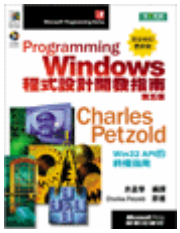


Windows程序设计

原著: Charles Petzold
翻译: 余孟学
PDF 整理: 涂德华

基础篇

第一章 开始

本书介绍了在Microsoft Windows 98、Microsoft Windows NT 4.0和Windows NT 5.0下程序写作的方法。这些程序用C语言编写并使用原始的Windows Application Programming Interface（API）。如在本章稍后所讨论的，这不是写作Windows程序的唯一方法。然而，无论最终您使用什么方式写作程序，了解Windows API都是非常重要的。

正如您可能知道的，Windows 98已成为使用Intel 32位微处理器（例如486和Pentium）的IBM兼容型个人计算机环境上最新的图形操作系统之代表。Windows NT是IBM PC兼容机种以及一些RISC（精简指令集计算机）工作站上使用的Windows工业增强型版本。

使用本书有三个先决条件。首先，您应该从使用者的角度熟悉Windows 98。不要期望可以在不了解Windows使用者接口的情形下开发其应用程序。因此，我建议您在开发程序（或在进行其它工作）时使用执行Windows的机器来跑Windows应用程序。

第二，您应了解C语言。如果要写Windows程序，一开始却不想了解C语言，那不是一个好主意。我建议您在文字控制台环境中，例如在Windows 98 MS-DOS命令提示窗口下提供的环境中学习C语言。Windows程序设计有时包括一些非文字模式程序设计的C语言部分；在这些情况下，我将针对这些问题提供讨论。但大多数情况下，您应非常熟悉该语言，特别是C语言的结构和指针。了解标准C语言执行期链接库的一些相关知识是有帮助的，但不是必要的。

第三，您应该在机器上安装一个适于进行Windows程序设计的32位C语言编译器和开发环境。在本书中，假定您正在使用Microsoft Visual C++ 6.0，该软件包可独立购买，也可作为Visual Studio 6.0软件包的一部分购买。

到此为止，我将不再假设您具有任何图形使用者接口（如Windows）的程序写作经验。

WINDOWS环境

Windows几乎不需要介绍。然而人们很容易忘记Windows给办公室和家庭桌上型计算机所带来的重大改变。Windows在其早期曾经走过一段坎坷的道路，征服桌上型计算机市场的前途一度相当渺茫。

Windows简史

在1981年秋天IBM PC推出之后不久，MS-DOS就已经很明显成为PC上的主流操作系统。MS-DOS代表Microsoft Disk Operating System（磁盘操作系统）。MS-DOS是一个小型的操作系统。MS-DOS提供给用户一种命令列接口，提供如DIR和TYPE的命令，也可以将应用程序加载内存执行。对于应用程序写作者，它提供了一组函数呼叫，进行文件的输入输出（I/O）。对于其它的外围处理 – 尤其是将文字或图形写到显示器上 – 应用程序可以直接存取PC的硬件。

由于内存和硬件的限制，成熟的图形环境缓慢地才到来。当苹果计算机公司不幸的Lisa计算机在1983年1月发表时，它提供了不同于文字模式环境的另一种选择，并在1984年1月成为Macintosh上图形环境的一种标准。尽管Macintosh的市场占有率在下降，但是它仍然被认为是衡量所有其它

图形环境的标准。包括Macintosh和Windows的所有图形环境，其实都要归功于Xerox Palo Alto Research Center（PARC）在70年代中期所作的开拓性研究工作。

Windows是由微软在1983年11月（在Lisa之后，Macintosh之前）宣布，并在两年后（1985年11月）发行。在此后的两年中，紧随着Microsoft Windows早期版本1.0之后，又推出了几种改进版本，以支持国际商业市场，并提供新型视讯显示器和打印机的驱动程序。

Windows版本2.0是在1987年11月正式在市场上推出的。该版本对使用者接口做了一些改进。这些改进中最有效的是使用了可重迭式窗口，而Windows 1.0中使用的是并排式窗口。Windows 2.0还增强了键盘和鼠标接口，特别是加入了菜单和对话框。

至此，Windows还只要求Intel 8086或者8088等级的微处理器，以「实际模式」执行，只能存取地址在1MB以下的内存。Windows/386（在Windows 2.0之后不久发行的）使用Intel 386微处理器的「虚拟8086」模式，实现将直接存取硬件的多个MS-DOS程序窗口化和多任务化。为了统一起见，Windows版本2.1被更名为Windows/286。

Windows 3.0是在1990年5月22日发表的。它将Windows/286和Windows/386结合到同一种产品中。Windows 3.0有了一个很大的改变，这就是对Intel的286、386和486微处理器保护模式的支持。这能使Windows和Windows应用程序能存取高达16MB的内存。Windows用于执行程序和维护文件的「外壳」程序得到了全面的改进。Windows 3.0是第一个在家用和办公室市场上取得立足点的版本。

任何Windows的历史介绍都必须包括一些OS/2的说明，OS/2是对DOS和Windows的另一种选择，最初是由Microsoft和IBM合作开发的。OS/2版本1.0（只有文字模式）在Intel 286（或者后来的）微处理器上运行，在1987年末发布。在1988年10月的OS/2版本1.1中出现了管理图形使用者接口的PM（Presentation Manager）。PM最初的设计构想是成为Windows的一种保护模式版本，但是图形API改变程度太大，致使软件生产厂商很难提供对这两种平台的支持。

到1990年9月，IBM和Microsoft之间的冲突达到了高峰，导致这两个公司最后分道扬镳。IBM接管了OS/2，而Microsoft明确表示Windows将是他们操作系统策略的中心。虽然OS/2仍然拥有一些狂热的崇拜者，但是它远不及Windows这样的普及程度。

Microsoft Windows版本3.1是1992年4月发布的，其中包括的几个重要特性是TrueType字体技术（给Windows带来可缩放的轮廓字体）、多媒体（声音和音乐）、对象连结和嵌入（OLE：Object Linking and Embedding）和通用对话框。跟OS/2一样，Windows 3.1只能在保护模式下运作，并且要求至少配置了1MB内存的286或386处理器。

在1993年7月发表的Windows NT是第一个支持Intel 386、486和Pentium微处理器32位保护模式的Windows版本。Windows NT提供32位平坦寻址，并使用32位的指令集。（本章后面我会谈到一些寻址空间的问题）。Windows NT还可以移植到非Intel处理器上，并在几种使用RISC芯片的工作站上执行。

Windows 95是在1995年8月发布的。和Windows NT一样，Windows 95也支持Intel 386或更高等级处理器的32位保护模式。虽然它缺少Windows NT中的某些功能，诸如高安全性和对RISC机器的可移植性等，但是Windows 95具有需要较少硬件资源的优点。

Windows 98在1998年6月发布，具有许多加强功能，包括执行效能的提高、更好的硬件支持以及与因特网和全球信息网（WWW）更紧密的结合。

Windows方面

Windows 98和Windows NT都是支持32位优先权式多任务（preemptive multitasking）及多

线程的图形操作系统。Windows拥有图形使用者接口（GUI），这种使用者界面也称作「可视化接口」或「图形窗口环境」。有关GUI的概念可追溯至70年代中期，在Alto和Star等机器上以及SmallTalk等环境中由Xerox PARC所作的研究工作。该项研究的成果后来被Apple Computer和Microsoft引入主流并流行起来。虽然有一些争议，但现在已非常清楚，GUI是（Microsoft的Charles Simonyi的说法）一个在个人计算机工业史上集各方面技术大成于一体的最重要产物。

所有GUI都在点矩阵对应的视讯显示器上处理图形。图形提供了使用屏幕的最佳方式、传递信息的可视化丰富多彩环境，以及能够WYSIWYG（what you see is what you get：所见即所得）的图形视讯显示和为书面文件准备好格式化文字输出内容。

在早期，视讯显示器仅用于响应使用者通过键盘输入的文字。在图形使用者接口中，视讯显示器自身成为使用者输入的一个来源。视讯显示器以图标和输入设备（例如按钮和滚动条）的形式显示多种图形对象。使用者可以使用键盘（或者更直接地使用鼠标等指向设备）直接在屏幕上操纵这些对象，拖动图形对象、按下鼠标按钮以及滚动滚动条。

因此，使用者与程序的交流变得更为亲密。这不再是一种从键盘到程序，再到视讯显示器的单向信息流动，使用者已经能够与显示器上的对象直接交互作用了。

使用者不再需要花费长时间学习如何使用计算机或掌握新程序了。Windows让这一切成真，因为所有应用程序都有相同的基本外观和感觉。程序占据一个窗口 – 屏幕上的一块矩形区域。每个窗口由一个标题栏标识。大多数程序功能由程序的菜单开始。用户可使用滚动条观察那些无法在一个屏幕中装下的信息。某些菜单项目触发对话框，用户可在其中输入额外的信息。几乎在每个大的Windows程序中都有一个用于开启文件的特殊对话框。该对话框在所有这些Windows程序中看起来都一样（或接近相同），而且几乎总是从同一菜单选项中启动。

一旦您了解使用一个Windows程序的方法，您就非常容易学习其它的Windows程序。菜单和对话框允许用户试验一个新程序并探究它的功能。大多数Windows程序同时具有键盘接口和鼠标接口。虽然Windows程序的大多数功能可通过键盘控制，但使用鼠标要容易得多。

从程序写作者的角度看，一致的使用者接口来自于Windows建构菜单和对话框的内置程序。所有菜单都有同样的键盘和鼠标接口，因为这项工作是由Windows处理，而不是由应用程序处理。

为便于多个程序的使用，以及这些程序间信息的交换，Windows支持多任务。在同一时刻能有多个Windows程序显示并运行。每个程序在屏幕上占据一个窗口。用户可在屏幕上移动窗口，改变它们的大小，在不同程序间切换，并从一个程序向另一个程序传送数据。因为这些窗口看起来有些像桌面上的纸（当然，这是计算机还未占据办公桌之前的年代），Windows有时被称作：一个显示多个程序的「具象化桌面」。

Windows的早期版本使用一种「非优先权式（non-preemptive）」的多任务系统。这意味着Windows不使用系统定时器将处理时间分配给系统中运行的多个应用程序，程序必须自愿放弃控制以便其它程序运行。在Windows NT和Windows 98中，多任务是优先权式的，而且程序自身可分割成近乎同时执行的多个执行绪。

操作系统不对内存进行管理便无法实现多任务。当新程序启动、旧程序终止时，内存会出现碎裂空间。系统必须能够将闲置的内存空间组织在一起，因此系统必须能够移动内存中的程序代码和数据块。

即使是在8088微处理器上跑的Windows 1.0也能进行这类内存管理。在实际模式限制下，这种能力被认为是软件工程一个令人惊讶的成就。在Windows 1.0中，PC硬件结构的640KB内存限制，在不要求任何额外内存的情况下被有效地扩展了。但Microsoft并未就此停步：Windows 2.0允许Windows应用程序存取扩充内存（EMS）；Windows 3.0在保护模式下，允许Windows应用程序存

取高达16MB的扩展内存。Windows NT和Windows 98通过成熟的32位操作系统及平坦寻址空间，摆脱了这些旧的限制。

Windows上执行的程序可共享在称为「动态链接库」的文件中的例程。Windows包括一个机制，能够在执行时连结使用动态链接库中例程的程序。Windows自身基本上就是一个动态链接库的集合。

Windows是一个图形接口，Windows程序能够在视讯显示器和打印机上充分利用图形和格式化文字。图形接口不仅在外观上更有吸引力，而且还能够让使用者传递高层次的信息。

Windows应用程序不能直接存取屏幕和打印机等图形显示设备硬件。相反，Windows提供一种图形程序语言（称作图形设备接口，或者GDI），使显示图形和格式化文字更容易。Windows虚拟化了显示硬件，使为Windows编写的程序可使用任何具有Windows设备驱动程序的视频卡或打印机，而程序无需确定系统相连的设备类型。

对Windows开发者来说，将与设备无关的图形接口输出到IBM PC上不是件轻松的事。PC的设计是基于开放式架构的原则，鼓励第三方硬件制造商为PC开发接口设备，而且开发了大量这样的设备。虽然出现了多种标准，PC上的传统MS-DOS程序仍不得不各自支持许多不同的硬设备。这对MS-DOS字处理软件来说非常普遍，它们连同1到2张有许多小文件的磁盘一同销售，每个文件支持一种特定的打印机。Windows程序不要求每个应用程序都自行开发这些驱动程序，因为这种支持是Windows的一部分。

动态链接

Windows运作机制的核心是一个称作「动态链接」的概念。Windows提供了应用程序丰富的可呼叫函数，大多数用于实作其使用者接口和在视讯显示器上显示文字和图形。这些函数采用动态链接库(Dynamic Linking Library，DLL)的方式撰写。这些动态链接库是些具有.DLL或者有时是.EXE扩展名的文件，在Windows 98中通常位于\WINDOWS\SYSTEM子目录中，在Windows NT中通常位于\WINNT\SYSTEM和\WINNT\SYSTEM32子目录中。

在早期，Windows的主要部分仅通过三个动态链接库实作。这代表了Windows的三个主要子系统，它们被称作Kernel、User和GDI。当子系统的数目在Windows最近版本中增多时，大多数典型的Windows程序产生的函数呼叫仍对应到这三个模块之一。Kernel（日前由16位的KRNL386.EXE和32位的KERNEL32.DLL实现）处理所有在传统上由操作系统核心处理的事务 – 内存管理、文件I/O和多任务管理。User（由16位的USER.EXE和32位的USER32.DLL实作）指使用者接口，实作所有窗口运作机制。GDI（由16位的GDI.EXE和32位的GDI32.DLL实作）是一个图形设备接口，允许程序在屏幕和打印机上显示文字和图形。

Windows 98支持应用程序可使用的上千种函数呼叫。每个函数都有一个描述名称，例如CreateWindow。该函数（如您所猜想的）为程序建立新窗口。所有应用程序可以使用的Windows函数都在表头文件里预先声明过。

在Windows程序中，使用Windows函数的方式通常与使用如strlen等C语言链接库函数的方式相同。主要的区别在于C语言链接库函数的机械码连结到您的程序代码中，而Windows函数的程序代码在您程序执行文件外的DLL中。

当您执行Windows程序时，它通过一个称作「动态链接」的过程与Windows相接。一个Windows的.EXE文件中有使用到的不同动态链接库的参考数据，所使用的函数即在那些动态链接库中。当Windows程序被加载到内存中时，程序中的呼叫被指向DLL函数的入口。如果该DLL不在内存中，就把它加载到内存中。

当您连结Windows程序以产生一个可执行文件时，您必须连结程序开发环境提供的特定「引用

链接库（import library）」。这些引用链接库包含了动态链接库名称和所有Windows函数呼叫的引用信息。连结程序使用该信息在.EXE文件中建立一个表格，在加载程序时，Windows使用它将呼叫转换为Windows函数。

WINDOWS程序设计选项

为说明Windows程序设计的多种技术，本书提供了许多范例程序。这些程序使用C语言撰写并原原本本的使用Windows API来开发程序。我将这种方法称作「古典」Windows程序设计。这是我们在1985年为Windows 1.0写程序的方法，它今天仍是写作Windows程序的有效方法。

API和内存模式

对于程序写作者来说，操作系统是由本身的API定义的。API包含了所有应用程序能够使用的操作系统函数呼叫，同时包含了相关的数据型态和结构。在Windows中，API还意味着一个特殊的程序架构，我们将在每章的开头进行研究。

一般而言，Windows API自Windows 1.0以来一直保持一致，没什么重大改变。具有Windows 98程序写作经验的Windows程序写作者会对Windows 1.0程序的原始码感觉非常熟悉。API改变的一种方式是进行增强。Windows 1.0支持不到450个函数呼叫，现在已有了上千种函数呼叫。

Windows API和它的语法的最大变化来自于从16位架构向32位架构转化的过程中。Windows从版本1.0到版本3.1使用16位Intel 8086、8088、和286微处理器上所谓的分段内存模式，由于兼容性的原因，从386开始的32位Intel微处理器也支持该模式。在这种模式下，微处理器缓存器的大小为16位，因此C的int数据型态也是16位宽。在分段内存模式下，内存地址由两个部分组成 – 一个16位段（segment）指针和一个16位偏移量（offset）指标。从程序写作者的角度看，这非常凌乱并带来了long或far指针（包括段地址和偏移量地址）和short或near指标（包括带有假定段地址的偏移量地址）的区别。

从Windows NT和Windows 95开始，Windows支持使用Intel 386、486和Pentium处理器32位模式下的32位平坦寻址内存模式。C语言的int数据型态也扩展为32位的值。为32位版本Windows编写的程序使用简单的平坦线性空间寻址的32位指针值。

用于16位版本Windows的API（Windows 1.0到Windows 3.1）现在称作Win16。用于32位版本Windows的API（Windows 95、Windows 98和所有版本的Windows NT）现在称作Win32。许多函数呼叫在从Win16到Win32的转变中保持相同，但有些需要增强。例如，图像坐标点由Win16中的16位值变为Win32中的32位值。此外，某些Win16函数呼叫返回一个包含在32位整数值中的二维坐标点。这在Win32中不可能，因此增加的新函数呼叫以不同方式运作。

所有32位版本的Windows都支持Win16 API（以确保和旧有应用程序兼容）和Win32 API（以运行新应用程序）。非常有趣的是，Windows NT与Windows 95及Windows 98的工作方式不同。在Windows NT中，Win16函数呼叫通过一个转换层被转化为Win32函数呼叫，然后被操作系统处理。在Windows 95和Windows 98中，该操作正相反：Win32函数呼叫通过转换层转换为Win16函数呼叫，再由操作系统处理。

在同一时刻有两个不同的Windows API集（至少名称不同）。Win32s （「s」代表「subset（子集）」）是一个API，允许程序写作者编写在Windows 3.1上执行的32位应用程序。该API仅支持已被Win16支持的32位函数版本。此外，Windows 95 API一度被称作Win32c（「c」代表「compatibility（兼容性）」），但该术语已被抛弃了。

现在，Windows NT和Windows 98都被认为能够支持Win32 API。然而，每个操作系统依然都

支持某些不被别的操作系统支持的某些功能特性。因为它们的相同之处是相当可观的，所以有可能编写在两个操作系统下都可执行的程序。而且，人们普遍认为这两个产品最终会合而为一。

语言选项

使用C语言和原始的API不是编写Windows 98程序的唯一方法。然而，这种方法却提供给您最佳的性能、最强大的功能和在发掘Windows特性方面最大的灵活性。可执行文件相对较小且运行时不要求外部链接库（自然，Windows DLL自身除外）。最重要的是，不管您最终以什么方式开发Windows应用程序，熟悉API会使您对Windows内部有更深入的了解。

虽然我认为学习古典的Windows程序设计对任何Windows程序写作者都是重要的，我没有必要建议使用C和API编写每个Windows应用程序。许多程序写作者，特别是那些为公司内部开发程序或在家编写娱乐程序的程序写作者喜欢轻松的开发环境，例如Microsoft Visual Basic或者Borland Delphi（它结合了对象导向的Pascal版本）。这些环境使程序写作者将精力集中于应用程序的使用者接口和相关使用者接口对象的程序代码上。要学习Visual Basic，您也许需要参考Microsoft Press的一些其它图书，例如Michael Halvorson1996年着的《Learn Visual Basic Now》。

在专业程序写作者中 – 特别是那些开发商业应用程序的程序写作者 – Microsoft Visual C++和Microsoft Foundation Class Library（MFC）是近年来流行的选择。MFC在一组C++对象类别中封装了许多Windows程序设计中的琐碎细节。Jeff Prosise的《Programming Windows with MFC, 第二版》（Microsoft Press,1999年）提供了MFC程序的写作指南。

最近，Internet和World Wide Web的流行大力推广着Sun Microsystems的Java，这是一个受C++启发却与微处理器无关的程序设计语言，而且结合了可在几个操作系统平台上执行的图形应用程序开发工具组。Microsoft Press有一本关于Microsoft J++（Microsoft的Java）开发工具的好书，《Programming Visual J++ 6.0》（1998年），由Stephen R. Davis着。

显然，很难说哪种方法更有利于开发Windows应用程序。更主要的是，也许是应用程序自身的特性决定了所使用的工具。不管您最后实际上使用什么工具写作程序，学习Windows API将使您更深入地了解Windows工作的方式。Windows是一个复杂的系统，在API上增加一个程序写作层并未减少它的复杂性，仅仅是掩盖了它，早晚您会碰到它。了解API会给您更好的补救机会。

在原始的Windows API之上的任何软件层都必定将您限制在全部功能的一个子集内。您也许发现，例如，使用Visual Basic编写应用程序非常理想，然而它不允许您做一个或两个很简单的基本工作。在这种情况下，您将不得不使用原始的API呼叫。API定义了作为Windows程序写作者所需的一切。没有什么方法比直接使用API更万能的了。

MFC尤其问题百出。虽然它大幅简化了某些工作（例如OLE），我却经常发现要让它们按我所想的去工作时，会在其它特性（例如Document/View架构）上碰壁。MFC还不是Windows程序设计者所追求的灵丹妙药，很少有人认为它是一个好的对象导向设计的模型。MFC程序写作者从他们使用的对象类别定义如何工作中受益颇深，并会发现他们经常参考MFC原始码，搞懂这些原始码是学习Windows API的好处之一。

程序开发环境

在本书中，假定您正使用Microsoft Visual C++ 6.0，标准版、专业版和企业版都可以。经济的标准版足以应付本书中的程序设计需求。Visual C++ 还是Visual Studio 6.0中的一部分。

Microsoft Visual C++ 软件包中包括C编译器和其它编译及连结Windows程序所需的文件和工具等。它还包括Visual C++ Developer Studio，一个可编辑原始码、以交谈方式建立资源（如图标和对话框）以及编辑、编译、执行和测试程序的环境。

如果您正使用Visual C++ 5.0，则需要为Windows 98和Windows NT 5.0更新表头文件和引用链接库，这些东西可从Microsoft的网站上得到。在 http://www.microsoft.com/msdn/，选择「Downloads」，然后选择「 Platform SDK」（软件开发套件），您就能在选择的目录中下载和安装更新文件。要让Microsoft Developer Studio浏览这些目录，可以从「Tool」菜单项选择「 Options」然后按下「Directories」标签。

Microsoft网站上的msdn部分代表「Microsoft Developer Network（Microsoft软件开发者网络）」。这是一个向程序写作者提供了经常更新的CD-ROM的计划，这些CD-ROM中包含了程序写作者在Windows开发中所需的最新东西。您也可以订阅MSDN，这样就避免经常得从Microsoft的网站下载文件。

API文件

本书不是Windows API权威的正式文件的替代品。那组文件不再以印刷形式出版，它仅能从CD-ROM或Internet上取得。

当您安装Visual C++ 6.0时，您将得到一个包括API文件的在线求助系统。您可通过订阅MSDN或使用Microsoft网站上的在线求助系统更新该文件。连接到http://www.microsoft.com/msdn/，并选择「MSDN Library Online」。

在Visual C++ 6.0中，从「Help」菜单项选择「Contents」项目开启MSDN窗口。API文件按树形结构组织，寻找标有「 Platform SDK」的部分，所有在本书中引用的文件都来自于该部分。我将向您介绍如何从「 Platform SDK」开始寻找以斜线分层分门别类的文件的位置。（我知道「Platform SDK」是整个MSDN知识库中较为晦涩的部分，但我敢保证那是Windows程序设计的基本核心。）例如，对于如何在Windows程序中使用鼠标的文件，您可参考/ Platform SDK / User Interface Services / User Input / Mouse Input。

我在前面提到Windows大致分为Kernel、User和GDI子系统。kernel接口在/ Platform SDK / Windows Base Services中，User界面函数在 / Platform SDK / User Interface Services中，GDI位于 / Platform SDK / Graphics and Multimedia Services / GDI中。

编写第一个WINDOWS程序

现在是开始写些程序的时候了。为了便于对比，让我们以一个非常短的Windows程序和一个简短的文字模式程序开始。这会帮助我们找到使用开发环境并感受建立和编译程序机制的正确方向。

文字模式（Character-Mode）模型

程序写作者们喜爱的一本书是《The C Programming Language》（Prentice Hall，1978年和1988年），由Brian W. Kernighan和Dennis M. Ritchie（亲切地称为K&R）编着。该书的第一章以一个显示「hello, world」的C语言程序开始。

这里是在《The C Programming Language》第一版第6页中出现的程序：

```
   main ()
  {
    printf ("hello, world\n") ;
  }
```

以前C程序写作者在使用printf等C执行期链接库函数时，无需先声明它们。但这是90年代，我们愿意给编译器一个在我们的程序中标出错误的机会。这里是在K&R第二版中修正的程序：

```
#include <stdio.h>
main ()
```

```
{
  printf ("hello, world\n") ;
}
```

该程序仍然是那么短。但它可通过编译并执行得很好，但当今许多程序写作者更愿意清楚地说明main函数的返回值，在这种情况下ANSI C规定该函数必须返回一个值：

```
#include <stdio.h>
int main ()
{
  printf ("hello, world\n") ;
  return 0 ;
}
```

我们还可以包括main的参数，把程序弄得更长一些，但让我们暂且这样就好了－包括一个include声明、程序的进入点、一个对执行期链接库函数的呼叫和一个return语句。

同样效果的Windows程序

Windows关于「hello, world」程序的等价程序有和文字模式版本完全相同的组件。它有一个include声明、一个程序进入点、一个函数呼叫和一个return语句。下面便是该程序：

```
/*------------------------------------------------------------------------
HelloMsg.c -- Displays "Hello, Windows 98!" in a message box
   (c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  MessageBox (NULL, TEXT ("Hello, Windows 98!"), TEXT ("HelloMsg"), 0);
  return 0 ;
}
```

在剖析该程序之前，让我们看一下在Visual C++ Developer Studio中建立新程序的方式。

首先，从File菜单中选New。在 New对话框中，单击Projects页面标签，选择 Win32 Application。在Location栏中，选择一个子目录，在 Project Name栏中，输入该项目的名称，此时该名称是HelloMsg，这便是在 Location栏中显示的目录的子目录。Create New Workspace复选框应该勾起来，Platforms部分应该显示 Win32，选择OK。

将会出现一个标题为Win32 Application - Step 1 Of 1的对话框，指出要建立一个Empty Project，并按下Finish按钮。

从File菜单中再次选择New。在 New对话框中，选择Files页面标签，选择 C++ Source File。Add To Project复选框应被选中，并应显示HelloMsg。在 File Name栏中输入HelloMsg.c，选中OK。

现在您可输入上面所示的HELLOMSG.C文件，您也可以选择Insert菜单和 File As Text选项从本书附带的CD-ROM上复制HELLOMSG.C的内容。

从结构上说，HELLOMSG.C与K&R的「hello,world」程序是相同的。表头文件STDIO.H已被WINDOWS.H所代替，进入点main被WinMain所代替，而且C语言执行时期链接库函数printf被Windows API函数MessageBox所代替。然而，在程序中有许多新东西，包括几个陌生的大写标识符。

让我们从头开始。

表头文件

HELLOMSG.C以一个前置处理器指示命令开始，实际上在每个用C编写的Windows程序的开头都可看到：

```
#include <windows.h>
```

WINDOWS.H是主要的含入文件，它包含了其它Windows表头文件，这些表头文件的某些也包含了其它表头文件。这些表头文件中最重要的和最基本的是：

WINDEF.H 基本型态定义。

WINNT.H 支持Unicode的型态定义。
WINBASE.H Kernel函数。
WINUSER.H 使用者接口函数。
WINGDI.H 图形设备接口函数。

这些表头文件定义了Windows的所有数据型态、函数呼叫、数据结构和常数标识符，它们是Windows文件中的一个重要部分。使用Visual C++ Developer Studio的**Edit**菜单中的**Find in Files**搜索这些表头文件非常方便。您还可以在Developer Studio中打开这些表头文件并直接阅读它们。

程序进入点

正如在C程序中的进入点是函数main一样，Windows程序的进入点是WinMain，总是像这样出现：

```
int WINAPI WinMain (HINSTANCE hInstance,HINSTANCE hPrevInstance,
                    PSTR szCmdLine,int iCmdShow)
```

该进入点在/ Platform SDK / User Interface Services / Windowing / Windows / Window Reference / Window Functions中有说明。它在WINBASE.H中声明如下：

```
int  WINAPI  WinMain(HINSTANCE hInstance,  HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,  int nShowCmd  );
```

您会注意到我在HELLOMSG.C中做了许多小改动。第三个参数在WINBASE.H中定义为LPSTR，我将它改为PSTR。这两种数据型态都定义在WINNT.H中，作为指向字符串的指针。LP前缀代表「长指针」，这是16位Windows下的产物。

我还在WinMain声明中改变了两个参数的名称。许多Windows程序中的变量名使用一种称作「匈牙利表示法」的命名系统，该系统在变量名称前面增加了表示变量数据型态的短前缀，我将在第三章更详细地讨论这个概念。现在仅需记住前缀i表示int、sz表示「以零结束的字符串」。

WinMain函数声明为返回一个int值。WINAPI标识符在WINDEF.H定义，语句如下：

```
   #define WINAPI __stdcall
```

该语句指定了一个呼叫约定，包括如何生产机械码以在堆栈中放置函数呼叫的参数。许多Windows函数呼叫声明为WINAPI。

WinMain的第一个参数被称作「执行实体句柄」。在Windows程序设计中，句柄仅是一个应用程序用来识别某些东西的数字。在这种情况下，该句柄唯一地标识该程序，还需要它在其它Windows函数呼叫中作为参数。在Windows的早期版本中，当同时运行同一程序多次时，您便创建了该程序的「多个执行实体（multiple instances）」。同一应用程序的所有执行实体共享程序和只读的内存（通常是例如菜单和对话框模板的资源）。程序通过检查hPrevInstance参数就能够确定自身的其它执行实体是否正在运行。然后它可以略过一些繁杂的工作并从前面的执行实体将某些数据移到自己的数据区域。

在32位Windows版本中，该概念已被抛弃。传给WinMain的第二个参数总是NULL（定义为0）。

WinMain的第三个参数是用于执行程序的命令列。某些Windows应用程序利用它在程序启动时将文件加载内存。WinMain的第四个参数指出程序最初显示的方式，可以是正常的或者是最大化地充满整个画面，或者是最小化显示在工作列中。我们将在 第三章中介绍使用该参数的方法。

MessageBox函数

MessageBox函数用于显示短信息。虽然，MessageBox显示的小窗口不具有什么功能，实际上它被认为是一个对话框。

MessageBox的第一个参数通常是窗口句柄，我们将在第三章介绍其含义。第二个参数是在消息框主体中显示的字符串，第三个参数是出现在消息框标题栏上的字符串。在HELLMSG.C中，这些文字字符串的每一个都被封装在一个TEXT宏中。通常您不必将所有字符串都封装在TEXT宏中，但如果想将您的程序转换为Unicode字符集，这确是一个好主意。我将在第二章详细讨论该问题。

MessageBox的第四个参数可以是在WINUSER.H中定义的一组以前缀MB_开始的常数的组合。您可从第一组中选择一个常数指出希望在对话框中显示的按钮：

```
#define MB_OK 0x00000000L
#define MB_OKCANCEL 0x00000001L
#define MB_ABORTRETRYIGNORE 0x00000002L
#define MB_YESNOCANCEL 0x00000003L
#define MB_YESNO 0x00000004L
#define MB_RETRYCANCEL 0x00000005L
```

如果在HELLOMSG中将第四个参数设置为0，则仅显示「OK」按钮。可以使用C语言的OR（|）操作符号将上面显示的一个常数与代表内定按钮的常数组合：

```
#define MB_DEFBUTTON1    0x00000000L
#define MB_DEFBUTTON2    0x00000100L
#define MB_DEFBUTTON3    0x00000200L
#define MB_DEFBUTTON4    0x00000300L
```

还可以使用一个常数指出消息框中图标的外观：

```
#define MB_ICONHAND       0x00000010L
#define MB_ICONQUESTION    0x00000020L
#define MB_ICONEXCLAMATION 0x00000030L
#define MB_ICONASTERISK    0x00000040L
```

这些图标中的某些有替代名称：

```
#define  MB_ICONWARNING    MB_ICONEXCLAMATION
#define  MB_ICONERROR     MB_ICONHAND
#define  MB_ICONINFORMATION   MB_ICONASTERISK
#define  MB_ICONSTOP     MB_ICONHAND
```

虽然只有少数其它MB_常数，但您可以自己参考表头文件或 / Platform SDK / User Interface Services / Windowing / Dialog Boxes / Dialog Box Reference / Dialog Box Functions里的文件。

在本程序中，MessageBox返回数值1，但更严格地说它返回IDOK，IDOK在WINUSER.H中定义，等于1。根据在消息框中显示的其它按钮，MessageBox函数还可返回IDYES、IDNO、IDCANCEL、IDABORT、 IDRETRY或IDIGNORE。

这个小的Windows程序真的与K&R的「hello, world」程序有着同等效果吗？您也许认为不是，因为MessageBox函数并没有「hello, world」中printf函数所具有的潜在格式化文字能力。但我们将在下一章中看到编写类似printf的MessageBox版本的方法。

编译、连结和执行

当您准备编译HELLOMSG时，您可从「Build」菜单中选择「 Build Hellomsg.exe」，或者按F7，或者在「 Build」工具列中选择「Build」图标。（该图标的外观显示在「 Build」菜单中。如果当前没有显示「Build」工具列，您可从「 Tools」菜单中选择「Customize」并选择「 Toolbars」页面标签，选中「Build」或者「 Build MiniBar」。）

另一种方法，您可从「Build」菜单中选择「Execute Hellomsg.exe」，或者按「Ctrl+F5」，或

者在「 Build」工具列单击「Execute Program」图标（该图标看上去像一个红的感叹号），就会弹出一个消息框询问是否编译该程序。

正常情况下，在编译阶段，编译器从C原始码文件产生一个.OBJ（目标）文件。在连结阶段，连结程序结合.OBJ文件和.LIB（库）文件以建立.EXE（可执行）文件。通过在「 Project」页面标签上选择「Settings」并单击「 Link」页面标签可以查看这些库文件的列表。特别地，您会注意到KERNEL32.LIB、USER32.LIB和GDI32.LIB。这些是三个主要Windows子系统的「引用链接库」。它们包含了动态链接库的名称以及放进.EXE文件的引用信息。Windows使用该信息处理程序对KERNEL32.DLL、USER32.DLL、GDI32.DLL动态链接库中函数的呼叫。

在Visual C++ Developer Studio中，您可用不同的设定编译和连结程序。内定情况下，它们是「Debug」和「Release」。可执行文件被存放在以这些名称命名的子目录下。在Debug设定下，信息被附加到 .EXE文件中，这些信息有助于测试程序和追踪原始码。

如果您喜欢在命令列下工作，附上的CD-ROM包含所有范例程序的.MAK（make）文件。（可通过「 Tools」菜单选择「Options」，再选择「 Build」页面标签，来告诉Developer Studio生成make文件。这里有一个复选框需要勾选）。您需要执行在Developer Studio的BIN子目录下的VCVARS32.BAT来设置环境变量。要从命令列执行make文件，可以转到HELLOMSG目录并执行：

NMAKE /f HelloMsg.mak CFG="HelloMsg - Win32 Debug"

或者

NMAKE /f HelloMsg.mak CFG="HelloMsg - Win32 Release"

然后您可通过输入：

DEBUG\HELLOMSG

或者

class=settRELEASE\HELLOMSG

从命令列执行.EXE文件。

我已经在本书附上的CD-ROM中对项目文件中的内定Debug设定做了一个改动。在「 Project Settings」对话框中，选择「C/C++」页面标签后，在「 Preprocessor Definitions」栏中，我已定义了标识符UNICODE。我将在下一章中对此有更多的解释。

第二章 Unicode简介

在第一章中，我已经预告，C语言中在Microsoft Windows程序设计中扮演着重要角色的任何部分都会讲述到，您也许在传统文字模式程序设计中还尚未遇到过这些问题。宽字符集和Unicode差不多就是这样的问题。

简单地说，Unicode扩展自ASCII字符集。在严格的ASCII中，每个字符用7位表示，或者计算机上普遍使用的每字符有8位宽；而Unicode使用全16位字符集。这使得Unicode能够表示世界上所有的书写语言中可能用于计算机通讯的字符、象形文字和其它符号。Unicode最初打算作为ASCII的补充，可能的话，最终将代替它。考虑到ASCII是计算机中最具支配地位的标准，所以这的确是一个很高的目标。

Unicode影响到了计算机工业的每个部分，但也许会对操作系统和程序设计语言的影响最大。从这方面来看，我们已经上路了。Windows NT从底层支持Unicode（不幸的是，Windows 98只是小部分支持Unicode）。先天即被ANSI束缚的C程序设计语言通过对宽字符集的支持来支持Unicode。下面将详细讨论这些内容。

自然，作为程序写作者，我们通常会面对许多繁重的工作。我已试图透过使本书中的所有程序「Unicode化」来减轻负担。其含义会随着本章对Unicode的讨论而清晰起来。

字符集简史

虽然不能确定人类开始讲话的时间，但书写已有大约6000年的历史了。实际上，早期书写的内容是象形文字。每个字符都对应于发声的字母表则出现于大约3000年前。虽然人们过去使用的多种书写语言都用得好好的，但19世纪的几个发明者还是看到了更多的需求。Samuel F. B. Morse在1838年到1854年间发明了电报，当时他还发明了一种电报上使用的代码。字母表中的每个字符对应于一系列短的和长的脉冲（点和破折号）。虽然其中大小写字母之间没有区别，但数字和标点符号都有了自己的代码。

Morse代码并不是以其它图画的或印刷的象形文字来代表书写语言的第一个例子。1821年到1824年之间，年轻的Louis Braille受到在夜间读写信息的军用系统的启发，发明了一种代码，它用纸上突起的点作为代码来帮助盲人阅读。Braille代码实际上是一种6位代码，它把字符、常用字母组合、常用单字和标点进行编码。一个特殊的escape代码表示后续的字符代码应解释为大写。一个特殊的shift代码允许后续代码被解释为数字。

Telex代码，包括Baudot （以一个法国工程师命名，该工程师死于1903年）以及一种被称为CCITT #2的代码（1931年被标准化），都是包括字符和数字的5位代码。

美国标准

早期计算机的字符码是从Hollerith卡片（号称不能被折迭、卷曲或毁伤）发展而来的，该卡片由Herman Hollerith发明并首次在1890年的美国人口普查中使用。6位字符码系统BCDIC（Binary-Coded Decimal Interchange Code：二进制编码十进制交换编码）源自Hollerith代码，在60年代逐步扩展为8位EBCDIC，并一直是IBM大型主机的标准，但没使用在其它地方。

美国信息交换标准码（ASCII：American Standard Code for Information Interchange）起始于50年代后期，最后完成于1967年。开发ASCII的过程中，在字符长度是6位、7位还是8位的问题上产生了很大的争议。从可靠性的观点来看不应使用替换字符，因此ASCII不能是6位编码，但由于费用的原因也排除了8位版本的方案（当时每位的储存空间成本仍很昂贵）。这样，最终的字符码就有26个小写字母、26个大写字母、10个数字、32个符号、33个句柄和一个空格，总共128个字符码。ASCII现在记录在ANSI X3.4-1986字符集 – 用于信息交换的7位美国国家标准码（7-Bit ASCII: 7-Bit American National Standard Code for Information Interchange），由美国国家标准协会（American National Standards Institute）发布。图2-1中所示的ASCII字符码与ANSI文件中的格式相似。

ASCII有许多优点。例如，26个字母代码是连续的（在EBCDIC代码中就不是这样的）；大写字母和小写字母可通过改变一位数据而相互转化；10个数字的代码可从数值本身方便地得到（在BCDIC代码中，字符「0」的编码在字符「9」的后面!）

最棒的是，ASCII是一个非常可靠的标准。在键盘、视讯显示卡、系统硬件、打印机、字体文件、操作系统和Internet上，其它标准都不如ASCII码流行而且根深蒂固。

	0-	1-	2-	3-	4-	5-	6-	7-
-0	NUL	DLE	SP	0	@	P	`	p
-1	SOH	DC1	!	1	A	Q	a	q
-2	STX	DC2	"	2	B	R	b	r
-3	ETX	DC3	#	3	C	S	c	s
-4	EOT	DC4	$	4	D	T	d	t
-5	ENQ	NAK	%	5	E	U	e	u
-6	ACK	SYN	&	6	F	V	f	v
-7	BEL	ETB	'	7	G	W	g	w
-8	BS	CAN	(	8	H	X	h	x
-9	HT	EM	)	9	I	Y	I	y
-A	LF	SUB	*	:	J	Z	j	z
-B	VT	ESC	+	;	K	[	k	{
-C	FF	FS	,	<	L	\	l	|
-D	CR	GS	-	=	M	]	m	}
-E	SO	RS	.	>	N	^	n	~
-F	SI	US	/	?	O	_	o	DEL

图 2-1 ASCII 字符集

国际方面

ASCII的最大问题就是该缩写的第一个字母。ASCII是一个真正的美国标准，所以它不能良好满足其它讲英语国家的需要。例如英国的英镑符号（£）在哪里?

英语使用拉丁（或罗马）字母表。在使用拉丁语字母表的书写语言中，英语中的单词通常很少需要重音符号（或读音符号）。即使那些传统惯例加上读音符号也无不当的英语单字，例如cⓞperate或者résumé，拼写中没有读音符号也会被完全接受。

但在美国以南、以北，以及大西洋地区的许多国家，在语言中使用读音符号很普遍。这些重音

符号最初是为使拉丁字母表适合这些语言读音不同的需要。在远东或西欧的南部旅游，您会遇到根本不使用拉丁字母的语言，例如希腊语、希伯来语、阿拉伯语和俄语（使用斯拉夫字母表）。如果您向东走得更远，就会发现中国象形汉字，日本和朝鲜也采用汉字系统。

ASCII的历史开始于1967年，此后它主要致力于克服其自身限制以更适合于非美国英语的其它语言。例如，1967年，国际标准化组织（ISO：International Standards Organization）推荐一个ASCII的变种，代码0x40、0x5B、0x5C、0x5D、0x7B、0x7C和0x7D「为国家使用保留」，而代码0x5E、0x60和0x7E标为「当国内要求的特殊字符需要8、9或10个空间位置时，可用于其它图形符号」。这显然不是一个最佳的国际解决方案，因为这并不能保证一致性。但这却显示了人们如何想尽办法为不同的语言来编码的。

扩展ASCII

在小型计算机开发的初期，就已经严格地建立了8位字节。因此，如果使用一个字节来保存字符，则需要128个附加的字符来补充ASCII。1981年，当最初的IBM PC推出时，视讯卡的ROM中烧有一个提供256个字符的字符集，这也成为IBM标准的一个重要组成部分。

最初的IBM扩展字符集包括某些带重音的字符和一个小写希腊字母表（在数学符号中非常有用），还包括一些块型和线状图形字符。附加的字符也被添加到ASCII控制字符的编码位置，这是因为大多数控制字符都不是拿来显示用的。

该IBM扩展字符集被烧进无数显示卡和打印机的ROM中，并被许多应用程序用于修饰其文字模式的显示方式。不过，该字符集并没有为所有使用拉丁字母表的西欧语言提供足够多的带重音字符，而且也不适用于Windows。Windows不需要图形字符，因为它有一个完全图形化的系统。

在Windows 1.0（1985年11月发行）中，Microsoft没有完全放弃IBM扩展字符集，但它已退居第二重要位置。因为遵循了ANSI草案和ISO标准，纯Windows字符集被称作「ANSI字符集」。ANSI草案和ISO标准最终成为ANSI/ISO 8859-1-1987，即「American National Standard for Information Processing-8-Bit Single-Byte Coded Graphic Character Sets-Part 1: Latin Alphabet No 1」，通常也简写为「Latin 1」。

在Windows 1.0的《Programmer's Reference》中印出了ANSI字符集的最初版本，如图2-2所示。

	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
0	□	□		0	@	P	`	p	□	□		°	À	Đ	à	ð
1	□	□	!	1	A	Q	a	q	□	□	¡	±	Á	Ñ	á	ñ
-2	□	□	"	2	B	R	b	r	□	□	¢	²	Â	Ò	â	ò
-3	□	□	#	3	C	S	c	s	□	□	£	³	Ã	Ó	ã	ó
4	□	□	$	4	D	T	d	t	□	□	¤	´	Ä	Ô	ä	ô
-5	□	□	%	5	E	U	e	u	□	□	¥	µ	Å	Õ	à	õ
6	□	□	&	6	F	V	f	v	□	□	¦	¶	Æ	Ö	æ	ö
-7	□	□	'	7	G	W	g	w	□	□	§	·	Ç	□	ç	□
-8	□	□	(	8	H	X	h	x	□	□	¨	¸	È	Ø	è	ø
9	□	□	)	9	I	Y	I	y	□	□	©	¹	É	Ù	é	ù
-A	□	□	*	:	J	Z	j	z	□	□	ª	º	Ê	Ú	é	ú
B	□	□	+	;	K	[	k	{	□	□	«	»	Ë	Û	ë	û
-C	□	□	,	<	L	\	l	|	□	□	¬	¼	Ì	Ü	ì	ü
-D	□	□	-	=	M	]	m	}	□	□	-	½	Í	Ý	í	ý
-E	□	□	.	>	N	^	n	~	□	□	®	¾	Î	Þ	î	þ
-F	□	□	/	?	O	_	o	DEL	□	□	¯	¿	Ï	ß	ï	ÿ

图2-2 Windows ANSI字符集（基于ANSI/ISO 8859-1）

空方框表示该位置未定义字符。这与ANSI/ISO 8859-1的最终定义一致。ANSI/ISO 8859-1仅显示了图形字符，而没有控制字符，因此没有定义DEL。此外，代码0xA0定义为一个非断开的空格（这意味着在编排格式时，该字符不用于断开一行），代码0xAD是一个软连字符（表示除非在行尾断开单词时使用，否则不显示）。此外，ANSI/ISO 8859-1将代码0xD7定义为乘号（*），0xF7为除号（/）。Windows中的某些字体也定义了从0x80到0x9F的某些字符，但这些不是ANSI/ISO 8859-1标准的一部分。

MS-DOS 3.3（1987年4月发行）向IBM PC用户引进了代码页（code page）的概念，Windows也使用此概念。代码页定义了字符的映像代码。最初的IBM字符集被称作代码页437，或者「MS-DOS Latin US）。代码页850就是「MS-DOS Latin 1」，它用附加的带重音字母（但**不是**图2-2所示的Latin 1 ISO/ANSI标准）代替了一些线形字符。其它代码页被其它语言定义。最低的128个代码总是相同的；较高的128个代码取决于定义代码页的语言。

在MS-DOS中，如果用户为PC的键盘、显示卡和打印机指定了一个代码页，然后在PC上创建、编辑和打印文件，一切都很正常，每件事都会保持一致。然而，如果用户试图与使用不同代码页的用户交换文件，或者在机器上改变代码页，就会产生问题。字符码与错误的字符相关联。应用程序能够将代码页信息与文件一起保存来试图减少问题的产生，但该策略包括了某些在代码页间转换的工作。

虽然代码页最初仅提供了不包括带重音符号字母的附加拉丁字符集，但最终代码页的较高的128个字符还是包括了完整的非拉丁字母，例如希伯来语、希腊语和斯拉夫语。自然，如此多样会导致代码页变得混乱；如果少数带重音的字母未正确显示，那么整个文字便会混乱不堪而不可阅读。

代码页的扩展正是基于所有这些原因，但是还不够。斯拉夫语的MS-DOS代码页855与斯拉夫语的Windows代码页1251以及斯拉夫语的Macintosh代码页10007不同。每个环境下的代码页都是对该环境所作的标准字符集修正。IBM OS/2也支援多种EBCDIC代码页。

但等一下，你会发现事情变得更糟糕。

双字节字符集

迄今为止，我们已经看到了256个字符的字符集。但中国、日本和韩国的象形文字符号有大约21,000个。如何容纳这些语言而仍保持和ASCII的某种兼容性呢?

解决方案（如果这个说法正确的话）是双字节字符集（DBCS：double-byte character set）。DBCS从256代码开始，就像ASCII一样。与任何行为良好的代码页一样，最初的128个代码是ASCII。然而，较高的128个代码中的某些总是跟随着第二个字节。这两个字节一起（称作首字节和跟随字节）定义一个字符，通常是一个复杂的象形文字。

虽然中文、日文和韩文共享一些相同的象形文字，但显然这三种语言是不同的，而且经常是同一个象形文字在三种不同的语言中代表三件不同的事。Windows支持四个不同的双字节字符集：代码页932（日文）、936（简体中文）、949（韩语）和950（繁体汉字）。只有为这些国家（地区）生产的Windows版本才支持DBCS。

双字符集问题并不是说字符由两个字节代表。问题在于一些字符（特别是ASCII字符）由1个字节表示。这会引起附加的程序设计问题。例如，字符串中的字符数不能由字符串的字节数决定。必须剖析字符串来决定其长度，而且必须检查每个字节以确定它是否为双字节字符的首字节。如果有一个指向DBCS字符串中间的指针，那么该字符串前一个字符的地址是什么呢？惯用的解决方案是从开始的指针分析该字符串!

Unicode解决方案

我们面临的基本问题是世界上的书写语言不能简单地用256个8位代码表示。以前的解决方案包括代码页和DBCS已被证明是不能满足需要的，而且也是笨拙的。那什么才是真正的解决方案呢?

身为程序写作者，我们经历过这类问题。如果事情太多，用8位数值已经不能表示，那么我们就试更宽的值，例如16位值。而且这很有趣的，正是Unicode被制定的原因。与混乱的256个字符代码映像，以及含有一些1字节代码和一些2字节代码的双字节字符集不同，Unicode是统一的16位系统，这样就允许表示65,536个字符。这对表示所有字符及世界上使用象形文字的语言，包括一系列的数学、符号和货币单位符号的集合来说是充裕的。

明白Unicode和DBCS之间的区别很重要。Unicode使用（特别在C程序设计语言环境里）「宽字符集」。「Unicode中的每个字符都是16位宽而不是8位宽。」在Unicode中，没有单单使用8位数值的意义存在。相比之下，在双字节字符集中我们仍然处理8位数值。有些字节自身定义字符，而某些字节则显示需要和另一个字节共同定义一个字符。

处理DBCS字符串非常杂乱，但是处理Unicode文字则像处理有秩序的文字。您也许会高兴地知道前128个Unicode字符（16位代码从0x0000到0x007F）就是ASCII字符，而接下来的128个Unicode字符（代码从0x0080到0x00FF）是ISO 8859-1对ASCII的扩展。Unicode中不同部分的字符都同样基于现有的标准。这是为了便于转换。希腊字母表使用从0x0370到0x03FF的代码，斯拉夫语使用从0x0400到0x04FF的代码，美国使用从0x0530到0x058F的代码，希伯来语使用从0x0590到0x05FF的代码。中国、日本和韩国的象形文字（总称为CJK）占用了从0x3000到0x9FFF的代码。

Unicode的最大好处是这里只有一个字符集，没有一点含糊。Unicode实际上是个人计算机行业中几乎每个重要公司共同合作的结果，并且它与ISO 10646-1标准中的代码是一一对应的。Unicode的重要参考文献是《The Unicode Standard，Version 2.0》（Addison-Wesley出版社，1996年）。这是一本特别的书，它以其它文件少有的方式显示了世界上书写语言的丰富性和多样性。此外，该书还提供了开发Unicode的基本原理和细节。

Unicode有缺点吗？当然有。Unicode字符串占用的内存是ASCII字符串的两倍。（然而压缩文件有助于极大地减少文件所占的磁盘空间。）但也许最糟的缺点是：人们相对来说还不习惯使用Unicode。身为程序写作者，这就是我们的工作。

宽字符和 C

对C程序写作者来说，16位字符的想法的确让人扫兴。一个char和一个字节同宽是最不能确定的事情之一。没几个程序写作者清楚ANSI/ISO 9899-1990，这是「美国国家标准程序设计语言－C」（也称作「ANSI C」）通过一个称作「宽字符」的概念来支持用多个字节代表一字符的字符集。这些宽字符与常用的字符完美地共存。

ANSI C也支持多字节字符集，例如中文、日文和韩文版本Windows支持的字符集。然而，这些多字节字符集被当成单字节构成的字符串看待，只不过其中一些字符改变了后续字符的含义而已。多字节字符集主要影响C语言程序执行时期链接库函数。相比之下，宽字符比正常字符宽，而且会引起一些编译问题。

宽字符不需要是Unicode。Unicode是一种可能的宽字符集。然而，因为本书的焦点是Windows而不是C执行的理论，所以我将把宽字符和Unicode作为同义语。

Char数据型态

假定我们都非常熟悉在C程序中使用char数据型态来定义和储存字符跟字符串。但为了便于理解C如何处理宽字符，让我们先回顾一下可能在Win32程序中出现的标准字符定义。

下面的语句定义并初始化了一个只包含一个字符的变量：

```
char c = 'A' ;
```

变量c需要1个字节来保存，并将用十六进制数0x41初始化，这是字母A的ASCII代码。

您可以像这样定义一个指向字符串的指针：

```
char * p ;
```

因为Windows是一个32位操作系统，所以指针变量p需要用4个字节保存。您还可初始化一个指向字符串的指针：

```
char * p = "Hello!" ;
```

像前面一样，变量p也需要用4个字节保存。该字符串保存在静态内存中并占用7个字节 – 6个字节保存字符串，另1个字节保存终止符号0。

您还可以像这样定义字符数组：

```
char a[10] ;
```

在这种情况下，编译器为该数组保留了10个字节的储存空间。表达式sizeof（a）将返回10。如果数组是整体变量（即在所有函数外定义），您可使用像下面的语句来初始化一个字符数组：

```
char a[] = "Hello!" ;
```

如果您将该数组定义为一个函数的区域变量，则必须将它定义为一个static变量，如下：

```
static char a[] = "Hello!" ;
```

无论哪种情况，字符串都储存在静态程序内存中，并在末尾添加0，这样就需要7个字节的储存空间。

宽字符

Unicode或者宽字符都没有改变char数据型态在C中的含义。char继续表示1个字节的储存空间，sizeof （char）继续返回1。理论上，C中1个字节可比8位长，但对我们大多数人来说，1个字节（也就是1个char）是8位宽。

C中的宽字符基于wchar_t数据型态，它在几个表头文件包括WCHAR.H中都有定义，像这样：

```
typedef unsigned short wchar_t ;
```

因此，wchar_t数据型态与无符号短整数型态相同，都是16位宽。

要定义包含一个宽字符的变量，可使用下面的语句：

```
wchar_t c = 'A' ;
```

变量c是一个双字节值0x0041，是Unicode表示的字母A。（然而，因为Intel微处理器从最小的字节开始储存多字节数值，该字节实际上是以0x41、0x00的顺序保存在内存中。如果检查Unicode文字的计算机储存应注意这一点。）

您还可定义指向宽字符串的指针：

```
wchar_t * p = L"Hello!" ;
```

注意紧接在第一个引号前面的大写字母L（代表「long」）。这将告诉编译器该字符串按宽字符

保存 – 即每个字符占用2个字节。通常，指针变量p要占用4个字节，而字符串变量需要14个字节 – 每个字符需要2个字节，末尾的0还需要2个字节。

同样，您还可以用下面的语句定义宽字符数组：

static wchar_t a[] = L"Hello!" ;

该字符串也需要14个字节的储存空间，sizeof (a) 将返回14。索引数组a可得到单独的字符。a[1] 的值是宽字符「e」，或者0x0065。

虽然看上去更像一个印刷符号，但第一个引号前面的L非常重要，并且在两个符号之间必须没有空格。只有带有L，编译器才知道您需要将字符串存为每个字符2字节。稍后，当我们看到使用宽字符串而不是变量定义时，您还会遇到第一个引号前面的L。幸运的是，如果忘记了包含L，C编译器通常会给提出警告或错误信息。

您还可在单个字符文字前面使用L前缀，来表示它们应解释为宽字符。如下所示：

wchar_t c = L'A' ;

但通常这是不必要的，C编译器会对该字符进行扩充，使它成为宽字符。

宽字符链接库函数

我们都知道如何获得字符串的长度。例如，如果我们已经像下面这样定义了一个字符串指针：

char * pc = "Hello!" ;

我们可以呼叫

iLength = strlen (pc) ;

这时变量iLength将等于6，也就是字符串中的字符数。

太好了！现在让我们试着定义一个指向宽字符的指针：

wchar_t * pw = L"Hello!" ;

再次呼叫strlen ：

iLength = strlen (pw) ;

现在麻烦来了。首先，C编译器会显示一条警告消息，可能是这样的内容：

'function' : incompatible types - from 'unsigned short *' to 'const char *'

这条消息的意思是：声明strlen函数时，该函数应接收char类型的指标，但它现在却接收了一个unsigned short类型的指标。您仍然可编译并执行该程序，但您会发现iLength等于1。为什么?

