD	预留
28H	DWORD	预留
2CH	DWORD	预留

0501H

分配扩充内存。
输入:BX:CX = 请求的内存长度(以字节为单位)
返回:BX:CX = 线性内存块的地址
SI:DI = 功能 0502H 和 0503H 的句柄
注意,此功能不能为内存块创建一个描述符。

0502H

释放扩展内存。
输入:SI:DI = 内存块句柄
返回:无

0503H

重分配扩展内存块。
输入:BX:CX = 新内存块长度
SI:DI = 内存块句柄
返回:BX:CX = 新线性内存块地址
SI:DI = 新内存块句柄
注意,内存块选择器必须被更新。

0600H

锁定线性地址区,以防止删除。
输入:BX:CX = 被锁定内存区域的线性初始地址
SI:DI = 内存区长度(以字节为单位)
返回:无
如果边界在页面的中间,整页都将被锁定。
DPMI 主机为每一页保留一个锁定计数。

0601H

解锁线性地址区以允许删除。
输入:BX:CX = 已解锁的内存块的线性初始地址
SI:DI = 内存区长度(以字节为单位)
如果边界在页内,整页都将被解锁。
锁定计数大于 0 的页保持锁定状态。

0602H

允许实模式内存删除。
输入:BX:CX = 将解锁的内存区的线性初始地址

SI:DI = 内存区长度(以字节为单位)

返回:无

注意,实模式内存无锁定计数。

0603H

重锁定实模式内存。

输入:BX:CX = 将被锁定的内存区的线性地址

SI:DI = 内存区长度(以字节为单位)

0604H

返回内存页的长度。

输入:无

返回:BX:CX = 页长度(以字节为单位)

0700H:预留

0701H:预留

0702H

为删除而建议的内存页。

输入:BX:CX = 内存区的线性初始地址

SI:DI = 内存区长度(以字节为单位)

返回:无

此功能实际上是一个建议,它不能导致内存区立即被删除。

0703H

拒收内存页的内容。

输入:BX:CX = 内存区的线性初始地址

SI:DI = 内存区长度(以字节为单位)

返回:无

0800H

重计算虚拟内存中的物理地址。

输入:BX:CX = 物理地址(在1兆字节以外)

SI:DI = 内存区长度

返回:BX:CX = 内存区的线性地址

0900H

读取和清除虚拟中断标志。

输入:无
返回:AL = 被清除的虚拟中断标志
AL = 1:先前置位
AL = 0:先前清除

0901H

读取和设置虚拟中断标志。
输入:无
返回:AL = 虚拟中断标志的设置
AL = 1:先前置位
AL = 0:先前清除

0902H

读取虚拟中断标志。
输入:无
返回:AL = 1:虚拟中断标志置位
AL = 0:虚拟中断标志清除

0A00H

返回特定于制造商的 API 的入口点。
输入:DS:(E)SI = 指向制造商名或注册号
返回:ES:(E)DI = 扩充的 API 入口点的地址
此 API 细节随制造商而变化。

0B00H

为调试定义观察点。
输入:BX:CX = 为观察点设置的线性地址
DL = 观察点长度(1,2 或 4 字节)
DH = 管理类型
DH = 0:执行
DH = 1:写访问
DH = 2:读/写访问
返回:BX = 观察点句柄

0B01H

释放观察点。
输入:BX = 观察点句柄
返回:无

0B02H

返回观察点状态。
输入:BX = 观察点句柄
返回:AX = 1:从上一次调用功能 0B00H 或 0B03H 后执行的观察点
AX = 0:未执行的观察点

0B03H

执行观察点复位。
输入:BX = 观察点句柄
返回:无

[General Information]
书名=高级Windows程序设计技术
作者=王浩　李莉 等
页数=786
SS号=10204554
出版日期=1996年1月第1版