字符串「Hello!」中的6个字符占用16位：

0x0048 0x0065 0x006C 0x006C 0x006F 0x0021

Intel处理器在内存中将其存为：

48 00 65 00 6C 00 6C 00 6F 00 21 00

假定strlen函数正试图得到一个字符串的长度，并把第1个字节作为字符开始计数，但接着假定如果下一个字节是0，则表示字符串结束。

这个小练习清楚地说明了C语言本身和执行时期链接库函数之间的区别。编译器将字符串L"Hello!" 解释为一组16位短整数型态数据，并将其保存在wchar_t数组中。编译器还处理数组索引和sizeof操作符，因此这些都能正常工作，但在连结时才添加执行时期链接库函数，例如strlen。这

些函数认为字符串由单字节字符组成。遇到宽字符串时，函数就不像我们所希望那样执行了。

您可能要说：「噢，太麻烦了！」现在每个C语言链接库函数都必须重写以接受宽字符。但事实上并不是每个C语言链接库函数都需要重写，只是那些有字符串参数的函数才需要重写，而且也不用由您来完成。它们已经重写完了。

strlen函数的宽字符版是wcslen（wide-character string length：宽字符串长度），并且在STRING.H（其中也说明了strlen）和WCHAR.H中均有说明。strlen函数说明如下：

size_t __cdecl strlen (const char *) ;

而wcslen函数则说明如下：

size_t __cdecl wcslen (const wchar_t *) ;

这时我们知道，要得到宽字符串的长度可以呼叫

iLength = wcslen (pw) ;

函数将返回字符串中的字符数6。请记住，改成宽字节后，字符串的字符长度不改变，只是位组长度改变了。

您熟悉的所有带有字符串参数的C执行时期链接库函数都有宽字符版。例如，wprintf是printf的宽字符版。这些函数在WCHAR.H和含有标准函数说明的表头文件中说明。

维护单一原始码

当然，使用Unicode也有缺点。第一点也是最主要的一点是，程序中的每个字符串都将占用两倍的储存空间。此外，您将发现宽字符执行时期链接库中的函数比常规的函数大。出于这个原因，您也许想建立两个版本的程序 – 一个处理ASCII字符串，另一个处理Unicode字符串。最好的解决办法是维护既能按ASCII编译又能按Unicode编译的单一原始码文件。

虽然只是一小段程序，但由于执行时期链接库函数有不同的名称，您也要定义不同的字符，这将在处理前面有L的字符串文字时遇到麻烦。

一个办法是使用Microsoft Visual C++包含的TCHAR.H表头文件。该表头文件不是ANSI C标准的一部分，因此那里定义的每个函数和宏定义的前面都有一条底线。TCHAR.H为需要字符串参数的标准执行时期链接库函数提供了一系列的替代名称（例如，_tprintf和_tcslen）。有时这些名称也称为「通用」函数名称，因为它们既可以指向函数的Unicode版也可以指向非Unicode版。

如果定义了名为_UNICODE的标识符，并且程序中包含了TCHAR.H表头文件，那么_tcslen就定义为wcslen：

#define _tcslen wcslen

如果没有定义UNICODE，则_tcslen定义为strlen：

#define _tcslen strlen

等等。TCHAR.H还用一个新的数据型态TCHAR来解决两种字符数据型态的问题。如果定义了_UNICODE标识符，那么TCHAR就是wchar_t：

typedef wchar_t TCHAR ;

否则，TCHAR就是Char：

typedef char TCHAR ;

现在开始讨论字符串文字中的L问题。如果定义了_UNICODE标识符，那么一个称作__T的宏就

定义如下：

#define __T(x) L##x

这是相当晦涩的语法，但合乎ANSI C标准的前置处理器规范。那一对井字号称为「粘贴符号（token paste）」，它将字母L添加到宏参数上。因此，如果宏参数是"Hello!"，则L##x就是L"Hello!"。

如果没有定义_UNICODE标识符，则__T宏只简单地定义如下：

#define __T(x) x

此外，还有两个宏与__T定义相同：

#define _T（x）__T（x）

#define _TEXT（x）__T（x）

在Win32 console程序中使用哪个宏，取决于您喜欢简洁还是详细。基本地，必须按下述方法在_T或_TEXT宏内定义字符串文字：

_TEXT ("Hello!")

这样做的话，如果定义了_UNICODE，那么该串将解释为宽字符的组合，否则解释为8位的字符字符串。

宽字符和 Windows

Windows NT从底层支援Unicode。这意味着Windows NT内部使用由16位字符组成的字符串。因为世界上其它许多地方还不使用16位字符串，所以Windows NT必须经常将字符串在操作系统内转换。Windows NT可执行为ASCII、Unicode或者ASCII和Unicode混合编写的程序。即，Windows NT支持不同的API函数呼叫，这些函数接受8位或16位的字符串（我们将马上看到这是如何动作的。）

相对于Windows NT，Windows 98对Unicode的支持要少得多。只有很少的Windows 98函数呼叫支持宽字符串（这些函数列在《Microsoft Knowledge Base article Q125671》中；它们包括MessageBox）。如果要发行的程序中只有一个.EXE文件要求在Windows NT和Windows 98下都能执行，那么就不应该使用Unicode，否则就不能在Windows 98下执行；尤其程序不能呼叫Unicode版的Windows函数。这样，将来发行Unicode版的程序时会处于更有利的位置，您应试着编写既为ASCII又为Unicode编译的原始码。这就是本书中所有程序的编写方式。

Windows表头文件类型

正如您在第一章所看到的那样，一个Windows程序包括表头文件WINDOWS.H。该文件包括许多其它表头文件，包括WINDEF.H，该文件中有许多在Windows中使用的基本型态定义，而且它本身也包括WINNT.H。WINNT.H处理基本的Unicode支持。

WINNT.H的前面包含C的表头文件CTYPE.H，这是C的众多表头文件之一，包括wchar_t的定义。WINNT.H定义了新的数据型态，称作CHAR和WCHAR：

typedef char CHAR ;

typedef wchar_t WCHAR ;　　// wc

当您需要定义8位字符或者16位字符时，推荐您在Windows程序中使用的数据型态是CHAR和WCHAR。WCHAR定义后面的注释是匈牙利标记法的建议：一个基于WCHAR数据型态的变量可在前面附加上字母wc以说明一个宽字符。

WINNT.H表头文件进而定义了可用做8位字符串指针的六种数据型态和四个可用做const 8位字符串指针的数据型态。这里精选了表头文件中一些实用的说明数据型态语句：

typedef CHAR * PCHAR, * LPCH, * PCH, * NPSTR, * LPSTR, * PSTR ;

typedef CONST CHAR * LPCCH, * PCCH, * LPCSTR, * PCSTR ;

前缀N和L表示「near」和「long」，指的是16位Windows中两种大小不同的指标。在Win32中near和long指标没有区别。

类似地，WINNT.H定义了六种可作为16位字符串指针的数据型态和四种可作为const 16位字符串指针的数据型态：

typedef WCHAR * PWCHAR, * LPWCH, * PWCH, * NWPSTR, * LPWSTR, * PWSTR ;

typedef CONST WCHAR * LPCWCH, * PCWCH, * LPCWSTR, * PCWSTR ;

至此，我们有了数据型态CHAR（一个8位的char）和WCHAR（一个16位的wchar_t），以及指向CHAR和WCHAR的指标。与TCHAR.H一样，WINNT.H将TCHAR定义为一般的字符类型。如果定义了标识符UNICODE（没有底线），则TCHAR和指向TCHAR的指标就分别定义为WCHAR和指向WCHAR的指标；如果没有定义标识符UNICODE，则TCHAR和指向TCHAR的指标就分别定义为char和指向char的指标：

```
#ifdef UNICODE
typedef WCHAR TCHAR, * PTCHAR ;
typedef LPWSTR LPTCH, PTCH, PTSTR, LPTSTR ;
typedef LPCWSTR LPCTSTR ;
#else
typedef char TCHAR, * PTCHAR ;
typedef LPSTR LPTCH, PTCH, PTSTR, LPTSTR ;
typedef LPCSTR LPCTSTR ;
#endif
```

如果已经在某个表头文件或者其它表头文件中定义了TCHAR数据型态，那么WINNT.H和WCHAR.H表头文件都能防止其重复定义。不过，无论何时在程序中使用其它表头文件时，都应在所有其它表头文件之前包含WINDOWS.H。

WINNT.H表头文件还定义了一个宏，该宏将L添加到字符串的第一个引号前。如果定义了UNICODE标识符，则一个称作 __TEXT的宏定义如下：

#define __TEXT(quote) L##quote

如果没有定义标识符UNICODE，则像这样定义__TEXT宏：

#define __TEXT(quote) quote

此外， TEXT宏可这样定义：

#define TEXT(quote) __TEXT(quote)

这与TCHAR.H中定义_TEXT宏的方法一样，只是不必操心底线。我将在本书中使用这个宏的TEXT版本。

这些定义可使您在同一程序中混合使用ASCII和Unicode字符串，或者编写一个可被ASCII或Unicode编译的程序。如果您希望明确定义8位字符变量和字符串，请使用CHAR、PCHAR（或者其它），以及带引号的字符串。为明确地使用16位字符变量和字符串，请使用WCHAR、PWCHAR，并将L添加到引号前面。对于是8位还是16位取决于UNICODE标识符的定义的变量或字符串，要使用TCHAR、PTCHAR和TEXT宏。

Windows函数呼叫

从Windows 1.0到Windows 3.1的16位Windows中，MessageBox函数位于动态链接库USER.EXE。在Windows 3.1软件开发套件的WINDOWS.H中，MessageBox函数定义如下：

int WINAPI MessageBox (HWND, LPCSTR, LPCSTR, UINT) ;

注意，函数的第二个、第三个参数是指向常数字符串的指针。当编译连结一个Win16程序时，Windows并不处理MessageBox呼叫。程序.EXE文件中的表格，允许Windows将该程序的呼叫与USER中的MessageBox函数动态链接起来。

32位的Windows（即所有版本的Windows NT，以及Windows 95和Windows 98）除了含有与16位兼容的USER.EXE以外，还含有一个称为USER32.DLL的动态链接库，该动态链接库含有32位使用者接口函数的进入点，包括32位的MessageBox。

这就是Windows支持Unicode的关键：在USER32.DLL中，没有32位MessageBox函数的进入点。实际上，有两个进入点，一个名为MessageBoxA（ASCII版），另一个名为MessageBoxW（宽字符版）。用字符串作参数的每个Win32函数都在操作系统中有两个进入点！幸运的是，您通常不必关心这个问题，程序中只需使用MessageBox。与TCHAR表头文件一样，每个Windows表头文件都有我们需要的技巧。

下面是MessageBoxA在WINUSER.H中定义的方法。这与MessageBox早期的定义很相似：

```
WINUSERAPI int WINAPI MessageBoxA (HWND hWnd, LPCSTR lpText,
                                   LPCSTR lpCaption, UINT uType) ;
```

下面是MessageBoxW：

```
 WINUSERAPI int WINAPI MessageBoxW (HWND hWnd, LPCWSTR lpText,
                                   LPCWSTR lpCaption, UINT uType) ;
```

注意，MessageBoxW函数的第二个和第三个参数是指向宽字符的指针。

如果需要同时使用并分别匹配ASCII和宽字符函数呼叫，那么您可在Windows程序中明确地使用MessageBoxA和MessageBoxW函数。但大多数程序写作者将继续使用MessageBox。根据是否定义了UNICODE，MessageBox将与MessageBoxA或MessageBoxW一样。在WINUSER.H中完成这一技巧时，程序相当琐碎：

```
#ifdef UNICODE
#define MessageBox MessageBoxW
#else
#define MessageBox MessageBoxA
#endif
```

这样，如果定义了UNICODE标识符，那么程序中所有的MessageBox函数呼叫实际上就是MessageBoxW函数；否则，就是MessageBoxA函数。

执行该程序时，Windows将程序中不同的函数呼叫与不同的Windows动态链接库的进入点连结。虽然只有少数例外，但是，在Windows 98中不能执行Unicode版的Windows函数。虽然这些函数有进入点，但通常返回错误代码。应用程序注意这些返回的错误并采取一些合理的动作。

Windows的字符串函数

正如前面谈到的，Microsoft C包括宽字符和需要字符串参数的C语言执行时期链接库函数的所有普通版本。不过，Windows复制了其中一部分。例如，下面是Windows定义的一组字符串函数，这些函数用来计算字符串长度、复制字符串、连接字符串和比较字符串：

```
iLength = lstrlen (pString) ;
pString = lstrcpy (pString1, pString2) ;
pString = lstrcpyn (pString1, pString2, iCount) ;
pString = lstrcat (pString1, pString2) ;
iComp = lstrcmp (pString1, pString2) ;
```

```
iComp = lstrcmpi (pString1, pString2) ;
```

这些函数与C链接库中对应的函数功能相同。如果定义了UNICODE标识符，那么这些函数将接受宽字符串，否则只接受常规字符串。宽字符串版的lstrlenW函数可在Windows 98中执行。

在Windows中使用printf

有文字模式、命令列C语言程序写作历史的程序写作者往往特别喜欢printf函数。即使可以使用更简单的命令（例如puts），但printf出现在Kernighan和Ritchie的「hello, world」程序中一点也不会令人惊奇。我们知道，增强后的「hello, world」最终还是需要printf的格式化输出，因此我们最好从头开始就使用它。

但有个坏消息：在Windows程序中不能使用printf。虽然Windows程序中可以使用大多数C的执行时期链接库 – 实际上，许多程序写作者更愿意使用C内存管理和文件I/O函数而不是Windows中等效的函数 – Windows对标准输入和标准输出没有概念。在Windows程序中可使用fprintf，而不是printf。

还有一个好消息，那就是仍然可以使用sprintf及sprintf系列中的其它函数来显示文字。这些函数除了将内容格式化输出到函数第一个参数所提供的字符串缓冲区以外，其功能与printfl相同。然后便可对该字符串进行操作（例如将其传给MessageBox）。

如果您从未使用过sprintf （我第一次开始写Windows程序时也没用过此函数），这里有一个简短的执行实体，printf函数说明如下：

int printf (const char * szFormat, ...) ;

第一个参数是一个格式字符串，后面是与格式字符串中的代码相对应的不同类型多个参数。

sprintf函数定义如下：

int sprintf (char * szBuffer, const char * szFormat, ...) ;

第一个参数是字符缓冲区；后面是一个格式字符串。Sprintf不是将格式化结果标准输出，而是将其存入szBuffer。该函数返回该字符串的长度。在文字模式程序设计中，

printf ("The sum of %i and %i is %i", 5, 3, 5+3) ;

的功能相同于

char szBuffer [100] ;

sprintf (szBuffer, "The sum of %i and %i is %i", 5, 3, 5+3) ;

puts (szBuffer) ;

在Windows中，使用MessageBox显示结果优于puts。

几乎每个人都经历过，当格式字符串与被格式化的变量不合时，可能使printf执行错误并可能造成程序当掉。使用sprintf时，您不但要担心这些，而且还有一个新的负担：您定义的字符串缓冲区必须足够大以存放结果。Microsoft专用函数_snprintf解决了这一问题，此函数引进了另一个参数，表示以字符计算的缓冲区大小。

vsprintf是sprintf的一个变形，它只有三个参数。vsprintf用于执行有多个参数的自订函数，类似printf格式。vsprintf的前两个参数与sprintf相同：一个用于保存结果的字符缓冲区和一个格式字符串。第三个参数是指向格式化参数数组的指针。实际上，该指针指向在堆栈中供函数呼叫的变量。va_list、va_start和va_end宏（在STDARG.H中定义）帮助我们处理堆栈指针。本章最后的SCRNSIZE程序展示了使用这些宏的方法。使用vsprintf函数，sprintf函数可以这样编写：

```
int sprintf (char * szBuffer, const char * szFormat, ...)
```

```
{
  int iReturn ;
  va_list pArgs ;
  va_start (pArgs, szFormat) ;
  iReturn = vsprintf (szBuffer, szFormat, pArgs) ;
  va_end (pArgs) ;
  return iReturn ;
}
```

va_start宏将pArg设置为指向一个堆栈变量，该变量地址在堆栈参数szFormat的上面。

由于许多Windows早期程序使用了sprintf和vsprintf，最终导致Microsoft向Windows API中增添了两个相似的函数。Windows的wsprintf和wvsprintf函数在功能上与sprintf和vsprintf相同，但它们不能处理浮点格式。

当然，随着宽字符的发表，sprintf类型的函数增加许多，使得函数名称变得极为混乱。表2-1列出了Microsoft的C执行时期链接库和Windows支持的所有sprintf函数。

表2-1

	ASCII	宽字符	常规
参数的变数个数			
标准版	sprintf	swprintf	_stprintf
最大长度版	_snprintf	_snwprintf	_sntprintf
Windows版	wsprintfA	wsprintfW	wsprintf
参数数组的指针			
标准版	vsprintf	vswprintf	_vstprintf
最大长度版	_vsnprintf	_vsnwprintf	_vsntprintf
Windows版	wvsprintfA	wvsprintfW	wvsprintf

在宽字符版的sprintf函数中，将字符串缓冲区定义为宽字符串。在宽字符版的所有这些函数中，格式字符串必须是宽字符串。不过，您必须确保传递给这些函数的其它字符串也必须由宽字符组成。

格式化消息框

程序2-1所示的SCRNSIZE程序展示了如何实作MessageBoxPrintf函数，该函数有许多参数并能像printf那样编排它们的格式。

程序2-1 SCRNSIZE

SCRNSIZE.C

```
/*---------------------------------------------------------------------------
SCRNSIZE.C -- Displays screen size in a message box
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
int CDECL MessageBoxPrintf (TCHAR * szCaption, TCHAR * szFormat, ...)
{
  TCHAR  szBuffer [1024] ;
  va_list pArgList ;
  // The va_start macro (defined in STDARG.H) is usually equivalent to:
  // pArgList = (char *) &szFormat + sizeof (szFormat) ;
  va_start (pArgList, szFormat) ;
  // The last argument to wvsprintf points to the arguments
  _vsntprintf ( szBuffer, sizeof (szBuffer) / sizeof (TCHAR),
    szFormat, pArgList) ;
  // The va_end macro just zeroes out pArgList for no good reason
  va_end (pArgList) ;
```

```
  return MessageBox (NULL, szBuffer, szCaption, 0) ;
}

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  int cxScreen, cyScreen ;
  cxScreen = GetSystemMetrics (SM_CXSCREEN) ;
  cyScreen = GetSystemMetrics (SM_CYSCREEN) ;
  MessageBoxPrintf (  TEXT ("ScrnSize"),
    TEXT ("The screen is %i pixels wide by %i pixels high."),
    cxScreen, cyScreen) ;
  return 0 ;
}
```

经由从GetSystemMetrics函数得到的信息，该程序以像素为单位显示了视讯显示的宽度和高度。GetSystemMetrics是一个能用来获得Windows中不同对象的尺寸信息的函数。事实上，我将在第四章用GetSystemMetrics函数向您展示如何在一个Windows窗口中显示和滚动多行文字。

本书与国际化

为国际市场准备的Windows程序不光要使用Unicode。国际化超出了本书的范围，但在Nadine Kano所写的《Developing International Software for Windows 95 and Windows NT》（Microsoft Press，1995年）一书中涉猎了许多。

本书中的程序写作时被限制成既可使用也可不使用定义的UNICODE标识符来编译。这包括对所有字符和字符串定义使用TCHAR，对字符串文字使用TEXT宏，以及注意不要混淆字节和字符。例如，注意SCRNSIZE中的 _vsntprintf呼叫。第二个参数是缓冲区的字符大小。通常，您使用sizeof（szBuffer）。但如果缓冲区中有宽字符，则返回的不是缓冲区的字符长度，而是缓冲区的字节大小。您必须用sizeof（TCHAR）将其分开。

通常，在Visual C++ Developer Studio中，可使用两种不同的设定来编译程序：Debug和Release。为简便起见，对本书的范例程序，我已修改了Debug设定，以便于定义UNICODE标识符。如果程序使用了需要字符串作参数的C链接库函数，那么_UNICODE标识符也在Debug设定中定义（要了解这是在哪里完成的，请从「Project」菜单中选择「Settings」，然后单击「C/C++」标签）。使用这种方式，这些程序就可以方便地被重新编译和连结以供测试。

本书中所有程序 – 无论是否为Unicode编译 – 都可以在Windows NT下执行。只有极少数情况例外。本书中按Unicode编译的程序不能在Windows 98中执行，而非Unicode版则可以。本章和第一章的程序就是两个特例。MessageBoxW是Windows 98支持的少数宽字符Windows函数之一。在SCRNSIZE.C中，如果用Windows函数wprintf代替了_vsntprintf（您还必须删除该函数的第二个参数），那么SCRNSIZE.C的Unicode版将不能在Windows 98下执行，这是因为Windows 98不支持wprintfW。

在本书的后面（特别在第六章，介绍键盘的使用时），我们将看到，编写能处理远东版Windows双字符集的Windows程序不是一件容易的事情。本书没有说明如何去做，并且基于这个原因，本书中的某些非Unicode版本的程序在远东版的Windows下不能正常执行。这也是Unicode对将来的程序设计如此重要的一条理由。Unicode允许程序更容易地跨越国界。

第三章 窗口和消息

在前两章，程序使用了同一个函数MessageBox来向使用者输出文字。MessageBox函数会建立一个「窗口」。在Windows中，「窗口」一词有确切的含义。一个窗口就是屏幕上的一个矩形区域，它接收使用者的输入并以文字或图形的格式显示输出内容。

MessageBox函数建立一个窗口，但这只是一个功能有限的特殊窗口。消息窗口有一个带关闭按钮的标题栏、一个选项图标、一行或多行文字，以及最多四个按钮。当然，必须选择Windows提供给您的图标与按钮。

MessageBox函数非常有用，但下面不会过多地使用它。我们不能在消息框中显示图形，而且也不能在消息框中添加菜单。要添加这些对象，就需要建立自己的窗口，现在就开始。

自己的窗口

建立窗口很简单，只需呼叫CreateWindow函数即可。

好啦，虽然建立窗口的函数的确名为CreateWindow，而且您也能在/Platform SDK/User Interface Services/Windowing/Windows/Window Reference/Window Functions找到此文件，但您将发现CreateWindow的第一个参数就是所谓的「窗口类别名称」，并且该窗口类别连接所谓的「窗口消息处理程序」。在我们呼叫CreateWindow之前，有一点背景知识会对您大有帮助。

总体结构

进行Windows程序设计，实际上是在进行一种对象导向的程序设计（OOP）。这一点在Windows中使用得最多的对象上表现最为明显。这种对象正是Windows之所以命名为「Windows」的原因，它具有人格化的特征，甚至可能会在您的梦中出现，这就是那个叫做「窗口」的东西。

桌面上最明显的窗口就是应用程序窗口。这些窗口含有显示程序名称的标题栏、菜单甚至可能还有工具列和滚动条。另一类窗口是对话框，它可以有标题栏也可以没有标题栏。

装饰对话框表面的还有各式各样的按键、单选按钮、复选框、清单方块、滚动条和文字输入区域。其中每一个小的视觉对象都是一个窗口。更确切地说，这些都称为「子窗口」或「控件窗口」或「子窗口控件」。

作为对象，使用者会在屏幕上看到这些窗口，并通过键盘和鼠标直接与它们进行交互操作。更有趣的是，程序写作者的观点与使用者的观点极其类似。窗口以「消息」的形式接收窗口的输入，窗口也用消息与其它窗口通讯。对讯息的理解将是学习如何写作Windows程序所必须越过的障碍之一。

这有一个Windows的消息范例：我们知道，大多数的Windows程序都有大小合适的应用程序窗口。也就是说，您能够通过鼠标拖动窗口的边框来改变窗口的大小。通常，程序将通过改变窗口中的内容来响应这种大小的变化。您可能会猜测（并且您也是正确的），是Windows本身而不是应用程序在处理与使用者重新调整窗口大小相关的全部杂乱程序。由于应用程序能改变其显示的样子，所以它也「知道」窗口大小改变了。

应用程序是如何知道使用者改变了窗口的大小的呢？由于程序写作者习惯了往常的文字模式

程序，操作系统没有设置将此类消息通知给使用者的机制。问题的关键在于理解Windows所使用的架构。当使用者改变窗口的大小时，Window给程序发送一个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呢？我们在这里讨论的是程序代码，而不是一个电子邮件系统。操作系统怎么给程序发送消息呢？

其实，所谓「Windows给程序发送消息」，是指Windows呼叫程序中的一个函数，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「窗口消息处理程序」。

无疑，读者对程序呼叫操作系统的做法是很熟悉的。例如，程序在打开磁盘文件时就要使用有关的系统呼叫。读者所不习惯的，可能是操作系统呼叫程序，而这正是Windows对象导向架构的基础。

程序建立的每一个窗口都有相关的窗口消息处理程序。这个窗口消息处理程序是一个函数，既可以在程序中，也可以在动态链接库中。Windows通过呼叫窗口消息处理程序来给窗口发送消息。窗口消息处理程序根据此消息进行处理，然后将控制传回给Windows。

更确切地说，窗口通常是在「窗口类别」的基础上建立的。窗口类别标识了处理窗口消息的窗口消息处理程序。使用窗口类别使多个窗口能够属于同一个窗口类别，并使用同一个窗口消息处理程序。例如，所有Windows程序中的所有按钮均依据同一个窗口类别。这个窗口类别与一个处理所有按钮消息的窗口消息处理程序（位于Windows的动态链接库中）联结。

在对象导向的程序设计中，对象是程序与数据的组合。窗口是一种对象，其程序是窗口消息处理程序。数据是窗口消息处理程序保存的信息和Windows为每个窗口以及系统中那个窗口类别保存的信息。

窗口消息处理程序处理给窗口发送消息。这些消息经常是告知窗口，使用者正使用键盘或者鼠标进行输入。这正是按键窗口知道它被「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，由其它消息通知窗口。

Windows程序开始执行后，Windows为该程序建立一个「消息队列」。这个消息队列用来存放该程序可能建立的各种不同窗口的消息。程序中有一小段程序代码，叫做「消息循环」，用来从队列中取出消息，并且将它们发送给相应的窗口消息处理程序。有些消息直接发送给窗口消息处理程序，不用放入消息队列中。

如果您对这段Windows架构过于简略的描述将信将疑，就让我们去看看在实际的程序中，窗口、窗口类别、窗口消息处理程序、消息队列、消息循环和窗口消息是如何相互配合的。这或许会对您有些帮助。

HELLOWIN程序

建立一个窗口首先需要注册一个窗口类别，那需要一个窗口消息处理程序来处理窗口消息。处理窗口消息对每个Windows程序都带来了些负担。程序3-1所示的HELLOWIN程序中整个做的事情差不多就是料理这些事情。

程序3-1 HELLOWIN

HELLOWIN.C

```
/*------------------------------------------------------------------------
HELLOWIN.C -- Displays "Hello, Windows 98!" in client area
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
```

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("HelloWin") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASwndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuNam = NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow( szAppName, // window class name
    TEXT ("The Hello Program"), // window caption
    WS_OVERLAPPEDWINDOW, // window style
    CW_USEDEFAULT,// initial x position
    CW_USEDEFAULT,// initial y position
    CW_USEDEFAULT,// initial x size
    CW_USEDEFAULT,// initial y size
    NULL, // parent window handle
    NULL, // window menu handle
    hInstance, // program instance handle
    NULL) ; // creation parameters
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;
  switch (message)
  {
  case WM_CREATE:
    PlaySound (TEXT ("hellowin.wav"), NULL, SND_FILENAME | SND_ASYNC) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    GetClientRect (hwnd, &rect) ;
    DrawText (hdc, TEXT ("Hello, Windows 98!"), -1, &rect,
      DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
```

```
}
```

程序建立一个普通的应用程序窗口，如图3-1所示。在窗口显示区域的中央显示「Hello, Windows 98!」。如果安装了声卡，那么您还可以听到相应的朗读声音。

图3-1 HELLOWIN窗口

提醒您注意：如果您使用Microsoft Visual C++ 为此程序建立新项目，那么您得加上连结程序所需的链接库文件。从Project菜单选择 Setting选项，然后选取Link页面标签。从 Category清单方块中选择General，然后在 Object/Library Modules文字方块添加WINMM.LIB（Windows multimedia – Windows多媒体）。您这样做是因为HELLOWIN将使用多媒体功能呼叫，而内定的项目中又不包括多媒体链接库文件。不然连结程序报告了错误信息，表明PlaySound函数不可用。

HELLOWIN将存取文件HELLOWIN.WAV，该文件在本书所附光盘的HELLOWIN目录中。执行HELLOWIN.EXE时，内定的目录必须是HELLOWIN。在Visual C++中执行此程序时，虽然执行文件会产生在HELLOWIN的RELEASE或DEBUG子目录中，但执行程序的目录还是必须在HELLOWIN中。

通盘考量

实际上，每一个Windows程序代码中都包括HELLOWIN.C程序的大部分。没人能真正记住此程序的全部写法；通常，Windows程序写作者在开始写一个新程序时总是会复制一个现有的程序，然后再做相应的修改。您可以按此习惯自由使用本书附带光盘中的程序。

上面提到，HELLOWIN将在其窗口的中央显示字符串。这种说法不是完全正确的。文字实际显示在程序显示区域的中央，它在图3-1中是标题栏和边界范围内的大片白色区域。这区别对我们来说很重要；显示区域就是程序自由绘图并且向使用者显示输出结果的窗口区域。

如果您认真思考一下，将会发现虽然只有80行程序代码，这个窗口却令人惊讶地具有许多功能。您可以用鼠标按住标题栏，在屏幕上移动窗口；可以按住大小边框，改变窗口的大小。在窗口大小改变时，程序自动地将「Hello, Windows 98!」字符串重新定位在显示区域的中央。您可以按最大

化按钮，放大HELLOWIN以充满整个屏幕；也可以按最小化按钮，将程序缩小成一个图标。您可以在系统菜单中执行所有选项(就是按下在标题栏最左端的小图标)；也可以从系统菜单中选择 Close 选项，或者单击标题栏最右端的关闭按钮，或者双击标题栏最左端的图标，来关闭窗口以终止程序的执行。

我们将在本章的余下部分对此程序作一详细的检查。当然，我们首先要从整体上看一下。

与前两章中的范例程序一样，HELLOWIN.C也有一个WinMain函数，但它还有另外一个函数，名为WndProc。这就是窗口消息处理程序。注意，在HELLOWIN.C中没有呼叫WndProc的程序代码。当然，在WinMain中有对WndProc的参考，而这就是该函数要在程序开头附近声明的原因。

Windows函数呼叫

HELLOWIN至少呼叫了18个Windows函数。下面以它们在HELLOWIN中出现的次序列出这些函数以及各自的简明描述：

LoadIcon 加载图标供程序使用。

LoadCursor 加载鼠标光标供程序使用。
GetStockObject 取得一个图形对象（在这个例子中，是取得绘制窗口背景的画刷对象)。
RegisterClass 为程序窗口注册窗口类别。
MessageBox 显示消息框。
CreateWindow 根据窗口类别建立一个窗口。
ShowWindow 在屏幕上显示窗口。
UpdateWindow 指示窗口自我更新。
GetMessage 从消息队列中取得消息。
TranslateMessage 转译某些键盘消息。
DispatchMessage 将消息发送给窗口消息处理程序。
PlaySound 播放一个声音文件。
BeginPaint 开始绘制窗口。
GetClientRect 取得窗口显示区域的大小。
DrawText 显示字符串。
EndPaint 结束绘制窗口。
PostQuitMessage 在消息队列中插入一个「退出程序」消息。
DefWindowProc 执行内定的消息处理。

这些函数均在Platform SDK文件中说明，并在不同的表头文件中声明，其中绝大多数声明在WINUSER.H中。

大写字母标识符

读者可能注意到，HELLOWIN.C中有几个大写的标识符，这些标识符是在Windows表头文件中定义的。有些标识符含有两个字母或者三个字母的前缀，这些前缀后头接着一个底线：

CS_HREDRAW	*DT_VCENTER*	*SND_FILENAME*
CS_VREDRAW	*IDC_ARROW*	*WM_CREATE*
CW_USEDEFAULT	*IDI_APPLICATION*	*WM_DESTROY*
DT_CENTER	*MB_ICONERROR*	*WM_PAINT*
DT_SINGLELINE	*SND_ASYNC*	*WS_OVERLAPPEDWINDOW*

这些是简单的数值常数。前缀指示该常数所属的类别，如表3-1所示。

表3-1

前缀	类别
CS	窗口类别样式
CW	建立窗口
DT	绘制文字
IDI	图标ID
IDC	光标ID
MB	消息框
SND	声音
WM	窗口消息
WS	窗口样式

奉劝程序写作者不要费力气去记忆Windows程序设计中的数值常数。实际上，Windows中使用的每个数值常数在表头文件中均有相应的标识符定义。

新的数据型态

HELLOWIN.C中的其它标识符是新的数据型态，也在Windows表头文件中使用typedef叙述或者#define叙述加以定义了。最初是为了便于将Windows程序从原来的16位系统上移植到未来的使用32位(或者其它)技术的操作系统上。这种作法并不如当时每个人想象的那样顺利，但是这种概念基本上是正确的。

有时这些新的数据型态只是为了方便缩写。例如，用于WndProc的第二个参数的UINT数据型态只是一个unsigned int （无正负号整数），在Windows 98中，这是一个32位的值。用于WinMain的第三个参数的PSTR数据型态是指向一个字符串的指针，即是一个char *。

其它数据型态的含义不太明显。例如，WndProc的第三和第四个参数分别被定义为WPARAM和LPARAM，这些名字的来源有点历史背景：当Windows还是16位系统时，WndProc的第三个参数被定义为一个WORD，这是一个16位的 **无正负号短**（unsigned short）整数，而第四个参数被定义为一个LONG，这是一个32位有正负号长整数，从而导致了文字「PARAM」前面加上了前置前缀「W」和「L」。当然，在32位的Windows中，WPARAM被定义为一个UINT，而LPARAM被定义为一个LONG（这就是C中的long整数型态），因此窗口消息处理程序的这两个参数都是32位的值。这也许有点奇怪，因为WORD数据型态在Windows98中仍然被定义为一种16位的 **无正负号**整数，因此「PARAM」前的「W」就有点误用了。

WndProc函数传回一个型态为LRESULT的值，该值简单地被定义为一个LONG。WinMain函数被指定了一个WINAPI型态（在表头文件中定义的所有Windows函数都被指定这种型态），而WndProc函数被指定一个CALLBACK型态。这两个标识符都被定义为_stdcall，表示在Windows本身和使用者的应用程序之间发生的函数呼叫的呼叫参数传递方式。

HELLOWIN还使用了Windows表头文件中定义的四种数据结构（我们将在本章稍后加以讨论）。这些数据结构如表3-2所示。

表3-2

结构	含义

MSG	消息结构
WNDCLASS	窗口类别结构
PAINTSTRUCT	绘图结构
RECT	矩形结构

前面两个数据结构在WinMain中使用，分别定义了两个名为msg和wndclass的结构，后面两个数据结构在WndProc中使用，分别定义了ps和rect结构。

句柄简介

最后，还有三个大写标识符（见表3-3），用于不同型态的「句柄」:

表3-3

标识符	含义
HINSTANCE	执行实体（程序自身）句柄
HWND	窗口句柄
HDC	设备内容句柄

句柄在Windows中使用非常频繁。在本章结束之前，我们将遇到HICON（图标句柄）、HCURSOR（鼠标光标句柄）和HBRUSH（画刷句柄）。

句柄是一个（通常为32位的）整数，它代表一个对象。Windows中的句柄类似传统C或者MS-DOS程序设计中使用的文件句柄。程序几乎总是通过呼叫Windows函数取得句柄。程序在其它Windows函数中使用这个句柄，以使用它代表的对象。代号的实际值对程序来说是无关紧要的。但是，向您的程序提供代号的Windows模块知道如何利用它来使用相对应的对象。

匈牙利表示法

读者可能注意到，HELLOWIN.C中有一些变量的名字显得很古怪。如szCmdLine，它是传递给WinMain的参数。

许多Windows程序写作者使用一种叫做「匈牙利表示法」的变量命名通则。这是为了纪念传奇性的Microsoft程序写作者Charles Simonyi。非常简单，变量名以一个或者多个小写字母开始，这些字母表示变量的数据型态。例如，szCmdLine中的sz代表「以0结尾的字符串」。在hInstance和hPrevInstance中的h前缀表示「句柄」；在iCmdShow中的i前缀表示「整数」。WndProc的后两个参数也使用匈牙利表示法。正如我在前面已经解释过的，尽管wParam应该更适当地被命名为uiParam（代表「无正负号整数」），但是因为这两个参数是使用数据型态WPARAM和LPARAM定义的，因此保留它们传统的名字。

在命名结构变量时，可以用结构名（或者结构名的一种缩写）的小写作为变量名的前缀，或者用作整个变量名。例如，在HELLOWIN. C的WinMain函数中，msg变量是MSG型态的结构；wndclass是WNDCLASSEX型态的一个结构。在WndPmc函数中，ps是一个PAINTSTRUCT结构，rect是一个RECT结构。

匈牙利表示法能够帮助程序写作者及早发现并避免程序中的错误。由于变量名既描述了变量的作用，又描述了其数据型态，就比较容易避免产生数据型态不合的错误。

表3-4列出了在本书中经常用到的变量前缀。

表3-4

前缀	数据型态

c	char或WCHAR或TCHAR
by	BYTE （无正负号字符）
n	short
i	int
x, y	int分别用作x坐标和y坐标
cx, cy	int分别用作x长度和y长度；C代表「计数器」
b或f	BOOL (int)；f代表「旗标」
w	WORD （无正负号短整数）
l	LONG （长整数）
dw	DWORD （无正负号长整数）
fn	function（函数）
s	string（字符串）
sz	以字节值0结尾的字符串
h	句柄
p	指标

注册窗口类别

窗口依照某一窗口类别建立，窗口类别用以标识处理窗口消息的窗口消息处理程序。

不同窗口可以依照同一种窗口类别建立。例如，Windows中的所有按钮窗口 – 包括按键、复选框，以及单选按钮 – 都是依据同一种窗口类别建立的。窗口类别定义了窗口消息处理程序和依据此类别建立的窗口的其它特征。在建立窗口时，要定义一些该窗口所独有的特征。

在为程序建立窗口之前，必须首先呼叫RegisterClass注册一个窗口类别。该函数只需要一个参数，即一个指向型态为WNDCLASS的结构指针。此结构包括两个指向字符串的字段，因此结构在WINUSER.H表头文件中定义了两种不同的方式，第一个是ASCII版的WNDCLASSA：

```
typedef struct tagWNDCLASSA
{
  UINT style ;
  WNDPROC lpfnWndProc ;
  int cbClsExtra ;
  int cbWndExtra ;
  HINSTANCE hInstance ;
  HICON hIcon ;
  HCURSOR hCursor ;
  HBRUSH hbrBackground ;
  LPCSTR lpszMenuName ;
  LPCSTR lpszClassName ;
}
WNDCLASSA, * PWNDCLASSA, NEAR * NPWNDCLASSA, FAR * LPWNDCLASSA ;
```

在这里提示一下数据型态和匈牙利表示法：其中的lpfn前缀代表「指向函数的长指标」。（在Win32 API中，长指标和短指标（或者近程指标）没有区别。这只是16位Windows的遗物。）cb前缀代表「字节数」而且通常作为一个常数来表示一个字节的大小。h前缀是一个句柄，而hbr前缀代表「一个画刷的代号」。lpsz前缀代表「指向以0结尾字符串的指针」。

Unicode版的结构定义如下：

```
typedef struct tagWNDCLASSW
{
  UINT style ;
  WNDPROC lpfnWndProc ;
  int cbClsExtra ;
```

```
  int cbWndExtra ;
  HINSTANCE hInstance ;
  HICON hIcon ;
  HCURSOR hCursor ;
  HBRUSH hbrBackground ;
  LPCWSTR lpszMenuName ;
  LPCWSTR lpszClassName ;
}
WNDCLASSW, * PWNDCLASSW, NEAR * NPWNDCLASSW, FAR * LPWNDCLASSW ;
```

与前者唯一的区别在于最后两个字段定义为指向宽字符串常数，而不是指向ASCII字符串常数。

WINUSER.H定义了WNDCLASSA和WNDCLASSW结构（以及指向结构的指针）以后，表头文件依据对UNICODE标识符的解释，定义了WNDCLASS和指向WNDCLASS的指标（包括一些向后兼容的程序代码）：

```
#ifdef UNICODE
typedef WNDCLASSW WNDCLASS ;
typedef PWNDCLASSW PWNDCLASS ;
typedef NPWNDCLASSW NPWNDCLASS ;
typedef LPWNDCLASSW LPWNDCLASS ;
#else
typedef WNDCLASSA WNDCLASS ;
typedef PWNDCLASSA PWNDCLASS ;
typedef NPWNDCLASSA NPWNDCLASS ;
typedef LPWNDCLASSA LPWNDCLASS ;
#endif
```

本书后面列出结构时，将只列出功用相同的结构定义，对WNDCLASS就像这样：

```
typedef struct
{
   UINT style ;
   WNDPROC lpfnWndProc ;
   int cbClsExtra ;
   int cbWndExtra ;
   HINSTANCE hInstance ;
   HICON hIcon ;
   HCURSOR hCursor ;
   HBRUSH hbrBackground ;
   LPCTSTR lpszMenuName ;
   LPCTSTR lpszClassName ;
}
WNDCLASS, * PWNDCLASS ;
```

我也不再着重说明指标的定义。一个程序写作者的程序不应该因为使用以LP或NP为前缀的不同指针型态而被搅乱。

在WinMain中为WNDCLASS定义一个结构，通常像这样：

```
WNDCLASS wndclass ;
```

然后，你就可以初始化该结构的10个字段，并呼叫RegisterClass。

在WNDCLASS结构中最重要的两个字段是第二个和最后一个，第二个字段(lpfnWndProc) 是依据这个类别来建立的所有窗口所使用的窗口消息处理程序的地址。在HELLOWIN.C中，这个是WndProc函数。最后一个字段是窗口类别的文字名称。程序写作者可以随意定义其名称。在只建立一个窗口的程序中，窗口类别名称通常设定为程序名称。

其它字段依照下面的方法描述了窗口类别的一些特征。让我们依次看看WNDCLASS结构中的每个字段。

wndclass.style = CS_HREDRAW | CS_VREDRAW ;

使用C的位「或」运算子结合了两个「窗口类别样式」标识符。在表头文件WINUSER.H中，已定义了一整组以CS为前缀的标识符：

```
#define  CS_VREDRAW    0x0001
#define  CS_HREDRAW   0x0002
#define  CS_KEYCVTWINDOW   0x0004
#define  CS_DBLCLKS   0x0008
#define  CS_OWNDC    0x0020
#define  CS_CLASSDC   0x0040
#define  CS_PARENTDC   0x0080
#define  CS_NOKEYCVT   0x0100
#define  CS_NOCLOSE   0x0200
#define  CS_SAVEBITS   0x0800
#define  CS_BYTEALIGNCLIENT  0x1000
#define  CS_BYTEALIGNWINDOW  0x2000
#define  CS_GLOBALCLASS  0x4000
#define  CS_IME    0x00010000
```

由于每个标识符都可以在一个复合值中设置一个位的值，所以按这种方式定义的标识符通常称为「位旗标」。通常我们只使用少数的窗口类别样式。HELLOWIN中用到的这两个标识符表示，所有依据此类别建立的窗口，每当窗口的水平方向大小(CS_HREDRAW)或者垂直方向大小(CS_VREDRAW)改变之后，窗口要完全重画。改变HELLOWIN的窗口大小，可以看到字符串仍然显示在窗口的中央，这两个标识符确保了这一点。不久我们就将看到窗口消息处理程序是如何得知这种窗口大小的变化的。

WNDCLASS结构的第二个字段由以下叙述进行初始化：

wndclass.lpfnWndProc = WndProc ;

这条叙述将这个窗口类别的窗口消息处理程序设定为WndProc，即HELLOWIN.C中的第二个函数。这个过程将处理依据这个窗口类别建立的所有窗口的全部消息。在C语言中，像这样在结构中使用函数名时，真正提供的是指向函数的指标。

下面两个字段用于在窗口类别结构和Windows内部保存的窗口结构中预留一些额外空间：

wndclass.cbClsExtra = 0 ;

wndclass.cbWndExtra = 0 ;

程序可以根据需要来使用预留的空间。HELLOWIN没有使用它们，所以设定值为0。否则，和匈牙利表示法所指示的一样，这个字段将被当成「预留的字节数」。(在第七章的程序CHECKER3将使用cbWndExtra字段。)

下一个字段就是程序的执行实体句柄（它也是WinMain的参数之一）：

wndclass.hInstance = hInstance ;

wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;

为所有依据这个窗口类别建立的窗口设置一个图标。图标是一个小的位图图像，它对使用者代表程序，将出现在Windows工作列中和窗口的标题栏的左端。在本书的后面，您将学习如何为您的Windows程序自订图标。现在，为了方便起见，我们将使用预先定义的图标。

要取得预先定义图标的句柄，可以将第一个参数设定为NULL来呼叫LoadIcon。在加载程序写作者自订的图标时（图标应该存放在磁盘上的.EXE程序文件中），这个参数应该被设定为程序的执

行实体句柄hInstance。第二个参数代表图标。对于预先定义图标，此参数是以IDI开始的标识符（「ID代表图标」），标识符在WINUSER.H中定义。IDI_APPLICATION图标是一个简单的窗口小图形。LoadIcon函数传回该图标的句柄。我们并不关心这个句柄的实际值，它只用于设置hIcon字段元的值。该字段在WNDCLASS结构中定义为HICON型态，此型态名的含义为「handle to an icon（图标句柄）」。

wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;

与前一条叙述非常相似。LoadCursor函数加载一个预先定义的鼠标光标（命名为IDC_ARROW），并传回该光标的句柄。该句柄被设定给WNDCLASS结构的hCursor字段。当鼠标光标在依据这个类别建立的窗口的显示区域上出现时，它变成一个小箭头。

下一个字段指定依据这个类别建立的窗口背景颜色。hbrBackground字段名称中的hbr前缀代表「handle to a brush（画刷句柄）」。画刷是个绘图词汇，指用来填充一个区域的着色样式。Windows有几个标准画刷，也称为「备用(stock)」画刷。这里所示的GetStockObject呼叫将传回一个白色画刷的句柄：

wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;

这意味着窗口显示区域的背景完全为白色，这是一种极其普遍的做法。

下一个字段指定窗口类别菜单。HELLOWIN没有应用程序菜单，所以该字段被设定为NULL：

wndclass.lpszMenuName = NULL ;

最后，必须给出一个类别名称。对于小程序，类别名称可以与程序名相同，即存放在szAppName变量中的「HelloWin」字符串。

wndclass.lpszClassName = szAppName ;

至于该字符串由ASCII字符组成或由Unicode字符组成，取决于是否定义了UNICODE标识符。

在初始化该结构的10个字段后，HELLOWIN呼叫RegisterClass来注册这个窗口类别。该函数只有一个参数，即指向WNDCLASS结构的指针。实际上，RegisterClassA函数将获得一个指向WNDCLASSA结构的指针，而RegisterClassW函数将获得一个指向WNDCLASSW结构的指针。程序要使用哪个函数来注册窗口类别，取决于发送给窗口的消息包含ASCII文字还是Unicode文字。

现在有一个问题：如果用定义的UNICODE标识符编译了程序，程序将呼叫RegisterClassW。该程序可以在Microsoft Windows NT中执行良好。但如果此程序在Windows 98上执行，RegisterClassW函数并未真地被执行到。函数有一个进入点，但函数呼叫后只传回0，表明错误。对于在Windows 98下执行的Unicode程序来说，这是一个通知使用者有问题并终止执行的好机会。这是本书中多数程序处理RegisterClass函数呼叫的方法：

```
if (!RegisterClass (&wndclass))
{
  MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
    szAppName, MB_ICONERROR) ;
  return 0 ;
}
```

由于MessageBoxW是可在Windows 98环境下执行的几个Unicode函数之一，所以其执行正常。

当然，这段程序假定RegisterClass不会因为其它原因而呼叫失败，诸如WNDCLASS结构中lpfnWndProc字段被设定成NULL之类的错误。GetLastError函数会帮助您确定在这样的情况下产生错误的原因。GetLastError是Windows中常用的函数，它可以在函数呼叫失败时获得更多错误信息。不同函数的文件将指出您是否能够用GetLastError来获得这些信息。在Windows 98中呼叫

RegisterClassW时，GetLastError将传回120。在WINERROR.H中您可以看到，值120与标识符ERROR_CALL_NOT_IMPLEMENTED 相等。您也可以在/Platform SDK/Windows Base Services/Debugging and Error Handling/Error Codes/System Errors - Numerical Order查看错误。

一些Windows程序写作者喜欢检查所有可能发生错误的函数呼叫的传回值。这么做确实有点道理，相信您也非常习惯在配置内存后检查错误。而许多Windows函数需要配置内存。例如，RegisterClass需要配置内存，以保存窗口类别的信息。如此一来，您就应该要检查这个函数的执行结果。另一方面说来，如果由于RegisterClass不能得到所需要的内存，它会声明呼叫失败，而Windows大概也快当掉了。

在本书的范例程序中，我做了最少的错误检查。这不是因为我认为错误检查不是一个好方法，而是因为这会让我们在程序举例中分心。

最后，一个老经验是：在一些Windows范例程序中，您可能在WinMain中看到以下程序代码：

```
if (!hPrevInstance)
{
  wndclass.cbStyle = CS_HREDRAW | CS_VREDRAW ;
  //初始化其它 wndclass
  RegisterClass (&wndclass) ;
}
```

这是出于「旧习难改」的原因。在16位的Windows中，如果您启动正在执行的程序的一个新执行实体，WinMain的hPrevInstance参数将是前一个执行实体的执行实体句柄。为节省内存，两个或多个执行实体就可能会共享相同的窗口类别。这样，窗口类别就只在hPrevInstance是NULL的时候才注册，这表明程序没有其它执行实体。

在32位的Windows中，hPrevInstance总是NULL。此程序代码会正常执行，而实际上也没必要检查hPrevInstance。

建立窗口

窗口类别定义了窗口的一般特征，因此可以使用同一窗口类别建立许多不同的窗口。实际呼叫CreateWindow建立窗口时，可能指定有关窗口的更详细的信息。

Windows程序设计新手有时会混淆窗口类别和窗口之间的区别，以及为什么一个窗口的所有特征不能被一次设定好。实际上，以这种方式分开这些样式信息是非常方便的。例如，所有的按钮窗口都可以依据同样的窗口类别来建立，与这个窗口类别相关的窗口消息处理程序位于Windows内部。由窗口类别来负责处理按钮的键盘和鼠标输入，并定义按钮在屏幕上的外观形象。从这一点看来，所有的按钮都是以同样的方式工作的。但是并非所有的按钮都是一样的。它们可以有不同的大小，不同的屏幕位置，以及不同的字符串。后面的这样一些特征是窗口定义的一部分，而不是窗口类别定义的。

传递给RegisterClass函数的信息会在一个数据结构中设定好，而传递给CreateWindow函数的信息会在函数单独的参数中设定好。下面是HELLOWIN.C中的CreateWindows呼叫，每一个字段都做了完整的说明：

```
hwnd = CreateWindow (szAppName, // window class name
                  TEXT ( "The Hello Program"), // window caption
                  WS_OVERLAPPEDWINDOW, // window style
                  CW_USEDEFAULT, // initial x position
                  CW_USEDEFAULT, // initial y position
                  CW_USEDEFAULT, // initial x size
                  CW_USEDEFAULT, // initial y size
                  NULL, // parent window handle
                  NULL, // window menu handle
```

```
                hInstance, // program instance handle
                NULL) ; // creation parameters
```

在这里，我不想提实际上有CreateWindowA函数和CreateWindowW函数，两个函数分别将前两个参数当成ASCII或者Unicode字符串来处理。

标记为「window class name」的参数是szAppName，它含有字符串「HelloWin」－这是程序注册的窗口类别名称。这就是我们建立的窗口联结窗口类别的方式。

此程序建立的窗口是一个普通的重迭式窗口。它含有一个标题栏，标题栏左边有一个系统菜单按钮，标题栏右边有缩小、放大和关闭图标，四周还有一个表示窗口大小的边框。这是标准样式的窗口，名为WS_OVERLAPPEDWINDOW，出现在CreateWindow的「窗口样式」参数中。如果看一下WINUSER.H，您将会发现此样式是几种位旗标的组合：

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED | \
                             WS_CAPTION | \
                             WS_SYSMENU | \
                             WS_THICKFRAME | \
                             WS_MINIMIZEBOX | \
                             WS_MAXIMIZEBOX)
```

「窗口标题」是显示在标题栏中的文字。

注释着「initial x position」和「initial y position」的参数指定了窗口左上角相对于屏幕左上角的初始位置。由于这些参数使用CW_USEDEFAULT标识符，指示Windows使用重迭窗口的内定位置。（CW_USEDEFAULT定义为0x80000000。）内定情况下，Windows依次对新建立的窗口定位，使各窗口左上角的垂直和水平距离在屏幕上按一定的大小递增。与此类似，注释着「initial x size」和「initial y size」的参数分别指定窗口的宽度和高度。同样使用了CW_USEDEFAULT标识符，表明希望Windows使用内定尺寸。

在建立一个「最上层」窗口，如应用程序窗口时，注释为「父窗口句柄」的参数设定为NULL。通常，如果窗口之间存在有父子关系，则子窗口总是出现在父窗口的上面。应用程序窗口出现在桌面窗口的上面，但不必为呼叫CreateWindow而找出桌面窗口的句柄。

因为窗口没有菜单，所以「窗口菜单句柄」也设定为NULL。「程序执行实体句柄」设定为执行实体句柄，它是作为WinMain的参数传递给这个程序的。最后，「建立参数」指标设定为NULL，可以用这个参数存取稍后程序中可能引用到的数据。

CreateWindow传回被建立的窗口的句柄，该句柄存放在变量hwnd中，后者被定义为HWND型态（「窗口句柄型态」）。Windows中的每个窗口都有一个句柄，程序用句柄来使用窗口。许多Windows函数需要使用hwnd作为参数，这样，Windows才能知道函数是针对哪个窗口的。如果一个程序建立了许多窗口，则每个窗口均有一个句柄。窗口句柄是Windows程序所处理最重要的句柄之一。

显示窗口

在CreateWindow呼叫传回之后，Windows内部已经建立了这个窗口。这就是说，Windows已经配置了一块内存，用来保存在CreateWindow呼叫中指定窗口的全部信息跟一些其它信息，而Windows稍后就是依据窗口句柄找到这些信息的。

然而，光是这样子，窗口并不会出现在视讯显示器上。您还需要两个函数呼叫，一个是：

ShowWindow (hwnd, iCmdShow) ;

第一个参数是刚刚用CreateWindow建立的窗口句柄。第二个参数是作为参数传给WinMain的iCmdShow。它确定最初如何在屏幕上显示窗口，是一般大小、最小化还是最大化。在开始菜单中安装程序时，使用者可能做出最佳选择。如果窗口按一般大小显示，那么WinMain接收到后传递

给 ShowWindow 的就是 SW_SHOWNORMAL；如果窗口是最大化显示的，则为 SW_SHOWMAXIMIZED。而如果窗口只显示在工作列上，则是SW_SHOWMINNOACTIVE。

ShowWindow 函数在显示器上显示窗口。如果 ShowWindow 的第二个参数是 SW_SHOWNORMAL，则窗口的显示区域就会被窗口类别中定义的背景画刷所覆盖。函数呼叫

UpdateWindow (hwnd) ;

会重画显示区域。它经由发送给窗口消息处理程序（即HELLOWIN.C中的WndProc函数）一个WM_PAINT消息做到这一点。后面，我们将说明WndProc如何处理这个消息。

消息循环

呼叫UpdateWindow之后，窗口就出现在视讯显示器上。程序现在必须准备读入使用者用键盘和鼠标输入的数据。Windows为当前执行的每个Windows程序维护一个「消息队列」。在发生输入事件之后，Windows将事件转换为一个「消息」并将消息放入程序的消息队列中。

程序通过执行一块称之为「消息循环」的程序代码从消息队列中取出消息：

```
while (GetMessage (&msg, NULL, 0, 0))
{
  TranslateMessage (&msg) ;
  DispatchMessage (&msg) ;
}
```

msg变量是型态为MSG的结构，型态MSG在WINUSER.H中定义如下：

```
typedef struct tagMSG
{
  HWND hwnd ;
  UINT message ;
  WPARAM wParam ;
  LPARAM lParam ;
  DWORD time ;
  POINT pt ;
}
MSG, * PMSG ;
```

POINT数据型态也是一个结构，它在WINDEF.H中定义如下：

```
typedef struct tagPOINT
{
  LONG x ;
  LONG y ;
}
POINT, * PPOINT;
```

消息循环以GetMessage呼叫开始，它从消息队列中取出一个消息：

GetMessage (&msg, NULL, 0, 0)

这一呼叫传给Windows一个指标，指向名为msg的MSG结构。第二、第三和第四个参数设定为NULL或者0，表示程序接收它自己建立的所有窗口的所有消息。Windows用从消息队列中取出的下一个消息来填充消息结构的各个字段，结构的各个字段包括：

hwnd 接收消息的窗口句柄。在HELLOWIN程序中，这一参数与CreateWindow传回的hwnd值相同，因为这是该程序拥有的唯一窗口。

message 消息标识符。这是一个数值，用以标识消息。对于每个消息，均有一个对应的标识符，这些标识符定义于Windows表头文件（其中大多数在WINUSER.H中），以前缀WM（「window message」，窗口消息）开头。例如，使用者将鼠标光标放在HELLOWIN显示区域之内，并按下鼠标左按钮，Windows就在消息队列中放入一个消息，该消息的message字段等于WM_LBUTTONDOWN。这是一个常数，其值为0x0201。

wParam 一个32位的「message parameter（消息参数）」，其含义和数值根据消息的不同而不同。

lParam 一个32位的消息参数，其值与消息有关。

time 消息放入消息队列中的时间。

pt 消息放入消息队列时的鼠标坐标。

只要从消息队列中取出消息的message字段不为WM_QUIT（其值为0x0012），GetMessage就传回一个非零值。WM_QUIT消息将导致GetMessage传回0。

TranslateMessage (&msg) ;

将msg结构传给Windows，进行一些键盘转换。（关于这一点，我们将在第六章中深入讨论。）

DispatchMessage (&msg) ;

又将msg结构回传给Windows。然后，Windows将该消息发送给适当的窗口消息处理程序，让它进行处理。这也就是说，Windows将呼叫窗口消息处理程序。在HELLOWIN中，这个窗口消息处理程序就是WndProe函数。处理完消息之后，WndProc传回到Windows。此时，Windows还停留在DispatchMessage呼叫中。在结束DispatchMessage呼叫的处理之后，Windows回到HELLOWIN，并且接着从下一个GetMessage呼叫开始消息循环。

窗口消息处理程序

以上我们所讨论的都是必要的负担：注册窗口类别，建立窗口，然后在屏幕上显示窗口，程序进入消息循环，然后不断从消息队列中取出消息来处理。

实际的动作发生在窗口消息处理程序中。窗口消息处理程序确定了在窗口的显示区域中显示些什么以及窗口怎样响应使用者输入。

在HELLOWIN中，窗口消息处理程序是命名为WndProc的函数。窗口消息处理程序可任意命名（只要求不和其它名字发生冲突）。一个Windows程序可以包含多个窗口消息处理程序。一个窗口消息处理程序总是与呼叫RegisterClass注册的特定窗口类别相关联。CreateWindow函数根据特定窗口类别建立一个窗口。但依据一个窗口类别，可以建立多个窗口。

窗口消息处理程序总是定义为如下形式：

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

注意，窗口消息处理程序的四个参数与MSG结构的前四个字段是相同的。第一个参数hwnd是接收消息的窗口的句柄，它与CreateWindow函数的传回值相同。对于与HELLOWIN相似的程序（只建立一个窗口），这个参数是程序所知道的唯一窗口句柄。如果程序是依据同一窗口类别（同时也是同一窗口消息处理程序）建立多个窗口，则hwnd标识接收消息的特定窗口。

第二个参数与MSG结构中的message字段相同，它是标识消息的数值。最后两个参数都是32位的消息参数，提供关于消息的更多信息。这些参数包含每个消息型态的详细信息。有时消息参数是两个存放在一起的16位值，而有时消息参数又是一个指向字符串或数据结构的指针。

程序通常不直接呼叫窗口消息处理程序，窗口消息处理程序通常由Windows本身呼叫。通过呼叫SendMessage函数，程序能够直接呼叫它自己的窗口消息处理程序。我们将在后面的章节讨论SendMessage函数。

处理消息

窗口消息处理程序所接受的每个消息均是用一个数值来标识的，也就是传给窗口消息处理程序

的message参数。Windows表头文件WINUSER.H为每个消息参数定义以「WM」（窗口消息）为前缀开头的标识符。

一般来说，Windows程序写作者使用switch和case结构来确定窗口消息处理程序接收的是什么消息，以及如何适当地处理它。窗口消息处理程序在处理消息时，必须传回0。窗口消息处理程序不予处理的所有消息应该被传给名为DefWindowProc的Windows函数。从DefWindowProc传回的值必须由窗口消息处理程序传回。

在HELLOWIN中，WndProc只选择处理三种消息：WM_CREATE、WM_PAINT和WM_DESTROY。窗口消息处理程序的结构如下：

```
switch (iMsg)
{
caseWM_CREATE :
//处理WM_CREATE消息
return 0 ;

caseWM_PAINT :
//处理WM_PAINT消息
return 0 ;

caseWM_DESTROY :
//处理WM_DESTROY消息
return 0 ;
}
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
```

呼叫DefWindowProc来为窗口消息处理程序不予处理的所有消息提供内定处理，这是很重要的。不然一般动作，如终止程序，将不会正常执行。

播放声音文件

窗口消息处理程序接收的第一个消息－也是WndProc选择处理的第一个消息－是WM_CREATE。当Windows在WinMain中处理CreateWindow函数时，WndProc接收这个消息。就是说，在HELLOWIN呼叫CreateWindow时，Windows将做一些它必须做的工作。在这些工作中，Windows呼叫WndProc，将第一个参数设定为窗口句柄，第二个参数设定为WM_CREATE。WndProc处理WM_CREATE消息并将控制传回给Windows。 Windows然后可以从CreateWindow呼叫中传回到HELLOWIN中，继续在WinMain中进行下一步的处理。

通常，窗口消息处理程序在WM_CREATE处理期间进行一次窗口初始化。HELLOWIN对这个消息的处理中播放一个名为HELLOWIN.WAV的声音文件。它使用简单的PlaySound函数来做到这一点。该函数说明在/Platform SDK/Graphics and Multimedia Services/Multimedia Audio/Waveform Audio中，而文件在/Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Functions中。

PlaySound的第一个参数是声音文件的名称（它也可能是在Control Panel的Sounds中定义的一种声音的别名，或者是一个程序资源）。第二个参数只有当声音文件是一种资源时才被使用。第三个参数指定一些选项。在这个例子中，我指定第一个参数是一个文件名，并且异步地播放声音，即PlaySound函数呼叫在声音文件开始播放时立即传回，而不会等待它的完成。在这种方法下，程序能够继续初始化。

WndProc通过从窗口消息处理程序中传回0，结束了整个WM_CREATE的处理。

WM_PAINT消息

WndProc处理的第二个消息为WM_PAINT。这个消息在Windows程序设计中是很重要的。当窗口显示区域的一部分显示内容或者全部变为「无效」，以致于必须「更新画面」时，将由这个消

息通知程序。

显示区域的显示内容怎么会变得无效呢？在最初建立窗口的时候，整个显示区域都是无效的，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中呼叫UpdateWindow时）指示窗口消息处理程序在显示区域上画一些东西。

在使用者改变HELLOWIN窗口的大小后，显示区域的显示内容重新变得无效。读者应该还记得，HELLOWIN中wndclass结构的style字段设定为标志CS_HREDRAW和CS_VREDRAW，这样的格式设定指示Windows，在窗口大小改变后，就把整个窗口显示内容当成无效。然后，窗口消息处理程序将收到一条WM_PAINT消息。

当使用者将HELLOWIN最小化，然后再次将窗口恢复为以前的大小时，Windows将不会保存显示区域的内容。在图形环境下，窗口显示区域涉及的数据量很大。因此，Windows令窗口无效，窗口消息处理程序接收一条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迭时，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。窗口消息处理程序接收到一条WM_PAINT消息，以更新窗口的内容。

对WM_PAINT的处理几乎总是从一个BeginPaint呼叫开始：

hdc = BeginPaint (hwnd, &ps) ;

而以一个EndPaint呼叫结束：

EndPaint (hwnd, &ps) ;

在这两个呼叫中，第一个参数都是程序的窗口句柄，第二个参数是指向型态为PAINTSTRUCT的结构指针。PAINTSTRUCT结构中包含一些窗口消息处理程序，可以用来更新显示区域的内容。我们将在下一章中讨论该结构的各个字段。现在我们只在BeginPaint和EndPaint函数中用到它。

在BeginPaint呼叫中，如果显示区域的背景还未被删除，则由Windows来删除。它使用注册窗口类别的WNDCLASS结构的hbrBackground字段中指定的画刷来删除背景。在HELLOWIN中，这是一个白色备用画刷。这意味着，Windows将通过把窗口背景设定为白色来删除窗口背景。BeginPaint呼叫令整个显示区域有效，并传回一个「设备内容句柄」。设备内容是指实体输出设备（如视讯显示器）及其设备驱动程序。在窗口的显示区域显示文字和图形需要设备内容句柄。但是从BeginPaint传回的设备内容句柄不能在显示区域之外绘图，读者可以试一试。EndPaint释放设备内容句柄，使之不再有效。

如果窗口消息处理程序不处理WM_PAINT消息（这是很罕见的），它们必须被传送给DefWindowProc。DefWindowProc只是依次呼叫BeginPaint和EndPaint，以使显示区域有效。

呼叫完BeginPaint之后，WndProc接着呼叫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标，指向一个RECT型态的rectangle结构。该结构有四个LONG字段，分别为left、top、right和bottom。GetClientRect将这四个字段设定为窗口显示区域的尺寸。left和top字段通常设定为0，right和bottom字段设定为显示区域的宽度和高度（像素点数）。

WndProc除了将该RECT结构指针作为DrawText的第四个参数传递外，不再对它做其它处理：

```
DrawText ( hdc, TEXT ("Hello, Windows 98!"), -1, &rect,
        DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
```

DrawText可以输出文字（正如其名字所表明的一样）。由于该函数要输出文字，第一个参数是

从BeginPaint传回的设备内容句柄，第二个参数是要输出的文字，第三个参数是 -1，指示字符串是以字节0终结的。

DrawText最后一个参数是一系列位旗标，它们均在WINUSER.H中定义（虽然由于其显示输出的效果，使得DrawText像一个GDI函数呼叫，但它确实因为相当高级的画图功能而成为User模块的一部分。此函数在/Platform SDK/Graphics and Multimedia Services/GDI/Fonts and Text中说明）。旗标指示了文字必须显示在一行上，水平方向和垂直方向都位于第四个参数指定的矩形中央。因此，这个函数呼叫将让字符串「Hello, Windows 98!」显示在显示区域的中央。

一旦显示区域变得无效（正如在改变大小时所发生的情况一样），WndProc就接收到一个新的WM_PAINT消息。WndProc通过呼叫GetClientRect取得变化后的窗口大小，并在新窗口的中央显示文字。

WM_DESTROY消息

WM_DESTROY消息是另一个重要消息。这一个消息指示，Windows正在根据使用者的指示关闭窗口。该消息是使用者单击Close按钮或者在程序的系统菜单上选择 Close时发生的（在本章的后面，我们将详细讨论WM_DESTROY消息是如何生效的）。

HELLOWIN通过呼叫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个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传回非0值。而当GetMessage得到一个WM_QUIT消息时，它传回0。这将导致WinMain退出消息循环，并终止程序。然后程序执行下面的叙述：

```
return msg.wParam ;
```

结构的wParam字段是传递给PostQuitMessage函数的值（通常是0）。然后return叙述将退出WinMain并终止程序。

Windows 程序设计的难点

即使有了对HELLOWIN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HELLOWIN中，WinMain只包含了注册窗口类别，建立窗口，从消息队列中取出消息和发送消息所必须的程序代码。

程序的所有实际动作均在窗口消息处理程序中发生。在HELLOWIN中，这些动作不多，WndProc只是简单地播放了一个声音文件并在窗口中显示一个字符串。但是在后面的章节中，读者将发现，Windows程序所作的一切，都是响应发送给窗口消息处理程序的消息。这是概念上的主要难点之一，在开始写作Windows程序之前，必须先搞清楚。

别呼叫我，我会呼叫您

前面我们提到过，程序写作者已经熟悉了使用操作系统呼叫的做法。例如，C程序写作者使用fopen函数打开文件。fopen函数最终通过呼叫操作系统来打开文件，这一点问题也没有。

但是Windows不同，尽管Windows有1000个以上的函数可供程序呼叫，但Windows也呼叫使用者程序，比如前面定义的窗口消息处理程序WndProc。窗口消息处理程序与窗口类别相关，窗口类别是程序呼叫RegisterClass注册的。依据该类别建立的窗口使用这个窗口消息处理程序来处理

窗口的所有消息。Windows通过呼叫窗口消息处理程序对窗口发送消息。

在第一次建立窗口时，Windows呼叫WndProc。在窗口关闭时，Windows也呼叫WndProc。窗口改变大小、移动或者变成图标时，从菜单中选择某一项目、挪动滚动条、按下鼠标按钮或者从键盘输入字符时，以及窗口显示区域必须被更新时，Windows都要呼叫WndProc。

所有这些WndProc呼叫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可以发送给窗口消息处理程序的消息通常都以WM开头的名字标识，并且都在WINUSER.H表头文件中定义。

实际上，从程序外呼叫程序内的例程这一种做法，在传统的程序设计中并非前所未闻。C中的signal函数可以拦截Ctrl-C中断或操作系统的其它中断。为MS-DOS编写的老程序中经常有拦截硬件中断的程序代码。

但在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消息处理程序。然后，窗口消息处理程序以某种方式响应这个消息，或者将消息传给DefWindowProc，进行内定处理。

在HELLOWIN中，窗口消息处理程序的wParam和lParam参数除了作为传递给DefWindowProc的参数外，不再有其它用处。这些参数给出了关于消息的其它信息，参数的含义与具体消息相关。

让我们来看一个例子。一旦窗口的显示区域大小发生了改变，Windows就呼叫窗口的窗口消息处理程序。窗口消息处理程序的hwnd参数是改变大小的窗口的句柄（请记住，一个窗口消息处理程序能处理依据同一个窗口类别建立的多个窗口的消息。参数hwnd让窗口消息处理程序知道是哪个窗口在接收消息）。参数message是WM_SIZE。消息WM_SIZE的参数wParam的值是SIZE_RESTORED、SIZE_MINIMIZED、SIZE_MAXIMIZED、SIZE_MAXSHOW或SIZE_MAXHIDE （在WINUSER.H表头文件中分别定义为数字0到4）。也就是说，参数wParam表明窗口是非最小化还是非最大化，是最小化、最大化，还是隐藏。

lParam参数包含了新窗口的大小，新宽度和新高度均为16位值，合在一起成为32位的lParam。WINDEF.H中提供了帮助程序写作者从lParam中取出这两个值的宏，我们将在下一章说明这个宏。

有时候，DefWindowProc处理完消息后会产生其它的消息。例如，假设使用者执行HELLOWIN，并且使用者最终单击了 Close按钮，或者假设用键盘或鼠标从系统菜单中选择了 Close，DefWindowProc处理这一键盘或者鼠标输入，在检测到使用者选择了Close选项之后，它给窗口消息处理程序发送一条WM_SYSCOMMAND消息。WndProc将这个消息传给DefWindowProc。DefWindowProc给窗口消息处理程序发送一条WM_CLOSE消息来响应之。WndProc再次将它传给DefWindowProc。DestroyWindow呼叫DestroyWindow来响应这条WM_CLOSE消息。DestroyWindow导致Windows给窗口消息处理程序发送一条WM_DESTROY消息。WndProc再呼叫PostQuitMessage，将一条WM_QUIT消息放入消息队列中，以此来响应此消息。这个消息导致WinMain中的消息循环终止，然后程序结束。

队列化消息与非队列化消息

我们已经谈到过，Windows给窗口发送消息，这意味着Windows呼叫窗口消息处理程序。但是，Windows程序也有一个消息循环，它呼叫GetMessage从消息队列中取出消息，并且呼叫DispatchMessage将消息发送给窗口消息处理程序。

那么，Windows程序是依次等待消息（类似于普通程序中相同的键盘输入），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。

消息能够被分为「队列化的」和「非队列化的」。队列化的消息是由Windows放入程序消息队列中的。在程序的消息循环中，重新传回并分配给窗口消息处理程序。非队列化的消息在Windows呼叫窗口时直接送给窗口消息处理程序。也就是说，队列化的消息被「发送」给消息队列，而非队列化的消息则「发送」给窗口消息处理程序。任何情况下，窗口消息处理程序都将获得窗口所有的消息--包括队列化的和非队列化的。窗口消息处理程序是窗口的「消息中心」。

队列化消息基本上是使用者输入的结果，以击键（如WM_KEYDOWN和WM_KEYUP消息）、击键产生的字符（WM_CHAR）、鼠标移动（WM_MOUSEMOVE）和鼠标按钮（WM_LBUTTONDOWN）的形式给出。队列化消息还包含时钟消息（WM_TIMER）、更新消息（WM_PAINT）和退出消息（WM_QUIT）。

非队列化消息则是其它消息。在许多情况下，非队列化消息来自呼叫特定的Windows函数。例如，当WinMain呼叫CreateWindow时，Windows将建立窗口并在处理中给窗口消息处理程序发送一个WM_CREATE消息。当WinMain呼叫ShowWindow时，Windows将给窗口消息处理程序发送WM_SIZE和WM_SHOWWINDOW消息。当WinMain呼叫UpdateWindow时，Windows将给窗口消息处理程序发送WM_PAINT消息。键盘或鼠标输入时发出的队列化消息信号，也能在非队列化消息中出现。例如，用键盘或鼠标选择了一个菜单项时，键盘或鼠标消息就是队列化的，而说明菜单项已选中的WM_COMMAND消息则可能就是非队列化的。

这一过程显然很复杂，但幸运的是，其中的大部分是由Windows解决的，不关我们的程序的事。从窗口消息处理程序的角度来看，这些消息是以一种有序的、同步的方式进出的。窗口消息处理程序可以处理它们，也可以不处理。

当我说消息是以一种有序的同步的方式进出时，我是说首先消息与硬件的中断不同。在一个窗口消息处理程序中处理消息时，程序不会被其它消息突然中断。

虽然Windows程序可以多线程执行，但每个执行绪的消息队列只为窗口消息处理程序在该执行绪中执行的窗口处理消息。换句话说，消息循环和窗口消息处理程序不是并发执行的。当一个消息循环从其消息队列中接收一个消息，然后呼叫DispatchMessage将消息发送给窗口消息处理程序时，直到窗口消息处理程序将控制传回给Windows，DispatchMessage才能结束执行。

当然，窗口消息处理程序能呼叫给窗口消息处理程序发送另一个消息的函数。这时，窗口消息处理程序必须在函数呼叫传回之前完成对第二个消息的处理。那时窗口消息处理程序将处理最初的消息。例如，当窗口过程调用UpdateWindow时，Windows将呼叫窗口消息处理程序来处理WM_PAINT消息。窗口消息处理程序处理WM_PAINT消息结束以后，UpdateWindow呼叫将把控制传回给窗口消息处理程序。

这也就是说窗口消息处理程序必须是可重入。在大多数情况下，这不会带来问题，但是程序写作者应该意识到这一点。例如，假设您在窗口消息处理程序中处理一个消息时设置了一个静态变量，然后呼叫了一个Windows函数。在这个函数传回时，您还能保证那个变数的值还是原来那个吗？难说--很可能您呼叫的Windows函数产生了另外一个消息，并且窗口消息处理程序在处理这个消息时改变了该变量的值。这也是在编译Windows程序时，有些编译最佳化选项必须关闭的原因之一。

在许多情况下，窗口消息处理程序必须保存它从消息中取得的信息，并在处理另一个消息时使用这些信息。这些信息可以储存在窗口的静态（static）变量或整体变量中。

当然，读者将在下面几章对此有一个更清楚的了解，因为窗口消息处理程序将处理更多的消息。

行动迅速

Windows 98和Windows NT都是优先权式的多任务环境。这意味着当一个程序在进行一项长时间工作时，Windows可以允许使用者将控制切换到另一个程序中。这是一件好事，也是现在的

Windows优越于以前16位Windows的地方。

然而，由于Windows设计的方式，这种优先权式多任务并不总是以您希望的样子工作。例如，假设您的程序花费一分钟左右来处理某一个消息。是的，使用者可以将控制切换到另一个程序，但是却无法对您的程序进行任何动作。使用者无法移动您的程序窗口、缩放它、最小化、关闭它、什么都不能做。这是因为您的窗口消息处理程序正忙于进行一项长时间的作业。表面上并不是窗口消息处理程序在执行它自己的移动和缩放操作，但实际上确实是它在做。这就是DefWindowProc部分的工作，它必须被考虑为您的窗口消息处理程序的一部分。

如果您的程序在处理某些消息时需要长时间的作业的话，可以选择我在第二十章里描述的那些方法来做得更有优雅一些。即使是在优先权式多任务环境中，也不应该让您的程序呆在屏幕上一动不动。这会让使用者讨厌的，他们会认为您的程序中有bug、不标准的动作，说明文件没写好。最好让使用者觉得程序只停了一下子就把全部消息中快速料理完了。

第四章 输出文字

在前一章，您看到了一个简单的Windows 98程序，它在窗口中央，或者更准确地说，在显示区域中央显示一行文字。正如我们学到的，显示区域是整个应用程序窗口中未被标题栏、窗口边框，以及可选的菜单列、工具列、状态列和滚动条占据的部分。简而言之，显示区域是窗口中可以由程序任意书写和传递视觉信息的部分。

对于程序的显示区域，您几乎可以为所欲为，只不过您不能假定窗口大小是某一特定尺寸，或者在程序执行时其大小会保持不变。如果您不熟悉图形窗口环境的程序设计，这些限制可能会使您感到惊讶：不能再假设屏幕上的一行文字一定有80个字符了。您的程序必须与其它Windows程序共享视讯显示器。Windows使用者控制程序窗口在屏幕上显示的方式。尽管可以建立固定大小的窗口（这对于计算器之类的应用是合理的），但在大多数情况下，使用者应该能够改变应用程序窗口的大小。您的程序必须能够接受指定给它的大小，并且合理地利用这一空间。

这有两种可能的情况。一种可能是，程序只有仅能显示「hello」的显示区域；还有另一种可能，即程序在一个大屏幕、高分辨率的系统上执行，其显示区域大得足以显示两整页文字。灵活地处理这两种极端是Windows程序设计的要点之一。

这一章，我们将讲述程序在显示区域显示信息的方式，但比上一章说明的显示方式更加复杂。当程序在显示区域显示文字或图形时，它经常要「绘制」它的显示区域。本章着重讲述绘制的方法。

尽管Windows为显示图形提供了强大的图形设备接口（GDI）函数，但在这一章中，我只介绍简单文字行的显示。我也将忽略Windows能够使用的不同字体外形及字体大小，仅使用Windows的内定系统字体。这看起来似乎是一种限制，其实不然，本章涉及和解决的问题适用于所有Windows程序设计。在混合显示文字和图形时，Windows内定字体的字符大小通常决定了图形的尺寸。

本章表面上是讨论绘图的方法，实际上是讨论与设备无关的程序设计基础。Windows程序只能对显示区域大小甚至字符的大小做很少的假定，相反地，必须使用Windows提供的功能来取得关于程序执行环境的信息。

绘制和更新

在文字模式环境下，程序可以在显示器的任意部分输出，程序输出到屏幕上的内容会停留在原处，不会神秘地消失。因此，程序可以丢掉重新生成屏幕显示时所需的信息。

在Windows中，只能在窗口的显示区域绘制文字和图形，而且不能确保在显示区域内显示的内容会一直保留到程序下一次有意地改写它时还保留在那里。例如，使用者可能会在屏幕上移动另一个程序的窗口，这样就可能覆盖您的应用程序窗口的一部分。Windows不会保存您的窗口中被其它程序覆盖的区域，当程序移开后，Windows会要求您的程序更新显示区域的这个部分。

Windows是一个消息驱动系统。它通过把消息投入应用程序消息队列中或者把消息发送给合适的窗口消息处理程序，将发生的各种事件通知给应用程序。Windows通过发送WM_PAINT消息通知窗口消息处理程序，窗口的部分显示区域需要绘制。

WM_PAINT消息

大多数Windows程序在WinMain中进入消息循环之前的初始化期间都要呼叫函数UpdateWindow。Windows利用这个机会给窗口消息处理程序发送第一个WM_PAINT消息。这个消息通知窗口消息处理程序：必须绘制显示区域。此后，窗口消息处理程序应在任何时刻都准备好处理其它WM_PAINT消息，必要的话，甚至重新绘制窗口的整个显示区域。在发生下面几种事件之一时，窗口消息处理程序会接收到一个WM_PAINT消息：

在使用者移动窗口或显示窗口时，窗口中先前被隐藏的区域重新可见。

使用者改变窗口的大小（如果窗口类别样式有着CS_HREDRAW和CS_VREDRAW位旗标的设定）。

程序使用ScrollWindow或ScrollDC函数滚动显示区域的一部分。

程序使用InvalidateRect或InvalidateRgn函数刻意产生WM_PAINT消息。

在某些情况下，显示区域的一部分被临时覆盖，Windows试图保存一个显示区域，并在以后恢复它，但这不一定能成功。在以下情况下，Windows可能发送WM_PAINT消息：

Windows擦除覆盖了部分窗口的对话框或消息框。

菜单下拉出来，然后被释放。

显示工具提示消息。

在某些情况下，Windows总是保存它所覆盖的显示区域，然后恢复它。这些情况是：

鼠标光标穿越显示区域。

图标拖过显示区域。

处理WM_PAINT消息要求程序写作者改变自己向显示器输出的思维方式。程序应该组织成可以保留绘制显示区域需要的所有信息，并且仅当「响应要求」– 即Windows给窗口消息处理程序发送WM_PAINT消息时才进行绘制。如果程序在其它时间需要更新其显示区域，它可以强制Windows产生一个WM_PAINT消息。这看来似乎是在屏幕上显示内容的一种舍近求远的方法。但您的程序结构可以从中受益。

有效矩形和无效矩形

尽管窗口消息处理程序一旦接收到WM_PAINT消息之后，就准备更新整个显示区域，但它经常只需要更新一个较小的区域（最常见的是显示区域中的矩形区域）。显然，当对话框覆盖了部分显示区域时，情况即是如此。在擦除对话框之后，需要重画的只是先前被对话框遮住的矩形区域。

这个区域称为「无效区域」或「更新区域」。正是显示区域内无效区域的存在，才会让Windows将一个WM_PAINT消息放在应用程序的消息队列中。只有在显示区域的某一部分失效时，窗口才会接受WM_PAINT消息。

Windows内部为每个窗口保存一个「绘图信息结构」，这个结构包含了包围无效区域的最小矩形的坐标以及其它信息，这个矩形就叫做「无效矩形」，有时也称为「无效区域」。如果在窗口消息处理程序处理WM_PAINT消息之前显示区域中的另一个区域变为无效，则Windows计算出一个包围两个区域的新的无效区域（以及一个新的无效矩形），并将这种变化后的信息放在绘制信息结构中。Windows不会将多个WM_PAINT消息都放在消息队列中。

窗口消息处理程序可以通过呼叫InvalidateRect使显示区域内的矩形无效。如果消息队列中已经包含一个WM_PAINT消息，Windows将计算出新的无效矩形。否则，它将一个新的WM_PAINT消息放入消息队列中。在接收到WM_PAINT消息时，窗口消息处理程序可以取得无效矩形的坐标（我们马上就会看到这一点）。通过呼叫GetUpdateRect，可以在任何时候取得这些坐标。

在处理WM_PAINT消息处理期间，窗口消息处理程序在呼叫了BeginPaint之后，整个显示区域即变为有效。程序也可以通过呼叫ValidateRect函数使显示区域内的任意矩形区域变为有效。如果这呼叫具有令整个无效区域变为有效的效果，则目前队列中的任何WM_PAINT消息都将被删除。

GDI 简介

要在窗口的显示区域绘图，可以使用Windows的图形设备接口（GDI）函数。Windows提供了几个GDI函数，用于将字符串输出到窗口的显示区域内。我们已经在上一章看过DrawText函数，但是目前使用最为普遍的文字输出函数是TextOut。该函数的格式如下：

TextOut (hdc, x, y, psText, iLength) ;

TextOut向窗口的显示区域写入字符串。psText参数是指向字符串的指针，iLength是字符串的长度。x和y参数定义了字符串在显示区域的开始位置（不久会讲述关于它们的详细情况）。hdc参数是「设备内容句柄」，它是GDI的重要部分。实际上，每个GDI函数都需要将这个句柄作为函数的第一个参数。

设备内容

读者可能还记得，句柄只不过是一个数值，Windows以它在内部使用对象。程序写作者从Windows取得句柄，然后在其它函数中使用该句柄。设备内容句柄是GDI函数的窗口「通行证」，有了这种设备内容句柄，程序写作者就能自如地在显示区域上绘图，使图形如自己所愿地变得好看或者难看。

设备内容（简称为「DC」）实际上是GDI内部保存的数据结构。设备内容与特定的显示设备（如视讯显示器或打印机）相关。对于视讯显示器，设备内容总是与显示器上的特定窗口相关。

设备内容中的有些值是图形「属性」，这些属性定义了GDI绘图函数工作的细节。例如，对于TextOut，设备内容的属性确定了文字的颜色、文字的背景色、x坐标和y坐标映像到窗口的显示区域的方式，以及显示文字时Windows使用的字体。

当程序需要绘图时，它必须先取得设备内容句柄。在取得了该句柄后，Windows用内定的属性值填入内部设备内容结构。在后面的章节中您会看到，可以通过呼叫不同的GDI函数改变这些默认值。利用其它的GDI函数可以取得这些属性的目前值。当然，还有其它的GDI函数能够在窗口的显示区域真正地绘图。

当程序在显示区域绘图完毕后，它必须释放设备内容句柄。句柄被程序释放后就不再有效，且不能再被使用。程序必须在处理单个消息处理期间取得和释放句柄。除了呼叫CreateDC（函数，在本章暂不讲述）建立的设备内容之外，程序不能在两个消息之间保存其它设备内容句柄。

Windows应用程序一般使用两种方法来取得设备内容句柄，以备在屏幕上绘图。

取得设备内容句柄：方法一

在处理WM_PAINT消息时，使用这种方法。它涉及BeginPaint和EndPaint两个函数，这两个函数需要窗口句柄（作为参数传给窗口消息处理程序）和PAINTSTRUCT结构的变量（在WINUSER.H表头文件中定义）的地址为参数。Windows程序写作者通常把这一结构变量命名为ps并且在窗口消息处理程序中定义它：

PAINTSTRUCT ps ;

在处理WM_PAINT消息时，窗口消息处理程序首先呼叫BeginPaint。BeginPaint函数一般在

准备绘制时导致无效区域的背景被擦除。该函数也填入ps结构的字段。BeginPaint传回的值是设备内容句柄，这一传回值通常被保存在叫做hdc的变量中。它在窗口消息处理程序中的定义如下：

HDC hdc ;

HDC数据型态定义为32位的无正负号整数。然后，程序就可以使用需要设备内容句柄的TextOut等GDI函数。呼叫EndPaint即可释放设备内容句柄。

一般地，处理WM_PAINT消息的形式如下：

```
case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;
  //使用GDI函数
  EndPaint (hwnd, &ps) ;
  return 0 ;
```

在处理WM_PAINT消息时，必须成对地呼叫BeginPaint和EndPaint。如果窗口消息处理程序不处理WM_PAINT消息，则它必须将WM_PAINT消息传递给Windows中DefWindowProc（内定窗口消息处理程序）。DefWindowProc以下列代码处理WM_PAINT消息：

```
case WM_PAINT:
  BeginPaint (hwnd, &ps) ;
  EndPaint (hwnd, &ps) ;
  return 0 ;
```

这两个BeginPaint和EndPaint呼叫之间中没有任何叙述，仅仅使先前无效区域变为有效。但以下方法是错误的：

```
case WM_PAINT:
  return 0 ; // WRONG !!!
```

Windows将一个WM_PAINT消息放到消息队列中，是因为显示区域的一部分无效。如果不呼叫BeginPaint和EndPaint（或者ValidateRect），则Windows不会使该区域变为有效。相反，Windows将发送另一个WM_PAINT消息，且一直发送下去。

绘图信息结构

前面提到过，Windows为每个窗口保存一个「绘图信息结构」，这就是PAINTSTRUCT，定义如下：

```
typedef struct tagPAINTSTRUCT
{
  HDC hdc ;
  BOOL fErase ;
  RECT rcPaint ;
  BOOL fRestore ;
  BOOL fIncUpdate ;
  BYTE rgbReserved[32] ;
} PAINTSTRUCT ;
```

在程序呼叫BeginPaint时，Windows会适当填入该结构的各个字段值。使用者程序只使用前三个字段，其它字段由Windows内部使用。hdc字段是设备内容句柄。在旧版本的Windows中，BeginPaint的传回值也曾是这个设备内容句柄。在大多数情况下， fErase被标志为FALSE(0)，这意味着Windows已经擦除了无效矩形的背景。这最早在BeginPaint函数中发生（如果要在窗口消息处理程序中自己定义一些背景擦除行为，可以自行处理WM_ERASEBKGND消息）。Windows使用WNDCLASS结构的hbrBackground字段指定的画刷来擦除背景，这个WNDCLASS结构是程序在WinMain初始化期间登录窗口类别时使用的。许多Windows程序使用白色画刷。以下叙述设定窗口类别结构字段值：

wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;

不过，如果程序通过呼叫Windows函数InvalidateRect使显示区域中的矩形失效，则该函数

的最后一个参数会指定是否擦除背景。如果这个参数为FALSE（即0），则Windows将不会擦除背景，并且在呼叫完BeginPaint后PAINTSTRUCT结构的fErase字段将为TRUE（非零）。

PAINTSTRUCT结构的rcPaint字段是RECT型态的结构。您已经在第三章中看到，RECT结构定义了一个矩形，其四个字段为left、top、right和bottom。PAINTSTRUCT结构的rcPaint字段定义了无效矩形的边界，如图4-1所示。这些值均以像素为单位，并相对于显示区域的左上角。无效矩形是应该重画的区域。

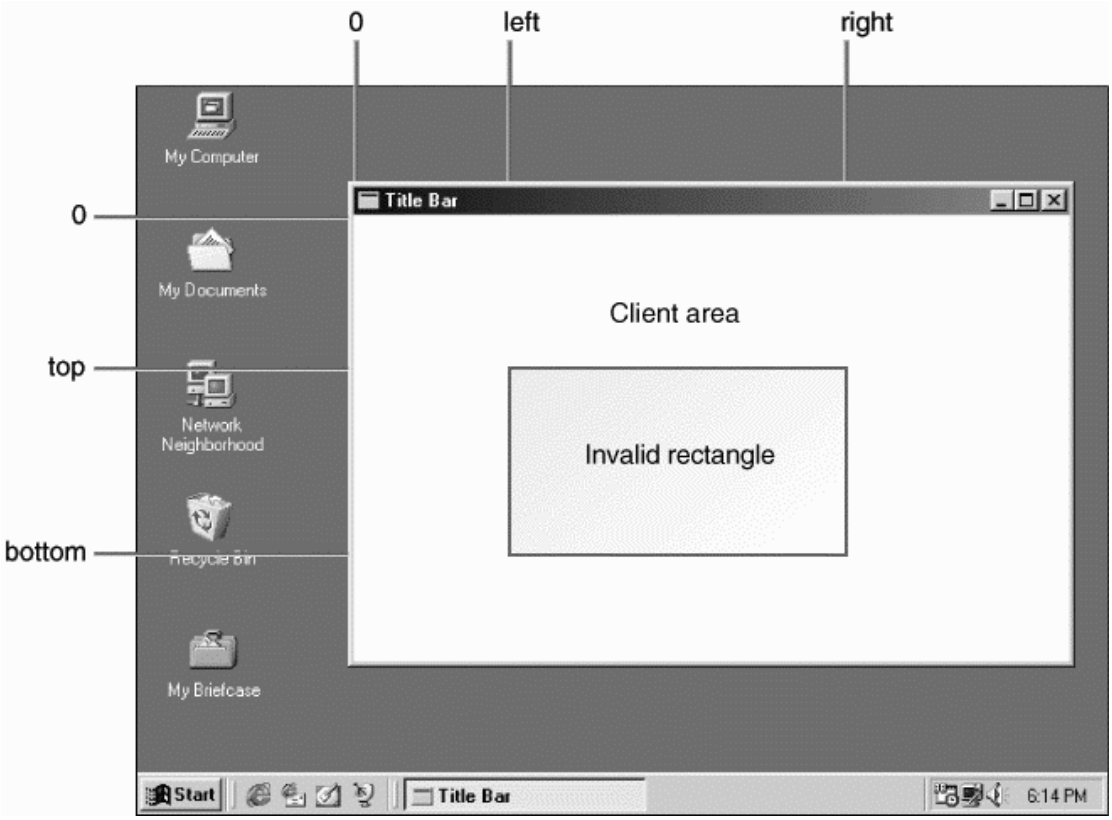


图4-1 无效矩形的边界

PAINTSTRUCT中的rcPaint矩形不仅是无效矩形，它还是一个「剪取」矩形。这意味着Windows将绘图操作限制在剪取矩形内（更确切地说，如果无效矩形区域不为矩形，则Windows将绘图操作限制在这个区域内）。

在处理WM_PAINT消息时，为了在更新的矩形外绘图，可以使用如下呼叫：

InvalidateRect (hwnd, NULL, TRUE) ;

该呼叫在BeginPaint呼叫之前进行，它使整个显示区域变为无效，并擦除背景。但是，如果最后一个参数等于FALSE，则不擦除背景，原有的东西将保留在原处。

通常这是Windows程序在无论何时收到WM_PAINT消息而不考虑rcPaint结构的情况下简单地重画整个显示区域最方便的方法。例如，如果在显示区域的显示输出中包括了一个圆，但是只有圆的一部分落到了无效矩形中，它就使仅绘制圆的无效部分变得没有意义。这需要画整个圆。在您使用从BeginPaint传回的设备内容句柄时，Windows不会绘制rcPaint矩形外的任何部分。

在第三章的HELLOWIN程序中，我们并不关心处理WM_PAINT消息时的无效矩形。如果文字显示区域恰巧在无效矩形内，则由DrawText恢复之。否则，在处理DrawText呼叫的某个时刻，

Windows会确定它无须向显示器上输出。不过，这一决定需要时间。关心程序性能和速度的程序写作者希望在处理WM_PAINT期间使用无效矩形范围，以避免不必要的GDI呼叫。如果绘制时需要存取例如位图这样的磁盘文件，则这就显得尤其重要。

取得设备内容句柄：方法二

虽然最好是在处理WM_PAINT消息处理期间更新整个显示区域，但是您也会发现在处理非WM_PAINT消息处理期间绘制显示区域的某个部分也是非常有用的。或者您需要将设备内容句柄用于其它目的，如取得设备内容的信息。

要得到窗口显示区域的设备内容句柄，可以呼叫GetDC来取得句柄，在使用完后呼叫ReleaseDC：

```
hdc = GetDC (hwnd) ;
//使用GDI函数
ReleaseDC (hwnd, hdc) ;
```

与BeginPaint和EndPaint一样，GetDC和ReleaseDC函数必须成对地使用。如果在处理某消息时呼叫GetDC，则必须在退出窗口消息处理程序之前呼叫ReleaseDC。不要在一个消息中呼叫GetDC却在另一个消息呼叫ReleaseDC。

与从BeginPaint传回设备内容句柄不同，GetDC传回的设备内容句柄具有一个剪取矩形，它等于整个显示区域。可以在显示区域的某一部分绘图，而不只是在无效矩形上绘图（如果确实存在无效矩形）。与BeginPaint不同，GetDC不会使任何无效区域变为有效。如果需要使整个显示区域有效，可以呼叫

ValidateRect (hwnd, NULL) ;

一般可以呼叫GetDC和ReleaseDC来对键盘消息（如在字处理程序中）和鼠标消息（如在画图程序中）作出反应。此时，程序可以立刻根据使用者的键盘或鼠标输入来更新显示区域，而不需要考虑为了窗口的无效区域而使用WM_PAINT消息。不过，一旦确实收到了WM_PAINT消息，程序就必须要收集足够的信息后才能更新显示。

与GetDC相似的函数是GetWindowDC。GetDC传回用于写入窗口显示区域的设备内容句柄，而GetWindowDC传回写入整个窗口的设备内容句柄。例如，您的程序可以使用从GetWindowDC传回的设备内容句柄在窗口的标题栏上写入文字。然而，程序同样也应该处理WM_NCPAINT （「非显示区域绘制」）消息。

TextOut：细节

TextOut是用于显示文字的最常用的GDI函数。语法是：

TextOut (hdc, x, y, psText, iLength) ;

以下将详细地讨论这个函数。

第一个参数是设备内容句柄，它既可以是GetDC的传回值，也可以是在处理WM_PAINT消息时BeginPaint的传回值。

设备内容的属性控制了被显示的字符串的特征。例如，设备内容中有一个属性指定文字颜色，内定颜色为黑色；内定设备内容还定义了白色的背景。在程序向显示器输出文字时，Windows使用这个背景色来填入字符周围的矩形空间（称为「字符框」）。

该文字背景色与定义窗口类别时设置的背景并不相同。窗口类别中的背景是一个画刷，它是一种纯色或者非纯色组成的画刷，Windows用它来擦除显示区域，它不是设备内容结构的一部分。在定义窗口类别结构时，大多数Windows应用程序使用WHITE_BRUSH，以便内定设备内容中的内定

文字背景颜色与Windows用以擦除显示区域背景的画刷颜色相同。

psText参数是指向字符串的指针，iLength是字符串中字符的个数。如果psText指向Unicode字符串，则字符串中的字节数就是iLength值的两倍。字符串中不能包含任何ASCII控制字符（如回车、换行、制表或退格），Windows会将这些控制字符显示为实心块。Text0ut不识别作为字符串结束标志的内容为零的字节（对于Unicode，是一个短整数型态的0），而需要由nLength参数指明长度。

TextOut中的x和y定义显示区域内字符串的开始位置，x是水平位置，y是垂直位置。字符串中第一个字符的左上角位于坐标点(x,y)。在内定的设备内容中，原点（x和y均为0的点）是显示区域的左上角。如果在TextOut中将x和y设为0，则将从显示区域左上角开始输出字符串。

当您阅读GDI绘图函数（例如TextOut）的文件时，就会发现传递给函数的坐标常常被称为「逻辑坐标」。在第五章会详细地解释这种情况。现在请注意，Windows有许多「坐标映像方式」，它们用来控制GDI函数指定的逻辑坐标转换为显示器的实际像素坐标的方式。映像方式在设备内容中定义，内定映像方式是MM_TEXT（使用WINGDI.H中定义的标识符）。在MM_TEXT映像方式下，逻辑单位与实际单位相同，都是像素；x的值从左向右递增，y的值从上向下递增（参看图4-2）。MM_TEXT坐标系与Windows在PAINTSTRUCT结构中定义无效矩形时使用的坐标系相同，这为我们带来了很多方便（但是，其它映像方式并非如此）。

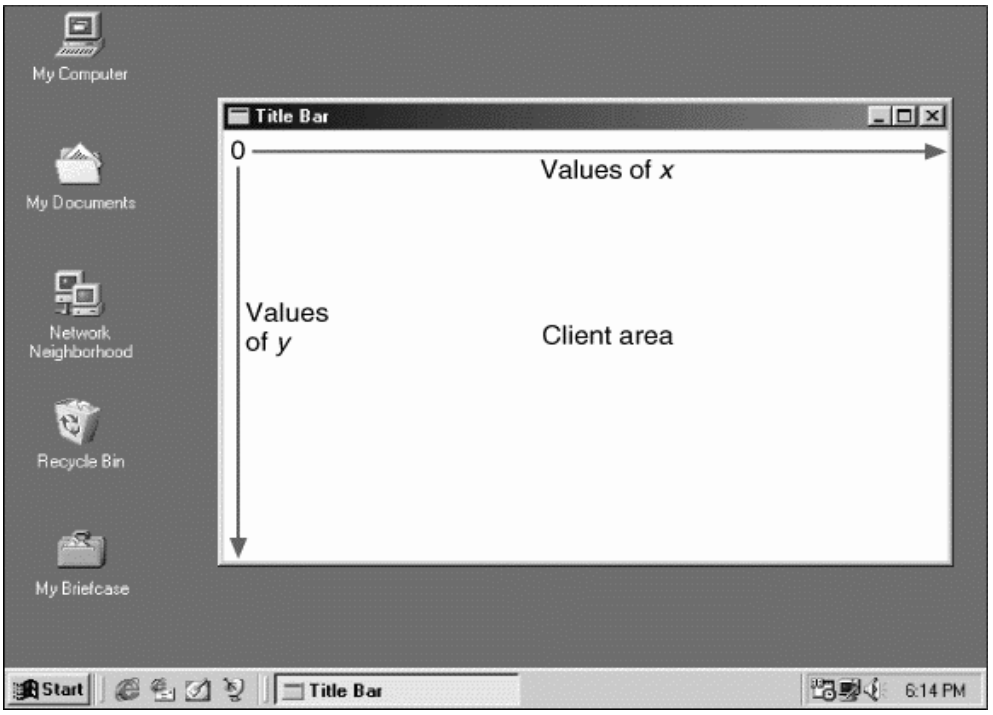


图4-2 MM_TEXT映像方式下的x坐标和y坐标

设备内容也定义了一个剪裁区域。您已经看到，对于从GetDC取得的设备内容句柄，内定剪裁区域是整个显示区域；而对于从BeginPaint取得的设备内容句柄，则为无效区域。Windows不会在剪裁区域之外的任何位置显示字符串。如果一个字符有一部分在剪裁区域外，则Windows将只显示此区域内的那部分。要想将输出写到窗口的显示区域之外不是那么容易的，所以不用担心会无意间出现这种事情。

系统字体

设备内容还定义了在您呼叫TextOut显示文字时Windows使用的字体。内定字体为「系统字

体」，或用Windows表头文件中的标识符，即SYSTEM_FONT。系统字体是Windows用来在标题栏、菜单和对话框中显示字符串的内定字体。

在Windows的早期版本中，系统字体是等宽(fixed-pitch)字体，这意味着所有字符均具有同样的宽度，非常类似于打字机。然而，从Windows 3.0开始，系统字体成为一种变宽(variable-pitch)字体，这意味着不同的字符具有不同的大小，比如，「W」要比「i」宽。变宽字体比等宽字体好读，这已经是公认的事实。不过，可以想见，这一转变使很多原来的Windows程序代码不再适用，从而要求程序写作者学习一些使用字体的新技术。

系统字体是一种「点阵字体」，这意味着字符被定义为像素块（在第十七章，将讨论TrueType字体，它是由轮廓定义的）。至于确切的大小，系统字体的字符大小取决于视讯显示器的大小。系统字体设计为至少能在显示器上显示25行80列文字。

字符大小

要用TextOut显示多行文字，就必须确定字体的字符大小，可以根据字符的高度来定位字符的后续行，以及根据字符的宽度来定位字符的后续列。

系统字体的字符高度和平均宽度是多少？这个问题取决于视讯显示器的像素大小。Windows需要的最小显示大小是640×480，但是许多使用者更喜欢800×600或1024×768的显示大小。另外，对于这些较大的显示尺寸，Windows允许使用者选择不同大小的系统字体。

程序可以呼叫GetSystemMetrics函数以取使用者接口上各类视觉组件大小的信息，呼叫GetTextMetrics取得字体大小。GetTextMetrics传回设备内容中目前选取的字体信息，因此它需要设备内容句柄。Windows将文字大小的不同值复制到在WINGDI.H中定义的TEXTMETRIC型态的结构中。TEXTMETRIC结构有20个字段，我们只使用前七个：

```
typedef struct tagTEXTMETRIC
{
  LONG tmHeight ;
  LONG tmAscent ;
  LONG tmDescent ;
  LONG tmInternalLeading ;
  LONG tmExternalLeading ;
  LONG tmAveCharWidth ;
  LONG tmMaxCharWidth ;
  //其它结构字段
}
TEXTMETRIC, * PTEXTMETRIC ;
```

这些字段值的单位取决于选定的设备内容映像方式。在内定设备内容下，映像方式是MM_TEXT，因此值的大小是以像素为单位。

要使用GetTextMetrics函数，需要先定义一个结构变量（通常称为tm）：

TEXTMETRIC tm ;

在需要确定文字大小时，先取得设备内容句柄，再呼叫GetTextMetrics：

```
hdc = GetDC (hwnd) ;
GetTextMetrics (hdc, &tm) ;
ReleaseDC (hwnd, hdc) ;
```

此后，您就可以查看文字尺寸结构中的值，并有可能保存其中的一些以备将来使用。

文字大小：细节

TEXTMETRIC结构提供了关于目前设备内容中选用的字体的丰富信息。但是，字体的纵向大小只由5个值确定，其中4个值如图4-3所示。

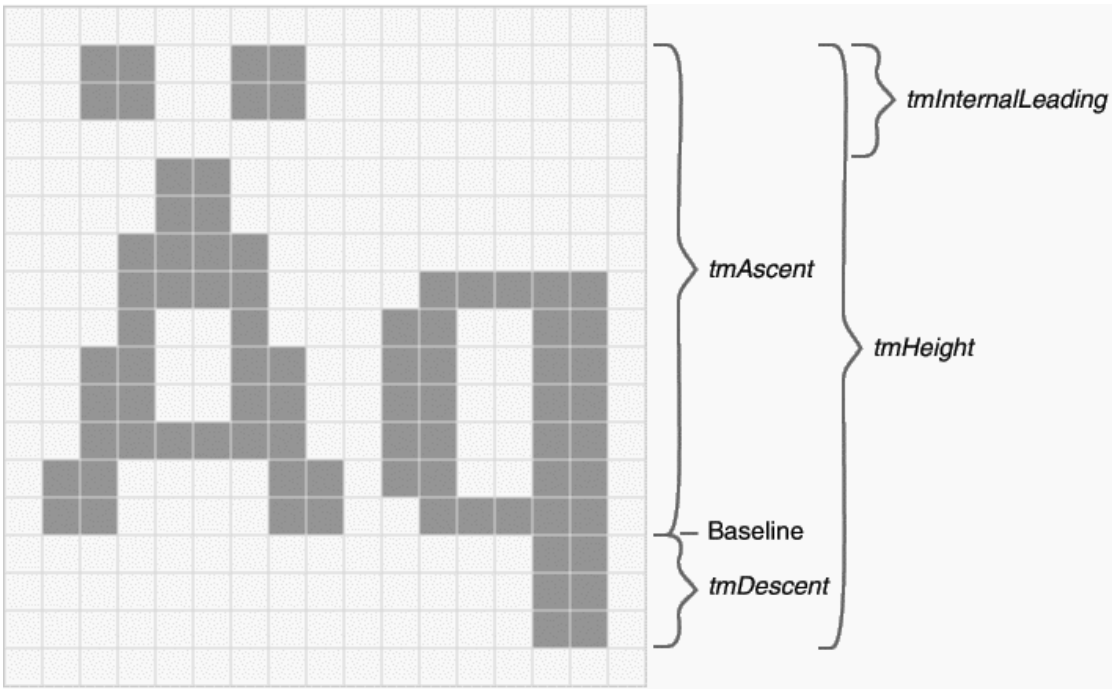


图4-3 定义字体中纵向字符大小的4个值

最重要的值是tmHeight，它是tmAscent和tmDescent的和。这两个值表示了基准在线下字符的最大纵向高度。「间距」(leading) 指打印机在两行文字间插入的空间。在TEXTMETRIC结构中，内部的间距包括在tmAscent中（因此也在tmHeight中），并且它经常是重音符号出现的地方。tmInternalLeading字段可被设成0，在这种情况下，加重音的字母会稍稍缩短以便容纳重音符号。

TEXTMETRIC结构还包括一个不包含在tmHeight值中的字段tmExternalLeading。它是字体设计者建议加在横向字符之间的空间大小。在安排文字行之间的空隙时，您可以接受设计者建议的值，也可以拒绝它。在系统字体中tmExternalLeading可以为0，因此我没有在图4-3中显示它。（尽管我不想告诉你们，图4-3确实就是Windows在640×480的显示分辨率中使用的系统字体。）

TEXTMETRICS结构包含有描述字符宽度的两个字段，即tmAveCharWidth（小写字母加权平均宽度）和tmMaxCharWidth（字体中最宽字符的宽度）。对于定宽字体，这两个值是相等的（图4-3中这些值分别为7和14）。

本章的范例程序还需要另一种字符宽度，即大写字母的平均宽度，这可以用tmAveCharWidth乘以150%大致计算出来。

必须认识到，系统字体的大小取决于Windows所执行的视讯显示器的分辨率，在某些情况下，取决于使用者选取的系统字体的大小。Windows提供了一个与设备无关的图形接口，但程序写作者还是有事情要处理的。不要想当然耳地猜测字体大小来写作Windows程序，也不要把值定死，您可以使用GetTextMetrics函数取得这一信息。

格式化文字

Windows启动后，系统字体的大小就不会发生改变，所以在程序执行过程中，程序写作者只需要呼叫一次GetTexMetrics。最好是在窗口消息处理程序中处理WM_CREATE消息时进行此呼叫，WM_CREATE消息是窗口消息处理程序接收的第一个消息。在WinMain中呼叫CreateWindow时，Windows会以一个WM_CREATE消息呼叫窗口消息处理程序。

假设要编写一个Windows程序，在显示区域显示几行文字，这需要先取得字符宽度和高度。您可以在窗口消息处理程序内定义两个变量来保存平均字符宽度(cxChar)和总的字符高度(cyChar)：

static int cxChar, cyChar ;

变量名的前缀c代表「count」，在这里指像素数，与x和y结合，分别指宽和高。这些变量定义为static静态变量，因为它们在窗口消息处理程序中处理其它消息（如WM_PAINT）时也应该是有效的。如果变量在函数外面定义，则不需要定义为static。

下面是取得系统字体的字符宽度和高度的WM_CREATE程序代码：

```
case WM_CREATE:
  hdc = GetDC (hwnd) ;
  GetTextMetrics (hdc, &tm) ;
  cxChar = tm.tmAveCharWidth ;
  cyChar = tm.tmHeight + tm.tmExternalLeading ;
  ReleaseDC (hwnd, hdc) ;
  return 0 ;
```

注意我在计算cyChar时包括了tmExternalLeading字段，虽然该字段在系统字体中为0，但是因为它使得文字的可读性更好，所以还是应该把它包括进去。沿着窗口向下每隔cyChar像素就会显示一行文字。

您会发现常常需要显示格式化的数字跟简单的字符串。我在第二章讲到过，您不能使惯用的工具（可爱的printf函数）来完成这项工作，但是可以使用sprintf和Windows版的sprintf – wsprintf。这些函数与printf相似，只是把格式化字符串放到字符串中。然后，可以用TextOut将字符串输出到显示器上。非常方便的是，从sprintf和wsprintf传回的值就是字符串的长度。您可以将这个值传递给TextOut作为iLength参数。下面的程序代码显示了wsprintf与TextOut的典型组合：

```
int iLength ;
TCHAR szBuffer [40] ;
//其它行程序
iLength = wsprintf (szBuffer, TEXT ("The sum of %i and %i is %i"),
                    iA, iB, iA + iB) ;
TextOut (hdc, x, y, szBuffer, iLength) ;
```

对于这样简单的情况，可以将nLength的定义值与TextOut放在同一条叙述中，从而无需定义iLength：

```
TextOut (hdc, x, y, szBuffer,
        wsprintf (szBuffer, TEXT ("The sum of %i and %i is %i"),
        iA, iB, iA + iB)) ;
```

虽然这样子写起来不好看，但是功能与前者是一样的。

综合使用

现在，我们似乎已经具备了在屏幕上显示多行文字所需要的所有知识。我们知道如何在WM_PAINT消息处理期间取得一个设备内容句柄，如何使用TextOut函数以及如何根据字符大小来安排字距，剩下的就是显示一点有意义的东西了。

在前一章里，我们大概知道从Windows的GetSystemMetrics函数中取得的信息是很有意义的，该函数传回Windows中不同视觉组件的大小信息，如图标、光标、标题栏和滚动条等。它们的大小因显示卡和驱动程序的不同而有所不同。GetSystemMetrics是在程序中完成与设备无关图形输出的重要函数。

该函数需要一个参数，叫做「索引」，在Windows表头文件定义了75个整数索引标识符（标识符的数量随着每个版本的Windows的发布而不断地增加，在Windows 1.0的程序写作者文件中仅列出了26个）。GetSystemMetrics传回一个整数，这个整数通常就是参数中指定的图形组件大小。

让我们来编写一个程序，显示一些可以从GetSystemMetrics呼叫中取得的信息，显示格式为每种视觉组件一行。如果我们建立一个表头文件，在表头文件中定义一个结构数组，此结构包含GetSystemMetrics索引对应的Windows表头文件标识符和呼叫所传回的每个值对应的字符串，这样处理起来要容易一些。表头文件名为SYSMETS.H，如程序4-1所示。

程序4-1 SYSMETS.H

```
/*-------------------------------------------------------
SYSMETS.H -- System metrics display structure
-------------------------------------------------------*/
#define NUMLINES ((int) (sizeof sysmetrics / sizeof sysmetrics [0]))
struct
{
  int Index ;
  TCHAR * szLabel ;
  TCHAR * szDesc ;
}
sysmetrics [] =
{
  SM_CXSCREEN, TEXT ("SM_CXSCREEN"),
    TEXT ("Screen width in pixels"),
    SM_CYSCREEN, TEXT ("SM_CYSCREEN"),
    TEXT ("Screen height in pixels"),
    SM_CXVSCROLL, TEXT ("SM_CXVSCROLL"),
    TEXT ("Vertical scroll width"),
    SM_CYHSCROLL, TEXT ("SM_CYHSCROLL"),
    TEXT ("Horizontal scroll height"),
    SM_CYCAPTION, TEXT ("SM_CYCAPTION"),
    TEXT ("Caption bar height"),
    SM_CXBORDER, TEXT ("SM_CXBORDER"),
    TEXT ("Window border width"),
    SM_CYBORDER, TEXT ("SM_CYBORDER"),
    TEXT ("Window border height"),
    SM_CXFIXEDFRAME,TEXT ("SM_CXFIXEDFRAME"),
    TEXT ("Dialog window frame width"),
    SM_CYFIXEDFRAME,TEXT ("SM_CYFIXEDFRAME"),
    TEXT ("Dialog window frame height"),
    SM_CYVTHUMB, TEXT ("SM_CYVTHUMB"),
    TEXT ("Vertical scroll thumb height"),
    SM_CXHTHUMB, TEXT ("SM_CXHTHUMB"),
    TEXT ("Horizontal scroll thumb width"),
    SM_CXICON, TEXT ("SM_CXICON"),
    TEXT ("Icon width"),
    SM_CYICON, TEXT ("SM_CYICON"),
    TEXT ("Icon height"),
    SM_CXCURSOR, TEXT ("SM_CXCURSOR"),
    TEXT ("Cursor width"),
    SM_CYCURSOR, TEXT ("SM_CYCURSOR"),
    TEXT ("Cursor height"),
    SM_CYMENU, TEXT ("SM_CYMENU"),
    TEXT ("Menu bar height"),
    SM_CXFULLSCREEN,TEXT ("SM_CXFULLSCREEN"),
    TEXT ("Full screen client area width"),
    SM_CYFULLSCREEN,TEXT ("SM_CYFULLSCREEN"),
    TEXT ("Full screen client area height"),
    SM_CYKANJIWINDOW,TEXT ("SM_CYKANJIWINDOW"),
    TEXT ("Kanji window height"),
    SM_MOUSEPRESENT, TEXT ("SM_MOUSEPRESENT"),
    TEXT ("Mouse present flag"),
    SM_CYVSCROLL,TEXT ("SM_CYVSCROLL"),
    TEXT ("Vertical scroll arrow height"),
    SM_CXHSCROLL,TEXT ("SM_CXHSCROLL"),
    TEXT ("Horizontal scroll arrow width"),
    SM_DEBUG, TEXT ("SM_DEBUG"),
    TEXT ("Debug version flag"),
    SM_SWAPBUTTON,TEXT ("SM_SWAPBUTTON"),
    TEXT ("Mouse buttons swapped flag"),
```

```
    SM_CXMIN, TEXT ("SM_CXMIN"),
    TEXT ("Minimum window width"),
    SM_CYMIN, TEXT ("SM_CYMIN"),
    TEXT ("Minimum window height"),
    SM_CXSIZE, TEXT ("SM_CXSIZE"),
    TEXT ("Min/Max/Close button width"),
    SM_CYSIZE, TEXT ("SM_CYSIZE"),
    TEXT ("Min/Max/Close button height"),
    SM_CXSIZEFRAME,TEXT ("SM_CXSIZEFRAME"),
    TEXT ("Window sizing frame width"),
    SM_CYSIZEFRAME,TEXT ("SM_CYSIZEFRAME"),
    TEXT ("Window sizing frame height"),
    SM_CXMINTRACK,TEXT ("SM_CXMINTRACK"),
    TEXT ("Minimum window tracking width"),
    SM_CYMINTRACK,TEXT ("SM_CYMINTRACK"),
    TEXT ("Minimum window tracking height"),
    SM_CXDOUBLECLK,TEXT ("SM_CXDOUBLECLK"),
    TEXT ("Double click x tolerance"),
    SM_CYDOUBLECLK,TEXT ("SM_CYDOUBLECLK"),
    TEXT ("Double click y tolerance"),
    SM_CXICONSPACING,TEXT ("SM_CXICONSPACING"),
    TEXT ("Horizontal icon spacing"),
    SM_CYICONSPACING,TEXT ("SM_CYICONSPACING"),
    TEXT ("Vertical icon spacing"),
    SM_MENUDROPALIGNMENT,TEXT ("SM_MENUDROPALIGNMENT"),
    TEXT ("Left or right menu drop"),
    SM_PENWINDOWS, TEXT ("SM_PENWINDOWS"),
    TEXT ("Pen extensions installed"),
    SM_DBCSENABLED, TEXT ("SM_DBCSENABLED"),
    TEXT ("Double-Byte Char Set enabled"),
    SM_CMOUSEBUTTONS, TEXT ("SM_CMOUSEBUTTONS"),
    TEXT ("Number of mouse buttons"),
    SM_SECURE, TEXT ("SM_SECURE"),
    TEXT ("Security present flag"),
    SM_CXEDGE, TEXT ("SM_CXEDGE"),
    TEXT ("3-D border width"),
    SM_CYEDGE, TEXT ("SM_CYEDGE"),
    TEXT ("3-D border height"),
    SM_CXMINSPACING, TEXT ("SM_CXMINSPACING"),
    TEXT ("Minimized window spacing width"),
    SM_CYMINSPACING, TEXT ("SM_CYMINSPACING"),
    TEXT ("Minimized window spacing height"),
    SM_CXSMICON, TEXT ("SM_CXSMICON"),
    TEXT ("Small icon width"),
    SM_CYSMICON, TEXT ("SM_CYSMICON"),
    TEXT ("Small icon height"),
    SM_CYSMCAPTION, TEXT ("SM_CYSMCAPTION"),
    TEXT ("Small caption height"),
    SM_CXSMSIZE, TEXT ("SM_CXSMSIZE"),
    TEXT ("Small caption button width"),
    SM_CYSMSIZE, TEXT ("SM_CYSMSIZE"),
    TEXT ("Small caption button height"),
    SM_CXMENUSIZE, TEXT ("SM_CXMENUSIZE"),
    TEXT ("Menu bar button width"),
    SM_CYMENUSIZE, TEXT ("SM_CYMENUSIZE"),
    TEXT ("Menu bar button height"),
    SM_ARRANGE, TEXT ("SM_ARRANGE"),
    TEXT ("How minimized windows arranged"),
    SM_CXMINIMIZED, TEXT ("SM_CXMINIMIZED"),
    TEXT ("Minimized window width"),
    SM_CYMINIMIZED, TEXT ("SM_CYMINIMIZED"),
    TEXT ("Minimized window height"),
    SM_CXMAXTRACK, TEXT ("SM_CXMAXTRACK"),
    TEXT ("Maximum draggable width"),
    SM_CYMAXTRACK, TEXT ("SM_CYMAXTRACK"),
    TEXT ("Maximum draggable height"),
    SM_CXMAXIMIZED, TEXT ("SM_CXMAXIMIZED"),
    TEXT ("Width of maximized window"),
    SM_CYMAXIMIZED, TEXT ("SM_CYMAXIMIZED"),
```

```
   TEXT ("Height of maximized window"),
   SM_NETWORK, TEXT ("SM_NETWORK"),
   TEXT ("Network present flag"),
   SM_CLEANBOOT, TEXT ("SM_CLEANBOOT"),
   TEXT ("How system was booted"),
   SM_CXDRAG, TEXT ("SM_CXDRAG"),
   TEXT ("Avoid drag x tolerance"),
   SM_CYDRAG, TEXT ("SM_CYDRAG"),
   TEXT ("Avoid drag y tolerance"),
   SM_SHOWSOUNDS, TEXT ("SM_SHOWSOUNDS"),
   TEXT ("Present sounds visually"),
   SM_CXMENUCHECK, TEXT ("SM_CXMENUCHECK"),
   TEXT ("Menu check-mark width"),
   SM_CYMENUCHECK, TEXT ("SM_CYMENUCHECK"),
   TEXT ("Menu check-mark height"),
   SM_SLOWMACHINE, TEXT ("SM_SLOWMACHINE"),
   TEXT ("Slow processor flag"),
   SM_MIDEASTENABLED, TEXT ("SM_MIDEASTENABLED"),
   TEXT ("Hebrew and Arabic enabled flag"),
   SM_MOUSEWHEELPRESENT,TEXT ("SM_MOUSEWHEELPRESENT"),
   TEXT ("Mouse wheel present flag"),
   SM_XVIRTUALSCREEN, TEXT ("SM_XVIRTUALSCREEN"),
   TEXT ("Virtual screen x origin"),
   SM_YVIRTUALSCREEN, TEXT ("SM_YVIRTUALSCREEN"),
   TEXT ("Virtual screen y origin"),
   SM_CXVIRTUALSCREEN, TEXT ("SM_CXVIRTUALSCREEN"),
   TEXT ("Virtual screen width"),
   SM_CYVIRTUALSCREEN, TEXT ("SM_CYVIRTUALSCREEN"),
   TEXT ("Virtual screen height"),
   SM_CMONITORS, TEXT ("SM_CMONITORS"),
   TEXT ("Number of monitors"),
   SM_SAMEDISPLAYFORMAT,TEXT ("SM_SAMEDISPLAYFORMAT"),
   TEXT ("Same color format flag")
} ;
```

显示信息的程序命名为SYSMETS1。SYSMETS1.C的原始码如程序4-2所示。现在大多数程序代码看起来都很熟悉。WinMain中的程序代码实际上与HELLOWIN中的程序代码相同，并且WndProc中的大部分程序代码都已经讨论过了。

程序4-2 SYSMETS1.C

```
/*------------------------------------------------------------------
SYSMETS1.C -- System Metrics Display Program No. 1
(c) Charles Petzold, 1998
------------------------------------------------------------------*/
#include <windows.h>
#include "sysmets.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("SysMets1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
```

```
     szAppName, MB_ICONERROR) ;

    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 1"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxChar, cxCaps, cyChar ;
  HDC hdc ;
  int i ;
  PAINTSTRUCT ps ;
  TCHAR szBuffer [10] ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar = tm.tmHeight + tm.tmExternalLeading ;
    ReleaseDC (hwnd, hdc) ;
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    for (i = 0 ; i < NUMLINES ; i++)
    {
      TextOut (hdc, 0, cyChar * i,
        sysmetrics[i].szLabel,
        lstrlen (sysmetrics[i].szLabel)) ;

      TextOut (hdc, 22 * cxCaps, cyChar * i,
        sysmetrics[i].szDesc,
        lstrlen (sysmetrics[i].szDesc)) ;

      SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
      TextOut (hdc, 22 * cxCaps + 40 * cxChar, cyChar * i, szBuffer,
        wsprintf (szBuffer, TEXT ("%5d"),
        GetSystemMetrics (sysmetrics[i].iIndex))) ;
      SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图4-4显示了在标准VGA上执行的SYSMETS1。在程序显示区域的前两行可以看到，屏幕宽度是640个像素，屏幕高度是480个像素，这两个值以及程序所显示的其它值可能会因视讯显示器型态的不同而不同。

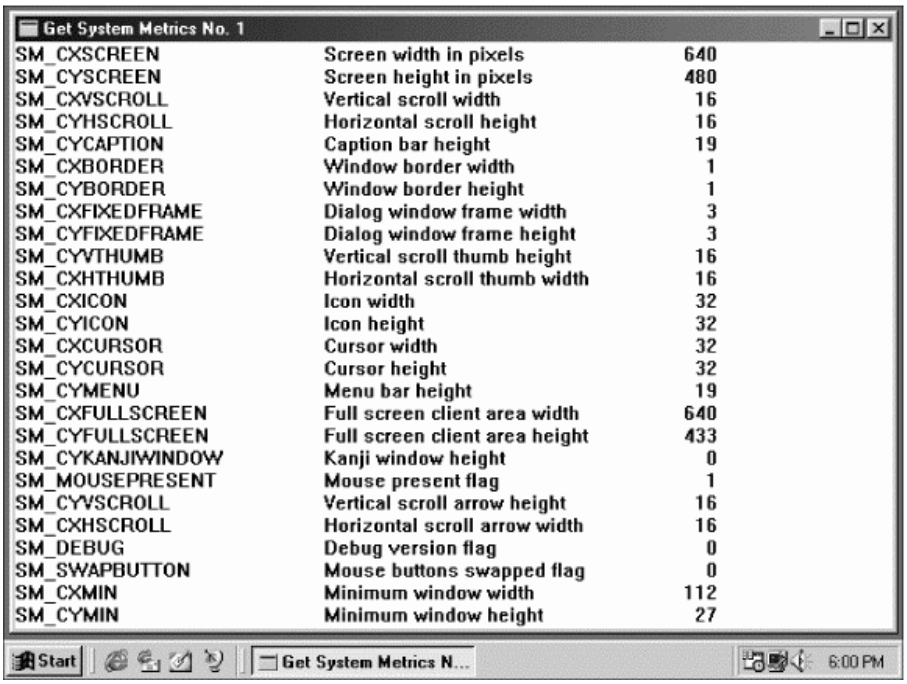


图4-4 SYSMETS1的显示

SYSMETS1.C窗口消息处理程序

SYSMETS1.C程序中的WndProc窗口消息处理程序处理三个消息：WM_CREATE、WM_PAINT和WM_DESTROY。WM_DESTROY消息的处理方法与第三章的HELLOWIN程序相同。

WM_CREATE消息是窗口消息处理程序接收到的第一个消息。在CreateWindow函数建立窗口时，Windows产生这个消息。在处理WM_CREATE消息时，SYSMETS1呼叫GetDC取得窗口的设备内容，并呼叫GetTextMetrics取得内定系统字体的文字大小。SYSMETS1将平均字符宽度保存在cxChar中，将字符的总高度（包括外部间距）保存在cyChar中。

SYSMETS1还将大写字母的平均宽度保存在静态变量cxCaps中。对于固定宽度的字体，cxCaps等于cxChar。对于可变宽度字体，cxCaps设定为cxChar乘以150%。对于可变宽度字体，TEXTMETRIC结构中的tmPitchAndFamily字段的低位为1，对于固定宽度字体，该值为0。SYSMETS1使用这个位从cxChar计算cxCaps：

```
cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
```

SYSMETS1在处理WM_PAINT消息处理期间完成所有窗口建立工作。通常，窗口消息处理程序先呼叫BeginPaint取得设备内容句柄，然后用一道for叙述对SYSMETS.H中定义的sysmetrics结构的每一行进行循环。三列文字用三个TextOut函数显示，对于每一列，TextOut的第三个参数都设定为：

```
cyChar * i
```

这个参数指示了字符串顶端相对于显示区域顶部的像素位置。

第一条TextOut叙述在第一列显示了大写标识符。TextOut的第二个参数是0，这是说文字从显示区域的左边缘开始。文字的内容来自sysmetrics结构的szLabel字段。我使用Windows函数lstrlen来计算字符串的长度，它是TextOut需要的最后一个参数。

第二条TextOut叙述显示了对系统尺寸值的描述。这些描述存放在sysmetrics结构的szDesc字

段中。在这种情况下，TextOut的第二个参数设定为：

22 * cxCaps

第一列显示的最长的大写标识符有20个字符，因此第二列必须在第一列文字开头向右20 × cxCaps处开始。我使用22，以在两列之间加一点多余的空间。

第三条TextOut叙述显示从GetSystemMetrics函数取得的数值。变宽字体使得格式化向右对齐的数值有些棘手。从0到9的数字具有相同的宽度，但是这个宽度比空格宽度大。数值可以比一个数字宽，所以不同的数值应该从不同的横向位置开始。

那么，如果我们指定字符串结束的像素位置，而不是指定字符串的开始位置，以此向右对齐数值，是否会容易一些呢？用SetTextAlign函数就可以做到这一点。在SYSMETS1呼叫：

SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;

之后，传给后续TextOut函数的坐标将指定字符串的右上角，而不是左上角。

显示列数的TextOut函数的第二个参数设定为：

22 * cxCaps + 40 * cxChar

值40*cxChar包含了第二列的宽度和第三列的宽度。在TextOut函数之后，另一个对SetTextAlign的呼叫将对齐方式设定回普通方式，以进行下次循环。

空间不够

在SYSMETS1程序中存在着一个很难处理的问题：除非您有一个大屏幕跟高分辨率的显示卡，否则就无法看到系统尺度列表的最后几行。如果窗口太窄，甚至根本看不到值。

SYSMETS1不知道这个问题。否则我们就会显示一个消息框说「抱歉!」程序甚至不知道它的显示区域有多大，它从窗口顶部开始输出文字，并仰赖Windows裁剪超出显示区域底部的内容。

显然，这很不理想。为了解决这个问题，我们的第一个任务是确定程序在显示区域内能输出多少内容。

显示区域的大小

如果您使用过现有的Windows应用程序，可能会发现窗口的尺寸变化极大。窗口最大化时（假定窗口只有标题栏并且没有菜单），显示区域几乎占据了整个屏幕。这一最大化了的显示区域的尺寸可以通过以SM_CXFULLSCREEN和SM_CYFULLSCREEN为参数呼叫GetSystemMetrics来获得。窗口的最小尺寸可以很小，有时甚至不存在，更不用说显示区域了。

在最近一章，我们使用GetClientRect函数来取得显示区域的大小。使用这个函数没有什么不好，但是在您每次要使用信息时就去呼叫它一遍是没有效率的。确定窗口显示区域大小的更好方法是在窗口消息处理程序中处理WM_SIZE消息。在窗口大小改变时，Windows给窗口消息处理程序发送一个WM_SIZE消息。传给窗口消息处理程序的lParam参数的低字组中包含显示区域的宽度，高字组中包含显示区域的高度。要保存这些尺寸，需要在窗口消息处理程序中定义两个静态变量：

static int cxClient, cyClient ;

与cxChar和cyChar相似，这两个变量在窗口消息处理程序内定义为静态变量，因为在以后处理其它消息时会用到它们。处理WM_SIZE的方法如下：

```
case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
```

```
  return 0 ;
```

实际上您会在每个Windows程序中看到类似的程序代码。LOWORD和HIWORD宏在Windows表头文件WINDEF.H中定义。这些宏的定义看起来像这样：

```
#define LOWORD(l) ((WORD)(l))
#define HIWORD(l) ((WORD)(((DWORD)(l) >> 16) & 0xFFFF))
```

这两个宏传回WORD值（16位的无正负号整数，范围从0到0xFFFF）。一般，将这些值保存在32位有号整数中。这就不会牵扯到任何转换问题，并使得这些值在以后需要的任何计算中易于使用。

在许多Windows程序中，WM_SIZE消息必然跟着一个WM_PAINT消息。为什么呢？因为在我们定义窗口类别时指定窗口类别样式为：

CS_HREDRAW | CS_VREDRAW

这种窗口类别样式告诉Windows，如果水平或者垂直大小发生改变， 则强制更新显示区域。

用如下公式计算可以在显示区域内显示的文字的总行数：

cyClient / cyChar

如果显示区域的高度太小以至无法显示一个完整的字符，这个公式的结果可以为0。类似地，在显示区域的水平方向可以显示的小写字符的近似数目为：

cxClient / cxChar

如果在处理WM_CREATE消息处理期间取得cxChar和cyChar，则不用担心在这两个计算公式中会出现被0除的情况。在WinMain呼叫CreateWindow时，窗口消息处理程序接收一个WM_CREATE消息。在WinMain呼叫ShowWindow之后接收到第一个WM_CREATE消息，此时cxChar和cyChar已经被赋予正的非零值了。

如果显示区域的大小不足以容纳所有的内容，那么，知道窗口显示区域的大小只是为使用者提供了在显示区域内卷动文字的第一步。如果您对其他有类似需求的Windows应用程序很熟悉，就很可能知道，这种情况下，我们需要使用「滚动条」。

滚动条

滚动条是图形使用者接口中最好的功能之一，它很容易使用，而且提供了很好的视觉回馈效果。您可以使用滚动条显示任何东西--无论是文字、图形、表格、数据库记录、图像或是网页，只要它所需的空间超出了窗口的显示区域所能提供的空间，就可以使用滚动条。

滚动条既有垂直方向的（供上下移动），也有水平方向的（供左右移动）。使用者可以使用鼠标在滚动条两端的箭头上或者在箭头之间的区域中点一下，这时，「卷动方块」在卷动列内的移动位置与所显示的信息在整个文件中的近似相关位置成比例。使用者也可以用鼠标拖动卷动方块到特定的位置。图4-5显示了垂直滚动条的建议用法。

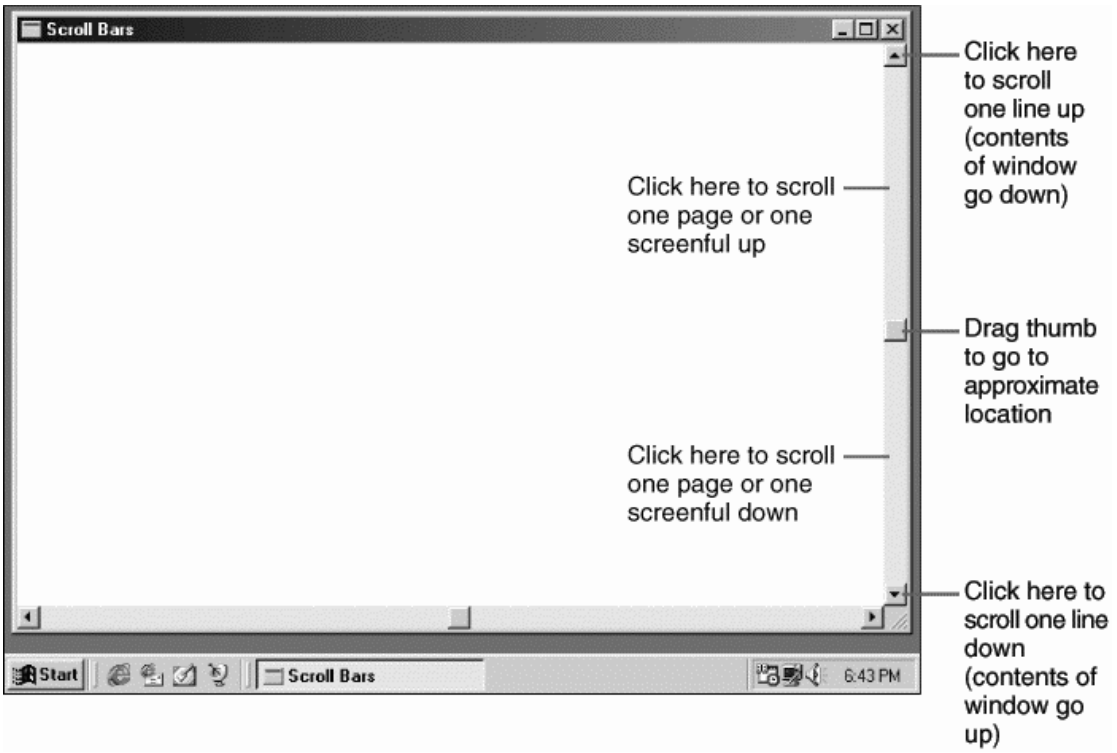


图4-5 垂直滚动条

有时，程序写作者对卷动概念很难理解，因为他们的观点与使用者的观点不同：使用者向下卷动是想看到文件较下面的部分；但是，程序实际上是将文件相对于显示窗口向上移动。Windows文件和表头文件标识符是依据使用者的观点：向上卷动意味着朝文件的开头移动；向下卷动意味着朝文件尾部移动。

很容易在应用程序中包含水平或者垂直的滚动条，程序写作者只需要在CreateWindow的第三个参数中包括窗口样式（WS）标识符WS_VSCROLL（垂直卷动）和/或WS_HSCROLL（水平卷动）即可。这些卷动列通常放在窗口的右部和底部，伸展为显示区域的整个长度或宽度。显示区域不包含卷动列所占据的空间。对于特定的显示驱动程序和显示分辨率，垂直卷动列的宽度和水平卷动列的高度是恒定的。如果需要这些值，可以使用GetSystemMetrics呼叫来取得（如前面的程序那样）。

Windows负责处理对滚动条的所有鼠标操作，但是，窗口滚动条没有自动的键盘接口。如果想用光标键来完成卷动功能，则必须提供这方面的程序代码（我们将在下一章另一个版本的SYSMETS程序中做到这一点）。

滚动条的范围和位置

每个滚动条均有一个相关的「范围」(这是一对整数，分别代表最小值和最大值）和「位置」(它是卷动方块在此范围内的位置)。当卷动方块在卷动列的顶部（或左部）时，卷动方块的位置是范围的最小值；在卷动列的底部（或右部）时，卷动方块的位置是范围的最大值。

在内定情况下，滚动条的范围是从0（顶部或左部）至100（底部或右部），但将范围改变为更方便于程序的数值也是很容易的：

SetScrollRange (hwnd, iBar, iMin, iMax, bRedraw) ;

参数iBar为SB_VERT或者SB_HORZ，iMin和iMax分别是范围的最小值和最大值。如果想要

Windows根据新范围重画滚动条，则设置bRedraw为TRUE（如果在呼叫SetScrollRange后，呼叫了影响滚动条位置的其它函数，则应该将bRedraw设定为FALSE以避免过多的重画）。

卷动方块的位置总是离散的整数值。例如，范围为0至4的滚动条具有5个卷动方块位置，如图4-6所示。

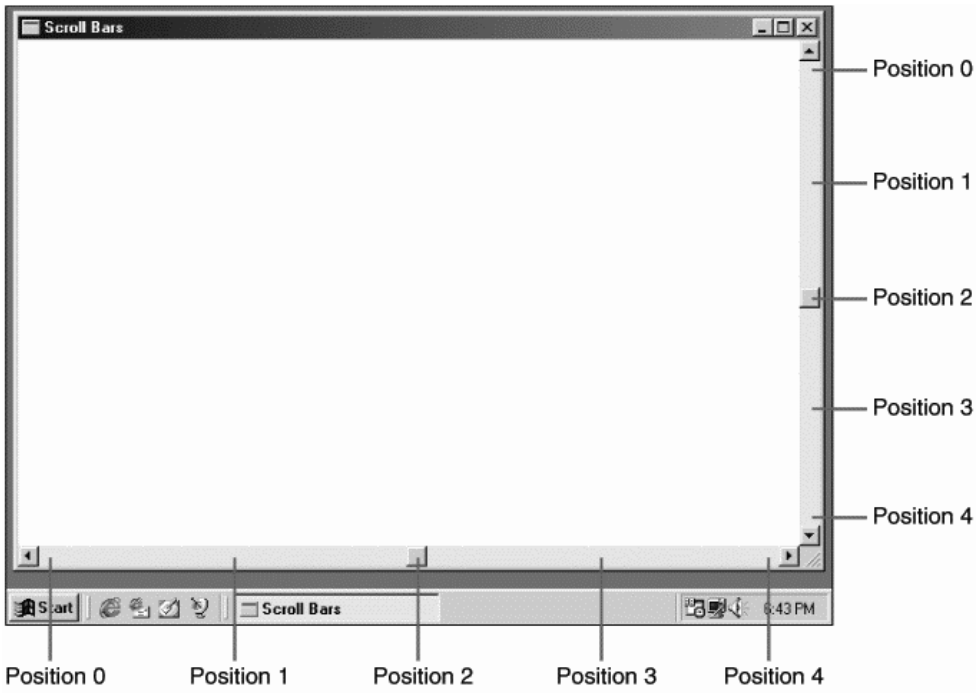


图4-6 具有5个卷动方块位置的卷动列

您可以使用SetScrollPos在滚动条范围内设置新的卷动方块位置：

SetScrollPos (hwnd, iBar, iPos, bRedraw) ;

参数iPos是新位置，它必须在iMin至iMax的范围内。Windows提供了类似的函数（GetScrollRange和GetScrollPos）来取得滚动条的目前范围和位置。

在程序内使用滚动条时，程序写作者与Windows共同负责维护滚动条以及更新卷动方块的位置。下面是Windows对滚动条的处理：

处理所有滚动条鼠标事件

当使用者在滚动条内单击鼠标时，提供一种「反相显示」的闪烁
当使用者在滚动条内拖动卷动方块时，移动卷动方块
为包含滚动条窗口的窗口消息处理程序发送滚动条消息

以下是程序写作者应该完成的工作：

初始化滚动条的范围和位置

处理窗口消息处理程序的滚动条消息
更新滚动条内卷动方块的位置
更改显示区域的内容以响应对滚动条的更改

像生活中的大多数事情一样，在我们看一些程序代码时这些会显得更加有意义。

滚动条消息

在用鼠标单击滚动条或者拖动卷动方块时，Windows给窗口消息处理程序发送WM_VSCROLL（供上下移动）和WM_HSCROLL（供左右移动）消息。在滚动条上的每个鼠标动作都至少产生两个消息，一条在按下鼠标按钮时产生，一条在释放按钮时产生。

和所有的消息一样，WM_VSCROLL和WM_HSCROLL也带有wParam和lParam消息参数。对于来自作为窗口的一部分而建立的滚动条消息，您可以忽略lParam；它只用于作为子窗口而建立的滚动条（通常在对话框内）。

wParam消息参数被分为一个低字组和一个高字组。wParam的低字组是一个数值，它指出了鼠标对滚动条进行的操作。这个数值被看作一个「通知码」。通知码的值由以SB（代表「scroll bar（滚动条）」）开头的标识符定义。以下是在WINUSER.H中定义的通知码：

```
#define SB_LINEUP 0
#define SB_LINELEFT 0
#define SB_LINEDOWN 1
#define SB_LINERIGHT 1
#define SB_PAGEUP 2
#define SB_PAGELEFT 2
#define SB_PAGEDOWN 3
#define SB_PAGERIGHT 3
#define SB_THUMBPOSITION 4
#define SB_THUMBTRACK 5
#define SB_TOP 6
#define SB_LEFT 6
#define SB_BOTTOM 7
#define SB_RIGHT 7
#define SB_ENDSCROLL 8
```

包含LEFT和RIGHT的标识符用于水平滚动条，包含UP、DOWN、TOP和BOTTOM的标识符用于垂直滚动条。鼠标在滚动条的不同区域单击所产生的通知码如图4-7所示。

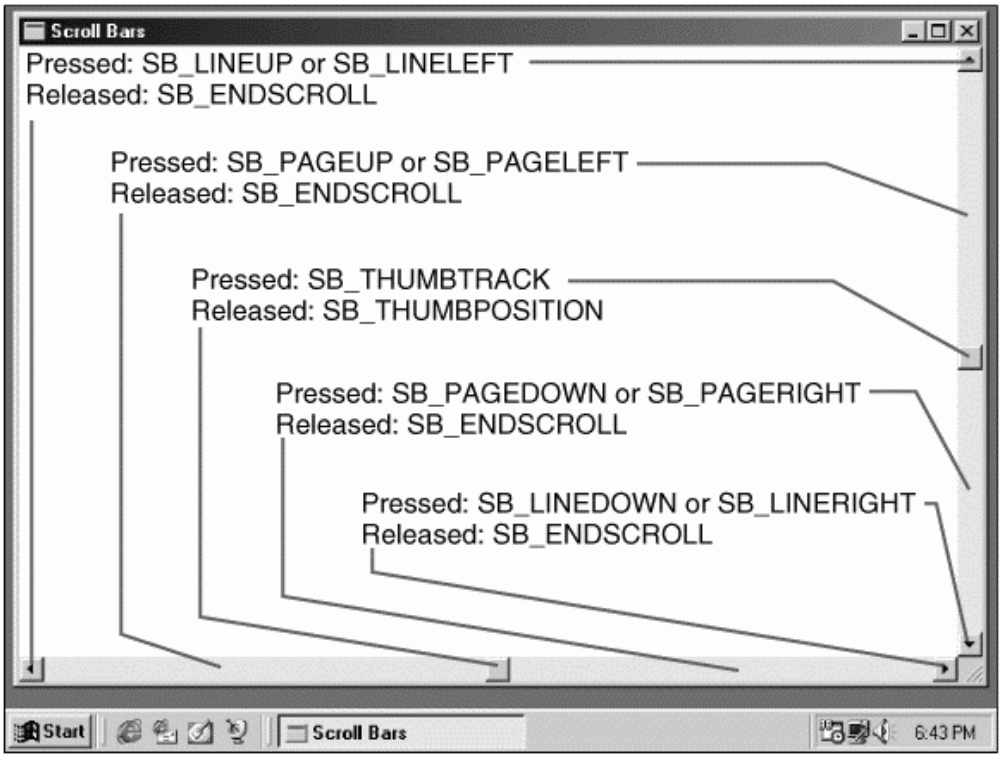


图4-7 用于滚动条消息的wParam值的标识符

如果在滚动条的各个部位按住鼠标键，程序就能收到多个滚动条消息。当释放鼠标键后，程序会收到一个带有SB_ENDSCROLL通知码的消息。一般可以忽略这个消息，Windows不会去改变卷动方块的位置，而您可以在程序中呼叫SetScrollPos来改变卷动方块的位置。

当把鼠标的光标放在卷动方块上并按住鼠标键时，您就可以移动卷动方块。这样就产生了带有SB_THUMBTRACK 和 SB_THUMBPOSITION 通知码的滚动条消息。在 wParam 的低字组是SB_THUMBTRACK时，wParam的高字组是使用者在拖动卷动方块时的目前位置。该位置位于卷动列范围的最小值和最大值之间。在wParam的低字组是SB_THUMBPOSITION时，wParam的高字组是使用者释放鼠标键后卷动方块的最终位置。对于其它的卷动列操作，wParam的高字组应该被忽略。

为了给使用者提供回馈，Windows在您用鼠标拖动卷动方块时移动它，同时您的程序会收到SB_THUMBTRACK 消息。然而，如果不通过呼叫 SetScrollPos 来处理 SB_THUMBTRACK 或SB_THUMBPOSITION消息，在使用者释放鼠标键后，卷动方块会迅速跳回原来的位置。

程序能够处理SB_THUMBTRACK或SB_THUMBPOSITION消息，但一般不同时处理两者。如果处理SB_THUMBTRACK消息，在使用者拖动卷动方块时您需要移动显示区域的内容。而如果处理SB_THUMBPOSITION消息，则只需在使用者停止拖动卷动方块时移动显示区域的内容。处理SB_THUMBTRACK消息更好一些（但更困难），对于某些型态的数据，您的程序可能很难跟上产生的消息。

WINUSER.H表头文件还包括SB_TOP、SB_BOTTOM、SB_LEFT和SB_RIGHT通知码，指出滚动条已经被移到了它的最小或最大位置。然而，对于作为应用程序窗口一部分而建立的滚动条来说，永远不会接收到这些通知码。

在滚动条范围使用32位的值也是有效的，尽管这不常见。然而，wParam的高字组只有16位的大小，它不能适当地指出SB_THUMBTRACK和SB_THUMBPOSITION操作的位置。在这种情况下，需要使用GetScrollInfo函数（在下面描述）来得到信息。

在SYSMETS中加入卷动功能

前面的说明已经很详尽了，现在，要将那些东西动手做做看了。让我们开始时简单些，从垂直卷动着手，因为我们实在太需要垂直卷动了，而暂时还可以不用水平卷动。SYSMET2如程序4-3所示。这个程序可能是滚动条的最简单的应用。

程序4-3 SYSMETS2.C

```
/*------------------------------------------------------------------
SYSMETS2.C -- System Metrics Display Program No. 2
(c) Charles Petzold, 1998
------------------------------------------------------------------*/
#include <windows.h>
#include "sysmets.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("SysMets2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 2"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxChar, cxCaps, cyChar, cyClient, iVscrollPos ;
  HDC hdc ;
  int i, y ;
  PAINTSTRUCT ps ;
  TCHAR szBuffer[10] ;
  TEXTMETRIC tm ;
  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar = tm.tmHeight + tm.tmExternalLeading ;

    ReleaseDC (hwnd, hdc) ;
    SetScrollRange (hwnd, SB_VERT, 0, NUMLINES - 1, FALSE) ;
    SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;
    return 0 ;
  case WM_SIZE:
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_VSCROLL:
    switch (LOWORD (wParam))
    {
    case SB_LINEUP:
      iVscrollPos -= 1 ;
      break ;
    case SB_LINEDOWN:
      iVscrollPos += 1 ;
      break ;
    case SB_PAGEUP:
      iVscrollPos -= cyClient / cyChar ;
      break ;
    case SB_PAGEDOWN:
      iVscrollPos += cyClient / cyChar ;
      break ;
    case SB_THUMBPOSITION:
      iVscrollPos = HIWORD (wParam) ;
      break ;
    default :
```

```
      break ;
    }

    iVscrollPos = max (0, min (iVscrollPos, NUMLINES - 1)) ;
    if (iVscrollPos != GetScrollPos (hwnd, SB_VERT))
    {
      SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;
    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;
      for (i = 0 ; i < NUMLINES ; i++)
      {
        y = cyChar * (i - iVscrollPos) ;
        TextOut (hdc, 0, y,
          sysmetrics[i].szLabel,
          lstrlen (sysmetrics[i].szLabel)) ;

        TextOut (hdc, 22 * cxCaps, y,
          sysmetrics[i].szDesc,
          lstrlen (sysmetrics[i].szDesc)) ;

        SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
        TextOut (hdc, 22 * cxCaps + 40 * cxChar, y, szBuffer,
          wsprintf (szBuffer, TEXT ("%5d"),
          GetSystemMetrics (sysmetrics[i].iIndex))) ;
        SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
      }
      EndPaint (hwnd, &ps) ;
      return 0 ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

新的CreateWindow呼叫在第三个参数中包含了WS_VSCROLL窗口样式，从而在窗口中加入了垂直滚动条，其窗口样式为：

WS_OVERLAPPEDWINDOW | WS_VSCROLL

WndProc窗口消息处理程序在处理WM_CREATE消息时增加了两条叙述，以设置垂直滚动条的范围和初始位置：

```
SetScrollRange (hwnd, SB_VERT, 0, NUMLINES - 1, FALSE) ;
SetScrollPos (hwnd, SB_VERT, iVscrollPos, TRUE) ;
```

sysmetrics结构具有NUMLINES行文字，所以滚动条范围被设定为0至NUMLINES-1。滚动条的每个位置对应于在显示区域顶部显示的一个文字行。如果卷动方块的位置为0，则第一行会被放置在显示区域的顶部。如果位置大于0，其它行就会出现在显示区域的顶部。当位置为NUMLINES-1时，则最后一行文字出现在显示区域的顶部。

为了有助于处理WM_VSCROLL消息，在窗口消息处理程序中定义了一个静态变量iVscrollPos，这一变量是滚动条内卷动方块的目前位置。对于SB_LINEUP和SB_LINEDOWN，只需要将卷动方块调整一个单位的位置。对于SB_PAGEUP和SB_PAGEDOWN，我们想移动一整面的内容，或者移动cyClient /cyChar个单位的位置。对于SB_THUMBPOSITION，新的卷动方块位置是wParam的高字组。SB_ENDSCROLL和SB_THUMBTRACK消息被忽略。

在程序依据收到的WM_VSCROLL消息计算出新的iVscrollPos值后，用min和max宏来调整iVscrollPos，以确保它在最大值与最小值之间。程序然后将iVscrollPos与呼叫GetScrollPos取得的先前位置相比较，如果卷动位置发生了变化，则使用SetScrollPos来进行更新，并且呼叫

InvalidateRect使整个窗口无效。

InvalidateRect呼叫产生一个WM_PAINT消息。SYSMETS1在处理WM_PAINT消息时，每一行的y坐标计算公式为：

cyChar * i

在SYSMETS2中，计算公式为：

cyChar * (i - iVscrollPos)

循环仍然显示NUMLINES行文字，但是对于非零值的iVscrollPos是负数。程序实际上在显示区域以外显示这些文字行。当然，Windows不会显示这些行，因此屏幕显得干净和漂亮。

前面说过，我们一开始不想弄得太复杂，这样的程序代码很浪费，效率很低。下面我们对此加以修改，但是先要考虑在WM_VSCROLL消息之后更新显示区域的方法。

绘图程序的组织

在处理完滚动条消息后，SYSMETS2不更新显示区域，相反，它呼叫InvalidateRect使显示区域失效。这导致Windows将一个WM_PAINT消息放入消息队列中。

最好能使Windows程序在响应WM_PAINT消息时完成所有的显示区域绘制功能。因为程序必须在一接收到WM_PAINT消息时就更新整个显示区域，如果在程序的其它部分也绘制的话，将很可能使程序代码重复。

首先，您可能对这种拐弯抹角的方式感到厌烦。在Windows的早期，因为这种方式与文字模式的程序设计差别太大，程序写作者感到这种概念很难理解。并且，程序要不断地通过马上绘制画面来响应键盘和鼠标。这样做既方便又有效，但是在很多情况下，这完全不必要。当您掌握了在响应WM_PAINT消息时积累绘制显示区域所需要的全部信息的原则之后，会对这种结果感到满意的。

如同SYSMETS2示范的，程序仍然需要在处理非WM_PAINT消息时更新特定的显示区域，使用InvalidateRect就很方便，您可以用它使显示区域内的特定矩形或者整个显示区域失效。

只将窗口显示区域标记为无效以产生WM_PAINT消息，对于某些应用程序来说也许不是完全令人满意的选择。在呼叫InvalidateRect之后，Windows将WM_PAINT消息放入消息队列中，最后由窗口消息处理程序处理它。然而，Windows将WM_PAINT消息当成低优先级消息，如果系统有许多其它的动作正在发生，那么也许会让您等待一会儿工夫。这时，当对话框消失时，将会出现一些空白的「洞」，程序仍然等待更新它的窗口。

如果您希望立即更新无效区域，可以在呼叫InvalidateRect之后呼叫UpdateWindow：

UpdateWindow (hwnd) ;

如果显示区域的任一部分无效，则UpdateWindow将导致Windows用WM_PAINT消息呼叫窗口消息处理程序（如果整个显示区域有效，则不呼叫窗口消息处理程序）。这一WM_PAINT消息不进入消息队列，直接由Windows呼叫窗口消息处理程序。窗口消息处理程序完成更新后立即退出，Windows将控制传回给程序中UpdateWindow呼叫之后的叙述。

您可能注意到，UpdateWindow与WinMain中用来产生第一个WM_PAINT消息的函数相同。最初建立窗口时，整个显示区域内容变为无效，UpdateWindow指示窗口消息处理程序绘制显示区域。

建立更好的滚动

SYSMETS2动作良好，但它只是模仿其它程序中的滚动条，并且效率很低。很快我将示范一个新的版本，改进它的不足。也许最有趣的是这个新版本不使用目前所讨论的四个滚动条函数。相反，它将使用Win32 API中才有的新函数。

滚动条信息函数

滚动条文件（在/Platform SDK/User Interface Services/Controls/Scroll Bars中）指出SetScrollRange、SetScrollPos、GetScrollRange和GetScrollPos函数是「过时的」，但这并不完全正确。这些函数在Windows 1.0中就出现了，在Win32 API中升级以处理32位参数。它们仍然具有良好的功能。而且，它们不与Windows程序设计中新函数相冲突，这就是我在此书中仍使用它们的原因。

Win32 API介绍的两个滚动条函数称作SetScrollInfo和GetScrollInfo。这些函数可以完成以前函数的全部功能，并增加了两个新特性。

第一个功能涉及卷动方块的大小。您可能注意到，卷动方块大小在SYSMETS2程序中是固定的。然而，在您可能使用到的一些Windows应用程序中，卷动方块大小与在窗口中显示的文件大小成比例。显示的大小称作「页面大小」。算法为：

$$\frac{\text{捲動方塊大小}}{\text{滾動長度}} \approx \frac{\text{頁面大小}}{\text{範圍}} \approx \frac{\text{顯示的文件數量}}{\text{文件的總大小}}$$

可以使用SetScrollInfo来设置页面大小(从而设置了卷动方块的大小)，如将要看到的SYSMETS3程序所示。

GetScrollInfo函数增加了第二个重要的功能，或者说它改进了目前API的不足。假设您要使用65,536或更大单位的范围，这在16位Windows中是不可能的。当然在Win32中，函数被定义为可接受32位参数，因此是没有问题的。(记住如果使用这样大的范围，卷动方块的实际物理位置数仍然由卷动列的像素大小限制)。然而，当使用SB_THUMBTRACK或SB_THUMBPOSITION通知码得到WM_VSCROLL或WM_HSCROLL消息时，只提供了16位数据来指出卷动方块的目前位置。通过GetScrollInfo函数可以取得真实的32位值。

SetScrollInfo和GetScrollInfo函数的语法是

```
SetScrollInfo (hwnd, iBar, &si, bRedraw) ;
GetScrollInfo (hwnd, iBar, &si) ;
```

像在其它滚动条函数中那样，iBar参数是SB_VERT或SB_HORZ，它还可以是用于滚动条控制的SB_CTL。SetScrollInfo的最后一个参数可以是TRUE或FALSE，指出了是否要Windows重新绘制计算了新信息后的滚动条。

两个函数的第三个参数是SCROLLINFO结构，定义为：

```
typedef struct tagSCROLLINFO
{
  UINT cbSize ;// set to sizeof (SCROLLINFO)
  UINT fMask ; // values to set or get
  int nMin ; // minimum range value
  int nMax ; // maximum range value
  UINT nPage ; // page size
  int nPos ; // current position
  int nTrackPos ;// current tracking position
}
```

```
SCROLLINFO, * PSCROLLINFO ;
```

在程序中，可以定义如下的SCROLLINFO结构型态：

SCROLLINFO si ;

在呼叫SetScrollInfo或GetScrollInfo之前，必须将cbSize字段设定为结构的大小：

si.cbSize = sizeof (si) ;

或

si.cbSize = sizeof (SCROLLINFO) ;

逐渐熟悉Windows后，您就会发现另外几个结构像这个结构一样，第一个字段指出了结构大小。这个字段使将来的Windows版本可以扩充结构并添加新的功能，并且仍然与以前编译的版本兼容。

把fMask字段设定为一个以上以SIF前缀开头的旗标，并且可以使用C的位操作OR运算子(|)组合这些旗标。

SetScrollInfo函数使用SIF_RANGE旗标时，必须把nMin和nMax字段设定为所需的滚动条范围。GetScrollInfo函数使用SIF_RANGE旗标时，应把nMin和nMax字段设定为从函数传回的目前范围。

SIF_POS旗标也一样。当通过SetScrollInfo使用它时，必须把结构的nPos字段设定为所需的位置。可以通过GetScrollInfo使用SIF_POS旗标来取得目前位置。

使用SIF_PAGE旗标能够取得页面大小。用SetScrollInfo函数把nPage设定为所需的页面大小。GetScrollInfo使用SIF_PAGE旗标可以取得目前页面的大小。如果不想得到比例化的滚动条，就不要使用该旗标。

当处理带有SB_THUMBTRACK或SB_THUMBPOSITION通知码的WM_VSCROLL或WM_HSCROLL消息时，通过GetScrollInfo只使用SIF_TRACKPOS旗标。从函数的传回中，SCROLLINFO结构的nTrackPos字段将指出目前的32位的卷动方块位置。

在SetScrollInfo函数中仅使用SIF_DISABLENOSCROLL旗标。如果指定了此旗标，而且新的滚动条参数使滚动条消失，则该滚动条就不能使用了（下面会有更多的解释）。

SIF_ALL旗标是SIF_RANGE、SIF_POS、SIF_PAGE和SIF_TRACKPOS的组合。在WM_SIZE消息处理期间设置滚动条参数时，这是很方便的（在SetScrollInfo函数中指定SIF_TRACKPOS后，它会被忽略）。这在处理滚动条消息时也是很方便的。

卷动范围

在SYSMETS2中，卷动范围设置最小为0，最大为NUMLINES-1。当滚动条位置是0时，第一行信息显示在显示区域的顶部；当滚动条的位置是NUMLINES-1时，最后一行显示在显示区域的顶部，并且看不见其它行。

可以说SYSMETS2卷动范围太大。事实上只需把信息最后一行显示在显示区域的底部而不是顶部即可。我们可以对SYSMETS2作出一些修改以达到此点。当处理WM_CREATE消息时不设置滚动条范围，而是等到接收到WM_SIZE消息后再做此工作：

```
iVscrollMax = max (0, NUMLINES - cyClient / cyChar) ;
SetScrollRange (hwnd, SB_VERT, 0, iVscrollMax, TRUE) ;
```

假定NUMLINES等于75，并假定特定窗口大小是：50（cyChar除以cyClient）。换句话说，我们有75行信息但只有50行可以显示在显示区域中。使用上面的两行程序代码，把范围设置最小为0，最大为25。当滚动条位置等于0时，程序显示0到49行。当滚动条位置等于1时，程序显示1到50行；

并且当滚动条位置等于25（最大值）时，程序显示25到74行。很明显需要对程序的其它部分做出修改，但这是可行的。

新滚动条函数的一个好的功能是当使用与滚动条范围一样大的页面时，它已经为您做掉了一大堆杂事。可以像下面的程序代码一样使用SCROLLINFO结构和SetScrollInfo：

```
si.cbSize = sizeof (SCROLLINFO) ;
si.cbMask = SIF_RANGE | SIF_PAGE ;
si.nMin = 0 ;
si.nMax = NUMLINES - 1 ;
si.nPage = cyClient / cyChar ;
SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
```

这样做之后，Windows会把最大的滚动条位置限制为si.nMax - si.nPage +1而不是si.nMax。像前面那样做出假设：NUMLINES等于75 （所以si.nMax等于74），si.nPage等于50。这意味着最大的滚动条位置限制为74 - 50 + 1，即25。这正是我们想要的。

当页面大小与滚动条范围一样大时，会发生什么情况呢？在这个例子中，就是nPage等于75或更大的情况。Windows通常隐藏滚动条，因为它并不需要。如果不想隐藏滚动条，可在呼叫SetScrollInfo时使用SIF_DISABLENOSCROLL，Windows只是让那个滚动条不能被使用，而不隐藏它。

新SYSMETS

SYSMETS3 – 此章中最后的SYSMETS程序版本 – 显示在程序4-4中。此版本使用SetScrollInfo和GetScrollInfo函数，添加左右卷动的水平滚动条，并能更有效地重画显示区域。

程序4-4 SYSMETS3

SYSMETS3.C

```
/*---------------------------------------------------------------------
SYSMETS3.C -- System Metrics Display Program No. 3
(c) Charles Petzold, 1998
---------------------------------------------------------------------*/
#include <windows.h>
#include "sysmets.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("SysMets3") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 3"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
```

```
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
  HDC hdc ;
  int i, x, y, iVertPos, iHorzPos, iPaintBeg, iPaintEnd ;
  PAINTSTRUCT ps ;
  SCROLLINFO si ;
  TCHAR szBuffer[10] ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar = tm.tmHeight + tm.tmExternalLeading ;
    ReleaseDC (hwnd, hdc) ;
    // Save the width of the three columns
    iMaxWidth = 40 * cxChar + 22 * cxCaps ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    // Set vertical scroll bar range and page size
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_RANGE | SIF_PAGE ;
    si.nMin = 0 ;
    si.nMax = NUMLINES - 1 ;
    si.nPage = cyClient / cyChar ;
    SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
    // Set horizontal scroll bar range and page size
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_RANGE | SIF_PAGE ;
    si.nMin = 0 ;
    si.nMax = 2 + iMaxWidth / cxChar ;
    si.nPage = cxClient / cxChar ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    return 0 ;
  case WM_VSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_ALL ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    // Save the position for comparison later on
    iVertPos = si.nPos ;
    switch (LOWORD (wParam))
    {
    case SB_TOP:
      si.nPos = si.nMin ;
      break ;
    case SB_BOTTOM:
      si.nPos = si.nMax ;
      break ;
```

```
    case SB_LINEUP:
      si.nPos - = 1 ;
      break ;
    case SB_LINEDOWN:
      si.nPos += 1 ;
      break ;
    case SB_PAGEUP:
      si.nPos -= si.nPage ;
      break ;
    case SB_PAGEDOWN:
      si.nPos += si.nPage ;
      break ;
    case SB_THUMBTRACK:
      si.nPos = si.nTrackPos ;
      break ;
    default:
      break ;
    }
    // Set the position and then retrieve it. Due to adjustments
    // by Windows it may not be the same as the value set.

    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;

    // If the position has changed, scroll the window and update it
    if (si.nPos != iVertPos)
    {
      ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),
        NULL, NULL) ;
      UpdateWindow (hwnd) ;
    }
    return 0 ;
    case WM_HSCROLL:
      // Get all the vertical scroll bar information
      si.cbSize = sizeof (si) ;
      si.fMask = SIF_ALL ;
      // Save the position for comparison later on
      GetScrollInfo (hwnd, SB_HORZ, &si) ;
      iHorzPos = si.nPos ;
      switch (LOWORD (wParam))
      {
      case SB_LINELEFT:
        si.nPos -= 1 ;
        break ;
      case SB_LINERIGHT:
        si.nPos += 1 ;
        break ;
      case SB_PAGELEFT:
        si.nPos -= si.nPage ;
        break ;
      case SB_PAGERIGHT:
        si.nPos += si.nPage ;
        break ;
      case SB_THUMBPOSITION:
        si.nPos = si.nTrackPos ;
        break ;
      default :
        break ;
      }
      // Set the position and then retrieve it. Due to adjustments
      // by Windows it may not be the same as the value set.
      si.fMask = SIF_POS ;
      SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
      GetScrollInfo (hwnd, SB_HORZ, &si) ;
      // If the position has changed, scroll the window
      if (si.nPos != iHorzPos)
      {
        ScrollWindow (hwnd, cxChar * (iHorzPos - si.nPos), 0,
```

```
        NULL, NULL) ;
    }
    return 0 ;
    case WM_PAINT :
      hdc = BeginPaint (hwnd, &ps) ;
      // Get vertical scroll bar position
      si.cbSize = sizeof (si) ;
      si.fMask = SIF_POS ;
      GetScrollInfo (hwnd, SB_VERT, &si) ;
      iVertPos = si.nPos ;
      // Get horizontal scroll bar position
      GetScrollInfo (hwnd, SB_HORZ, &si) ;
      iHorzPos = si.nPos ;
      // Find painting limits
      iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
      iPaintEnd = min ( NUMLINES - 1,
        iVertPos + ps.rcPaint.bottom / cyChar) ;
      for (i = iPaintBeg ; i <= iPaintEnd ; i++)
      {
        x = cxChar * (1 - iHorzPos) ;
        y = cyChar * (i - iVertPos) ;

        TextOut (hdc, x, y,
          sysmetrics[i].szLabel,
          lstrlen (sysmetrics[i].szLabel)) ;

        TextOut (hdc, x + 22 * cxCaps, y,
          sysmetrics[i].szDesc,
          lstrlen (sysmetrics[i].szDesc)) ;

        SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
        TextOut (hdc, x + 22 * cxCaps + 40 * cxChar, y, szBuffer,
          wsprintf (szBuffer, TEXT ("%5d"),
          GetSystemMetrics (sysmetrics[i].iIndex))) ;

        SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
      }
      EndPaint (hwnd, &ps) ;
      return 0 ;
    case WM_DESTROY :
      PostQuitMessage (0) ;
      return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这个版本的程序仰赖Windows保存滚动条信息并做边界检查。在WM_VSCROLL和WM_HSCROLL处理的开始，它取得所有的滚动条信息，根据通知码调整位置，然后呼叫SetScrollInfo设置其位置。程序然后呼叫GetScrollInfo。如果该位置超出了SetScrollInfo呼叫的范围，则由Windows来纠正该位置并且在GetScrollInfo呼叫中传回正确的值。

SYSMETS3使用ScrollWindow函数在窗口的显示区域中卷动信息而不是重画它。虽然该函数很复杂（在新版本的Windows中已被更复杂的ScrollWindowEx所替代），SYSMETS3仍以相当简单的方式使用它。函数的第二个参数给出了水平卷动显示区域的数值，第三个参数是垂直卷动显示区域的数值，单位都是像素。

ScrollWindow的最后两个参数设定为NULL，这指出了要卷动整个显示区域。Windows自动把显示区域中未被卷动操作覆盖的矩形设为无效。这会产生WM_PAINT消息。再也不需要InvalidateRect了。注意ScrollWindow不是GDI函数，因为它不需设备内容句柄。它是少数几个非GDI的Windows函数之一，它可以改变窗口的显示区域外观。很特殊但不方便，它是随滚动条函数一起记载在文件中。

WM_HSCROLL处理拦截SB_THUMBPOSITION通知码并忽略SB_THUMBTRACK。因而，如果使

用者在水平滚动条上拖动卷动方块，在使用者释放鼠标按钮之前，程序不会水平卷动窗口的内容。

WM_VSCROLL 的方法与之不同：程序拦截 SB_THUMBTRACK 消息并忽略 SB_THUMBPOSITION。因而，程序随使用者在垂直滚动条上拖动卷动方块而垂直地滚动内容。这种想法很好，但应注意：一旦使用者发现程序会立即响应拖动的卷动方块，他们就会不断地来回拖动卷动方块。幸运的是现在的PC快得可以胜任这种严酷的测试。但是在较慢的机器上，可以考虑为GetSystemMetrics使用SB_SLOWMACHINE参数来替代这种处理。

加快WM_PAINT处理的一个方法由SYSMETS3展示：WM_PAINT处理程序确定无效区域中的文字行并仅仅重画这些行。当然，程序代码复杂一些，但速度很快。

不用鼠标怎么办

在Windows的早期，有大量的使用者不喜欢使用鼠标，而且，Windows自身也不要求必须有鼠标。虽然，没有鼠标的PC现在走上了单色显示器和点阵打印机的没落之路，但我仍然建议您编写可以使用键盘来产生与鼠标操作相同效果的程序，尤其对于像滚动条这样的基本操作对象更是如此。因为我们的键盘有一组光标移动键，所以应该实作同样的操作。

在下一章，您将学习使用键盘和在SYSMETS3中增加键盘接口的方法。您可能会注意到，SYSMETS3似乎在通知码等于SB_TOP和SB_BOTTOM时处理了WM_VSCROLL消息。前面已经提到过，窗口消息处理程序不从滚动条接收这些消息，所以，目前这是多余的程序代码。当我们在下一章再次回到这个程序时，您将会明白这样做的原因。

第五章 图形基础

图形设备接口（GDI：Graphics Device Interface）是Windows的子系统，它负责在视讯显示器和打印机上显示图形。正如您所认为的那样，GDI是Windows非常重要的部分。不只您为Windows编写的应用系统在显示视觉信息时使用GDI，就连Windows本身也使用GDI来显示使用者接口对象，诸如菜单、滚动条、图标和鼠标光标。

不幸的是，如果要对GDI进行全面的讲述，将需要一整本书－当然不是这本书。在本章中，我只是想向您提供画线和填入区域的基本知识，这对于理解下面几章的GDI已经足够了。在后面几章中会讲述GDI支持的位图、metafile以及格式化文字。

GDI 的结构

从程序写作者的观点来看，GDI由几百个函数呼叫和一些相关的数据型态、宏和结构组成。但是在开始讲述这些函数的细节之前，让我们先从巨观上了解一下GDI的整体结构。

GDI原理

Windows 98和Microsoft Windows NT中的图形主要由GDI32.DLL动态链接库输出的函数来处理。在Windows 98中，这个GDI32.DLL实际是利用16位GDI.EXE动态链接库来执行许多函数。在Windows NT中，GDI.EXE只用于16位的程序。

这些动态链接库呼叫您安装的视讯显示器和任何打印机呼叫驱动程序中的例程。视讯驱动程序存取视讯显示器的硬件，打印机驱动程序将GDI命令转换为各种打印机能够理解的代码或者命令。显然，不同的视讯显示卡和打印机要求不同的设备驱动程序。

因为PC兼容机种上可以连接许多种不同的视讯设备，所以，GDI的主要目的之一是支持与设备无关的图形。Windows程序应该能够毫无困难地在Windows支持的任意一种图形输出设备上执行，GDI通过将您的程序和不同输出设备的特性隔离开来的方法来达到这一目的。

图形输出设备分为两大类：位映像设备和向量设备。大多数PC的输出设备是位映像设备，这意味着它们以图点构成的数组来表示图像，这类设备包括视讯显示卡、点阵打印机和激光打印机。向量设备使用线来绘制图像，通常局限于绘图机。

许多传统的计算机图形程序设计方式都是完全以向量为主的，这意味着使用向量图形系统的程序与硬件有着一定层次的隔离。输出设备用像素表示图形，但是程序与程序接口之间并不是用像素进行沟通的。您当然可以使用Windows GDI作为一个高阶的向量绘制系统，同时也可以将它用于比较低阶的像素操作。

从这方面来看，Windows GDI和传统的图形接口语言之间的关系，就如同C和其它程序设计语言之间的关系一样。C以它在不同操作系统和环境之间的高度可移植性而闻名，然而C也以允许程序写作者进行低阶系统呼叫而闻名，这些呼叫在其它高级语言中通常是不可能的。正如C有时被认为是一种「高级汇编语言」一样，您可以认为GDI是图形设备硬件之间的一种高阶界面。

您已经看到，Windows内定使用像素坐标系统。大多数传统的图形语言使用「虚拟」坐标系，其水平和垂直轴的范围在0到32,767之间。虽然有些图形语言不让您使用像素坐标，但是Windows

GDI允许您使用两种坐标系统之一（甚至依据实际度量衡的坐标系）。您可以使用虚拟坐标系以便让程序独立于硬件之外，或者也可以使用设备坐标系而完全迎合硬设备提供的环境。

某些程序写作者认为一旦开始使用操作像素的程序设计方式，就放弃了设备无关性。我们在 上一章看到，这不完全是正确的，其中的诀窍是在与设备无关的方式中使用像素。这要求图形接口语言为程序提供一些方法来确定设备的硬件特征，并进行适当的调节。例如，在SYSMETS程序中，我们根据标准系统字体字符的像素大小来确定屏幕上的文字间距，这种方法允许程序针对分辨率、文字大小和方向比例各不相同的显示卡进行相应的调节。您将在本章看到一些用于确定显示尺寸的其它方法。

早期，许多使用者在单色显示器上执行Windows。即使是几年前，笔记本计算机也还只有灰阶显示。为此，GDI的设计保证了您可以在编写一个程序时不必太担心色彩问题 – 也就是说，Windows可以将色彩转换为灰阶显示。甚至在今天，Windows 98使用的视讯显示已经具有了不同的色彩能力（16色、256色、「high-Color」以及「true-color」）。虽然，彩色喷墨打印机的成本已经很低了，但是大多数使用者仍然坚持使用黑白打印机。盲目地使用这些设备是可以的，但是您的程序也应该能决定在某种显示设备上有多少色彩可以使用，从而最佳利用硬件功能。

当然，就如同您编写C程序时，为了使它在其它计算机上执行而遇到一些微妙的移植性问题一样，您也可能不小心让设备依赖性溜进您的Windows程序，这就是不与硬件完全隔离的代价。您还应该知道Windows GDI的局限。虽然可以在显示器上到处移动图形对象，但GDI通常是一个静态的显示系统，只有有限的动画支持。如果需要为游戏编写复杂的动画，就应该研究一下Microsoft DirectX，它提供了您需要的支持。

GDI函数呼叫

组成GDI的几百个函数呼叫可以分为几大类：

取得（或者建立）和释放（或者清除）设备内容的函数 我们在前面的章节中已经看到过，您在绘图时需要设备内容句柄。GetDC和RealseDC函数让您在非WM_PAINT的消息处理期间来做到这一点，而BeginPaint和EndPaint函数（虽然在技术上它们是USER模块而不是GDI模块的一部分）在进行绘图的WM_PAINT消息处理期间使用。我们马上还会介绍有关设备内容的其它一些函数。

取得有关设备内容信息的函数再以第四章中SYSMETS程序为例，我们使用GetTextMetrics函数来取得有关设备内容中目前所选字体的尺寸信息。在本章后面，我们将看到一个取得非常广泛的设备内容信息的DEVCAPS1程序。

绘图函数显然，在所有前提条件都得以满足之后，这些函数是真正重要的部分。在上一章中，我们使用TextOut函数在窗口的显示区域显示一些文字。我们将看到，其它GDI函数还可以让您画线、填入区域。在第十四章和第十五章还会看到如何建立位图图像。

设定和取得设备内容参数的函数设备内容的「属性」决定有关绘图函数如何工作的细节。例如，用SetTextColor来指定TextOut（或者其它文字输出函数）所绘制的文字色彩。在第四章中SYSMETS程序中，我们使用SetTextAlign来告诉GDI：TextOut函数中的字符串的开始位置应该在字符串的右边而不是内定的左边。设备内容的所有属性都有默认值，取得设备内容时这些默认值就设定好了。对于所有的Set函数，都有相应的Get函数，以允许您取得目前设备内容属性。

使用GDI对象的函数GDI在这里变得有点混乱。首先举一个例子：内定时使用GDI绘制的所有直线都是实线并具有一个标准的宽度。您可能希望绘制更细的直线，或者是由一系列的点或短划线组成的直线。这种线的宽度和这种线的画笔样式不是设备内容的属性，而是一个「逻辑画笔」的特征。您可以通过在CreatePen、 CreatePenIndirect或ExtCreatePen函数中指定

这些特征来建立一个逻辑画笔，这些函数传回一个逻辑画笔的句柄（虽然这些函数被认为是GDI的一部分，但是和大多数GDI函数呼叫不一样，它们不要求设备内容的句柄）。要使用这个画笔，就要将画笔句柄选进设备内容。我们认为，设备内容中目前选中的画笔就是设备内容的一个属性。这样，您画任何线都使用这个画笔，然后，您可以取消设备内容中的画笔选择，并清除画笔对象。清除画笔对象是必要的，因为画笔定义占用了分配的内存空间。除了画笔以外，GDI对象还用于建立填入封闭区域的画刷、字体、位图以及GDI的其它一些方面。

GDI基本图形

您在屏幕或打印机上显示的图形型态本身可以被分为几类，通常被称为「基本图形」，它们是：

直线和曲线线条是所有向量图形绘制系统的基础。GDI支持直线、矩形、椭圆（包括椭圆的子集，也就是我们所说的「圆」）、椭圆圆周上的部分曲线即所谓的「弧」以及贝塞尔曲线（Bezier spline），我们将在本章中分别对它们进行介绍。所有更复杂的曲线可由折线（polyline）代替，折线通过一组非常短的直线来定义一条曲线。线条用设备内容中选中的目前画笔绘制。

填入区域当一系列直线或者曲线封闭了一个区域时，该区域可以使用目前GDI画刷对象进行填图。这个画刷可以是实心色彩、图案（可以是一系列的水平、垂直或者对角标记）或者是在区域内垂直或者水平重复的位图图像。

位图位图是位的矩形数组，这些位对应于显示设备上的像素，它们是位映像图形的基础工具。位图通常用于在视讯显示器或者打印机上显示复杂（一般都是真实的）图像。位图还可以用于显示必须快速绘制的小图像（诸如图标、鼠标光标以及在应用工具条中出现的按钮等）。GDI支持两种型态的位图 – 旧式的（虽然还非常有用）「设备相关」位图，是GDI对象；和新的（如Windows 3.0的）「设备无关」位图（或者DIB），可以储存在磁盘文件中。第十四章和第十五章讨论位图。

文字文字的数学味道不像计算机图形的其它方面那样浓。文字和几百年的传统印刷术有关，它被许多印刷工人看作为一门艺术。因此，文字通常不仅是所有的计算机图形系统中最复杂的部分，而且（如果识字还是社会基本要求的话）也是最重要的部分。用于定义GDI字体对象和取得字体信息的数据结构是Windows中最庞大的部分之一。从Windows 3.1开始，GDI开始支持TrueType字体，该字体是在填入轮廓线基础上建立的，这样的填入轮廓线可由其它GDI函数处理。依据兼容性和储存大小的考虑，Windows 98继续支持旧式的点阵字体。我会在第十七章讨论字体。

其它部分

GDI的其它部分无法这么容易地分类，它们是：

映像模式和变换虽然内定以像素为单位进行绘图，但是您并非局限于此。GDI映像模式允许您以英寸（或者甚至以几分之一英寸）、毫米或者任何您想使用的单位来绘图（Windows NT还支持传统的以三乘三矩阵表示的「坐标变换」， 这允许倾斜和旋转图形对象。不幸的是，在Windows 98中不支持坐标变换）。

MetafileMetafile是以二进制形式储存的GDI命令集合。Metafile主要用于通过剪贴板传输向量图形。第十八章会讨论metafile。

绘图区域绘图区域是形状任意的复杂区域，通常定义为较简单的绘图区域组合。在GDI内部，绘图区域除了储存为最初用来定义绘图区域的线条组合以外，还以一系列扫描线的形式储存。您可以将绘图区域用于绘制轮廓、填入图形和剪裁。

路径路径是GDI内部储存的直线和曲线的集合。路径可以用于绘图、填入图形和剪裁，还可以转换为绘图区域。

剪裁绘图可以限制在显示区域的某一部分中，这就是所谓的剪裁。剪裁区域是不是矩形都

可以，剪裁通常是通过区域或者路径来定义的。

调色盘自订调色盘通常限于显示256色的显示器。Windows仅保留这些色彩之中的20种以供系统使用，您可以改变其它236种色彩，以准确显示按位图形式储存的真实图像。第十六章会讨论调色盘。

打印虽然本章限于讨论视讯显示，但是您在本章中所学到的全部知识都适用于打印。第十三章会讨论打印。

设备内容

在开始绘图之前，让我们比第四章更精确地讨论一下设备内容。

当您想在一个图形输出设备（诸如屏幕或者打印机）上绘图时，您首先必须获得一个设备内容（或者DC）的句柄。将句柄传回给程序时，Windows就给了您使用设备的权限。然后您在GDI函数中将这个句柄作为一个参数，向Windows标识您想在其上进行绘图的设备。

设备内容中包含许多确定GDI函数如何在设备上工作的目前「属性」，这些属性允许传递给GDI函数的参数只包含起始坐标或者尺寸信息，而不必包含Windows在设备上显示对象时需要的所有其它信息。例如，呼叫TextOut时，您只需要在函数中给出设备内容句柄、起始坐标、文字和文字的长度。您不必指定字体、文字颜色、文字后面的背景色彩以及字符间距，因为这些属性都是设备内容的一部分。当您想改变这些属性之一时，您呼叫一个可以改变设备内容中属性的函数，以后针对该设备内容的TextOut呼叫来使用改变后的属性。

取得设备内容句柄

Windows提供了几种取得设备内容句柄的方法。如果在处理一个消息时取得了设备内容句柄，应该在退出窗口函数之前释放它（或者删除它）。一旦释放了句柄，它就不再有效了。对于打印机设备内容句柄，规则就没有这么严格。在第十三章会讨论打印。

最常用的取得并释放设备内容句柄的方法是，在处理WM_PAINT消息时，使用BeginPaint和EndPaint呼叫：

```
hdc = BeginPaint (hwnd, &ps) ;
//其它行程序
EndPaint (hwnd, &ps) ;
```

变量ps是型态为PAINTSTRUCT的结构，该结构的hdc字段是BeginPaint传回的设备内容句柄。PAINTSTRUCT结构又包含一个名为rcPaint的RECT（矩形）结构，rcPaint定义一个包围窗口显示区域无效范围的矩形。使用从BeginPaint获得的设备内容句柄，只能在这个区域内绘图。BeginPaint呼叫使该区域有效。

Windows程序还可以在处理非WM_PAINT消息时取得设备内容句柄：

```
hdc = GetDC (hwnd) ;
//其它行程序
ReleaseDC (hwnd, hdc) ;
```

这个设备内容适用于窗口句柄为hwnd的显示区域。这些呼叫与BeginPaint和EndPaint的组合之间的基本区别是，利用从GetDC传回的句柄可以在整个显示区域上绘图。当然， GetDC和ReleaseDC不使显示区域中任何可能的无效区域变成有效。

Windows程序还可以取得适用于整个窗口（而不仅限于窗口的显示区域）的设备内容句柄：

```
hdc = GetWindowDC (hwnd) ;
//其它行程序
ReleaseDC (hwnd, hdc) ;
```

这个设备内容除了显示区域之外，还包括窗口的标题栏、菜单、滚动条和框架（frame）。GetWindowDC函数很少使用，如果想尝试用一用它，则必须拦截处理WM_NCPAINT消息，Windows使用该消息在窗口的非显示区域上绘图。

BeginPaint、GetDC和GetWindowDC获得的设备内容都与视讯显示器上的某个特定窗口相关。取得设备内容句柄的另一个更通用的函数是CreateDC：

```
hdc = CreateDC (pszDriver, pszDevice, pszOutput, pData) ;
//其它行程序
DeleteDC (hdc) ;
```

例如，您可以通过下面的呼叫来取得整个屏幕的设备内容句柄：

```
   hdc = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
```

在窗口之外写入画面一般是不恰当的，但对于一些不同寻常的应用程序来说，这样做很方便（您还可通过在呼叫GetDC时使用一个NULL参数，从而取得整个屏幕的设备内容句柄，不过这在文件中已经提到了）。在第十三章中，我们将使用CreateDC函数来取得一个打印机设备内容句柄。

有时您只是需要取得关于某设备内容的一些信息而并不进行任何绘画，在这种情况下，您可以使用CreateIC来取得一个「信息内容」的句柄，其参数与CreateDC函数相同，例如：

```
   hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
```

您不能用这个信息内容句柄往设备上写东西。

使用位图时，取得一个「内存设备内容」有时是有用的：

```
hdcMem = CreateCompatibleDC (hdc) ;
//其它行程序
DeleteDC (hdcMem) ;
```

您可以将位图选进内存设备内容，然后使用GDI函数在位图上绘画。我将在第十四章讨论这些技术。

前面已经提到过，metafile是一些GDI呼叫的集合，以二进制形式编码。您可以通过取得metafile设备内容来建立metafile：

```
hdcMeta = CreateMetaFile (pszFilename) ;
//其它行程序
hmf = CloseMetaFile (hdcMeta) ;
```

在metafile设备内容有效期间，任何用hdcMeta所做的GDI呼叫都变成metafile的一部分而不会显示。在呼叫CloseMetaFile之后，设备内容句柄变为无效，函数传回一个指向metafile（hmf）的句柄。我会在第十八章讨论metafile。

取得设备内容信息

一个设备内容通常是指一个实际显示设备，如视讯显示器和打印机。通常，您需要取得有关该设备的信息，包括显示器的大小（单位为像素或者实际长度单位）和色彩显示能力。您可以通过呼叫GetDeviceCaps（「取得设备功能」）函数来取得这些信息：

iValue = GetDeviceCaps (hdc, iIndex) ;

其中，参数iIndex取值为WINGDI.H表头文件中定义的29个标识符之一。例如，iIndex为HORZRES时将使GetDeviceCaps传回设备的宽度（单位为像素）；iIndex为VERTRES时将让GetDeviceCaps传回设备的高度（单位为像素）。如果hdc是打印机设备内容的句柄，则GetDeviceCaps传回打印机显示区域的高度和宽度，它们也是以像素为单位的。

还可以使用GetDeviceCaps来确定设备处理不同型态图形的能力，这对于视讯显示器并不很重要，但是对于打印设备却是非常重要。例如，大多数绘图机不能画位图图像，GetDeviceCaps

就可以将这一情况告诉您。

DEVCAPS1程序

程序5-1所示的DEVCAPS1程序显示了以一个视讯显示器的设备内容为参数时，可以从GetDeviceCaps函数中获得的部分信息（该程序的另一个扩充版本DEVCAPS2将在第十三章给出，用于取得打印机信息）。

程序5-1 DEVCAPS1

DEVCAPS1.C

```
/*--------------------------------------------------------------------------
DEVCAPS1.C -- Device Capabilities Display Program No. 1
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#define NUMLINES ((int) (sizeof devcaps / sizeof devcaps [0]))
struct
{
  int iIndex ;
  TCHAR *szLabel ;
  TCHAR *szDesc ;
}
devcaps [] =
{
  HORZSIZE, TEXT ("HORZSIZE"),TEXT ("Width in millimeters:"),
    VERTSIZE, TEXT ("VERTSIZE"),TEXT ("Height in millimeters:"),
    HORZRES, TEXT ("HORZRES"), TEXT ("Width in pixels:"),
    VERTRES, TEXT ("VERTRES"), TEXT ("Height in raster lines:"),
    BITSPIXEL, TEXT ("BITSPIXEL"),TEXT ("Color bits per pixel:"),
    PLANES, TEXT ("PLANES"), TEXT ("Number of color planes:"),
    NUMBRUSHES, TEXT ("NUMBRUSHES"), TEXT ("Number of device brushes:"),
    NUMPENS, TEXT ("NUMPENS"), TEXT ("Number of device pens:"),
    NUMMARKERS, TEXT ("NUMMARKERS"), TEXT ("Number of device markers:"),
    NUMFONTS, TEXT ("NUMFONTS"), TEXT ("Number of device fonts:"),
    NUMCOLORS, TEXT ("NUMCOLORS"), TEXT ("Number of device colors:"),
    PDEVICESIZE, TEXT ("PDEVICESIZE"),TEXT ("Size of device structure:"),
    ASPECTX, TEXT ("ASPECTX"), TEXT ("Relative width of pixel:"),
    ASPECTY, TEXT ("ASPECTY"), TEXT ("Relative height of pixel:"),
    ASPECTXY, TEXT ("ASPECTXY"), TEXT ("Relative diagonal of pixel:"),
    LOGPIXELSX, TEXT ("LOGPIXELSX"), TEXT ("Horizontal dots per inch:"),
    LOGPIXELSY, TEXT ("LOGPIXELSY"), TEXT ("Vertical dots per inch:"),
    SIZEPALETTE, TEXT ("SIZEPALETTE"),TEXT ("Number of palette entries:"),
    NUMRESERVED, TEXT ("NUMRESERVED"),TEXT ("Reserved palette entries:"),
    COLORRES, TEXT ("COLORRES"), TEXT ("Actual color resolution:")
} ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("DevCaps1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
```

```
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Device Capabilities"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxChar, cxCaps, cyChar ;
  TCHAR szBuffer[10] ;
  HDC hdc ;
  int i ;
  PAINTSTRUCT ps ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar= tm.tmAveCharWidth ;
    cxCaps= (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar= tm.tmHeight + tm.tmExternalLeading ;
    ReleaseDC (hwnd, hdc) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    for (i = 0 ; i < NUMLINES ; i++)
    {
      TextOut ( hdc, 0, cyChar * i,
        devcaps[i].szLabel,
        lstrlen (devcaps[i].szLabel)) ;

      TextOut ( hdc, 14 * cxCaps, cyChar * i,
        devcaps[i].szDesc,
        lstrlen (devcaps[i].szDesc)) ;

      SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;
      TextOut (hdc, 14*cxCaps+35*cxChar, cyChar*i, szBuffer,
        wsprintf (szBuffer, TEXT ("%5d"),
        GetDeviceCaps (hdc, devcaps[i].iIndex))) ;

      SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

可以看到，这个程序非常类似第四章的SYSMETS1。为了保持程序代码的短小，我没有使用滚动条，因为我知道信息可以在一个画面上显示出来。在256色，640×480的VGA上显示的结果如图5-1所示。

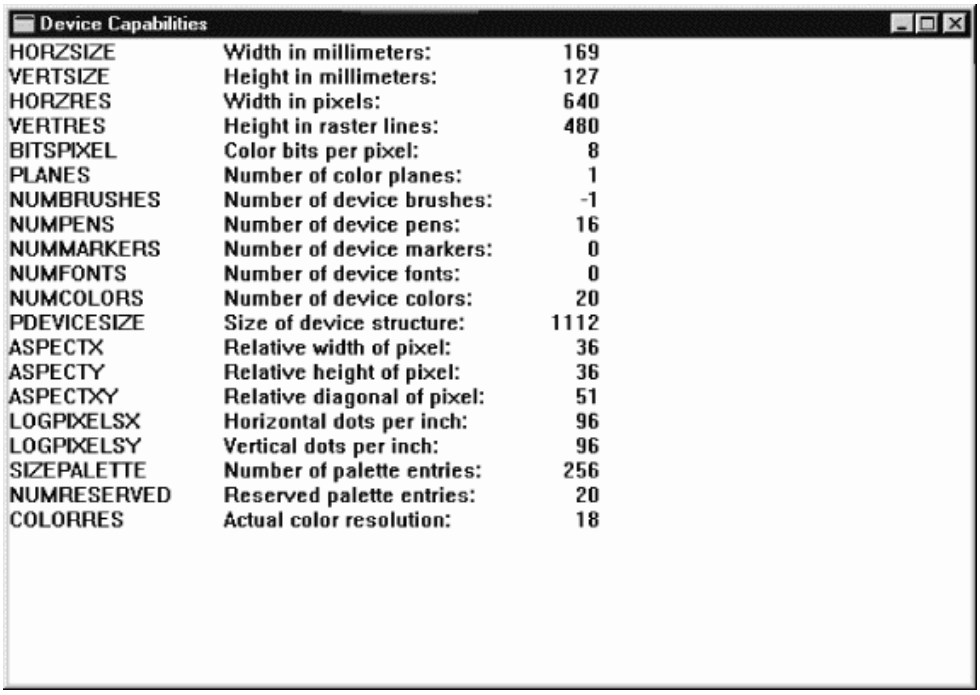


图5-1 256色，640×480VGA上的DEVCAPS1显示

设备的大小

假定要绘制边长为1英寸的正方形，您（程序写作者）或Windows（操作系统）需要知道视讯显示上1英寸对应多少像素。使用GetDeviceCaps函数能取得有关如视讯显示器和打印机之类输出设备的实际显示大小信息。

视讯显示器和打印机是两个不同的设备。但也许最不明显的区别是「分辨率」与设备联系起来的方式。对于打印机，我们经常用「每英寸的点数（dpi）」表示分辨率。例如，大多数激光打印机有300或600dpi的分辨率。然而，视讯显示器的分辨率是以水平和垂直的总像素数来表示的，例如，1024×768。大多数人不会告诉您他的打印机在一张纸上水平和垂直打印多少像素或他们的视讯显示器上每英寸有多少像素。

在本书中，我用「分辨率」来严格定义每度量单位（一般为英寸）内的像素数。我使用「像素大小」或「像素尺寸」表示设备水平或垂直显示的总像素数。「度量大小」或「度量尺寸」是以英寸或毫米为单位的设备显示区域的大小。（对于打印机页面，它不是整个页面，只是可打印的区域。）像素大小除以度量大小就得到分辨率。

现在Windows使用的大多数视讯显示器的屏幕都是宽比高多33%。这就表示纵横比为1.33:1或（一般写法）4:3。历史上，该比例可追溯到Thomas Edison制作电影的年代。它一直作为电影的标准纵横比，直到1953年出现各种型态的宽银幕投影机。电视机屏幕的纵横比也是4:3。

然而，Windows应用程序不应假设视讯显示器具有4:3的纵横比。人们进行文字处理时希望视讯显示器与一张纸的长和宽类似。最普通的选择是把4:3变为3:4显示，把标准显示翻转一下。

如果设备的水平分辨率与垂直分辨率相等，就称设备具有「正方形像素」。现在，Windows普遍使用的视讯显示器都具有正方形像素，但也有例外。（应用程序也不应假设视讯显示器总是具有正方形像素。）Windows第一次发表时，标准显示卡卡是IBM Color Graphics Adapter（CGA），它有640×200的像素大小；Enhanced Graphics Adapter（EGA）有640×350的像素大小；Hercules Graphics Card有720×348的像素大小。所有这些显示卡都使用4:3纵横比的显示器，但是水平和垂直像素数的比值都不是4:3。

执行Windows的使用者很容易确定视讯显示器的像素大小。在「控制台」中执行「显示器」，并选择「设定」页面标签。在标有「桌面区域」的字段中，可以看到这些像素尺寸之一：

640×480像素

800×600像素
1024×768像素
1280×1024像素
1600×1200像素

所有这些都是4:3。（除了1280×1024像素大小。这不但有些不好，还有些令人反感。所有这些像素尺寸都认为在4:3的显示器上会产生正方形的像素。）

Windows应用程序可以使用SM_CXSCREEN和SM_CYSCREEN参数从GetSystemMetrics得到像素尺寸。从DEVCAPS1程序中您会注意到，程序可以用HORZRES（水平分辨率）和VERTRES参数从GetDeviceCaps中得到同样的值。这里「分辨率」指的是像素大小而不是每度量单位的像素数。

这些是设备大小的简单部分，现在开始复杂的部分。

前两个设备能力，HORZSIZE和VERTSIZE，文件中称为「以毫米计的实际屏幕的宽度」及「以毫米计的实际屏幕的高度」（在/Platform SDK/Graphics和Multimedia Services/GDI/Device Contexts/Device Context Reference/Device Context Functions/GetDeviceCaps中）。这些看起来更像直接的定义。例如，给出视讯显示卡和显示器的接口特性，Windows如何真正知道显示器的大小呢？如果您有台膝上型计算机（它的视讯驱动程序能知道准确的屏幕大小）并且连接了外部显示器，又是哪种情况呢？如果把视讯投影机连接到计算机上呢？

在Windows的16位版本中（及在Windows NT中），Windows为HORZSIZE和VERTSIZE使用「标准」的显示大小。然而，从Windows 95开始，HORZSIZE和VERTSIZE值是从HORZRES、VERTRES、LOGPIXELSX和LOGPIXELSY值中衍生出来的。这是它的工作方式。

当您在「控制台」中使用「显示器」程序选择显示的像素大小时，也可以选择系统字体的大小。这个选项的原因是用于640×480显示的字体在提升到1024×768或更大时字太小，而您可能想要更大的系统字体。这些系统字体大小指「显示器」程序的「设定」页面卷标中的「小字体」和「大字体」。

在传统的排版中，字体的字母大小由「点」表示。1点大约1/72英寸，在计算机排版中1点正好为1/72英寸。

理论上，字体的点值是从字体中最高的字符顶部到例如j、p、q和y等字母下部的字符底部的距离，其中不包括重音符号。例如，在10点的字体中此距离是10/72英寸。根据TEXTMETRIC结构，字体的点值等于tmHeight字段减去tmInternalLeading字段，如图5-2所示（该图与上一章的图4-3一样）。

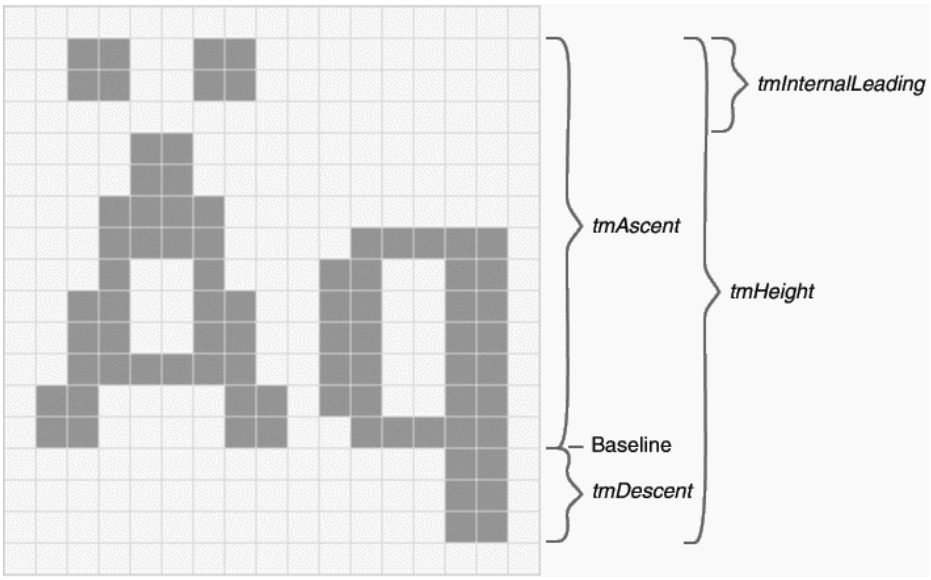


图5-2 小字体和TEXTMETRIC字段。

在真正的排版中，字体的点值与字体字母的实际大小并不正好相等。字体的设计者做出的实际字符比点值指示的要大一些或小一些。毕竟，字体设计是一种艺术而不是科学。

TEXTMETRIC结构的tmHeight字段指出文字的连续行在屏幕或打印机上间隔的方式。这也可以用点来测量。例如，12点的行距指出文字连续行的基准线应该间隔12/72（或1/6）英寸。不应该为10点字体使用10点行距，因为文字的连续行会碰到一起。

10点字体读起来很舒服。小于10点的字体不益于长时间阅读。

Windows系统字体 – 不考虑是大字体还是小字体，也不考虑所选择的视频像素大小 – 固定假设为10点字体和12点行距。这听起来很奇怪，如果字体都是10点，为什么还把它们称为大字体和小字体呢?

解答是：**当您在「控制台」的「显示」程序上选择小字体或大字体时，实际上是选择了一个假定的视讯显示分辨率，单位是每英寸的点数** 。当选择小字体时，即要Windows假定视讯显示分辨率为每英寸96点。当选择大字体时，即要Windows假定视讯显示分辨率为每英寸120点。

再看看图5-2。那是小字体，它依据的显示分辨率为每英寸96点。我说过它是10点字体。10点即是10/72英寸，如果乘以96点，每英寸大概就为13像素。这即是tmHeight减去tmInternalLeading的值。行距是12点，或12/72英寸，它乘以96点，每英寸就为16像素。这即是tmHeight的值。

图5-3显示大字体。这是依据每英寸120点的分辨率。同样，它是10点字体，10/72乘以120点，每英寸等于16像素，即是tmHeight减tmInternalLeading的值。12点行距等于20像素，即是tmHeight的值。（像第四章一样，再次强调所显示的是实际的度量大小，因此您可以理解它工作的方式。不要在您的程序中对此写作程序。）

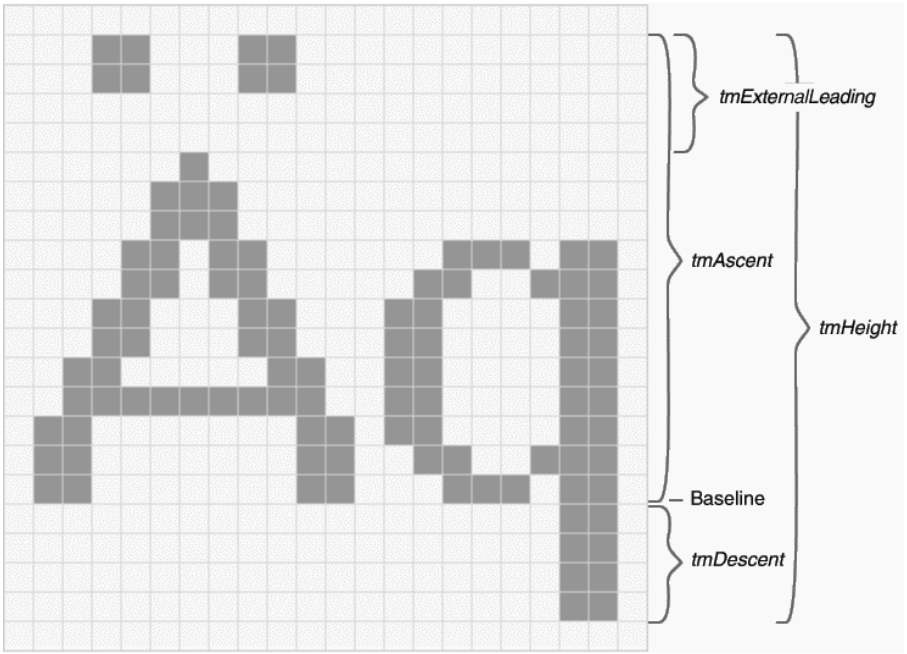


图5-3 大字体和FONTMETRIC字段

在Windows程序中，您可以使用GetDeviceCaps函数取得使用者在「控制台」的「显示器」程序中选择的以每英寸的点数为单位的假定分辨率。要得到这些值（如果视讯显示器不具有正方形像素，在理论上这些值是不同的），可以使用索引LOGPIXELSX和LOGPIXELSY。LOGPIXELS指逻辑像素，它的基本意思是「以每英寸的像素数为单位的非实际分辨率」。

用HORZSIZE和VERTSIZE索引从GetDeviceCaps得到的设备能力，在文件上称为「实际屏幕的宽度，单位毫米」及「实际屏幕的高度，单位毫米」。因为这些值是从HORZRES、VERTRES、LOGPIXELSX和LOGPIXELSY值中衍生出来的，所以它们应该称为「逻辑宽度」和「逻辑高度」。公式是：

$$\text{水平大小(mm)} = 25.4 \times \frac{\text{水平解析度(圖素)}}{\text{邏輯圖素}X\text{(每英寸的點數)}}$$

$$\text{垂直大小(mm)} = 25.4 \times \frac{\text{垂直解析度(圖素)}}{\text{邏輯圖素}Y\text{(每英寸的點數)}}$$

常数25.4用于把英寸转变为毫米。

这看起来是种不合逻辑的退步。毕竟，视讯显示器是可以用尺以毫米为单位的大小（至少是近似的）衡量的。但是Windows 98并不关心这个大小。相反，它以使用者选择的显示像素大小和系统字体大小为基础计算以毫米为单位的显示大小。更改显示的像素大小并根据GetDeviceCaps更改度量大小。这有什么意义呢?

这非常有意义。假定有一个17英寸的显示器。实际的显示大小大约是12英寸乘9英寸。假定在最小要求的640×480像素大小下执行Windows。这意味着实际的分辨率是每英寸53点。10点字体（在纸上便于阅读）在屏幕上从A的顶部到q的底部只有7个像素。这样的字体很难看而且不易读。

（可问问那些在旧的Color Graphics Adapter上执行Windows的人们。）

现在，把您的计算机接上视讯投影机。投影的视讯显示器是4英尺宽，3英尺高。同样的640×480像素大小现在是大约每英寸13点的分辨率。在这种条件下试图显示10点的字体是很可笑的。

10点字体在视讯显示器上应是可读的，因为它在打印时是肯定可读的。所以10点字体就成为一个重要的参照。当Windows应用程序确保10点屏幕字体为平均大小时，就能够使用8点字体显示较小的文字（仍可读），或用大于10点的字体显示较大的文字。因而，视频分辨率（以每英寸的点数为单位）由10点字体的像素大小来确定是很有意义的。

然而，在Windows NT中，用老的方法定义HORZSIZE和VERTSIZE值。这种方法与Windows的16位版本一致。HORZRES和VERTRES值仍然表示水平和垂直像素的数值，LOGPIXELSX和LOGPIXELSY仍然与在「控制台」的「显示器」程序中选择的字体有关。在Windows 98中，LOGPIXELSX和LOGPIXELSY的典型值是96和120 dpi，这取决于您选择的是小字体还是大字体。

在Windows NT中的区别是HORZSIZE和VERTSIZE值固定表示标准显示器大小。对于普通的显示卡，取得的HORZSIZE和VERTSIZE值分别是320和240毫米。这些值是相同的，与选择的像素大小无关。因此，这些值与用HORZRES、VERTRES、LOGPIXELSX和LOGPIXELSY索引从GetDeviceCaps中得到的值不同。然而，可以用前面的公式计算在Windows 98下的HORZSIZE和VERTSIZE值。

如果程序需要实际的视讯显示大小该怎么办？也许最好的解决方法是用对话框让使用者输入它们。

最后，来自GetDeviceCaps的另三个值与视讯大小有关。ASPECTX、ASPECTY和ASPECTXY值是每一个像素的相对宽度、高度和对角线大小，四舍五入到整数。对于正方形像素，ASPECTX和ASPECTY值相同。无论如何，ASPECTXY值应等于ASPECTX与ASPECTY平方和的平方根，就像直角三角形一样。

关于色彩

如果视讯显示卡仅显示黑色像素和白色像素，则每个像素只需要内存中的一位。彩色显示器中每个像素需要多个位。位数越多，色彩越多，或者更具体地说，可以同时显示的不同色彩的数目等于2的位数次方。

「Full-Color」视讯显示器的分辨率是每个像素24位 – 8位红色、8位绿色以及8位蓝色。红、绿、蓝即「色光三原色」。混合这三种基本颜色可以生成许多其它的颜色，您通过放大镜看显示屏，就可以看出来。

「High-Color」显示分辨率是每个像素16位 – 5位红色、6位绿色以及5位蓝色。绿色多一位是因为人眼对绿色更敏感一些。

显示256种颜色的显示卡每个像素需要8位。然而，这些8位的值一般由定义实际颜色的调色盘组织的。我会在第十六章详细地讨论它们。

最后，显示16种颜色的显示卡每个像素需要4位。这16种颜色一般固定分为暗的或亮的红、黑、蓝、青、紫、黄、两种灰色。这16种颜色要回溯到老式的IBM CGA。

祇有在某些怪异的程序中才需要知道视讯显示卡上的内存是如何组织的，但是GetDeviceCaps使程序写作者可以知道显示卡的储存组织以及它能够表示的色彩数目，下面的呼叫传回色彩平面的数目：

```
iPlanes = GetDeviceCaps (hdc, PLANES) ;
```

下面的呼叫传回每个像素的色彩位数：

iBitsPixel = GetDeviceCaps (hdc, BITSPIXEL) ;

大多数彩色图形显示设备使用多个色彩平面或每像素有多个色彩位的设计，但是不能同时一齐使用这两种方式；换句话说，这两个呼叫必有一个传回1。显示卡能够表示的色彩数可以用如下公式来计算：

iColors = 1 << (iPlanes * iBitsPixel) ;

这个值与用NUMCOLORS参数得到的色彩数值可能一样，也可能不一样：

iColors = GetDeviceCaps (hdc, NUMCOLORS) ;

我提到过，256色的显示卡使用色彩调色盘。在那种情况下，以NUMCOLORS为参数时，GetDeviceCaps传回由Windows保留的色彩数，值为20，剩余的236种颜色可以由Windows程序用调色盘管理器设定。对于High-Color和True-Color显示分辨率，带有NUMCOLORS参数的GetDeviceCaps通常传回-1，这样就无法得到需要的信息，因此应该使用前面所示的带有PLANES和BITSPIXEL值的iColors公式。

在大多数GDI函数呼叫中，使用COLORREF值（只是一个32位的无正负号长整数）来表示一种色彩。COLORREF值按照红、绿和蓝色的亮度指定了一种颜色，通常叫做「RGB色彩」 。32位的COLORREF值的设定如图5-4所示。

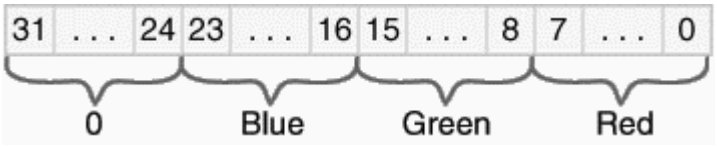


图5-4 32位COLORREF值

注意最前面是标为0的8个位，并且每种原色都指定为一个8位的值。理论上，COLORREF可以指定二的二十四次方种或一千六百万种色彩。

这个无正负号长整数常常称为一个「RGB色彩」。Windows表头文件WINGDI.H提供了几种使用RGB色彩值的宏。RGB宏要求三个参数分别代表红、绿和蓝值，然后将它们组合为一个无正负号长整数：

```
#define RGB(r,g,b) ((COLORREF)(((BYTE)(r) | \
  ((WORD)((BYTE)(g)) << 8)) | \
  (((DWORD)(BYTE)(b)) << 16)))
```

注意三个参数的顺序是红、绿和蓝。因此，值：

RGB (255, 255, 0)

是0x0000FFFF，或黄色（红色和绿色的合成）。当所有三个参数设定为0时，色彩为黑色；当所有参数设定为255时，色彩为白色。GetRValue、GetGValue和GetBValue宏从COLORREF值中抽取出原色值。当您在使用传回RGB色彩值的Windows函数时，这些宏有时会很方便。

在16色或256色显示卡上，Windows可以使用「混色」来模拟设备能够显示的颜色之外的色彩。混色利用了由多种色彩的像素组成的像素图案。可以呼叫GetNearestColor来决定与某一色彩最接近的纯色：

crPureColor = GetNearestColor (hdc, crColor) ;

设备内容属性

前面已经提到过，Windows使用设备内容来保存控制GDI函数在显示器上如何操作的「属性」。

例如，在用TextOut函数显示文字时，程序写作者不必指定文字的色彩和字体，Windows从设备内容取得这个信息。

程序取得一个设备内容的句柄时，Windows用默认值设定所有的属性（在下一节会看到如何取代这种设定）。表5-1列出了Windows 98支持的设备内容属性，程序可以改变或者取得任何一种属性。

表5-1

设备内容属性	默认值	修改该值的函数	取得该值的函数
Mapping Mode	MM_TEXT	SetMapMode	GetMapMode
Window Origin	(0, 0)	SetWindowOrgEx OffsetWindowOrgEx	GetWindowOrgEx
Viewport Origin	(0, 0)	SetViewportOrgEx OffsetViewportOrgEx	GetViewportOrgEx
Window Extents	(1, 1)	SetWindowExtEx SetMapMode ScaleWindowExtEx	GetWindowExtEx
Viewport Extents	(1, 1)	SetViewportExtEx SetMapMode ScaleViewportExtEx	GetViewportExtEx
Pen	BLACK_PEN	SelectObject	SelectObject
Brush	WHITE_BRUSH	SelectObject	SelectObject
Font	SYSTEM_FONT	SelectObject	SelectObject
Bitmap	None	SelectObject	SelectObject
Current Position	(0, 0)	MoveToEx LineTo PolylineTo PolyBezierTo	GetCurrentPositionEx
Background Mode	OPAQUE	SetBkMode	GetBkMode
Background Color	White	SetBkColor	GetBkColor
Text Color	Black	SetTextColor	GetTextColor
Drawing Mode	R2_COPYPEN	SetROP2	GetROP2
Stretching Mode	BLACKONWHITE	SetStretchBltMode	GetStretchBltMode
Polygon Fill Mode	ALTERNATE	SetPolyFillMode	GetPolyFillMode
Intercharacter Spacing	0	SetTextCharacterExtra	GetTextCharacterExtra
Brush Origin	(0, 0)	SetBrushOrgEx	GetBrushOrgEx
Clipping Region	None	SelectObject	GetClipBox

		SelectClipRgn IntersectClipRgn OffsetClipRgn ExcludeClipRect SelectClipPath	

保存设备内容

通常，在您呼叫GetDC或BeginPaint时，Windows用默认值建立一个新的设备内容，您对属性所做的一切改变在设备内容用ReleaseDC或EndPaint呼叫释放时，都会丢失。如果您的程序需要使用非内定的设备内容属性，则您必须在每次取得设备内容句柄时初始化设备内容：

```
case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;
  //设备内容属性
  //绘制窗口显示区域
  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，但是您还可能想要在释放设备内容之后，仍然保存程序中对设备内容属性所做的改变，以便在下一次呼叫GetDC和BeginPaint时它们仍然能够起作用。为此，可在登录窗口类别时，将CS_OWNDC旗标纳入窗口类别的一部分：

```
   wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC ;
```

现在，依据这个窗口类别所建立的每个窗口都将拥有自己的设备内容，它一直存在，直到窗口被删除。如果使用了CS_OWNDC风格，就只需初始化设备内容一次，可以在处理WM_CREATE消息处理期间完成这一操作：

```
case  WM_CREATE:
  hdc = GetDC (hwnd) ;
  //初始化设备内容属性
  ReleaseDC (hwnd, hdc) ;
```

这些属性在改变之前一直有效。

CS_OWNDC风格只影响GetDC和BeginPaint获得的设备内容，不影响其它函数（如GetWindowDC）获得的设备内容。以前不提倡使用CS_OWNDC风格，因为它需要内存；现在，在处理大量图形的Windows NT应用程序中，它可以提高性能。即使用了CS_OWNDC，您仍然应该在退出窗口消息处理程序之前释放设备内容。

某些情况下，您可能想改变某些设备内容属性，用改变后的属性进行绘图，然后恢复原来的设备内容。要简化这一过程，可以通过如下呼叫来保存设备内容的状态：

idSaved = SaveDC (hdc) ;

现在，可以改变一些属性，在想要回到呼叫SaveDC前存在的设备内容时，呼叫：

RestoreDC (hdc, idSaved) ;

您可以在呼叫RestoreDC之前呼叫SaveDC数次。

大多数程序写作者以不同的方式使用SaveDC和RestoreDC。然而，更像汇编语言中的PUSH和POP指令，当您呼叫SaveDC时，不需要保存传回值：

SaveDC (hdc) ;

然后，您可以更改某些属性并再次呼叫SaveDC。要将设备内容恢复到一个已经保存的状态，

呼叫：

RestoreDC (hdc, -1) ;

这就将设备内容恢复到最近由SaveDC函数保存的状态中。

画点和线

在第一章，我们谈论过Windows图形设备接口将图形输出设备的设备驱动程序与计算机连在一起的方式。在理论上，只要提供SetPixel和GetPixel函数，就可以使用图形设备驱动程序绘制一切东西了。其余的一切都可以使用GDI模块中实作的更高阶的例程来处理。例如，画线时，只需GDI呼叫SetPixel数次，并适当地调整x和y坐标。

在实际情况中，也的确可以仅使用SetPixel和GetPixel函数进行您需要的任何绘制。您也可以在这些函数的基础上设计出简洁和构造良好的图形编程系统。唯一的问题是启能。如果一个函数通过几次呼叫才能到达SetPixel函数，那么它执行起来会非常慢。如果一个图形系统画线和进行其它复杂的图形操作是在设备驱动程序的层次上，它就会更有效得多，因为设备驱动程序对完成这些操作的程序代码进行了最佳化。此外，一些显示卡包含了图形协处理器，它允许视讯硬件自己绘制图形。

设定像素

即使Windows GDI包含了SetPixel和GetPixel函数，但很少使用它们。在本书，仅在第七章的CONNECT程序中使用了SetPixel函数，仅在第八章的WHATCLR程序中使用了GetPixel函数。尽管如此，由它们开始来研究图形仍是非常方便。

SetPixel函数在指定的x和y坐标以特定的颜色设定像素：

SetPixel (hdc, x, y, crColor) ;

如同在任何绘图函数中一样，第一个参数是设备内容的句柄。第二个和第三个参数指明了坐标位置。通常要获得窗口显示区域的设备内容，并且x和y相对于该显示区域的左上角。最后一个参数是COLORREF型态指定了颜色。如果在函数中指定的颜色视讯显示器不支持，则函数将像素设定为最接近的纯色并从函数传回该值。

GetPixel函数传回指定坐标处的像素颜色：

crColor = GetPixel (hdc, x, y) ;

直线

Windows可以画直线、椭圆线（椭圆圆周上的曲线）和贝塞尔曲线。Windows 98支援的7个画线函数是：

LineTo 画直线。

Polyline和PolylineTo 画一系列相连的直线。
PolyPolyline 画多组相连的线。
Arc 画椭圆线。
PolyBezier和PolyBezierTo 画贝塞尔曲线。

另外，Windows NT还支持3种画线函数：

ArcTo和AngleArc 画椭圆线。

PolyDraw 画一系列相连的线以及贝塞尔曲线。

这三个函数Windows 98不支援。

在本章的后面我将介绍一些既画线也填入所画图形的封闭区域的函数，这些函数是：

Rectangle 画矩形。

Ellipse 画椭圆。

RoundRect 画带圆角的矩形。

Pie 画椭圆的一部分，使其看起来像一个扇形。

Chord 画椭圆的一部分，以呈弓形。

设备内容的五个属性影响着用这些函数所画线的外观：目前画笔的位置（仅用于LineTo、PolylineTo、PolyBezierTo和ArcTo ）、画笔、背景方式、背景色和绘图模式。

画一条直线，必须呼叫两个函数。第一个函数指定了线的开始点，第二个函数指定了线的终点：

```
MoveToEx (hdc, xBeg, yBeg, NULL) ;
LineTo (hdc, xEnd, yEnd) ;
```

MoveToEx实际上不会画线，它只是设定了设备内容的「目前位置」属性。然后LineTo函数从目前的位置到它所指定的点画一条直线。目前位置只是用于其它几个GDI函数的开始点。在内定的设备内容中，目前位置最初设定在点（0,0）。如果在呼叫LineTo之前没有设定目前位置，那么它将从显示区域的左上角开始画线。

小历史：

Windows的16位版本中，用来改变目前位置的函数是MoveTo。该函数只调整三个参数－设备内容句柄、x和y坐标。函数通过两个16位数拼成的32位无正负号长整数传回先前的目前位置。然而，在Windows的32位版本中，坐标是32位的数值，而C的32位版本中又没有定义64位的整数数据型态，因此这种改变意味着MoveTo在其传回值中不再指出先前的目前位置。在实际的程序写作中，由MoveTo传回的值几乎从来不用，因此就需要一个新函数，这就是MoveToEx。

MoveToEx的最后一个参数是指向POINT结构的指针。从该函数传回后，POINT结构的x和y字段指出了先前的目前位置。如果您不需要这种信息（通常如此），可以简单地如上面的例子所示的那样将最后一个参数设定为NULL。

警告：

尽管Windows 98中的坐标值看起来是32位的，实际上却只用到了低16位，坐标值实际上被限制在-32,768到32,767之间。在Windows NT中，使用完整的32位值。

如果您需要目前位置，就可以通过以下呼叫获得：

GetCurrentPositionEx (hdc, &pt) ;

其中，pt是POINT结构的。

下面的程序代码从窗口的左上角开始，在显示区域中画一个网格，线与线之间相隔100个像素，其中hwnd是窗口句柄，hdc是设备内容句柄，而x和y是整数：

```
GetClientRect (hwnd, &rect) ;
for ( x = 0 ; x < rect.right ; x+= 100)
{
```

```
  MoveToEx (hdc, x, 0, NULL) ;
  LineTo (hdc, x, rect.bottom) ;
}
for (y = 0 ; y < rect.bottom ; y += 100)
{
  MoveToEx (hdc, 0, y, NULL) ;
  LineTo (hdc, rect.right, y) ;
}
```

虽然用两个函数来画一条直线显得有些麻烦，但是在希望画一组相连的直线时，目前画笔位置属性又会变得很有用。例如，您可能想定义一个包含5个点（10个值）的数组，来画一个矩形的边界框：

POINT apt[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，只需要使用MoveToEx移到第一个点，并对后面的点使用LineTo：

```
MoveToEx (hdc, apt[0].x, apt[0].y, NULL) ;
for ( i = 1 ; i < 5 ; i++)
LineTo (hdc, apt[i].x, apt[i].y) ;
```

由于LineTo从目前位置画到（但不包括）LineTo函数中给出的点，所以这段程序代码没有在任何坐标处画两次。虽然在显示器上多输出几次不存在问题，但是在绘图机上或者在其它绘图方式（下面马上会讲到）下，视觉效果就不太好了。

当您要将数组中的点连接成线时，使用Polyline函数要简单得多。下面这条叙述画出与上面一段程序代码相同的矩形：

Polyline (hdc, apt, 5) ;

最后一个参数是点的数目。我们还可以使用(sizeof (apt) / sizeof (POINT))来表示这个值。Polyline与一个MoveToEx函数后面加几个LineTo函数的效果相同，但是，Polyline既不使用也不改变目前位置。PolylineTo有些不同，这个函数使用目前位置作为开始点，并将目前位置设定为最后一根线的终点。下面的程序代码画出与上面所示一样的矩形：

```
MoveToEx (hdc, apt[0].x, apt[0].y, NULL) ;
PolylineTo (hdc, apt + 1, 4) ;
```

您可以对几条线使用Polyline和PolylineTo，这些函数在绘制复杂曲线最有用了。您使用由几百甚至几千条线组成的极短线段，把它们连在一起就像一条曲线一样。例如，画正弦波就是这样的，程序5-2所示的SINEWAVE程序显示了如何做到这一点。

程序5-2　SINEWAVE

SINEWAVE.C

```
/*------------------------------------------------------------------------
SINEWAVE.C -- Sine Wave Using Polyline
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#define NUM 1000
#define TWOPI (2 * 3.14159)
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("SineWave") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
```

```
  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Sine Wave Using Polyline"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxClient, cyClient ;
  HDC hdc ;
  int i ;
  PAINTSTRUCT ps ;
  POINT apt [NUM] ;

  switch (message)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    MoveToEx (hdc, 0, cyClient / 2, NULL) ;
    LineTo (hdc, cxClient, cyClient / 2) ;

    for (i = 0 ; i < NUM ; i++)
    {
      apt[i].x = i * cxClient / NUM ;
      apt[i].y = (int) (cyClient / 2 * (1 - sin (TWOPI * i / NUM))) ;
    }
    Polyline (hdc, apt, NUM) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这个程序有一个含有1000个POINT结构的数组。随着for循环从0增加到999，结构的x成员设定

为从0递增到数值cxClient。结构的y成员设定为一个周期的正弦曲线值，并被放大以填满显示区域。整个曲线的绘制仅仅使用了一个Polyline呼叫。因为Polyline函数是在设备驱动程序层次上实作的，因此它要比呼叫1000次LineTo快得多，结果如图5-5所示。

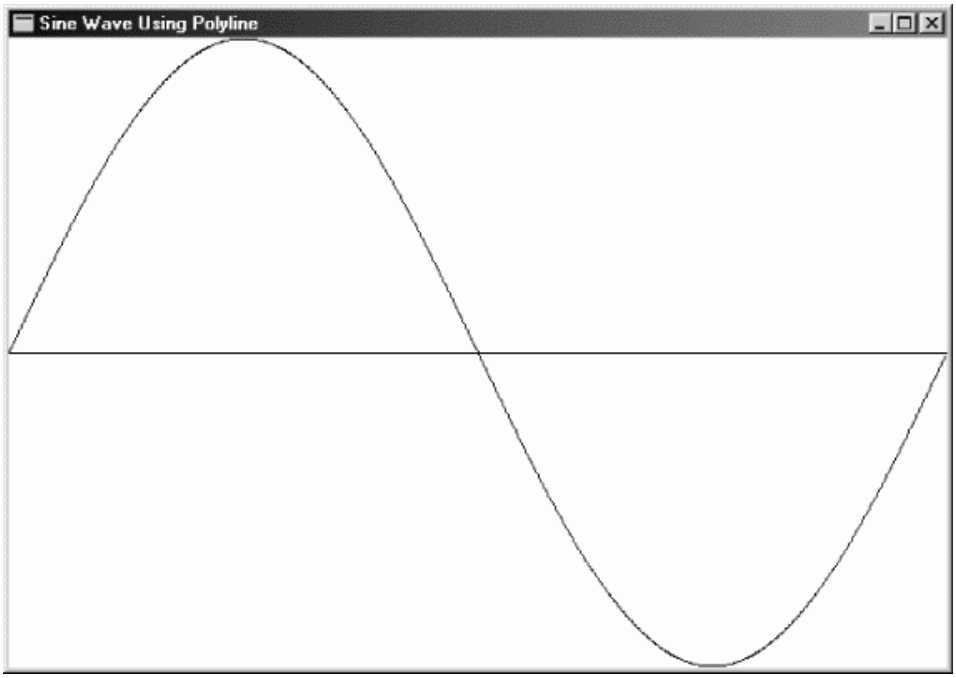


图5-5 SINEWAVE显示

边界框函数

下面我想讨论的是Arc函数，它绘制椭圆曲线。然而，如果不先讨论一下Ellipse函数，那么Arc函数将难以理解；而如果不先讨论Rectangle函数，那么Ellipse函数又将难以理解；而如果讨论Ellipse和Rectangle函数，那么我又会讨论RoundRect、Chord和Pie函数。

问题在于，Rectangle、Ellipse、RoundRect、Chord和Pie函数严格来说不是画线函数。没错，这些函数是在画线，但它们同时又填入画刷填入一个封闭区域。这个画刷内定为白色，因此当您第一次使用这些函数时，您可能不会注意到它们不只是画线。严格地说，这些函数属于后面「填入区域」的小节，不过，我还是在这里讨论它们。

上面提到的函数有一个共同特性，即它们都是依据一个矩形边界框来绘图的。您定义一个包含该对象的框，即「边界框(bounding box)」；Windows就在这个框内画出该物件。

这些函数中最简单的就是画一个矩形：

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。用函数Rectangle画出的图形如图5-6所示，矩形的边总是平行于显示器的水平和垂直边。

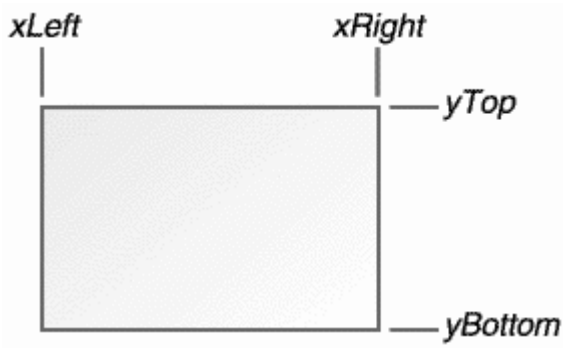


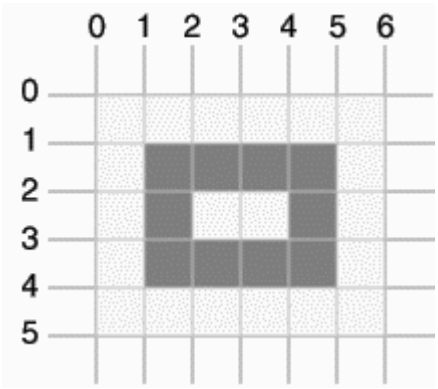
图5-6 使用Rectangle函数画出的图形

以前写过图形程序的程序写作者熟悉像素偏差的问题。有些图形系统画出的图形包含右坐标和底坐标，而有些则只画到（而不包含）右坐标和底坐标。Windows采用后一种方法，不过有一种更简单的方法来思考这个问题。

考虑下面的函数呼叫：

Rectangle (hdc, 1, 1, 5, 4) ;

上面我们提到，Windows在边界框内画图。可以将显示器想象成一个网格，其中，每个像素都在一个网格单元内。边界框画在网格上，然后在边界框内画矩形，下面说明了图形画出来时的样子：



将矩形和显示区域左上角分开的区域有l个像素宽。

我以前提到过，Rectangle严格地说不是画线函数，GDI也填入封闭区域。然而，因为内定用白色填入区域，因此GDI填入区域并不明显。

您知道了如何画矩形，也就知道了如何画椭圆，因为它们使用的参数都是相同的：

Ellipse (hdc, xLeft, yTop, xRight, yBottom) ;

用Ellipse函数画出的图形如图5-7所示（加上了虚线构成的边界框）。

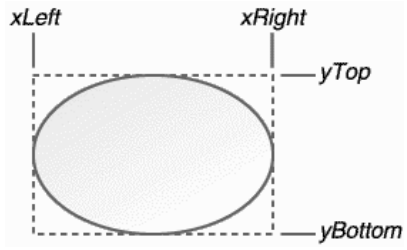


图5-7 用Ellipse函数画出的图形

画圆角矩形的函数使用与函数Rectangle及Ellipse函数相同的边界框，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5-8所示。

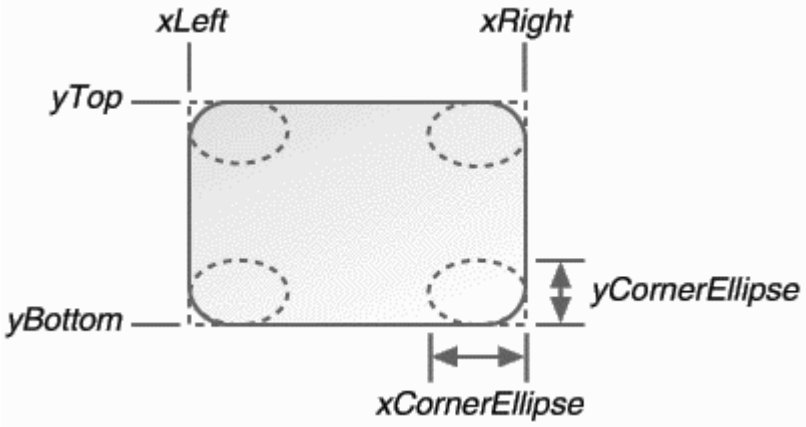


图5-8 用RoundRect函数画出的图形

Windows使用一个小椭圆来画圆角，这个椭圆的宽为xCornerEllipse，高为yCornerEllipse。可以想象这个小椭圆分为了四个部分，一个象限一个，每个刚好用在矩形的一个角上。xCornerEllipse和yCornerEllipse的值越大，角就越明显。如果xCornerEllipse等于xLeft与xRight的差，且yCornerEllipse等于yTop与yBottom的差，那么RoundRect函数将画出一个椭圆。

在绘制图5-8所示的圆角矩形时，用了下面的公式来计算角上椭圆的尺寸。

xCornerEllipse = (xRight - xLeft) / 4 ;
yCornerEllipse = (yBottom- yTop) / 4 ;

这是一种简单的方法，但是结果看起来有点不对劲，因为角的弯曲部分在矩形长的一边要大些。要矫正这一问题，您可以让xCornerEllipse与yCornerEllipse的值相等。

Arc、Chord和Pie函数都只要相同的参数：

```
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5-9所示；用Chord和Pie函数画出的线分别如图5-10和5-11所示。Windows用一条假想的线将(xStart，yStart)与椭圆的中心连接，从该线与边界框的交点开始，Windows按反时针方向，沿着椭圆画一条弧。Windows还用另一条假想的线将(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。

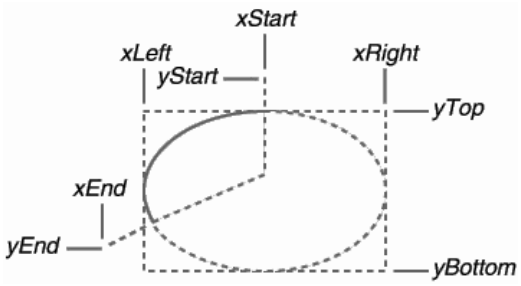


图5-9 Arc函数画出的线

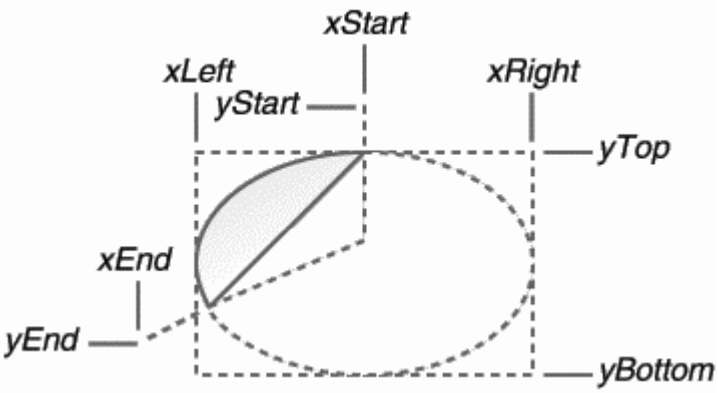


图5-10 Chord函数画出的线

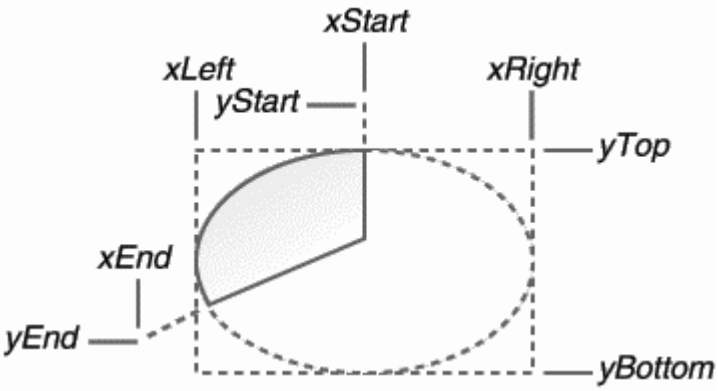


图5-11 Pie函数画出的线

对于Arc函数，这样就结束了。因为弧只是一条椭圆形的线而已，而不是一个填入区域。对于Chord函数，Windows连接弧线的端点。而对于Pie函数，Windows将弧的两个端点与椭圆的中心相连接。弦与扇形图的内部以目前画刷填入。

您可能不太明白在Arc、Chord和Pie函数中开始和结束位置的用法，为什么不简单地在椭圆的周在线指定开始和结束点呢？是的，您可以这么做，但是您将不得不算出这些点。Windows的方法在不要求这种精确性的条件下，却完成了相同的工作。

程序5-3 LINEDEMO画一个矩形、一个椭圆、一个圆角矩形和两条线段，不过不是按这一顺序。程序表明了定义封闭区域的函数实际上对这些区域进行了填入，因为在椭圆后面的线被遮住了，结果如图5-12中所示。

程序5-3 LINEDEMO

LINEDEMO.C

```
/*------------------------------------------------------------
LINEDEMO.C -- Line-Drawing Demonstration Program
(c) Charles Petzold, 1998
------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("LineDemo") ;
  HWND hwnd ;
  MSG msg ;
```

```
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Line Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxClient, cyClient ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    Rectangle (hdc, cxClient / 8, cyClient / 8,
      7 * cxClient / 8, 7 * cyClient / 8) ;
    MoveToEx (hdc, 0, 0, NULL) ;
    LineTo (hdc, cxClient, cyClient) ;

    MoveToEx (hdc, 0, cyClient, NULL) ;
    LineTo (hdc, cxClient, 0) ;

    Ellipse (hdc, cxClient / 8, cyClient / 8,
      7 * cxClient / 8, 7 * cyClient / 8) ;

    RoundRect (hdc, cxClient / 4, cyClient / 4,
      3 * cxClient / 4, 3 * cyClient / 4,
      cxClient / 4, cyClient / 4) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
```

```
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

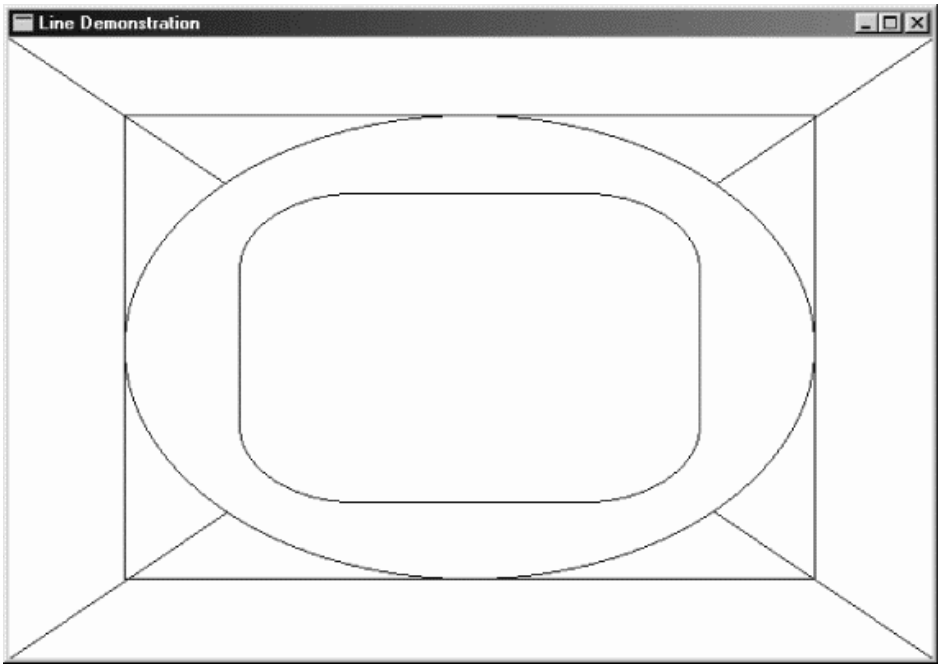


图5-12 LINEDEMO显示

贝塞尔曲线

「曲尺」这个词从前指的是一片木头、橡皮或者金属，用来在纸上画曲线。比如说，如果您有一些不同图点，您想要在它们之间画一条曲线（内插或者外插），您首先将这些点描在绘图纸上，然后，将曲尺定在这些点上，并用铅笔沿着曲尺绕着这些点弯曲的方向画曲线。

当然，时至今日，曲尺已经数学公式化了。有很多种不同的曲尺公式，它们各有千秋。贝塞尔曲线是计算机程序设计中用得最广的曲尺公式之一，它是直到最近才加到操作系统层次的图形支持中的。在六十年代Renault汽车公司进行了由手工设计车体（要用到粘土）到计算机辅助设计的转变。他们需要一些数学工具，而Pierm Bezier找到了一套公式，最后显示出这套公式应付这样的工作非常有用。

此后，二维的贝塞尔曲线成了计算机图学中最有用的曲线（在直线和椭圆之后）。在PostScript中，所有曲线都用贝塞尔曲线表示 – 椭圆线用贝塞尔曲线来逼近。贝塞尔曲线也用于定义PostScript字体的字符轮廓（TrueType使用一种更简单更快速的曲尺公式）。

一条二维的贝塞尔曲线由四个点定义 – 两个端点和两个控制点。曲线的端点在两个端点上，控制点就好像「磁石」一样把曲线从两个端点间的直线处拉走。这一点可以由底下的BEZIER互动交谈程序做出最好的展示，如程序5-4所示。

程序5-4　BEZIER

BEZIER.C

```
/*------------------------------------------------------------------------
BEZIER.C -- Bezier Splines Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
```

```
{
  static TCHAR szAppName[] = TEXT ("Bezier") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Bezier Splines"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void DrawBezier (HDC hdc, POINT apt[])
{
  PolyBezier (hdc, apt, 4) ;
  MoveToEx (hdc, apt[0].x, apt[0].y, NULL) ;
  LineTo (hdc, apt[1].x, apt[1].y) ;

  MoveToEx (hdc, apt[2].x, apt[2].y, NULL) ;
  LineTo (hdc, apt[3].x, apt[3].y) ;
}
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static POINT apt[4] ;
  HDC hdc ;
  int cxClient, cyClient ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    apt[0].x = cxClient / 4 ;
    apt[0].y = cyClient / 2 ;

    apt[1].x = cxClient / 2 ;
    apt[1].y = cyClient / 4 ;
```

```
    apt[2].x = cxClient / 2 ;
    apt[2].y = 3 * cyClient / 4 ;

    apt[3].x = 3 * cxClient / 4 ;
    apt[3].y = cyClient / 2 ;

    return 0 ;
  case WM_LBUTTONDOWN:
  case WM_RBUTTONDOWN:
  case WM_MOUSEMOVE:
    if (wParam & MK_LBUTTON || wParam & MK_RBUTTON)
    {
      hdc = GetDC (hwnd) ;
      SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
      DrawBezier (hdc, apt) ;

      if (wParam & MK_LBUTTON)
      {
        apt[1].x = LOWORD (lParam) ;
        apt[1].y = HIWORD (lParam) ;
      }

      if (wParam & MK_RBUTTON)
      {
        apt[2].x = LOWORD (lParam) ;
        apt[2].y = HIWORD (lParam) ;
      }

      SelectObject (hdc, GetStockObject (BLACK_PEN)) ;
      DrawBezier (hdc, apt) ;
      ReleaseDC (hwnd, hdc) ;
    }
    return 0 ;
  case WM_PAINT:
    InvalidateRect (hwnd, NULL, TRUE) ;

    hdc = BeginPaint (hwnd, &ps) ;
    DrawBezier (hdc, apt) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

由于这个程序要用到一些在第七章才讲的鼠标处理方式，所以我不在这里讨论它的内部运作（不过，这也是简单的），而是用这个程序来实验性地操纵贝塞尔曲线。在这个程序中，两个顶点设定在显示区域的上下居中、左右位于1/4和3/4处的位置；两个控制点可以改变，按住鼠标左键或右键并拖动鼠标可以分别改动两个控制点之一。图5-13是一个典型的例子。

除了贝塞尔曲线本身，程序还从第一个控制点向左边的第一个端点（也叫做开始点）画一条直线，并从第二个控制点向右边的端点画一条直线。

由于下面几个特点，贝塞尔曲线在计算机辅助设计中非常有用。首先，经过少量练习，就可以把曲线调整到与想要的形状非常接近。

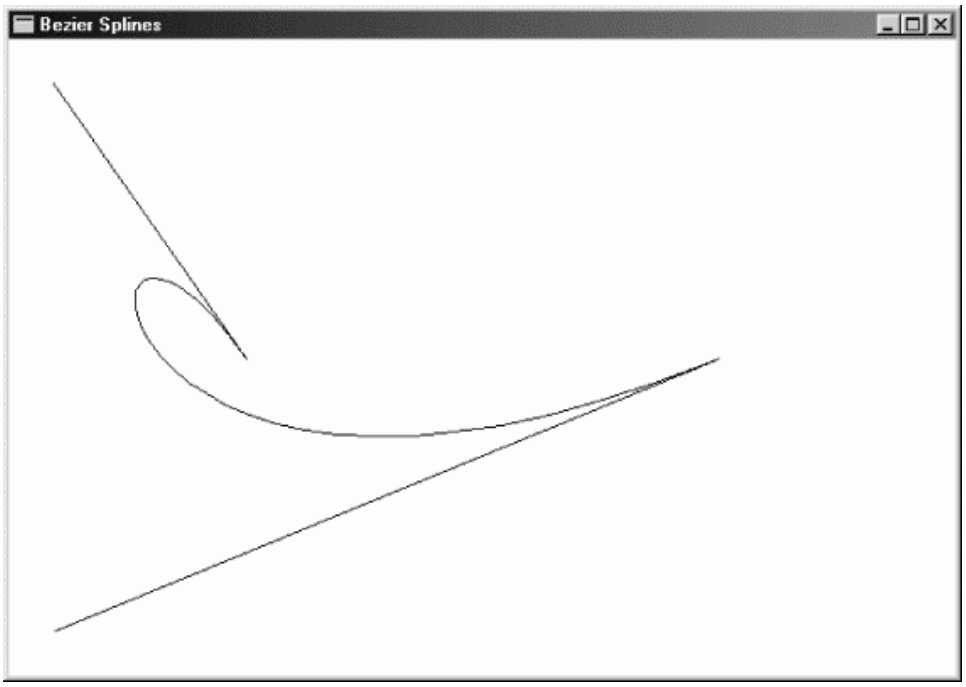


图5-13 BEZIER程序的显示

其次，贝塞尔曲线非常好控制。对于有的曲尺种类来说，曲线不经过任何一个定义该曲线的点。贝塞尔曲线总是由其两个端点开始和结束的（这是在推导贝塞尔公式时所做的假设之一）。另外，有些形式的曲尺公式有奇异点，在这些点处曲线趋向无穷远，这在计算机辅助设计中通常是很不合适的。事实上，贝塞尔曲线总是受限于一个四边形（叫做「凸包」），这个四边形由端点和控制点连接而成。

第三个特点涉及端点和控制点之间的关系。曲线总是与第一个控制点到起点的直线相切，并保持同一方向；同时，也与第二个控制点到终点的直线相切，并保持同一方向。这是用于推导贝塞尔公式时所做的另外两个假设。

第四，贝塞尔曲线通常比较具有美感。我知道这是一个主观评价的问题，不过，并非只有我才这样想。

在32位的Windows版本之前，您必须利用Polyline来自己建立贝塞尔曲线，并且还需要知道下面的贝塞尔曲线的参数方程。起点是(x0,y0)，终点是(x3,y3)，两个控制点是(x1,y1)和(x2,y2)，随着t的值从0到1的变化，就可以画出曲线：

x(t) = (1 - t)3 x0 + 3t (1 - t)2 x1 + 3t2 (1 - t) x2 + t3 x3

y(t) = (1 - t)3 y0 + 3t (1 - t)2 y1 + 3t2 (1 - t) y2 + t3 y3

在Windows 98中，您不需要知道这些公式。要画一条或多条连接的贝塞尔曲线，只需呼叫：

PolyBezier (hdc, apt, iCount) ;

或

PolyBezierTo (hdc, apt, iCount) ;

两种情况下，apt都是POINT结构的数组。对PolyBezier，前四个点（按照顺序）给出贝塞尔曲线的起点、第一个控制点、第二个控制点和终点。此后的每一条贝塞尔曲线只需给出三个点，因为后一条贝塞尔曲线的起点就是前一条贝塞尔曲线的终点，如此类推。iCount参数等于1加上您所绘制的这些首尾相接曲线条数的三倍。

PolyBezierTo函数使用目前点作为第一个起点，第一条以及后续的贝塞尔曲线都只需要给出三个点。当函数传回时，目前点设定为最后一个终点。

一点提示：在画一系列相连的贝塞尔曲线时，只有当第一条贝塞尔曲线的第二个控制点、第一条贝塞尔曲线的终点（也就是第二条曲线的起点）和第二条贝塞尔曲线的第一个控制点线性相关时，也就是说这三个点在同一条直线上时，曲线在连接点处才是光滑的。

使用现有画笔（Stock Pens）

当您呼叫这一节中讨论的任何画线函数时，Windows使用设备内容中目前选中的「画笔」来画线。画笔决定线的色彩、宽度和画笔样式，画笔样式可以是实线、点划线或者虚线，内定设备内容中画笔为BLACK_PEN。不管映像方式是什么，这种画笔都画出一个像素宽的黑色实线来。BLACK_PEN是Windows提供的三种现有画笔之一，其它两种是WHITE_PEN和NULL_PEN，NULL_PEN什么都不画。您也可以自己自订画笔。

Windows程序以句柄来使用画笔。 Windows表头文件WINDEF.H中包含一个叫做HPEN的型态定义，即画笔的句柄，可以定义这个型态的变量（例如hPen）：

```
HPEN hPen ;
```

呼叫GetStockObject，可以获得现有画笔的句柄。例如，假设您想使用名为WHITE_PEN的现有画笔，可以如下取得画笔的句柄：

```
hPen = GetStockObject (WHITE_PEN) ;
```

现在必须将画笔选进设备内容：

```
SelectObject (hdc, hPen) ;
```

目前的画笔是白色。在这个呼叫后，您画的线将使用WHITE_PEN，直到您将另外一个画笔选进设备内容或者释放设备内容句柄为止。

您也可以不定义hPen变量，而将GetStockObject和SelectObject呼叫合并成一个叙述：

```
SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
```

如果想恢复到使用BLACK_PEN的状态，可以用一个叙述取得这种画笔的句柄，并将其选进设备内容：

```
SelectObject (hdc, GetStockObject (BLACK_PEN)) ;
```

SelectObject的传回值是此呼叫前设备内容中的画笔句柄。如果启动一个新的设备内容并呼叫

```
hPen = SelectObject (hdc, GetStockobject (WHITE_PEN)) ;
```

则设备内容中的目前画笔将为WHITE_PEN，变量hPen将会是BLACK_PEN的句柄。以后通过呼叫

```
SelectObject (hdc, hPen) ;
```

就能够将BLACK_PEN选进设备内容。

画笔的建立、选择和删除

尽管使用现有画笔非常方便，但却受限于实心的黑画笔、实心的白画笔或者没有画笔这三种情况。如果想得到更丰富多彩的效果，就必须建立自己的画笔。

这一过程通常是：使用函数CreatePen或CreatePenIndirect建立一个「逻辑画笔」，这仅仅

是对画笔的描述。这些函数传回逻辑画笔的句柄；然后，呼叫SelectObject将画笔选进设备内容。现在，就可以使用新的画笔来画线了。在任何时候，都只能有一种画笔选进设备内容。在释放设备内容（或者在选择了另一种画笔到设备内容中）之后，就可以呼叫DeleteObject来删除所建立的逻辑画笔了。在删除后，该画笔的句柄就不再有效了。

逻辑画笔是一种「GDI对象」，它是您可以建立的六种GDI对象之一，其它五种是画刷、位图、区域、字体和调色盘。除了调色盘之外，这些对象都是通过SelectObject选进设备内容的。

在使用画笔等GDI对象时，应该遵守以下三条规则：

最后要删除自己建立的所有GDI对象。

当GDI对象正在一个有效的设备内容中使用时，不要删除它。

不要删除现有对象。

这些规则当然是有道理的，而且有时这道理还挺微妙的。下面我们将举些例子来帮助理解这些规则。

CreatePen函数的语法形如：

hPen = CreatePen (iPenStyle, iWidth, crColor) ;

其中，iPenStyle参数确定画笔是实线、点线还是虚线，该参数可以是WINGDI.H表头文件中定义的以下标识符，图5-14显示了每种画笔产生的画笔样式。

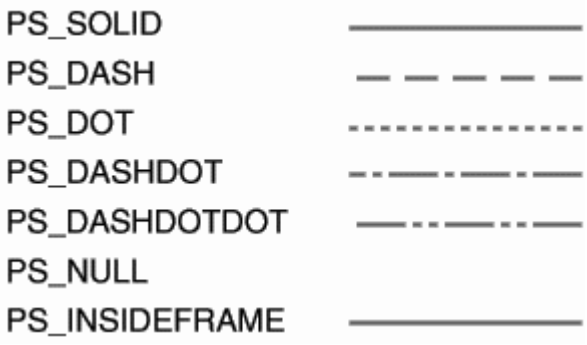


图5-14 七种画笔样式

对于PS_SOLID、PS_NULL和PS_INSIDEFRAME画笔样式，iWidth参数是画笔的宽度。iWidth值为0则意味着画笔宽度为一个像素。现有画笔是一个像素宽。如果指定的是点划线或者虚线式画笔样式，同时又指定一个大于1的实际宽度，那么Windows将使用实线画笔来代替。

CreatePen的crColor参数是一个COLORREF值，它指定画笔的颜色。对于除了PS_INSIDEFRAME之外的画笔样式，如果将画笔选入设备内容中，Windows会将颜色转换为设备所能表示的最相近的纯色。PS_INSIDEFRAME是唯一一种可以使用混色的画笔样式，并且只有在宽度大于1的情况下才如此。

在与定义一个填入区域的函数一起使用时，PS_INSIDEFRAME画笔样式还有另外一个奇特之处：对于除了PS_INSIDEFRAME以外的所有画笔样式来说，如果用来画边界框的画笔宽度大于1个像素，那么画笔将居中对齐在边界框在线，这样边界框线的一部分将位于边界框之外；而对于PS_INSIDEFRAME画笔样式来说，整条边界框线都画在边界框之内。

您也可以通过建立一个型态为LOGPEN（「逻辑画笔」）的结构，并呼叫CreatePenIndirect来建立画笔。如果您的程序使用许多能在原始码中初始化的画笔，那么使用这种方法将有效得多。

要使用CreatePenIndirect，首先定义一个LOGPEN型态的结构：

LOGPEN logpen ;

此结构有三个成员：lopnStyle（无正负号整数或UINT）是画笔样式，lopnWidth（POINT结构）是按逻辑单位度量的画笔宽度，lopnColor (COLORREF)是画笔颜色。Windows只使用lopnWidth结构的x值作为画笔宽度，而忽略y值。

将结构的地址传递给CreatePenIndirect结构就可以建立画笔了：

hPen = CreatePenIndirect (&logpen) ;

注意，CreatePen和CreatePenIndirect函数不需要设备内容句柄作为参数。这些函数建立与设备内容没有联系的逻辑画笔。直到呼叫SelectObject之后，画笔才与设备内容发生联系。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。

下面是建立、选择和删除画笔的一种方法。假设您的程序使用三种画笔 – 一种宽度为1的黑画笔、一种宽度为3的红画笔和一种黑色点式画笔，您可以先定义三个变量来存放这些画笔的句柄：

static HPEN hPen1, hPen2, hPen3 ;

在处理WM_CREATE期间，您可以建立这三种画笔：

```
hPen1 = CreatePen (PS_SOLID, 1, 0) ;
hPen2 = CreatePen (PS_SOLID, 3, RGB (255, 0, 0)) ;
hPen3 = CreatePen (PS_DOT, 0, 0) ;
```

在处理WM_PAINT期间，或者是在拥有一个设备内容有效句柄的任何时间里，您都可以将这三个画笔之一选进设备内容并用它来画线：

SelectObject (hdc, hPen2) ;

画线函数

SelectObject (hdc, hPen1) ;

其它画线函数

在处理WM_DESTROY期间，您可以删除您建立的三种画笔：

DeleteObject (hPen1) ;

DeleteObject (hPen2) ;

DeleteObject (hPen3) ;

这是建立、选择和删除画笔最直接的方法。但是您的程序必须知道执行期间需要哪些逻辑画笔，为此，您可能想要在每个WM_PAINT消息处理期间建立画笔，并在呼叫EndPaint之后删除它们（您可以在呼叫EndPaint之前删除它们，但是要小心，不要删除设备内容中目前选择的画笔）。

您可能还希望随时建立画笔，并将CreatePen和SelectObject呼叫组合到同一个叙述中：

SelectObject (hdc, CreatePen (PS_DASH, 0, RGB (255, 0, 0))) ;

现在再开始画线，您将使用一个红色虚线画笔。在画完红色虚线之后，可以删除画笔。糟了！由于没有保存画笔句柄，怎么才能删除这些画笔呢？不要紧，请记住，SelectObject将传回设备内容中上一次选择的画笔句柄。所以，您可以通过呼叫SelectObject将BLACK_PEN选进设备内容，并删除从SelectObject传回的值：

DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;

下面是另一种方法，在将新建立的画笔选进设备内容时，保存SelectObject传回的画笔句柄：

hPen = SelectObject (hdc, CreatePen (PS_DASH, 0, RGB (255, 0, 0))) ;

现在hPen是什么呢？如果这是在取得设备内容之后第一次呼叫SelectObject，则hPen是BLACK_PEN对象的句柄。现在，可以将hPen选进设备内容，并删除所建立的画笔（第二次SelectObject呼叫传回的句柄），只要一道叙述即可：

DeleteObject (SelectObject (hdc, hPen)) ;

如果有一个画笔的句柄，就可以通过呼叫GetObject取得LOGPEN结构各个成员的值：

GetObject (hPen, sizeof (LOGPEN), (LPVOID) &logpen) ;

如果需要目前选进设备内容的画笔句柄，可以呼叫：

hPen = GetCurrentObject (hdc, OBJ_PEN) ;

在第十七章将讨论另一个建立画笔的函数ExtCreatePen。

填入空隙

使用点式画笔和虚线画笔会产生一个有趣的问题：点和虚线之间的空隙会怎样呢？您所需要的是什么？

空隙的着色取决于设备内容的两个属性 – 背景模式和背景颜色。内定背景模式为OPAQUE，在这种方式下，Windows使用背景色来填入空隙，内定的背景色为白色。这与许多程序在窗口类别中用WHITE_BRUSH来擦除窗口背景的做法是一致的。

您可以通过如下呼叫来改变Windows用来填入空隙的背景色：

SetBkColor (hdc, crColor) ;

与画笔色彩所使用的crColor参数一样，Windows将这里的背景色转换为纯色。可以通过用GetBkColor来取得设备内容中定义的目前背景色。

通过将背景模式转换为TRANSPARENT，可以阻止Windows填入空隙：

SetBkMode (hdc, TRANSPARENT) ;

此后，Windows将忽略背景色，并且不填入空隙，可以通过呼叫GetBkMode来取得目前背景模式（TRANSPARENT或者OPAQUE）。

绘图方式

设备内容中定义的绘图方式也影响显示器上所画线的外观。设想画这样一条直线，它的色彩由画笔色彩和画线区域原来的色彩共同决定。设想用同一种画笔在白色表面上画出黑线而在黑色表面上画出白线，而且不用知道表面是什么色彩。这样的功能对您有用吗？通过绘图方式的设定，这些都可以实作。

当Windows使用画笔来画线时，它实际上执行画笔像素与目标位置处原来像素之间的某种位布尔运算。像素间的位布尔运算叫做「位映像运算」，简称为「ROP」。由于画一条直线只涉及两种像素（画笔和目标），因此这种布尔运算又称为「二元位映像运算」，简记为「ROP2」。Windows定义了16种ROP2代码，表示Windows组合画笔像素和目标像素的方式。在内定设备内容中，绘图方式定义为R2_COPYPEN，这意味着Windows只是将画笔像素复制到目标像素，这也是我们通常所熟知的。此外，还有15种ROP2码。

16种不同的ROP2码是怎样得来的呢？为了示范的需要，我们假设使用单色系统，目标色（窗口显示区域的色彩）为黑色（用0来表示）或者白色（用1来表示），画笔也可以为黑色或者白色。用黑色或者白色画笔在黑色或者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。

画笔在目标上绘制后会得到什么呢？一种可能是不管画笔和目标的色彩，画出的线总是黑色的，这种绘图方式由ROP2代码R2_BLACK表示。另一种可能是只有当画笔与目标都为黑色时，画出的结果才是白色，其它情况下画出的都是黑色。尽管这似乎有些奇怪，Windows还是为这种方式起了一个名字，叫做R2_NOTMERGEPEN。Windows执行目标像素与画笔像素的位「或」运算，然后翻转所得色彩。

表5-2显示了所有16种ROP2绘图方式，表中指示了画笔色彩(P)与目标色彩(D)是如何组合而成结果色彩的。在标有「布尔操作」的那一栏中，用C语言的表示法给出了目标像素与画笔像素的组合方式。

表5-2

画笔(P):目标(D):	11	10	01	00	**布尔操作**	**绘图模式**
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（内定）
	1	1	0	1	P | ~D	R2_MERGEPENNOT
	1	1	1	0	P | D	R2_MERGEPEN
	1	1	1	1	1	R2_WHITE

可以通过以下呼叫在设备内容中设定新的绘图模式:

SetROP2 (hdc, iDrawMode) ;

iDrawMode参数是表中「绘图模式」一栏中给出的值之一。您可以用函数:

iDrawMode = GetROP2 (hdc) ;

来取得目前绘图方式。设备内容中的内定设定为R2_COPYPEN，它用画笔色彩替代目标色彩。在R2_NOTCOPYPEN方式下，若画笔为黑色，则画成白色；若画笔为白色，则画成黑色。R2_BLACK方式下，不管画笔和背景色为何种色彩，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，我们已经讨论了单色系统。然而，大多数系统是彩色的。在彩色系统中，Windows为画笔和目标像素的每个颜色位执行绘图方式的位运算，并再次使用上表描述的16种ROP2代码。R2_NOT绘图方式总是翻转目标色彩来决定线的颜色，而不管画笔的色彩是什么。例如，在青色目标上的线会变成紫色。R2_NOT方式总是产生可见的画笔，除非画笔在中等灰度的背景上绘图。我将在第七章的BLOKOUT程序中展示R2_NOT绘图方式的使用。

绘制填入区域

现在再更进一步，从画线到画图形。Windows中七个用来画带边缘的填入图形的函数列于表5-3中。

表5-3

函数	图形
Rectangle	直角矩形
Ellipse	椭圆
RoundRect	圆角矩形
Chord	椭圆周上的弧，两端以弦连接
Pie	椭圆上的饼图
Polygon	多边形
PolyPolygon	多个多边形

Windows用设备内容中选择的目前画笔来画图形的边界框，边界框还使用目前背景方式、背景色彩和绘图方式，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界框。

图形以目前设备内容中选择的画刷来填入。内定情况下，使用现有对象，这意味着图形内部将画为白色。Windows定义六种现有画刷：WHITE_BRUSH、LTGRAY_BRUSH、GRAY_BRUSH、DKGRAY_BRUSH、BLACK_BRUSH和NULL_BRUSH （也叫HOLLOW_BRUSH）。您可以将任何一种现有画刷选入您的设备内容中，就和您选择一种画笔一样。Windbws将HBRUSH定义为画刷的句柄，所以可以先定义一个画刷句柄变量:

HBRUSH hBrush ;

您可以通过呼叫GetStockObject来取得GRAY_BRUSH的句柄:

hBrush = GetStockObject (GRAY_BRUSH) ;

您可以呼叫SelectObject将它选进设备内容:

SelectObject (hdc, hBrush) ;

现在，如果您要画上表中的任一个图形，则其内部将为灰色。

如果您想画一个没有边界框的图形，可以将NULL_PEN选进设备内容：

SelectObject (hdc, GetStockObject (NULL_PEN)) ;

如果您想画出图形的边界框，但不填入内部，则将NULL_BRUSH选进设备内容：

SelectObject (hdc, GetStockobject (NULL_BRUSH) ;

您也可以自订画刷，就如同您自订画笔一样。我们将马上谈到这个问题。

Polygon函数和多边形填入方式

我已经讨论过了前五个区域填入函数，Polygon是第六个画带边界框的填入图形的函数，该函数的呼叫与Polyline函数相似：

Polygon (hdc, apt, iCount) ;

其中，apt参数是POINT结构的一个数组，iCount是点的数目。如果该数组中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不会这么做）。PolyPolygon函数如下所示：

PolyPolygon (hdc, apt, aiCounts, iPolyCount) ;

该函数绘制多个多边形。最后一个参数给出了所画的多边形的个数。对于每个多边形，aiCounts数组给出了多边形的端点数。apt数组具有全部多边形的所有点。除传回值以外，PolyPolygon在功能上与下面的代码相同：

```
for (i = 0, iAccum = 0 ; i < iPolyCount ; i++)
{
  Polygon (hdc, apt + iAccum, aiCounts[i]) ;
  iAccum += aiCounts[i] ;
}
```

对于Polygon和PolyPolygon函数，Windows使用定义在设备内容中的目前画刷来填入这个带边界的区域。至于填入内部的方式，则取决于多边形填入方式，您可以用SetPolyFillMode函数来设定：

SetPolyFillMode (hdc, iMode) ;

内定情况下，多边形填入方式是ALTERNATE，但是您可以将它设定为WINDING。两种方式的区别参见图5-15所示。

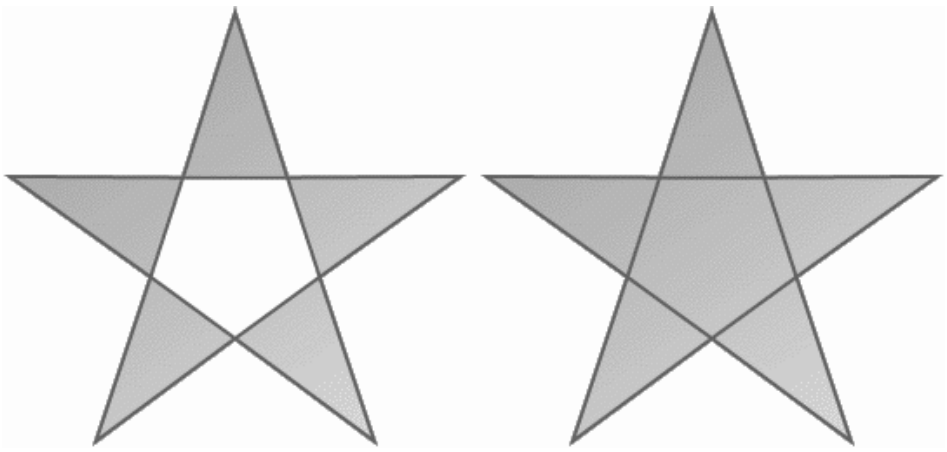


图5-15 用两种多边形填入方式画出的图：ALTERNATE（左）和WINDING（右）

首先，ALTERNATE和WINDING方式之间的区别很容易察觉。对于ALTERNATE方式，您可以设

想从一个无穷大的封闭区域内部的点画线，只有假想的线穿过了奇数条边界线时，才填入封闭区域。这就是填入了星的角而中心没被填入的原因。

五角星的例子使得WINDING方式看起来比实际上更简单一些。在绘制单个的多边形时，大多数情况下，WINDING方式会填入所有封闭的区域。但是也有例外。

在WINDING方式下要确定一个封闭区域是否被填入，您仍旧可以设想从那个无穷大的区域画线。如果假想的线穿过了奇数条边界线，区域就被填入，这和ALTERNATE方式一样。如果假想的线穿过了偶数条边界线，则区域可能被填入也可能不被填入。如果一个方向（相对于假想线）的边界线数与另一个方向的边界线数不相等，就填入区域。

例如，考虑图5-16中的物体。在线的箭头指出了画线的方向。两种方式都会填入三个封闭的L形区域，号码从1到3。号码为4和5的两个小内部区域，在ALTERNATE方式下不会被填入。但是，在WINDING方式下，号码为5的区域会被填入，因为从区域内必须穿过两条相同方向的线才能到达图形外部。号码为4的区域不会被填入，因为必须穿过两条方向相反的线。

如果您怀疑Windows没有这么聪明，那么程序5-5 ALTWIND会展示给您看。

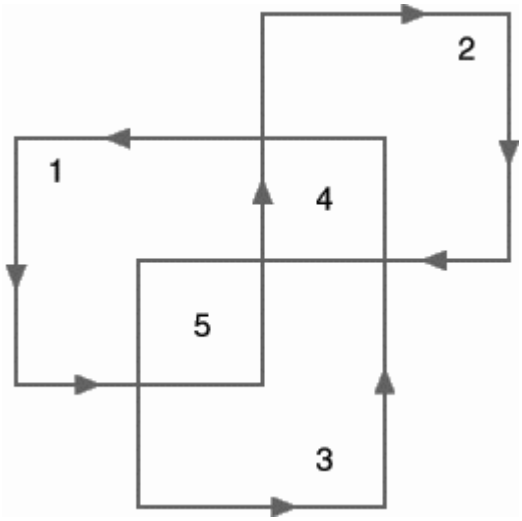


图5-16 WINDING方式不能填入所有内部区域的图形

程序5-5　ALTWIND

ALTWIND.C

```
/*---------------------------------------------------------------------
ALTWIND.C -- Alternate and Winding Fill Modes
(c) Charles Petzold, 1998
---------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("AltWind") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
```

```
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Alternate and Winding Fill Modes"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static POINT aptFigure [10] = {10,70, 50,70, 50,10, 90,10, 90,50,
    30,50, 30,90, 70,90, 70,30, 10,30 };
  static int cxClient, cyClient ;
  HDC hdc ;
  int i ;
  PAINTSTRUCT ps ;
  POINT apt[10] ;

  switch (message)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectObject (hdc, GetStockObject (GRAY_BRUSH)) ;

    for (i = 0 ; i < 10 ; i++)
    {
      apt[i].x = cxClient * aptFigure[i].x / 200 ;
      apt[i].y = cyClient * aptFigure[i].y / 100 ;
    }
    SetPolyFillMode (hdc, ALTERNATE) ;
    Polygon (hdc, apt, 10) ;

    for (i = 0 ; i < 10 ; i++)
    {
      apt[i].x += cxClient / 2 ;
    }
    SetPolyFillMode (hdc, WINDING) ;
    Polygon (hdc, apt, 10) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
```

```
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图形的坐标（划分为100×100个单位）储存在aptFigure数组中。这些坐标是依据显示区域的宽度和高度划分的。程序显示图形两次，一次使用ALTERNATE填入方式，另一次使用WINDING方式。结果见图5-17。

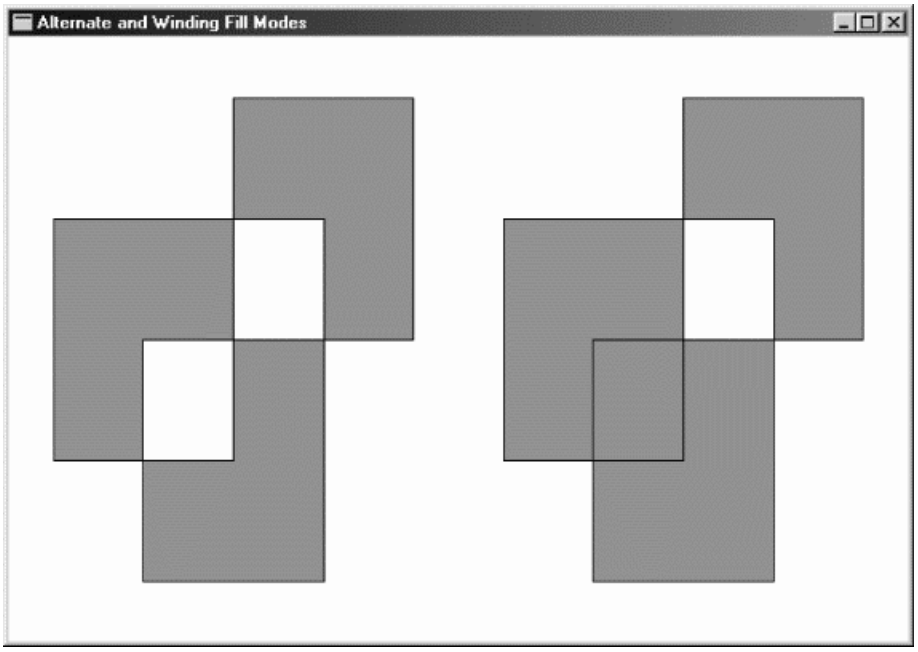


图5-17 ALTWIND的显示

用画刷填入内部

Rectangle、RoundRect、Ellipse、Chord、Pie、Polygon和PolyPolygon图形的内部是用选进设备内容的目前画刷（也称为「图样」）来填入的。画刷是一个8×8的位图，它水平和垂直地重复使用来填入内部区域。

当Windows用混色的方法来显示多于可从显示器上得到的色彩时，实际上是将画刷用于色彩。在单色系统上，Windows能够使用黑色和白色像素的混色建立64种不同的灰色，更精确地说，Windows能够建立64种不同的单色画刷。对于纯黑色，8×8位图中的所有位均为0。第一种灰色有一位为1，第二种灰色有两位为1，以此类推，直到8×8位图中所有位均为1，这就是白色。在16色或256色显示系统上，混色也是位图，并且可以得到更多的色彩。

Windows还有五个函数，可以让您建立逻辑画刷，然后就可使用SelectObject将画刷选进设备内容。与逻辑画笔一样，逻辑画刷也是GDI对象。您建立的所有画刷都必须被删除，但是当它还在设备内容中时不能将其删除。

下面是建立逻辑画刷的第一个函数：

hBrush = CreateSolidBrush (crColor) ;

函数中的Solid并不是指画刷为纯色。在将画刷选入设备内容中时，Windows建立一个混色色的位图，并为画刷使用该位图。

您还可以使用由水平、垂直或者倾斜的线组成的「影线标记(hatch marks)」来建立画刷，这种风格的画刷对着色条形图的内部和在绘图机上进行绘图最有用。建立影线画刷的函数为：

hBrush = CreateHatchBrush (iHatchStyle, crColor) ;

iHatchStyle参数描述影线标记的外观。图5-18显示了六种可用的影线标记风格。

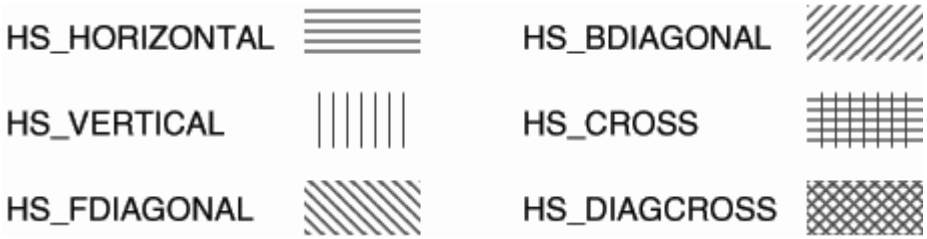


图5-18 六种影线画刷风格

CreateHatchBrush中的crColor参数是影线的色彩。在将画刷选进设备内容时，Windows将这种色彩转换为与之最相近的纯色。影线之间的区域根据设备内容中定义的背景方式和背景色来着色。如果背景方式为OPAQUE，则用背景色（它也被转换为纯色）来填入线之间的空间。在这种情况下，影线和填入色都不能是混色而成的颜色。如果背景方式为TRANSPARENT，则Windows只画出影线，不填入它们之间的区域。

您也可以使用CreatePatternBrush和CreateDIBPatternBrushPt建立自己的位图画刷。

建立逻辑画刷的第五个函数包含其它四个函数：

hBrush = CreateBrushIndirect (&logbrush) ;

变量logbrush是一个型态为LOGBRUSH（「逻辑画刷」）的结构，该结构的三个字段如表5-4所示，lbStyle字段的值确定了Windows如何解释其它两个字段的值：

表5-4

lbStyle (UINT)	lbColor (COLORREF)	lbHatch (LONG)
BS_SOLID	画刷的色彩	忽略
BS_HOLLOW	忽略	忽略
BS_HATCHED	影线的色彩	影线画刷风格
BS_PATTERN	忽略	位图的句柄
BS_DIBPATTERNPT	忽略	指向DIB的指标

前面我们用SelectObject将逻辑画笔选进设备内容，用DeleteObject删除画笔，用GetObject来取得逻辑画笔的信息。对于画刷，同样能使用这三个函数。一旦您取得到了画刷句柄，就可以使用SelectObject将该画刷选进设备内容：

SelectObject (hdc, hBrush) ;

然后，您可以使用DeleteObject函数删除所建立的画刷：

DeleteObject (hBrush) ;

但是，不要删除目前选进设备内容的画刷。

如果您需要取得画刷的信息，可以呼叫GetObject：

GetObject (hBrush, sizeof (LOGBRUSH), (LPVOID) &logbrush) ;

其中，logbrush是一个型态为LOGBRUSH的结构。

GDI 映像方式

到目前为止，所有的程序都是相对于显示区域的左上角，以像素为单位绘图的。这是内定情况，但不是唯一选择。事实上，「映像方式」是一种几乎影响任何显示区域绘图的设备内容属性。另外有四种设备内容属性 – 窗口原点、视端口原点、窗口范围和视端口范围 – 与映像方式密切相关。

大多数GDI绘图函数需要坐标值或大小。例如，下面是TextOut函数：

TextOut (hdc, x, y, psText, iLength) ;

参数x和y分别表示文字的开始位置。参数x是在水平轴上的位置，参数y是在垂直轴上的位置，通常用(x,y)来表示这个点。

在TextOut中，以及在几乎所有GDI函数中，这些坐标值使用的都是一种「逻辑单位」。Windows必须将逻辑单位转换为「设备单位」，即像素。这种转换是由映像方式、窗口和视端口的原点以及窗口和视端口的范围所控制的。映像方式还指示着x轴和y轴的方向(orientation)；也就是说，它确定了当您在向显示器的左或者右移动时x的值是增大还是减小，以及在上下移动时y的值是增大还是减小。

Windows定义了8种映像方式，它们在WINGDI.H中相应的标识符和含义如表5-5所示。

表5-5

映像方式	逻辑单位	增加值	
		x值	y值
MM_TEXT	像素	右	下
MM_LOMETRIC	0.1 mm	右	上
MM_HIMETRIC	0.01 mm	右	上
MM_LOENGLISH	0.01 in.	右	上
MM_HIENGLISH	0.001 in.	右	上
MM_TWIPS	1/1440 in.	右	上
MM_ISOTROPIC	任意(x = y)	可选	可选
MM_ANISOTROPIC	任意(x != y)	可选	可选

METRIC和ENGLISH指一般通行的度量衡系统，点是印刷的测量单位，约等于1/72英寸，但在图形程序设计中假定为正好1/72英寸。「Twip」等于1/20点，也就是1/1440英寸。「Isotropic」和「anisotropic」是真正的单字，意思是「等方性」（同方向）和「异方性」（不同方向）。

您可以使用下面的叙述来设定映射方式：

SetMapMode (hdc, iMapMode) ;

其中，iMapMode是8个映像方式标识符之一。您可以通过以下呼叫取得目前的映像方式：

iMapMode = GetMapMode (hdc) ;

内定映像方式为MM_TEXT。在这种映像方式下，逻辑单位与实际单位相同，这样我们可以直接以像素为单位进行操作。在TextOut呼叫中，它看起来像这样：

TextOut (hdc, 8, 16, TEXT ("Hello"), 5) ;

文字从距离显示区域左端8像素、上端16像素的位置处开始。

如果映像方式设定为MM_LOENGLISH：

SetMapMode (hdc, MM_LOENGLISH) ;

则逻辑单位是百分之一。现在，TextOut呼叫如下：

TextOut (hdc, 50, -100, TEXT ("Hello"), 5) ;

文字从距离显示区域左端0.5英寸、上端1英寸的位置处开始。至于y坐标前面的负号，随着我们对映像方式更详细的讨论，将逐渐清楚。其它映像方式允许程序按照毫米、打印机的点大小或者任意单位的坐标轴来指定坐标。

如果您认为使用像素进行工作很合适，那么就不要使用内定的MM_TEXT方式外的任何映像方式。如果需要以英寸或者毫米尺寸显示图像，那么可以从GetDeviceCaps中取得所需要的信息，自己再进行缩放。其它映像方式都是避免您自己进行缩放的一个方便途径而已。

虽然您在GDI函数中指定的坐标是32位的值，但是仅有Windows NT能够处理全32位。在Windows 98中，坐标被限制为16位，范围从-32,768到32,767。一些使用坐标表示矩形的开始点和结束点的Windows函数也要求矩形的宽和高小于或者等于32,767。

设备坐标和逻辑坐标

您也许会问：如果使用MM_LOENGLISH映射方式，是不是将会得到以百分之一英寸为单位的WM_SIZE消息呢？绝对不会。Windows对所有消息（如WM_MOVE、WM_SIZE和WM_MOUSEMOVE），对所有非GDI函数，甚至对一些GDI函数，永远使用设备坐标。可以这样来考虑：由于映像方式是一种设备内容属性，所以，只有对需要设备内容句柄作参数的GDI函数，映像方式才会起作用。GetSystemMetrics不是GDI函数，所以它总是以设备单位（即像素）为量度来传回大小的。尽管GetDeviceCaps是GDI函数，需要一个设备内容句柄作为参数，但是Windows仍然对HORZRES和VERTRES以设备单位作为传回值，因为该函数的目的之一就是给程序提供以像素为单位的设备大小。

不过，从GetTextMetrics呼叫中传回的TEXTMETRIC结构的值是使用逻辑单位的。如果在进行此呼叫时映像方式为MM_LOENGLISH，则GetTextMetrics将以百分之一英寸为单位提供字符的宽度和高度。在呼叫GetTextMetrics以取得关于字符的宽度和高度信息时，映像方式必须设定成根据这些信息输出文字时所使用的映像方式，这样就可以简化工作。

设备坐标系

Windows将GDI函数中指定的逻辑坐标映像为设备坐标。在讨论以各种不同的映像方式使用逻辑坐标系之前，我们先来看一下Windows为视讯显示器区域定义的不同的设备坐标系。尽管我们大多数时间在窗口的显示区域内工作，但Windows在不同的时间使用另外两种设备坐标区域。所有设备坐标系都以像素为单位，水平轴（即x轴）上的值从左到右递增，垂直轴（即y轴）上的值从上到下递增。

当我们使用整个屏幕时，就根据「屏幕坐标」进行操作。屏幕的左上角为(0,0)点，屏幕坐标用在WM_MOVE消息（对于非子窗口）以及下列Windows函数中：CreateWindow和MoveWindow（都是对于非子窗口）、GetMessagePos、GetCursorPos、SetCursorPos、GetWindowRect以及WindowFromPoint（这不是全部函数的列表）。它们或者是与窗口无关的函数（如两个光标函数），或者是必须相对于某个屏幕点来移动（或者寻找）窗口的函数。如果以DISPLAY为参数呼叫CreateDC，以取得整个屏幕的设备内容，则内定情况下GDI呼叫中指定的逻辑坐标将被映像为屏幕坐标。

「全窗口坐标」以程序的整个窗口为基准，如标题栏、菜单、滚动条和窗口框都包括在内。而

对于普通窗口，点（0,0）是缩放边框的左上角。全窗口坐标在Windows中极少使用，但是如果用GetWindowDC取得设备内容，GDI函数中的逻辑坐标就会转换为显示区域坐标。

第三种坐标系是我们最常使用的「显示区域坐标系」。点（0,0）是显示区域的左上角。当使用GetDC或BeginPaint取得设备内容时，GDI函数中的逻辑坐标就会内定转换为显示区域坐标。

用函数ClientToScreen和ScreenToClient可以将显示区域坐标转换为屏幕坐标，或者反过来，将屏幕坐标转换为显示区域坐标。也可以使用GetWindowRect函数取得屏幕坐标下的整个窗口的位置和大小。这三个函数为一种设备坐标转换为另一种提供了足够的信息。

视端口和窗口

映像方式定义了Windows如何将GDI函数中指定的逻辑坐标映像为设备坐标，这里的设备坐标系取决于您用哪个函数来取得设备内容。要继续讨论映像方式，我们需要一些术语：映像方式用于定义从「窗口」（逻辑坐标）到「视端口」（设备坐标）的映像。

「窗口」和「视端口」这两个词用得并不恰当。在其它图形接口语言中，视端口通常包含有剪裁区域的意思，并且，我们已经用窗口来指程序在屏幕上占据的区域。在这里的讨论中，我们必须把关于这些词的先入之见丢到一边。

「视端口」是依据设备坐标（像素）的。通常，视端口和显示区域相同，但是，如果您已经用GetWindowDC或CreateDC取得了一个设备内容，则视端口也可以是指整窗口坐标或者屏幕坐标。点(0,0)是显示区域（或者整个窗口或屏幕）的左上角，x的值向右增加，y的值向下增加。

「窗口」是依据逻辑坐标的，逻辑坐标可以是像素、毫米、英寸或者您想要的任何其它单位。您在GDI绘图函数中指定逻辑窗口坐标。

但是在真正的意义上，视端口和窗口仅是数学上的概念。对于所有的映像方式，Windows都用下面两个公式来将窗口（逻辑）坐标转化为视埠（设备）坐标：

$$xViewport = (xWindow - xWinOrg) \times \frac{xViewExt}{xWinExt} + xViewOrg$$

$$yViewport = (yWindow - yWinOrg) \times \frac{yViewExt}{yWinExt} + yViewOrg$$

其中，(xWindow,yWindow)是待转换的逻辑点，(xViewport,yViewport)是转换后的设备坐标点，一般情形下差不多就是显示区域坐标了。

这两个公式使用了分别指定窗口和视端口「原点」的点：(xWinOrg，yWinOrg)是逻辑坐标的窗口原点；(xViewOrg,yViewOrg)是设备坐标的视端口原点。在内定的设备内容中，这两个点均被设定为(0,0)，但是它们可以改变。此公式意味着，逻辑点(xWinOrg,yWinOrg)总被映像为设备点(xViewOrg,yViewOrg)。如果窗口和视端口的原点是默认值(0,0)，则公式简化为：

$$xViewport = xWindow \times \frac{xViewExt}{xWinExt}$$

$$yViewport = yWindow \times \frac{yViewExt}{yWinExt}$$

此公式还使用了两点来指定「范围」：(xWinExt,yWinExt)是逻辑坐标的窗口范围；(xViewExt,yViewExt)是设备坐标的窗口范围。在多数映像方式中，范围是映像方式所隐含的，不能够改变。每个范围自身没有什么意义，但是视端口范围与窗口范围的比例是逻辑单位转换为设备单位的换算因子。

例如，当您设定MM_LOENGLISH映像方式时，Windows将xViewExt设定为某个像素数而将xWinExt设定为xViewExt像素占据的一英寸内有几百像素的长度。比值给出了一英寸内有几百个像素的数值。为了提高转换效能，换算因子表示为整数比而不是浮点数。

范围可以为负，也就是说，逻辑x轴上的值不一定非得在向右时增加；逻辑y轴上的值不一定非得在向下时增加。

Windows也能将视埠（设备）坐标转换为窗口（逻辑）坐标：

$$xWindow = (xViewport - xViewOrg) \times \frac{xWinExt}{xViewExt} + xWinOrg$$

$$yWindow = (yViewport - yViewOrg) \times \frac{yWinExt}{yViewExt} + yWinOrg$$

Windows提供了两个函数来让您将设备点转换为逻辑点以及将逻辑点转换为设备点。下面的函数将设备点转换为逻辑点：

DPtoLP (hdc, pPoints, iNumber) ;

其中，pPoints是一个指向POINT结构数组的指针，而iNumber是要转换的点的个数。您会发现这个函数对于将GetClientRect（它总是使用设备单位）取得的显示区域大小转换为逻辑坐标很有用：

GetClientRect (hwnd, &rect) ;

DPtoLP (hdc, (PPOINT) &rect, 2) ;

下面的函数将逻辑点转换为设备点：

LPtoDP (hdc, pPoints, iNumber) ;

处理MM_TEXT

对于MM_TEXT映像方式，内定的原点和范围如下所示：

窗口原点：(0, 0) 可以改变

视埠原点：(0, 0) 可以改变

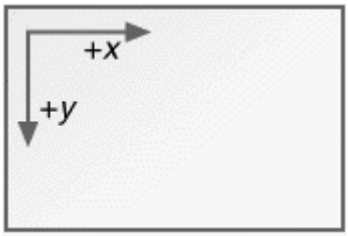
窗口范围：(1, 1) 不可改变

视埠范围：(1, 1) 不可改变

视端口范围与窗口范围的比例为1，所以不用在逻辑坐标与设备坐标之间进行缩放。上面所给出的公式可以简化为：

$$xViewport = xWindow - xWinOrg + xViewOrg$$
$$yViewport = yWindow - yWinOrg + yViewOrg$$

这种映像方式称为「文字」映像方式，不是因为它对于文字最适合，而是由于轴的方向。我们读文字是从左至右，从上至下的，而MM_TEXT以同样的方向定义轴上值的增长方向：



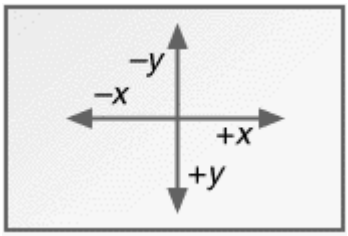
Windows提供了函数SetViewportOrgEx和SetWindowOrgEx，用来改变视端口和窗口的原点，这些函数都具有改变轴的效果，以致(0,0)不再指左上角。一般来说，您会使用SetViewportOrgEx或SetWindowOrgEx之一，但不会同时使用二者。

我们来看一看这些函数有何效果：如果将视埠原点改变为(xViewOrg,yViewOrg)，则逻辑点(0.0)就会映像为设备点(xViewOrg,yViewOrg)。如果将窗口原点改变为(xWinOrg,yWinOrg)，则逻辑点(xWinOrg,yWinOrg)将会映像为设备点(0,0)，即左上角。不管对窗口和视端口原点作什么改变，设备点(0,0)始终是显示区域的左上角。

例如，假设显示区域为cxClient个像素宽和cyClient个像素高。如果想将逻辑点(0，0)定义为显示区域的中心，可进行如下呼叫：

```
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

SetViewportOrgEx的参数总是使用设备单位。现在，逻辑点(0,0)将映像为设备点(cxClient/2,cyClient/2)，而显示区域的坐标系变成如下形状：



逻辑x轴的范围从-cxClient/2到+cxClient/2，逻辑y轴的范围从-cyClient/2到+cyClient/2，显示区域的右下角为逻辑点(cxClient/2,cyClient/2)。如果您想从显示区域的左上角开始显示文字。则需要使用负坐标：

```
TextOut (hdc, -cxClient / 2, -cyClient / 2, "Hello", 5) ;
```

用下面的SetWindowOrgEx叙述可以获得与上面使用SetViewportOrgEx同样的效果：

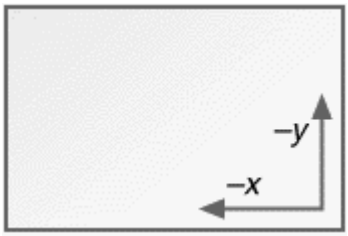
```
SetWindowOrgEx (hdc, -cxClient / 2, -cyClient / 2, NULL) ;
```

SetWindowOrgEx的参数总是使用逻辑单位。在这个呼叫之后，逻辑点(-cxClient / 2,-cyClient / 2)映像为设备点(0,0)，即显示区域的左上角。

您不会将这两个函数一起用，除非您知道这么做的结果：

```
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
SetWindowOrgEx (hdc, -cxClient / 2, -cyClient / 2, NULL) ;
```

这意味着逻辑点(-cxClient/2,-cyClient/2)将映像为设备点(cxClient/2, cyClient/2)，结果是如下所示的坐标系：



您可以使用下面两个函数取得目前视端口和窗口的原点：

GetViewportOrgEx (hdc, &pt) ;

GetWindowOrgEx (hdc, &pt) ;

其中pt是POINT结构。由GetViewportOrgEx传回的值是设备坐标，而由GetWindowOrgEx传回的值是逻辑坐标。

您可能想改变视端口或者窗口的原点，以改变窗口显示区域内的显示输出－例如，响应使用者在滚动条内的输入。但是，改变视端口和窗口原点并不能立即改变显示输出，而必须在改变原点之后更新输出。例如，在第四章的SYSMETS2程序中，我们使用了iVscrollPos值（垂直滚动条的目前位置）来调整显示输出的y坐标：

```
case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;

  for (i = 0 ; i < NUMLINES ; i++)
  {
    y = cyChar * (i - iVscrollPos) ;
    // 显示文字
  }
  EndPaint (hwnd, &ps) ;
  return 0 ;
    我们可以使用SetWindowOrgEx获得同样的效果:
case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;
  SetWindowOrgEx (hdc, 0, cyChar * iVscrollPos) ;

  for (i = 0 ; i < NUMLINES ; i++)
  {
    y = cyChar * i ;
    // 显示文字
  }
  EndPaint (hwnd, &ps) ;
  return 0 ;
```

现在，TextOut函数的y坐标的计算不需要iVscrollPos的值。这意味着您可以将文字输出函数放到一个例程中，不用将iVscrollPos值传给该例程，因为我们是通过改变窗口原点来调整文字显示的。

如果您有使用直角坐标系（即笛卡尔坐标系）的经验，那么将逻辑点（0,0）移到显示区域的中央（像我们上面所说的那样）的确值得考虑。但是，对于MM_TEXT映像方式来说，还存在着一个小小的问题：笛卡尔坐标系中，y值是随着上移而增加的，而MM_TEXT定义为下移时y值增加。从这一点来看，MM_TEXT有点古怪，而下面这五种映射方式都使用通常的增值方法。

「度量」映像方式

Windows包含五种以实际尺寸来表示逻辑坐标的映像方式。由于x轴和y轴的逻辑坐标映像为相

同的实际单位，这些映像方式能使您画出不变形的圆和矩形。

这五种「度量」映像方式在表5-6中列出，按照从低精度到高精度的顺序排列。右边的两列分别给出了以英寸和毫米为单位时逻辑单位的大小，以便比较。

表5-6

映像方式	逻辑单位	英寸	毫米
MM_LOENGLISH	0.01 in.	0.01	0.254
MM_LOMETRIC	0.1 mm.	0.00394	0.1
MM_HIENGLISH	0.001 in.	0.001	0.0254
MM_TWIPS	1/1400 in.	0.000694	0.0176
MM_HIMETRIC	0.01 mm.	0.000394	0.01

内定窗口及视端口的原点和范围如下所示：

窗口原点：(0, 0) 可以改变

视埠原点：(0, 0) 可以改变

窗口范围：(1, 1) 不可改变

视埠范围：(1, 1) 不可改变

问号表示窗口和视端口的范围依赖于映像方式和设备的分辨率。前面已经提到过，这些范围本身并不重要，但是表示比例时就必须知道。下面是窗口坐标到视端口坐标的转换公式：

$$xViewport = (xWindow - xWinOrg) \times \frac{xViewExt}{xWinExt} + xViewOrg$$

$$yViewport = (yWindow - yWinOrg) \times \frac{yViewExt}{yWinExt} + yViewOrg$$

例如，对于MM_LOENGLISH，Windows计算的范围如下：

$$\frac{xViewExt}{xWinExt} = 0.01\ in\ 中的水平圖素數$$

$$\frac{-yViewExt}{yWinExt} = 0.01\ in\ 中的垂直圖素數$$

Windows使用这些来自GetDeviceCaps的有用信息设定范围。只是在Windows 98和Windows NT之间有一点差别。

首先，来看看Windows 98是如何做的：假设您使用「控制台」的「显示」程序选择了96 dpi的系统字体。GetDeviceCaps对于LOGPIXELSX和LOGPIXELSY索引都将传回值96。Windows为视埠范围使用这些值并以表5-7的方式设定视端口和窗口的范围。

表5-7

映像方式	视埠范围(x,y)	窗口范围(x,y)
MM_LOMETRIC	(96, 96)	(254, -254)
MM_HIMETRIC	(96, 96)	(2540, -2540)
MM_LOENGLISH	(96, 96)	(100, -100)
MM_HIENGLISH	(96, 96)	(1000, -1000)
MM_TWIPS	(96, 96)	(1440, -1440)

这样，对MM_LOENGLISH来说，96除以100的比值是0.01英寸中的像素数。对MM_LOMETRIC来说，96除以254的比值是0.1毫米中的像素数。

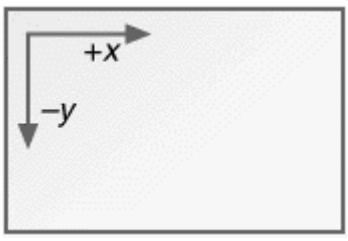
Windows NT使用不同的方法设定视端口和窗口的范围（与早期16位版本的Windows一致的方法）。视端口范围依据屏幕的像素尺寸。可以使用HORZRES和VERTRES索引从GetDeviceCaps取得这种信息。窗口范围依据假定的显示大小，它是您使用HORZSIZE和VERTSIZE索引时由GetDeviceCaps传回的。我在前面提到过，这些值一般是320和240毫米。如果您将显示器的像素尺寸设定为1024×768，则表5-8就是Windows NT报告的视端口和窗口范围的值。

表5-8

映像方式	视埠范围(x,y)	窗口范围(x,y)
MM_LOMETRIC	(1024, -768)	(3,200, 2,400)
MM_HIMETRIC	(1024, -768)	(32,000, 24,000)
MM_LOENGLISH	(1024, -768)	(1,260, 945)
MM_HIENGLISH	(1024, -768)	(12,598, 9,449)
MM_TWIPS	(1024, -768)	(18,142, 13,606)

这些窗口范围表示包含显示器全部宽度和高度的逻辑单位元数值。320毫米宽的屏幕也为1260 MM_LOENGLISH单位或12.6英寸（320除以25.4毫米/英寸）。

范围中，y前面的负号表示改变了轴的方向。对于这五种映像方式，y值随上升而增加，然而注意内定的窗口和视端口原点均为(0,0)。这个事实有一个有趣的结果。当一开始改变为五种映像方式之一时，坐标系如下：



要想在显示区域显示任何东西，必须使用负的y值。例如下面的程序代码：

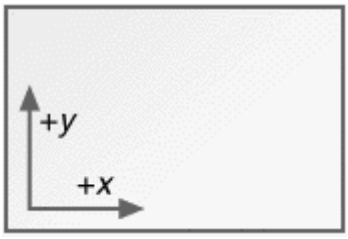
```
SetMapMode (hdc, MM_LOENGLISH) ;

TextOut (hdc, 100, -100, "Hello", 5) ;
```

将把文字显示在距离显示区域左边和上边各一英寸的地方。

为了使自己保持头脑清醒，您可能想避免这样做。一种解决办法是将逻辑的(0,0)点设为显示区域的左下角，您可以通过呼叫SetViewportOrgEx来完成（假设cyClient是以像素为单位的显示区域的高度）：

SetViewportOrgEx (hdc, 0, cyClient, NULL) ;

此时的坐标系如下：

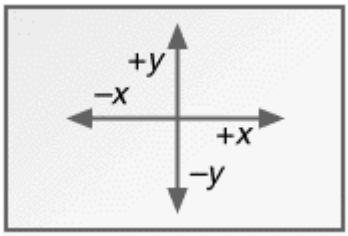


这是直角坐标系的右上象限。

另一种方法是将逻辑(0,0)点设为显示区域的中心：

SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;

此时的坐标系如下所示：



现在，我们有了一个真正的4象限笛卡尔坐标系，在x轴和y轴上有相等的按英寸、毫米或twip计算的逻辑单位。

您还可以使用SetWindowOrgEx函数来改变逻辑(0,0)点，但是这稍微困难一些，因为SetWindowOrgEx的参数必须使用逻辑单位，先要将(cxClient,cyClient)用DPtoLP函数转换为逻辑坐标。假设变量pt是型态为POINT的结构，下面的代码将逻辑(0,0)点改变到显示区域的中央：

```
pt.x = cxClient ;
pt.y = cyClient ;
DptoLP (hdc, &pt, 1) ;
SetWindowOrgEx (hdc, -pt.x / 2, -pt.y / 2, NULL) ;
```

「自行决定」的映像方式

剩下的两种映像方式为MM_ISOTROPIC和MM_ANISOTROPIC。只有这两种映像方式可以让您改变视端口和窗口范围，也就是说可以改变Windows用来转换逻辑和设备坐标的换算因子。「isotropic」的意思是「同方向性」;「anisotropic」的意思是「异方向性」。与上面所讨论的度量映射方式相似，MM_ISOTROPIC使用相同的轴，x轴上的逻辑单位与y轴上的逻辑单位的实际尺寸相等。这对您建立纵横比与显示比无关的图像是有帮助的。

MM_ISOTROPIC与度量映像方式之间的区别是，使用MM_ISOTROPIC，您可以控制逻辑单位的实际尺寸。如果愿意，您可以根据显示区域的大小来调整逻辑单位的实际尺寸，从而使所画的图像总是包含在显示区域内，并相应地放大或缩小。例如，第八章的两个时钟程序就是方向同性的例子。在您改变窗口大小时，时钟也相应地调整。

Windows程序完全可以通过调整窗口和视端口范围来处理图像大小的变化。因此，不管窗口尺寸怎样变，程序都可以在绘图函数中使用相同的逻辑单位。

有时候MM_TEXT和度量映像方式称为「完全局限性」映像方式，这就是说，您不能改变窗口和视端口的范围以及Windows将逻辑坐标换算为设备坐标的方法。MM_ISOTROPIC是一种「半局限

性」的映像方式，Windows允许您改变窗口和视端口范围，但只是调整它们，以便x和y逻辑单位代表同样的实际尺寸。MM_ANISOTROPIC映像方式是「非局限性」的，您可以改变窗口和视端口范围，但是Windows不调整这些值。

MM_ISOTROPIC映像方式

如果想要在使用任意的轴时都保证两个轴上的逻辑单位相同，则MM_ISOTROPIC映像方式就是理想的映像方式。这时，具有相同逻辑宽度和高度的矩形显示为正方形，具有相同逻辑宽度和高度的椭圆显示为圆。

当您刚开始将映像方式设定为MM_ISOTROPIC时，Windows使用与MM_LOMETRIC同样的窗口和视端口范围（但是，不要对此有所依赖）。区别在于，您现在可以呼叫SetWindowExtEx和SetViewportExtEx来根据自己的偏好改变范围了，然后，Windows将调整范围的值，以便两条轴上的逻辑单位有相同的实际距离。

一般说来，您可以用所期望的逻辑窗口的逻辑尺寸作为SetWindowExtEx的参数，用显示区域的实际宽和高作为SetViewportExtEx的参数。Windows在调整这些范围时，必须让逻辑窗口适应实际窗口，这就有可能导致显示区域的一段落到了逻辑窗口的外面。必须在呼叫SetViewportExtEx之前呼叫SetWindowExtEx，以便最有效地使用显示区域中的空间。

例如，假设您想要一个「传统的」单象限虚拟坐标系，其中(0,0)在显示区域的左下角，宽度和高度的范围都是从0到32,767，并且希望x和y轴的单位具有同样的实际尺寸。以下就是所需的程序：

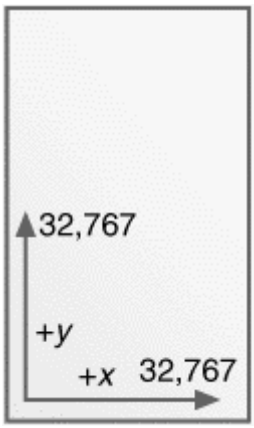
```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 32767, 32767, NULL) ;
SetViewportExtEx (hdc, cxClient, -cyClient, NULL) ;
SetViewportOrgEx (hdc, 0, cyClient, NULL) ;
```

如果其后用GetWindowExtEx和GetViewportExtEx函数获得了窗口和视端口的范围，可以发现，它们并不是先前指定的值。Windows将根据显示设备的纵横比来调整范围，以便两条轴上的逻辑单位表示相同的实际尺寸。

如果显示区域的宽度大于高度（以实际尺寸为准），Windows将调整x的范围，以便逻辑窗口比显示区域视端口窄。这样，逻辑窗口将放置在显示区域的左边：

Windows 98不允许在显示区域的右边超越x轴的范围之外显示任何东西，因为这需要一个大于16位所能表示的坐标。Windows NT使用全32位坐标，您可以在超出右边显示一些东西。

如果显示区域的高度大于宽度（以实际尺寸为准），那么Windows将调整y的范围。这样，逻辑窗口将放置在显示区域的下边：



Windows 98不允许在显示区域的顶部显示任何东西。

如果您希望逻辑窗口总是放在显示区域的左上部，那么将前面给出的程序代码改为：

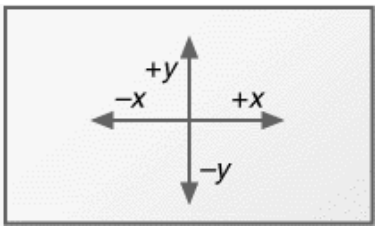
```
SetMapMode (MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 32767, 32767, NULL) ;
SetViewportExtEx (hdc, cxClient, -cyClient, NULL) ;
SetWindowOrgEx (hdc, 0, 32767, NULL) ;
```

在呼叫SetWindowOrgEx中，我们要求将逻辑点(0, 32767)映像为设备点(0,0)。现在，如果显示区域的高大于宽，则坐标系将安排为：

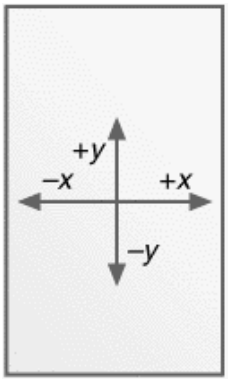
对于时钟程序，您也许想要使用一个四象限的笛卡尔坐标系，四个方向的坐标尺度可以任意指定，(0,0) 必须居于显示区域的中央。如果您想要每条轴的范围从0到1000，则可以使用以下程序代码：

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 1000, 1000, NULL) ;
SetViewportExtEx (hdc, cxClient / 2, -cyClient / 2, NULL) ;
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

如果显示区域的宽度大于高度，则逻辑坐标系形如：



如果显示区域的高度大于宽度，那么逻辑坐标也会居中：



记住，窗口或者视端口范围并不意味着要进行剪裁。在呼叫GDI函数时，您仍然对以随便地使用小于-1000和大于1000的x和y值。根据显示区域的外形，这些点可能看得见，也可能看不见。

在MM_ISOTROPIC映像方式下，可以使逻辑单位大于像素。例如，假设您想要一种映像方式，使点(0,0)显示在屏幕的左上角，y的值向下增长（和MM_TEXT相似），但是逻辑坐标单位为1/16英寸。以下是一种方法：

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 16, 16, NULL) ;
SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
                  GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
```

SetWindowExtEx函数的参数指出了每一英寸中逻辑单位数。SetViewportExtEx函数的参数指出了每一英寸中实际单位数（像素）。

然而，这种方法与Windows NT中的度量映像方式不一致。这些映射方式使用显示器的像素大小和公制大小。要与度量映像方式保持一致，可以这样做：

```
SetMapMode (hdc, MM_ISOTROPIC) ;
SetWindowExtEx (hdc, 160 * GetDeviceCaps (hdc, HORZSIZE) / 254,
                160 * GetDeviceCaps (hdc, VERTSIZE) / 254, NULL) ;
SetViewportExtEx (hdc, GetDeviceCaps (hdc, HORZRES),
                  GetDeviceCaps (hdc, VERTRES), NULL) ;
```

在这个程序代码中，视埠范围设定为按像素计算的整个屏幕的大小，窗口范围则必须设定为以1/16英寸为单位的整个屏幕的大小。GetDeviceCaps以HORZRES和VERTRES为参数，传回以毫米为单位的设备尺寸。如果我们使用浮点数，将把毫米数除以25.4，转换为英寸，然后，再乘以16以转换为l/16英寸。但是，由于我们使用的是整数，所以先乘以160，再除以254。

当然，这种坐标系会使逻辑单位大于实际单位。在设备上输出的所有东西都将映像为按1/16英寸增量的坐标值。当然，这样就不能画两条间隔l/32英寸的水平直线，因为这样将需要小数逻辑坐标。

MM_ANISOTROPIC：根据需要放缩图像

在MM_ISOTROPIC映像方式下设定窗口和视端口范围时，Windows会调整范围，以便两条轴上的逻辑单位具有相同的实际尺度。在MM_ANISOTROPIC映射方式下，Windows不对您所设定的值进行调整，这就是说，MM_ANISOTROPIC不需要维持正确的纵横比。

使用MM_ANISOTROPIC的一种方法是对显示区域使用任意坐标，就像我们对MM_ISOTROPIC所做的一样。下面的程序代码将点(0,0)设定为显示区域的左下角，x轴和y轴都从0到32,767：

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 32767, 32767, NULL) ;
SetViewportExtEx (hdc, cxClient, -cyClient, NULL) ;
SetViewportOrgEx (hdc, 0, cyClient, NULL) ;
```

在MM_ISOTROPIC方式下，相似的程序代码导致显示区域的一部分在轴的范围之外。但是对于MM_ANISOTROPIC，不论其尺度多大，显示区域的右上角总是(32767, 32767)。如果显示区域不是正方形的，则逻辑x和y的单位具有不同的实际尺度。

前一节在MM_ISOTROPIC映像方式下，我们讨论了在显示区域中画一个类似时钟的图像，x和y轴的范围都是从-1000到+1000。对于MM_ANISOTROPIC，也可以写出类似的程序：

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 1000, 1000, NULL) ;
SetViewportExtEx (hdc, cxClient / 2, -cyClient / 2, NULL) ;
SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
```

与MM_ANISOTROPIC方式不同的是，这个时钟一般是椭圆形的，而不是圆形的。

另一种使用MM_ANISOTROPIC的方法是将x和y轴的单位固定，但其值不相等。例如，如果有一个只显示文字的程序，您可能想根据单个字符的高度和宽度设定一种粗刻度的坐标：

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 1, 1, NULL) ;
SetViewportExtEx (hdc, cxChar, cyChar, NULL) ;
```

当然，这里假设cxChar和cyChar分别是那种字体的字符宽度和高度。现在，您可以按字符行和列指定坐标。下面的叙述在距离显示区域左边三个字符，上边二个字符处显示文字：

TextOut (hdc, 3, 2, TEXT ("Hello"), 5) ;

如果您使用固定大小的字体时会更加方便，就像下面的WHATSIZE程序所示的那样。

当您第一次设定MM_ANISOTROPIC映像方式时，它总是继承前面所设定的映像方式的范围，这会很方便。可以认为MM_ANISOTROPIC不「锁定」范围；也就是说，它允许您任意改变窗口范围。例如，假设您想用MM_LOENGLISH映像方式，因为希望逻辑单位为0.01英寸，但您不希望y轴的值向上增加，喜欢如MM_TEXT那样的方向，即y轴的值向下增加，可以使用如下的代码：

SIZE size ;

其它行程序

```
SetMapMode (hdc, MM_LOENGLISH) ;
SetMapMode (hdc, MM_ANISOTROPIC) ;
GetViewportExtEx (hdc, &size) ;
SetViewportExtEx (hdc, size.cx, -size.cy, NULL) ;
```

我们首先将映像方式设定为MM_LOENGLISH，然后，通过将映像方式设定为MM_ANISOTROPIC让范围可以自由改变。GetViewportExtEx取得视埠范围并放到一个SIZE结构中，然后，我们使用范围来呼叫SetViewportExtEx，只是要将y范围取反。

WHATSIZE程序

Windows的小历史：第一篇如何写作Windows程序的介绍文章出现在《Microsoft Systems Journal》1986年12月号上。在那篇文章中，范例程序叫做WSZ（「what size：什么尺寸」），它以像素、英寸和毫米为单位显示了显示区域的大小。那个程序的更简易版本是WHATSIZE，如程序5-6所示。程序显示了以五种度量映像方式显示的窗口显示区域的大小。

程序5-6　WHATSIZE

WHATSIZE.C

```
/*------------------------------------------------------------
WHATSIZE.C -- What Size is the Window?
(c) Charles Petzold, 1998
------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("WhatSize") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName= szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("What Size is the Window?"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
void Show (HWND hwnd, HDC hdc, int xText, int yText, int iMapMode,
           TCHAR * szMapMode)
{
  TCHAR szBuffer [60] ;
  RECT rect ;

  SaveDC (hdc) ;
  SetMapMode (hdc, iMapMode) ;
  GetClientRect (hwnd, &rect) ;
  DPtoLP (hdc, (PPOINT) &rect, 2) ;

  RestoreDC (hdc, -1) ;
  TextOut ( hdc, xText, yText, szBuffer,
    wsprintf (szBuffer, TEXT ("%-20s %7d %7d %7d %7d"), szMapMode,
    rect.left, rect.right, rect.top, rect.bottom)) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static TCHAR szHeading [] =
    TEXT ("Mapping Mode Left Right Top Bottom") ;
  static TCHAR szUndLine [] =
    TEXT ("------------ ---- ----- --- ------") ;
  static int cxChar, cyChar ;
  HDC hdc ;
  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_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
    SetMapMode (hdc, MM_ANISOTROPIC) ;
```

```
    SetWindowExtEx (hdc, 1, 1, NULL) ;
    SetViewportExtEx (hdc, cxChar, cyChar, NULL) ;

    TextOut (hdc, 1, 1, szHeading, lstrlen (szHeading)) ;
    TextOut (hdc, 1, 2, szUndLine, lstrlen (szUndLine)) ;

    Show (hwnd, hdc, 1, 3, MM_TEXT, TEXT ("TEXT (pixels)")) ;
    Show (hwnd, hdc, 1, 4, MM_LOMETRIC, TEXT ("LOMETRIC (.1 mm)")) ;
    Show (hwnd, hdc, 1, 5, MM_HIMETRIC, TEXT ("HIMETRIC (.01 mm)")) ;
    Show (hwnd, hdc, 1, 6, MM_LOENGLISH, TEXT ("LOENGLISH (.01 in)")) ;
    Show (hwnd, hdc, 1, 7, MM_HIENGLISH,TEXT ("HIENGLISH (.001 in)")) ;
    Show (hwnd, hdc, 1, 8, MM_TWIPS, EXT ("TWIPS (1/1440 in)")) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

为了便于用TextOut函数显示信息，WHATSIZE使用了一种固定间距的字体。下面一条简单的叙述就可以切换为固定间距的字体（在Windows 3.0中它是优先使用的）：

SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

有两个同样的函数用于选取画笔和画刷。像前面提到的，WHATSIZE也使用MM_ANISTROPIC映像方式将逻辑单位设定为字符大小。

当WHATSIZE需要取得六种映像方式之一的显示区域的大小时，它保存目前的设备内容，设定一种新的映像方式，取得显示区域坐标，将它们转换为逻辑坐标，然后在显示信息之前，恢复原映像方式。底下这些程序代码在WHATSIZE的Show函数里：

```
SaveDC (hdc) ;
SetMapMode (hdc, iMapMode) ;
GetClientRect (hwnd, &rect) ;
DptoLP (hdc, (PPOINT) &rect, 2) ;
RestoreDC (hdc, -1) ;
```

图5-19显示了WHATSIZE的典型输出。

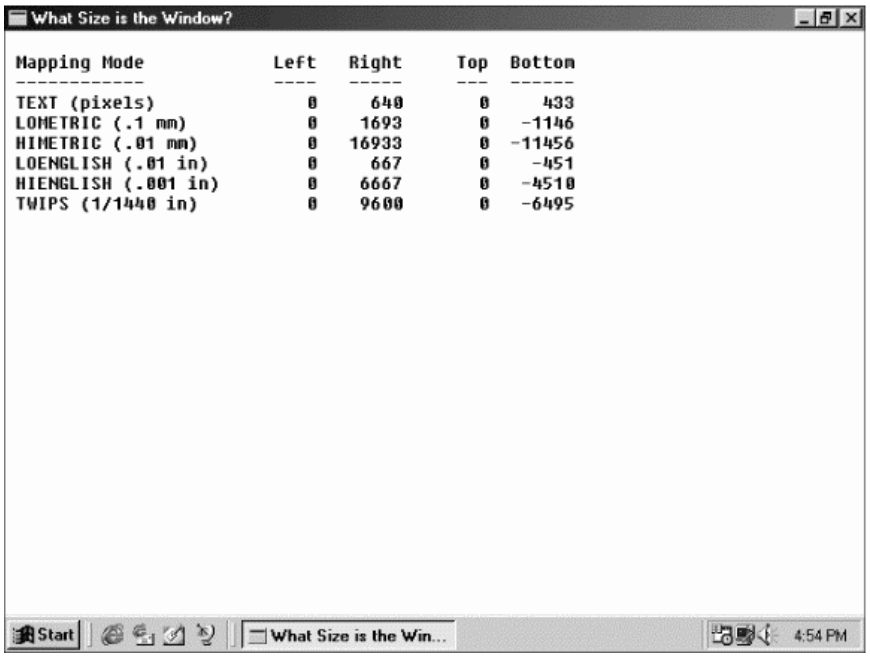


图5-19 典型的WHATSIZE显示

矩形、区域和剪裁

Windows包含了几种使用RECT（矩形）结构和「区域」的绘图函数。区域就是屏幕上的一块地方，它是矩形、多边形和椭圆的组合。

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型态的结构，它包含有4个字段：left、top、right和bottom。这个结构中的坐标被当作逻辑坐标。

FillRect用指定画刷来填入矩形（直到但不包含right和bottom坐标），该函数不需要先将画刷选进设备内容。

FrameRect使用画刷画矩形框，但是不填入矩形。使用画刷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），其边线都是用目前画笔绘制的。FrameRect允许使用者画一个不一定为纯色的矩形框。该边界框为一个逻辑单位元宽。如果逻辑单位大于设备单位，则边界框将会为2个像素宽或者更宽。

InvertRect将矩形中所有像素翻转，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，使您可以更容易、更清楚地操作RECT结构。例如，要将RECT结构的四个字段设定为特定值，通常使用如下的程序段：

```
rect.left = xLeft ;
rect.top = xTop ;
rect.right = xRight ;
rect.bottom = xBottom ;
```

但是，通过呼叫SetRect函数，只需要一道叙述就可以得到同样的结果：

SetRect (&rect, xLeft, yTop, xRight, yBottom) ;

在您想要做以下事情之一时，可以很方便地选用其它8个函数：

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) ;

矩形各字段设定为0：

SetRectEmpty (&rect) ;

将矩形复制给另一个矩形：

CopyRect (&DestRect, &SrcRect) ;

取得两个矩形的交集：

IntersectRect (&DestRect, &SrcRect1, &SrcRect2) ;

取得两个矩形的联集：

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) ;

大多数情况下，与这些函数相同作用的程序代码很简单。例如，您可以用下列叙述来替代CopyRect函数呼叫：

DestRect = SrcRect ;

随机矩形

在图形系统中，有这么一个「永远」有人执行的有趣程序，它简单地使用随机的大小和色彩绘制一系列矩形。您可以在Windows中建立一个这样的程序，但是它并不像乍看起来那样容易编写。我希望您能认识到，您不能简单地在WM_PAINT消息中使用一个while(TRUE)循环。当然，它能够执行，但是程序将停止对其他消息的处理，同时，这个程序不能中止或者最小化。

一种可以接受的方法是设定一个Windows定时器，给窗口程序发送WM_TIMER消息（我将在第八章中讨论定时器）。对于每条WM_TIMER消息，您使用GetDC取得一个设备内容，画一个随机的矩形，然后用ReleaseDC释放设备内容。但是这样又降低了程序的趣味性，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，而这又依赖于系统时钟的分辨率。

在Windows中一定有很多「闲置时间」，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者鼠标输入。我们能否在闲置时间内获得控制，绘制矩形，并且只在有消息加入程序的消息队列之后才释放控制呢？这就是PeekMessage函数的目的之一。下面是PeekMessage呼叫的一个例子：

PeekMessage (&msg, NULL, 0, 0, PM_REMOVE) ;

前面的四个参数（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、四个参数设定为NULL或0时，表明我们想让PeekMessage传回程序中所有窗口的所有消息。如果要将消息从消息队列中删除，则将PeekMessage的最后一个参数设定为PM_REMOVE。如果您不希望删除消息，那么您可以将这个参数设定为PM_NOREMOVE。这就是为什么Peek_Message是「偷看」而不是「取得」的原因，它使得程序可以检查程序的队列中的下一个消息，而不实际删除它。

GetMessage不将控制传回给程序，直到从程序的消息队列中取得消息，但是PeekMessage总是立刻传回，而不论一个消息是否出现。当消息队列中有一个消息时，PeekMessage的传回值为TRUE（非0），并且将按通常方式处理消息。当队列中没有消息时，PeekMessage传回FALSE (0)。

这使得我们可以改写普通的消息循环。我们可以将如下所示的循环：

```
while (GetMessage (&msg, NULL, 0, 0))
{
  TranslateMessage (&msg) ;
  DispatchMessage (&msg) ;
}
return msg.wParam ;
```

替换为下面的循环：

```
while (TRUE)
{
  if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    if (msg.message == WM_QUIT)
      break ;
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  else
  {
    // 完成某些工作的其它行程序
  }
}
return msg.wParam ;
```

注意，WM_QUIT消息被另外挑出来检查。在普通的消息循环中您不必这么作，因为如果GetMessage接收到一个WM_QUIT消息，它将传回0，但是PeekMessage用它的传回值来指示是否得到一个消息，所以需要对WM_QUIT进行检查。

如果PeekMessage的传回值为TRUE，则消息按通常方式进行处理。如果传回值为FALSE，则在将控制传回给Windows之前，还可以作一点工作（如显示另一个随机矩形）。

（尽管Windows文件上说，您不能用PeekMessage从消息队列中删除WM_PAINT消息，但是这并不是什么大不了的问题。毕竟，GetMessage并不从消息队列中删除WM_PAINT消息。从队列中删除WM_PAINT消息的唯一方法是令窗口显示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。如果您在使用PeekMessage从队列中取出WM_PAINT消息后，同平常一样处理它，那么就不会有问题了。所不能作的是使用如下所示的程序代码来清除消息队列中的所有消息：

while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE)) ;

这行叙述从消息队列中删除WM_PAINT之外的所有消息。如果队列中有一个WM_PAINT消息，程序就会永远地陷在while循环中。）

PeekMessage在Windows的早期版本中比在Windows 98中要重要得多。这是因为Windows的16位版本使用的是非优先权式的多任务（我将在第二十章中讨论这一点）。Windows的Terminal程序在从通讯端口接收输入后，使用一个PeekMessage循环。打印管理器程序使用这个技术来进行打印，其它的Windows打印应用程序通常都会使用一个PeekMessage循环。在Windows 98优先权式的多任务环境下，程序可以建立多个线程，我们将第二十章看到这一点。

不管怎样，有了PeekMessage函数，我们就可以编写一个不停地显示随机矩形的程序。这个RANDRECT如程序5-7中所示。

程序5-7 RANDRECT

RANDRECT.C

```
/*------------------------------------------------------------------------
RANDRECT.C -- Displays Random Rectangles
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <stdlib.h>// for the rand function

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
void DrawRectangle (HWND) ;

int cxClient, cyClient ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
```

```
{
  static TCHAR szAppName[] = TEXT ("RandRect") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Random Rectangles"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  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 ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  switch (iMsg)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
void DrawRectangle (HWND hwnd)
{
  HBRUSH hBrush ;
  HDC hdc ;
  RECT rect ;

  if (cxClient == 0 || cyClient == 0)
```

```
    return ;
  SetRect (&rect, rand () % cxClient, rand () % cyClient,
    rand () % cxClient, rand () % cyClient) ;
  hBrush = CreateSolidBrush (
    RGB (rand () % 256, rand () % 256, rand () % 256)) ;
  hdc = GetDC (hwnd) ;
  FillRect (hdc, &rect, hBrush) ;
  ReleaseDC (hwnd, hdc) ;
  DeleteObject (hBrush) ;
}
```

这个程序在现在的计算机上执行得非常快，看起来都不像是一系列随机矩形了。程序使用我在上面讨论过的SetRect和FillRect函数，根据由C的rand函数得到的随机数决定矩形坐标和实心画刷的色彩。我将在第二十章中提供这个程序的多线程版本。

建立和绘制剪裁区域

剪裁区域是对显示器上一个范围的描述，这个范围是矩形、多边形和椭圆的组合。剪裁区域可以用于绘制和剪裁，通过将剪裁区域选进设备内容，就可以用剪裁区域来进行剪裁（就是说，将可以绘图的范围限制为显示区域的一部分）。与画笔、画刷和位图一样，剪裁区域是GDI对象，您应该呼叫DeleteObject来删除您所建立的剪裁区域。

当您建立一个剪裁区域时，Windows传回一个该剪裁区域的句柄，型态为HRGN。最简单的剪裁区域是矩形，有两种建立矩形的方法：

hRgn = CreateRectRgn (xLeft, yTop, xRight, yBottom) ;

或者

hRgn = CreateRectRgnIndirect (&rect) ;

您也可以建立椭圆剪裁区域：

hRgn = CreateEllipticRgn (xLeft, yTop, xRight, yBottom) ;

或者

hRgn = CreateEllipticRgnIndirect (&rect) ;

CreateRoundRectRgn建立圆角的矩形剪裁区域。

建立多边形剪裁区域的函数类似于Polygon函数：

hRgn = CreatePolygonRgn (&point, iCount, iPolyFillMode) ;

point参数是一个POINT型态的结构数组，iCount是点的数目，iPolyFillMode是ALTERNATE或者WINDING。您还可以用CreatePolyPolygonRgn来建立多个多边形剪裁区域。

那么，您会问，剪裁区域究竟有什么特别之处？下面这个函数才真正显示出了剪裁区域的作用：

iRgnType = CombineRgn (hDestRgn, hSrcRgn1, hSrcRgn2, iCombine) ;

这一函数将两个剪裁区域（hSrcRgn1和hSrcRgn2）组合起来并用句柄hDestRgn指向组合成的剪裁区域。这三个剪裁区域句柄都必须是有效的，但是hDestRgn原来所指向的剪裁区域被破坏掉了（当您使用这个函数时，您可能要让hDestRgn在初始时指向一个小的矩形剪裁区域）。

iCombine参数说明hSrcRgn1和hSrcRgn2如何组合，见表5-9。

表5-9

iCombine值	新剪裁区域

RGN_AND	两个剪裁区域的公共部分
RGN_OR	两个剪裁区域的全部
RGN_XOR	两个剪裁区域的全部除去公共部分
RGN_DIFF	hSrcRgn1不在hSrcRgn2中的部分
RGN_COPY	hSrcRgn1的全部（忽略hSrcRgn2）

从CombineRgn传回的iRgnType值是下列之一：NULLREGION，表示得到一个空剪裁区域；SIMPLEREGION，表示得到一个简单的矩形、椭圆或者多边形；COMPLEXREGION，表示多个矩形、椭圆或多边形的组合；ERROR，表示出错了。

剪裁区域的句柄可以用于四个绘图函数：

```
FillRgn (hdc, hRgn, hBrush) ;
FrameRgn (hdc, hRgn, hBrush, xFrame, yFrame) ;
InvertRgn (hdc, hRgn) ;
PaintRgn (hdc, hRgn) ;
```

FillRgn、FrameRgn和InvertRgn类似于FillRect、FrameRect和InvertRect。FrameRgn的xFrame和yFrame参数是画在区域周围的边框的宽度和高度。PaintRgn函数用设备内容中目前画刷填入所指定的区域。所有这些函数都假定区域是用逻辑坐标定义的。

在您用完一个区域后，可以像删除其它GDI对象那样删除它：

DeleteObject (hRgn) ;

矩形与区域的剪裁

区域也在剪裁中扮演了一个角色。InvalidateRect函数使显示的一个矩形区域失效，并产生一个WM_PAINT消息。例如，您可以使用InvalidateRect函数来清除显示区域并产生一个WM_PAINT消息：

InvalidateRect (hwnd, NULL, TRUE) ;

您可以通过呼叫GetUpdateRect来取得失效矩形的坐标，并且可以使用ValidateRect函数使显示区域的矩形有效。当您接收到一个WM_PAINT消息时，无效矩形的坐标可以从PAINTSTRUCT结构中得到，该结构是用BeginPaint函数填入的。这个无效矩形还定义了一个「剪裁区域」，您不能在剪裁区域外绘图。

Windows有两个作用于剪裁区域而不是矩形的函数，它们类似于InvalidateRect和ValidateRect：

InvalidateRgn (hwnd, hRgn, bErase) ;

和

ValidateRgn (hwnd, hRgn) ;

当您接收到一个由无效区域引起的WM_PAINT消息时，剪裁区域不一定是矩形。

您可以使用以下两个函数之一：

SelectObject (hdc, hRgn) ;

或

SelectClipRgn (hdc, hRgn) ;

通过将一个剪裁区域选进设备内容来建立自己的剪裁区域，这个剪裁区域使用设备坐标。

GDI为剪裁区域建立一份副本，所以在将它选进设备内容之后，使用者可以删除它。Windows

还提供了几个对剪裁区域进行操作的函数，如ExcludeClipRect用于将一个矩形从剪裁区域里排除掉，IntersectClipRect用于建立一个新的剪裁区域，它是前一个剪裁区域与一个矩形的交，OffsetClipRgn用于将剪裁区域移动到显示区域的另一部分。

CLOVER程序

CLOVER程序用四个椭圆组成一个剪裁区域，将这个剪裁区域选进设备内容中，然后画出从窗口显示区域的中心出发的一系列直线，这些直线只出现在剪裁区域所限定的范围，结果显示如图5-20所示。

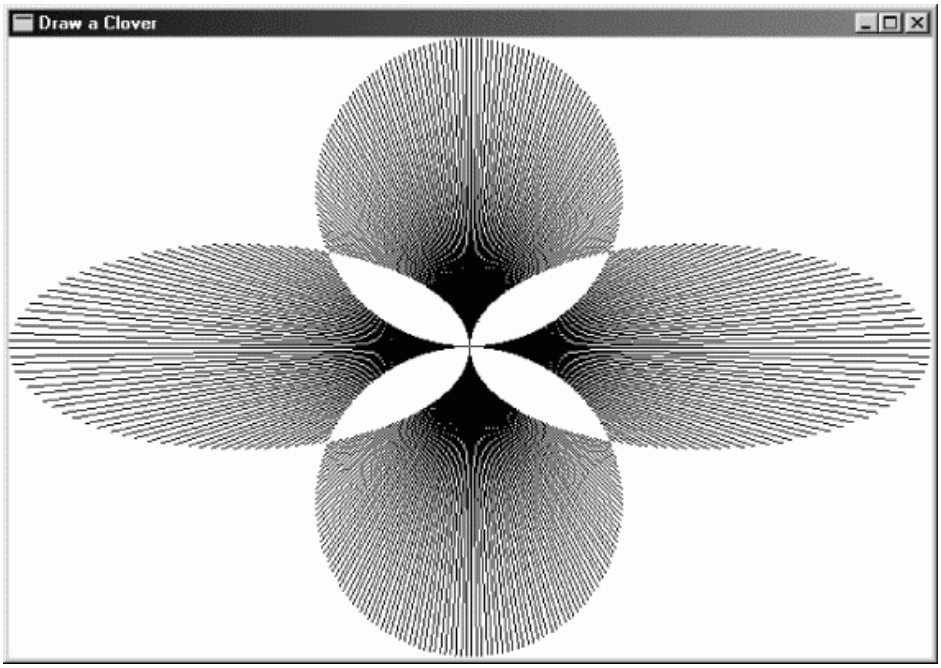


图5-20 CLOVER利用复杂的剪裁区域画出的图像

要用常规的方法画出这个图形，就必须根据椭圆的边线公式计算出每条直线的端点。利用复杂的剪裁区域，可以直接画出这些线条，而让Windows确定其端点。CLOVER如程序5-8所示。

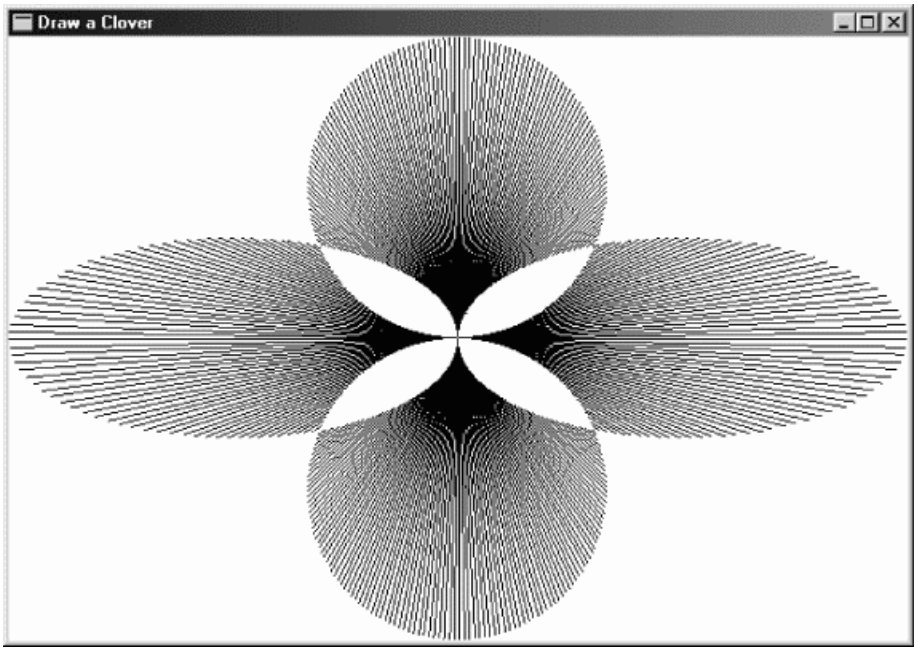
程序5-8 CLOVER

CLOVER.C

```
/*------------------------------------------------------------------------
CLOVER.C -- Clover Drawing Program Using Regions
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#define TWO_PI (2.0 * 3.14159)
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Clover") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Draw a Clover"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  static HRGN hRgnClip ;
  static int cxClient, cyClient ;
  double fAngle, fRadius ;
  HCURSOR hCursor ;
  HDC hdc ;
  HRGN hRgnTemp[6] ;
  int i ;
  PAINTSTRUCT ps ;
  switch (iMsg)
  {
  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) ;

    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
    SelectClipRgn (hdc, hRgnClip) ;
    fRadius = _hypot (cxClient / 2.0, cyClient / 2.0) ;
    for (fAngle = 0.0 ; fAngle < TWO_PI ; fAngle += TWO_PI / 360)
    {
      MoveToEx (hdc, 0, 0, NULL) ;
      LineTo (hdc, (int) ( fRadius * cos (fAngle) + 0.5),
        (int) (-fRadius * sin (fAngle) + 0.5)) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    DeleteObject (hRgnClip) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```



由于剪裁区域总是使用设备坐标，CLOVER程序必须在每次接收到WM_SIZE消息时重新建立剪裁区域。几年前，这可能需要几秒钟。现在的快速机器在一瞬间就可以画出来。

CLOVER从建立四个椭圆剪裁区域开始，这四个椭圆存放在hRgnTemp数组的头四个元素中，然后建立三个「空」剪裁区域：

```
hRgnTemp [4]= CreateRectRgn (0, 0, 1, 1) ;
hRgnTemp [5]= CreateRectRgn (0, 0, 1, 1) ;
hRgnClip = CreateRectRgn (0, 0, 1, 1) ;
```

显示区域左右的两个椭圆区域组合起来：

CombineRgn (hRgnTemp [4], hRgnTemp [0], hRgnTemp [1], RGN_OR) ;

同样，显示区域上下两个椭圆区域组合起来：

CombineRgn (hRgnTemp [5], hRgnTemp [2], hRgnTemp [3], RGN_OR) ;

最后，两个组合后的区域再组合到hRgnClip中：

CombineRgn (hRgnClip, hRgnTemp [4], hRgnTemp [5], RGN_XOR) ;

RGN_XOR标识符用于从结果区域中排除重迭部分。最后，删除6个临时区域：

for (i = 0 ; i < 6 ; i++)

DeleteObject (hRgnTemp [i]) ;

与画出的图形比起来，WM_PAINT的处理很简单。视端口原点设定为显示区域的中心（使画直线更容易一些），在WM_SIZE消息处理期间建立的区域选择为设备内容的剪裁区域：

SetViewportOrg (hdc, xClient / 2, yClient / 2) ;

SelectClipRgn (hdc, hRgnClip) ;

现在，剩下的就是画直线了，共360条，每隔一度画一条。每条线的长度为变量fRadius，这是从中心到显示区域的角落的距离：

```
fRadius = hypot (xClient / 2.0, yClient / 2.0) ;
for (fAngle = 0.0 ; fAngle < TWO_PI ; fAngle += TWO_PI / 360)
{
  MoveToEx (hdc, 0, 0, NULL) ;
  LineTo (hdc, (int) ( fRadius * cos (fAngle) + 0.5),
    (int) (-fRadius * sin (fAngle) + 0.5)) ;
}
```

在处理WM_DESTROY消息时，删除该剪裁区域：

DeleteObject (hRgnClip) ;

这不是本书关于图形程序设计的最后内容。第十三章讨论打印，第十四章和十五章讨论位图，第十七章讨论文字和字体，第十八章讨论MetaFile。

第六章 键盘

在Microsoft Windows 98中，键盘和鼠标是两个标准的使用者输入来源，在一些连贯操作中常产生互补作用。当然，鼠标在今天的应用程序中比十年前使用得更为广泛。甚至在一些应用程序中，我们更习惯于使用鼠标，例如在游戏、画图程序、音乐程序以及Web浏览器等程序中就是这样。然而，我们可以不使用鼠标，但绝对不能从一般的PC中把键盘拆掉。

相对于个人计算机的其它组件，键盘有非常久远的历史，它起源于1874年的第一台Remington打字机。早期的计算机程序员用键盘在Hollerith卡片上打孔，后来在终端机上用键盘直接与大型主机沟通。PC上的键盘在某些方面进行了扩充，加上了功能键、光标移动键和单独的数字键盘，但它们的输入原理基本相同。

键盘基础

您大概已经猜到Windows程序是如何获得键盘输入的：键盘输入以消息的形式传递给程序的窗口消息处理程序。实际上，第一次学习消息时，键盘事件就是一个消息如何将不同型态信息传递给应用程序的显例。

Windows用八种不同的消息来传递不同的键盘事件。这好像太多了，但是（就像我们所看到的一样）程序可以忽略其中至少一半的消息而不会有任何问题。并且，在大多数情况下，这些消息中包含的键盘信息会多于程序所需要的。处理键盘的部分工作就是识别出哪些消息是重要的，哪些是不重要的。

忽略键盘

虽然键盘是Windows程序中使用者输入的主要来源，但是程序不必对它接收的所有消息都作出响应。Windows本身也能处理许多键盘功能。

例如，您可以忽略那些属于系统功能的按键，它们通常用到Alt键。程序不必监视这些按键，因为Windows会将按键的作用通知程序（当然，如果程序想这么做，它也能监视这些按键）。虽然呼叫程序菜单的按键将通过窗口的窗口消息处理程序，但通常内定的处理方式是将按键传递给DefWindowProc。最终，窗口消息处理程序将获得一个消息，表示一个菜单项被选择了。通常，这是所有窗口消息处理程序需要知道的（在第十章将介绍菜单）。

有些Windows程序使用「键盘快捷键」来启动通用菜单项。快捷键通常是功能键或字母同Ctrl键的组合（例如，Ctrl-S用于保存文件）。这些键盘快捷键与程序菜单一起在程序的资源描述文件中定义（我们可以在第十章看到）。Windows将这些键盘快捷键转换为菜单命令消息，您不必自己去进行转换。

对话框也有键盘接口，但是当对话框处于活动状态时，应用程序通常不必监视键盘。键盘接口由Windows处理，Windows把关于按键作用的消息发送给程序。对话框可以包含用于输入文字的编辑控件。它们一般是小方框，使用者可以在框中键入字符串。Windows处理所有编辑控件逻辑，并在输入完毕后，将编辑控件的最终内容传送给程序。关于对话框的详细信息，请参见第十一章。

编辑控件不必局限于单独一行，而且也不限于只在对话框中。一个在程序主窗口内的多行编辑

控件就能够作为一个简单的文字编辑器了（参见第九、十、十一和十三章的POPPAD程序）。Windows甚至有一个Rich Text文字编辑控件，允许您编辑和显示格式化的文字（请参见/Platform SDK/User Interface Services/Controls/Rich Edit Controls）。

您将会发现，在开发Windows程序时，可以使用处理键盘和鼠标输入的子窗口控件来将较高层的信息传递回父窗口。只要这样的控件用得够多，您就不会因处理键盘消息而烦恼了。

谁获得了焦点

与所有的个人计算机硬件一样，键盘必须由在Windows下执行的所有应用程序共享。有些应用程序可能有多个窗口，键盘必须由该应用程序内的所有窗口共享。

回想一下，程序用来从消息队列中检索消息的MSG结构包括hwnd字段。此字段指出接收消息的窗口控件码。消息循环中的DispatchMessage函数向窗口消息处理程序发送该消息，此窗口消息处理程序与需要消息的窗口相联系。在按下键盘上的键时，只有一个窗口消息处理程序接收键盘消息，并且此消息包括接收消息的窗口控件码。

接收特定键盘事件的窗口具有输入焦点。输入焦点的概念与活动窗口的概念很相近。有输入焦点的窗口是活动窗口或活动窗口的衍生窗口（活动窗口的子窗口，或者活动窗口子窗口的子窗口等等）。

通常很容易辨别活动窗口。它通常是顶层窗口 – 也就是说，它的父窗口句柄是NULL。如果活动窗口有标题栏，Windows将突出显示标题栏。如果活动窗口具有对话框架（对话框中很常见的格式）而不是标题栏，Windows将突出显示框架。如果活动窗口目前是最小化的，Windows将在工作列中突出显示该项，其显示就像一个按下的按钮。

如果活动窗口有子窗口，那么有输入焦点的窗口既可以是活动窗口也可以是其子窗口。最常见的子窗口有类似以下控件：出现在对话框中的下压按钮、单选钮、复选框、滚动条、编辑方块和清单方块。子窗口不能自己成为活动窗口。只有当它是活动窗口的衍生窗口时，子窗口才能有输入焦点。子窗口控件一般通过显示一个闪烁的插入符号或虚线来表示它具有输入焦点。

有时输入焦点不在任何窗口中。这种情况发生在所有程序都是最小化的时候。这时，Windows将继续向活动窗口发送键盘消息，但是这些消息与发送给非最小化的活动窗口的键盘消息有不同的形式。

窗口消息处理程序通过拦截WM_SETFOCUS和WM_KILLFOCUS消息来判定它的窗口何时拥有输入焦点。WM_SETFOCUS指示窗口正在得到输入焦点，WM_KILLFOCUS表示窗口正在失去输入焦点。我将在本章的后面详细说明这些消息。

队列和同步

当使用者按下并释放键盘上的键时，Windows和键盘驱动程序将硬件扫描码转换为格式消息。然而，这些消息并不保存在消息队列中。实际上，Windows在所谓的「系统消息队列」中保存这些消息。系统消息队列是独立的消息队列，它由Windows维护，用于初步保存使用者从键盘和鼠标输入的信息。只有当Windows应用程序处理完前一个使用者输入消息时，Windows才会从系统消息队列中取出下一个消息，并将其放入应用程序的消息队列中。

此过程分为两步：首先在系统消息队列中保存消息，然后将它们放入应用程序的消息队列，其原因是需要同步。就像我们刚才所学的，假定接收键盘输入的窗口就是有输入焦点的窗口。使用者的输入速度可能比应用程序处理按键的速度快，并且特定的按键可能会使焦点从一个窗口切换到另一个窗口，后来的按键就输入到了另一个窗口。但如果后来的按键已经记下了目标窗口的地址，并放入了应用程序消息队列，那么后来的按键就不能输入到另一个窗口。

按键和字符

应用程序从Windows接收的关于键盘事件的消息可以分为按键和字符两类,这与您看待键盘的两种方式一致。

首先，您可以将键盘看作是键的集合。键盘只有唯一的A键，按下该键是一次按键，释放该键也是一次按键。但是键盘也是能产生可显示字符或控制字符的输入设备。根据Ctrl、 Shift和Caps Lock键的状态，A键能产生几个字符。通常情况下，此字符为小写a。如果按下Shift键或者打开了Caps Lock，则该字符就变成大写A。如果按下了Ctrl，则该字符为Ctrl-A（它在ASCII中有意义，但在Windows中可能是某事件的键盘快捷键）。在一些键盘上，A按键之前可能有「死字符键(dead-character key)」或者Shift、Ctrl或者Alt的不同组合，这些组合可以产生带有音调标记的小写或者大写，例如，à、á、狻20、或拧?/p>

对产生可显示字符的按键组合，Windows不仅给程序发送按键消息，而且还发送字符消息。有些键不产生字符，这些键包括shift键、功能键、光标移动键和特殊字符键如Insert和Delete。对于这些键，Windows只产生按键消息。

按键消息

当您按下一个键时，Windows把WM_KEYDOWN或者WM_SYSKEYDOWN消息放入有输入焦点的窗口的消息队列；当您释放一个键时，Windows把WM_KEYUP或者WM_SYSKEYUP消息放入消息队列中。

表6-1

	键按下	键释放
非系统键	WM_KEYDOWN	WM_KEYUP
系统键	WM_SYSKEYDOWN	WM_SYSKEYUP

通常「down（按下）」和「up（放开）」消息是成对出现的。不过，如果您按住一个键使得自动重复功能生效，那么当该键最后被释放时，Windows会给窗口消息处理程序发送一系列WM_KEYDOWN（或者WM_SYSKEYDOWN）消息和一个WM_KEYUP（或者WM_SYSKEYUP）消息。像所有放入队列的消息一样，按键消息也有时间信息。通过呼叫GetMessageTime，您可以获得按下或者释放键的相对时间。

系统按键与非系统按键

WM_SYSKEYDOWN和WM_SYSKEYUP中的「SYS」代表「系统」，它表示该按键对Windows比对Windows应用程序更加重要。WM_SYSKEYDOWN和WM_SYSKEYUP消息经常由与Alt相组合的按键产生，这些按键启动程序菜单或者系统菜单上的选项，或者用于切换活动窗口等系统功能（Alt-Tab或者Alt-Esc），也可以用作系统菜单快捷键（Alt键与一个功能键相结合，例如Alt-F4用于关闭应用程序）。程序通常忽略WM_SYSKEYUP和WM_SYSKEYDOWN消息，并将它们传送到DefWindowProc。由于Windows要处理所有Alt键的功能，所以您无需拦截这些消息。您的窗口消息处理程序将最后收到关于这些按键结果（如菜单选择）的其它消息。如果您想在自己的窗口消息处理程序中加上拦截系统按键的程序代码（如本章后面的KEYVIEW1和KEYVIEW2程序所作的那样），那么在处理这些消息之后再传送到DefWindowProc，Windows就仍然可以将它们用于通常的目的。

但是，请再考虑一下，几乎所有会影响使用者程序窗口的消息都会先通过使用者窗口消息处理

程序。只有使用者把消息传送到DefWindowProc，Windows才会对消息进行处理。例如，如果您将下面几行叙述：

```
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
case WM_SYSCHAR:
  return 0 ;
```

加入到一个窗口消息处理程序中，那么当您的程序主窗口拥有输入焦点时，就可以有效地阻止所有Alt键操作（我将在本章的后面讨论WM_SYSCHAR），其中包括Alt-Tab、Alt-Esc以及菜单操作。虽然我怀疑您会这么做，但是，我相信您会感到窗口消息处理程序的强大功能。

WM_KEYDOWN和WM_KEYUP消息通常是在按下或者释放不带Alt键的键时产生的，您的程序可以使用或者忽略这些消息，Windows本身并不处理这些消息。

对所有四类按键消息，wParam是虚拟键代码，表示按下或释放的键，而lParam则包含属于按键的其它数据。

虚拟键码

虚拟键码保存在WM_KEYDOWN、WM_KEYUP、WM_SYSKEYDOWN和WM_SYSKEYUP消息的wParam参数中。此代码标识按下或释放的键。

哈，又是「虚拟」，您喜欢这个词吗？虚拟指的是假定存在于思想中而不是现实世界中的一些事物，也只有熟练使用DOS汇编语言编写应用程序的程序写作者才有可能指出，为什么对Windows键盘处理如此基本的键码是虚拟的而不是真实的。

对于早期的程序写作者来说，真实的键码由实际键盘硬件产生。在Windows文件中将这些键码称为「扫描码(scan codes)」。在IBM兼容机种上，扫描码16是Q键，17是W键，18是E、19是R，20是T，21是Y等等。这时您会发现，扫描码是依据键盘的实际布局的。Windows开发者认为这些代码过于与设备相关了，于是他们试图通过定义所谓的虚拟键码，以便经由与设备无关的方式处理键盘。其中一些虚拟键码不能在IBM兼容机种上产生，但可能会在其它制造商生产的键盘中找到，或者在未来的键盘上找到。

您使用的大多数虚拟键码的名称在WINUSER.H表头文件中都定义为以VK_开头。表6-2列出了这些名称和数值（十进制和十六进制），以及与虚拟键相对应的IBM兼容机种键盘上的键。下表也标出了Windows执行时是否需要这些键。下表还按数字顺序列出了虚拟键码。

前四个虚拟键码中有三个指的是鼠标键：

表6-2

十进制	十六进制	WINUSER.H 标识符	必需?	IBM兼容键盘
1	01	VK_LBUTTON		鼠标左键
2	02	VK_RBUTTON		鼠标右键
3	03	VK_CANCEL	√	Ctrl-Break
4	04	VK_MBUTTON		鼠标中键

您永远都不会从键盘消息中获得这些鼠标键代码。在下一章可以看到，我们能够从鼠标消息中获得它们。VK_CANCEL代码是一个虚拟键码，它包括同时按下两个键(Ctrl-Break)。Windows应用程序通常不使用此键。

表6-3中的键--Backspace、Tab、Enter、Escape和Spacebar – 通常用于Windows程序。不过，Windows一般用字符消息（而不是键盘消息）来处理这些键。

表6-3

十进制	十六进制	WINUSER.H 标识符	必需?	IBM兼容键盘
8	08	VK_BACK	√	Backspace
9	09	VK_TAB	√	Tab
12	0C	VK_CLEAR		Num Lock关闭时的数字键盘5
13	0D	VK_RETURN	√	Enter （或者另一个）
16	10	VK_SHIFT	√	Shift （或者另一个）
17	11	VK_CONTROL	√	Ctrl （或者另一个）
18	12	VK_MENU	√	Alt （或者另一个）
19	13	VK_PAUSE		Pause
20	14	VK_CAPITAL	√	Caps Lock
27	1B	VK_ESCAPE	√	Esc
32	20	VK_SPACE	√	Spacebar

另外，Windows程序通常不需要监视Shift、Ctrl或Alt键的状态。

表6-4列出的前八个码可能是与VK_INSERT和VK_DELETE一起最常用的虚拟键码：

表6-4

十进制	十六进制	WINUSER.H 标识符	必需?	IBM兼容键盘
33	21	VK_PRIOR	√	Page Up
34	22	VK_NEXT	√	Page Down
35	23	VK_END	√	End
36	24	VK_HOME	√	Home
37	25	VK_LEFT	√	左箭头
38	26	VK_UP	√	上箭头
39	27	VK_RIGHT	√	右箭头
40	28	VK_DOWN	√	下箭头
41	29	VK_SELECT		
42	2A	VK_PRINT		
43	2B	VK_EXECUTE		
44	2C	VK_SNAPSHOT		Print Screen
45	2D	VK_INSERT	√	Insert
46	2E	VK_DELETE	√	Delete
47	2F	VK_HELP		

注意，许多名称（例如VK_PRIOR和VK_NEXT）都与键上的标志不同，而且也与滚动条中的标识符不统一。Print Screen键在平时都被Windows应用程序所忽略。Windows本身响应此键时会将视讯显示的位图影本存放到剪贴板中。假使有键盘提供了VK_SELECT、VK_PRINT、VK_EXECUTE和VK_HELP，大概也没几个人看过那样的键盘。

Windows也包括在主键盘上的字母和数字键的虚拟键码（数字键盘将单独处理）。

表6-5

十进制	十六进制	WINUSER.H 标识符	必需?	IBM兼容键盘
48-57	30-39	无	√	主键盘上的0到9
65-90	41-5A	无	√	A到Z

注意，数字和字母的虚拟键码是ASCII码。Windows程序几乎从不使用这些虚拟键码；实际上，程序使用的是ASCII码字符的字符消息。

表6-6所示的代码是由Microsoft Natural Keyboard及其兼容键盘产生的：

表6-6

十进制	十六进制	WINUSER.H 标识符	必需?	IBM兼容键盘
91	5B	VK_LWIN		左Windows键
92	5C	VK_RWIN		右Windows键
93	5D	VK_APPS		Applications键

Windows用VK_LWIN和VK_RWIN键打开「开始」菜单或者（在以前的版本中）启动「工作管理员程序」。这两个都可以用于登录或注销Windows（只在Microsoft Windows NT中有效），或者登录或注销网络（在Windows for Applications中）。应用程序能够通过显示辅助信息或者当成快捷方式键看待来处理application键。

表6-7所示的代码用于数字键盘上的键（如果有的话）：

表6-7

十进制	十六进制	WINUSER.H 标识符	必需?	IBM兼容键盘
96-105	60-69	VK_NUMPAD0 到 VK_ NUMPAD9		NumLock 打开时数字键盘上的0到9
106	6A	VK_MULTIPLY		数字键盘上的*
107	6B	VK_ADD		数字键盘上的+
108	6C	VK_SEPARATOR		
109	6D	VK_SUBTRACT		数字键盘上的-
110	6E	VK_DECIMAL		数字键盘上的.
111	6F	VK_DIVIDE		数字键盘上的/

最后，虽然多数的键盘都有12个功能键，但Windows只需要10个，而位旗标却有24个。另外，程序通常用功能键作为键盘快捷键，这样，它们通常不处理表6-8所示的按键：

表6-8

十进制	十六进制	WINUSER.H标识符	必需?	IBM兼容键盘
112-121	70-79	VK_F1到VK_F10	√	功能键F1到F10
122-135	7A-87	VK_F11到VK_F24		功能键F11到F24

144	90	VK_NUMLOCK		Num Lock
145	91	VK_SCROLL		Scroll Lock

另外，还定义了一些其它虚拟键码，但它们只用于非标准键盘上的键，或者通常在大型主机终端机上使用的键。查看/ Platform SDK / User Interface Services / User Input / Virtual-Key Codes，可得到完整的列表。

lParam信息

在四个按键消息（WM_KEYDOWN、WM_KEYUP、WM_SYSKEYDOWN和WM_SYSKEYUP）中，wParam消息参数含有上面所讨论的虚拟键码，而lParam消息参数则含有对了解按键非常有用的其它信息。lParam的32位分为6个字段，如图6-1所示。

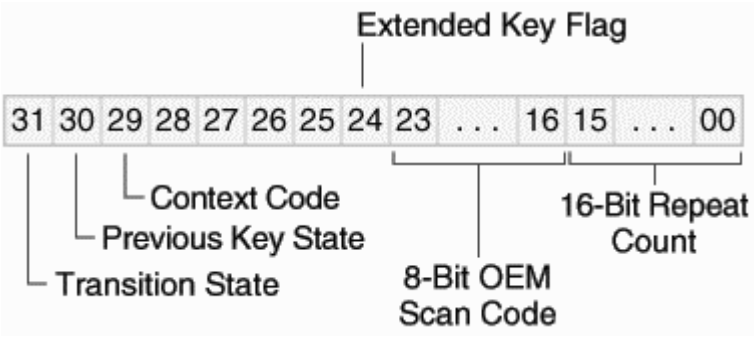


图6-1 lParam变量的6个按键消息字段

重复计数

重复计数是该消息所表示的按键次数，大多数情况下，重复计数设定为1。不过，如果按下一个键之后，您的窗口消息处理程序不够快，以致不能处理自动重复速率（您可以在「控制台」的「键盘」中进行设定）下的按键消息，Windows就把几个WM_KEYDOWN或者WM_SYSKEYDOWN消息组合到单个消息中，并相应地增加重复计数。WM_KEYUP或WM_SYSKEYUP消息的重复计数总是为1。

因为重复计数大于1指示按键速率大于您程序的处理能力，所以您也可能想在处理键盘消息时忽略重复计数。几乎每个人都有文书处理或执行电子表格时画面卷过头的经验，因为多余的按键堆满了键盘缓冲区，所以当程序用一些时间来处理每一次按键时，如果忽略您程序中的重复计数，就能够解决此问题。不过，有时可能也会用到重复计数，您应该尝试使用两种方法执行程序，并从中找出一种较好的方法。

OEM扫描码

OEM扫描码是由硬件（键盘）产生的代码。这对中古时代的汇编程序写作者来说应该很熟悉，它是从PC相容机种的ROM BIOS服务中所获得的值（OEM指的是PC的原始设备制造商（Original Equipment Manufacturer）及其与「IBM标准」同步的内容）。在此我们不需要更多的信息。除非需要依赖实际键盘布局的样貌，不然Windows程序可以忽略掉几乎所有的OEM扫描码信息，参见第二十二章的程序KBMIDI。

扩充键旗标

如果按键结果来自IBM增强键盘的附加键之一，那么扩充键旗标为1（IBM增强型键盘有101或102个键。功能键在键盘顶端，光标移动键从数字键盘中分离出来，但在数字键盘上还保留有光标移动键的功能）。对键盘右端的Alt和Ctrl键，以及不是数字键盘那部分的光标移动键（包括Insert和Delete键）、数字键盘上的斜线（／）和Enter键以及Num Lock键等，此旗标均被设定为1。Windows程序通常忽略扩充键旗标。

内容代码

右按键时，假如同时压下ALT键，那么内容代码为1。对WM_SYSKEYUP与WM_SYSKEYDOWN而言，此位总视为1；而对WM_SYSKEYUP与WM_KEYDOW消息而言，此位为0。除了两个之外：

如果活动窗口最小化了，则它没有输入焦点。这时候所有的按键都会产生WM_SYSKEYUP和WM_SYSKEYDOWN消息。如果Alt键未被按下，则内容代码字段被设定为0。Windows使用WM_SYSKEYUP和WM_SYSKEYDOWN消息，从而使最小化了的活动窗口不处理这些按键。

对于一些外国语文（非英文）键盘，有些字符是通过Shift、Ctrl或者Alt键与其它键相组合而产生的。这时内容代码为1，但是此消息并非系统按键消息。

键的先前状态

如果在此之前键是释放的，则键的先前状态为0，否则为1。对WM_KEYUP或者WM_SYSKEYUP消息，它总是设定为1；但是对WM_KEYDOWN或者WM_SYSKEYDOWN消息，此位可以为0，也可以为1。如果为1，则表示该键是自动重复功能所产生的第二个或者后续消息。

转换状态

如果键正被按下，则转换状态为0；如果键正被释放，则转换状态为1。对WM_KEYDOWN或者WM_SYSKEYDOWN消息，此字段为0；对WM_KEYUP或者WM_SYSKEYUP消息，此字段为1。

位移状态

在处理按键消息时，您可能需要知道是否按下了位移键（Shift、Ctrl和Alt）或开关键（Caps Lock、Num Lock和Scroll Lock）。通过呼叫GetKeyState函数，您就能获得此信息。例如：

iState = GetKeyState (VK_SHIFT) ;

如果按下了Shift，则iState值为负（即设定了最高位置位）。如果Caps Lock键打开，则从

iState = GetKeyState (VK_CAPITAL) ;

传回的值低位被设为1。此位与键盘上的小灯保持一致。

通常，您在使用GetKeyState时，会带有虚拟键码VK_SHIFT、VK_CONTROL和VK_MENU（在说明Alt键时呼叫）。使用GetKeyState时，您也可以用下面的标识符来确定按下的Shift、Ctrl或Alt键是左边的还是右边的：VK_LSHIFT、VK_RSHIFT、VK_LCONTROL、VK_RCONTROL、VK_LMENU、VK_RMENU。这些标识符只用于GetKeyState和GetAsyncKeyState（下面将详细说明）。

使用虚拟键码VK_LBUTTON、VK_RBUTTON和VK_MBUTTON，您也可以获得鼠标键的状态。不过，大多数需要监视鼠标键与按键相组合的Windows应用程序都使用其它方法来做到这一点 – 即在接收到鼠标消息时检查按键。实际上，位移状态信息包含在鼠标信息中，正如您在下一章中将看到的一样。

请注意GetKeyState的使用，它并非实时检查键盘状态，而只是检查直到目前为止正在处理的消息的键盘状态。多数情况下，这正符合您的要求。如果您需要确定使用者是否按下了Shift-Tab，请在处理Tab键的WM_KEYDOWN消息时呼叫GetKeyState，带有参数VK_SHIFT。如果GetKeyState传回的值为负，那么您就知道在按下Tab键之前按下了Shift键。并且，如果在您开始处理Tab键之前，已经释放了Shift键也没有关系。您知道，在按下Tab键的时候Shift键是按下的。

GetKeyState不会让您获得独立于普通键盘消息的键盘信息。例如，您或许想暂停窗口消息处理程序的处理，直到您按下F1功能键为止：

```
while (GetKeyState (VK_F1) >= 0) ;      // WRONG !!!
```

不要这么做！这将让程序当死（除非在执行此叙述之前早就从消息队列中接收到了F1的WM_KEYDOWN）。如果您确实需要知道目前某键的状态，那么您可以使用GetAsyncKeyState。

使用按键消息

如果程序能够获得每个按键的信息，这当然很理想，但是大多数Windows程序忽略了几乎所有的按键，而只处理部分的按键消息。WM_SYSKEYDOWN和WM_SYSKEYUP消息是由Windows系统函数使用的，您不必为此费心，就算您要处理WM_KEYDOWN消息，通常也可以忽略WM_KEYUP消息。

Windows程序通常为不产生字符的按键使用WM_KEYDOWN消息。虽然您可能认为借助按键消息和位移键状态信息能将按键消息转换为字符消息，但是不要这么做，因为您将遇到国际键盘间的差异所带来的问题。例如，如果您得到wParam等于0x33的WM_KEYDOWN消息，您就可以知道使用者按下了键3，到此为止一切正常。这时，如果用GetKeyState发现Shift键被按下，您就可能会认为使用者输入了#号，这可不一定。比如英国使用者就是在输入£。

对于光标移动键、功能键、Insert和Delete键，WM_KEYDOWN消息是最有用的。不过，Insert、Delete和功能键经常作为菜单快捷键。因为Windows能把菜单快捷键翻译为菜单命令消息，所以您就不必自己来处理按键。

在Windows之前的MS-DOS应用程序中大量使用功能键与Shift、Ctrl和Alt键的组合，同样地，您也可以在Windows程序中使用（实际上，Microsoft Word将大量的功能键用作命令快捷方式），但并不推荐这样做。如果您确实希望使用功能键，那么这些键应该是重复菜单命令。Windows的目标之一就是提供不需要记忆或者使用复杂命令流程的使用者接口。

因此，可以归纳如下：多数情况下，您将只为光标移动键（有时也为Insert和Delete键）处理WM_KEYDOWN消息。在使用这些键的时候，您可以通过GetKeyState来检查Shift键和Ctrl键的状态。例如，Windows程序经常使用Shift与光标键的组合键来扩大文书处理里选中的范围。Ctrl键常用于修改光标键的意义。例如，Ctrl与右箭头键相组合可以表示光标右移一个字。

决定您的程序中使用键盘方式的最佳方法之一是了解现有的Windows程序使用键盘的方式。如果您不喜欢那些定义，当然可以对其加以修改，但是这样做不利于其它人很快地学会使用您的程序。

为SYSMETS加上键盘处理功能

在编写第四章中三个版本的SYSMETS程序时，我们还不了解键盘，只能使用滚动条和鼠标来卷动文字。现在我们知道了处理键盘消息的方法，那么不妨在程序中加入键盘接口。显然，这是处理光标移动键的工作。我们将大多数光标键（Home、End、Page Up、Page Down、Up Arrow和Down Arrow）用于垂直卷动，左箭头键和右箭头键用于不太重要的水平卷动。

建立键盘接口的一种简单方法是在窗口消息处理程序中加入与WM_VSCROLL和WM_HSCROLL处理方式相仿，而且本质上相同的WM_KEYDOWN处理方法。不过这样子做是不聪明的，因为如果要修改滚动条的做法，就必须相对应地修改WM_KEYDOWN。

为什么不简单地将每一种WM_KEYDOWN消息都翻译成同等效用的WM_VSCROLL或者WM_HSCROLL消息呢？通过向窗口消息处理程序发送假冒消息，我们可能会让WndProc认为它获得了卷动信息。

在Windows中，这种方法是可行的。发送消息的函数叫做SendMessage，它所用的参数与传递到窗口消息处理程序的参数是相同的：

SendMessage (hwnd, message, wParam, lParam) ;

在呼叫SendMessage时，Windows呼叫窗口句柄为hwnd的窗口消息处理程序，并把这四个

参数传给它。当窗口消息处理程序完成消息处理之后，Windows把控制传回到SendMessage呼叫之后的下一道叙述。您发送消息过去的窗口消息处理程序，可以是同一个窗口消息处理程序、同一程序中的其它窗口消息处理程序或者其它应用程序，中的窗口消息处理程序。

下面说明在SYSMETS程序中使用SendMessage处理WM_KEYDOWN代码的方法：

```
case WM_KEYDOWN:
  switch (wParam)
  {
  case VK_HOME:
    SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
    break ;
  case VK_END:
    SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;
    break ;
  case VK_PRIOR:
    SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0) ;
    break ;
```

至此，您已经有了大概观念了吧。我们的目标是为滚动条添加键盘接口，并且也正在这么做。通过把卷动消息发送到窗口消息处理程序，我们实作了用光标移动键进行卷动列的功能。现在您知道在SYSMETS3中为WM_VSCROLL消息加上SB_TOP和SB_BOTTOM处理码的原因了吧。在那里并没有用到它，但是现在处理Home和End键时就有用了。如程序6-1所示的SYSENTS4就加上了这些变化。编译这个程序时还需要用到第四章的SYSMETS.H文件。

程序6-1 SYSMETS4

SYSMETS4.C

```
/*---------------------------------------------------------------------------
SYSMETS4.C -- System Metrics Display Program No. 4
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "sysmets.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("SysMets4") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics No. 4"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
```

```
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
  HDC hdc ;
  int i, x, y, iVertPos, iHorzPos, iPaintBeg, iPaintEnd ;
  PAINTSTRUCT ps ;
  SCROLLINFO si ;
  TCHAR szBuffer[10] ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;

    GetTextMetrics (hdc, &tm) ;
    cxChar= tm.tmAveCharWidth ;
    cxCaps= (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar= tm.tmHeight + tm.tmExternalLeading ;

    ReleaseDC (hwnd, hdc) ;
    // Save the width of the three columns

    iMaxWidth = 40 * cxChar + 22 * cxCaps ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    // Set vertical scroll bar range and page size

    si.cbSize = sizeof (si) ;
    si.fMask = SIF_RANGE | SIF_PAGE ;
    si.nMin = 0 ;
    si.nMax = NUMLINES - 1 ;
    si.nPage = cyClient / cyChar ;
    SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
    // Set horizontal scroll bar range and page size

    si.cbSize = sizeof (si) ;
    si.fMask = SIF_RANGE | SIF_PAGE ;
    si.nMin = 0 ;
    si.nMax = 2 + iMaxWidth / cxChar ;
    si.nPage = cxClient / cxChar ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    return 0 ;
  case WM_VSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_ALL ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    // Save the position for comparison later on

    iVertPos = si.nPos ;
    switch (LOWORD (wParam))
    {
    case SB_TOP:
```

```
    si.nPos = si.nMin ;
    break ;
  case SB_BOTTOM:
    si.nPos = si.nMax ;
    break ;
  case SB_LINEUP:
    si.nPos -= 1 ;
    break ;
  case SB_LINEDOWN:
    si.nPos += 1 ;
    break ;
  case SB_PAGEUP:
    si.nPos -= si.nPage ;
    break ;
  case SB_PAGEDOWN:
    si.nPos += si.nPage ;
    break ;
  case SB_THUMBTRACK:
    si.nPos = si.nTrackPos ;
    break ;
  default:
    break ;
  }
  // Set the position and then retrieve it. Due to adjustments
  // by Windows it might not be the same as the value set.

  si.fMask = SIF_POS ;
  SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
  GetScrollInfo (hwnd, SB_VERT, &si) ;

  // If the position has changed, scroll the window and update it

  if (si.nPos != iVertPos)
  {
    ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),
      NULL, NULL) ;
    UpdateWindow (hwnd) ;
  }
  return 0 ;
  case WM_HSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_ALL ;

    // Save the position for comparison later on
    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    iHorzPos = si.nPos ;

    switch (LOWORD (wParam))
    {
    case SB_LINELEFT:
      si.nPos -= 1 ;
      break ;
    case SB_LINERIGHT:
      si.nPos += 1 ;
      break ;
    case SB_PAGELEFT:
      si.nPos -= si.nPage ;
      break ;
    case SB_PAGERIGHT:
      si.nPos += si.nPage ;
      break ;
    case SB_THUMBPOSITION:
      si.nPos = si.nTrackPos ;
      break ;
    default:
      break ;
    }
    // Set the position and then retrieve it. Due to adjustments
```

```
    // by Windows it might not be the same as the value set.

    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    // If the position has changed, scroll the window

    if (si.nPos != iHorzPos)
    {
      ScrollWindow (hwnd, cxChar * (iHorzPos - si.nPos), 0,
        NULL, NULL) ;
    }
    return 0 ;
    case WM_KEYDOWN:
      switch (wParam)
      {
      case VK_HOME:
        SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
        break ;
      case VK_END:
        SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;
        break ;
      case VK_PRIOR:
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0) ;
        break ;
      case VK_NEXT:
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0) ;
        break ;
      case VK_UP:
        SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;
        break ;
      case VK_DOWN:
        SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0) ;
        break ;
      case VK_LEFT:
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEUP, 0) ;
        break ;
      case VK_RIGHT:
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEDOWN, 0) ;
        break ;
      }
      return 0 ;
      case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        // Get vertical scroll bar position
        si.cbSize = sizeof (si) ;
        si.fMask = SIF_POS ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;
        iVertPos = si.nPos ;
        // Get horizontal scroll bar position

        GetScrollInfo (hwnd, SB_HORZ, &si) ;
        iHorzPos = si.nPos ;
        // Find painting limits

        iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
        iPaintEnd = min (NUMLINES - 1,
          iVertPos + ps.rcPaint.bottom / cyChar) ;

        for (i = iPaintBeg ; i <= iPaintEnd ; i++)
        {
          x = cxChar * (1 - iHorzPos) ;
          y = cyChar * (i - iVertPos) ;

          TextOut (hdc, x, y,
            sysmetrics[i].szLabel,
            lstrlen (sysmetrics[i].szLabel)) ;

          TextOut (hdc, x + 22 * cxCaps, y,
```

```
            sysmetrics[i].szDesc,
            lstrlen (sysmetrics[i].szDesc)) ;

          SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;

          TextOut (hdc, x + 22 * cxCaps + 40 * cxChar, y, szBuffer,
            wsprintf (szBuffer, TEXT ("%5d"),
            GetSystemMetrics (sysmetrics[i].iIndex))) ;

          SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;
      case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

字符消息

前面讨论了利用位移状态信息把按键消息翻译为字符消息的方法，并且提到，仅利用转换状态信息还不够，因为还需要知道与国家/地区有关的键盘配置。由于这个原因，您不应该试图把按键消息翻译为字符代码。Windows会为您完成这一工作，在前面我们曾看到过以下的程序代码：

```
while (GetMessage (&msg, NULL, 0, 0))
{
  TranslateMessage (&msg) ;
  DispatchMessage (&msg) ;
}
```

这是WinMain中典型的消息循环。GetMessage函数用队列中的下一个消息填入msg结构的字段。DispatchMessage以此消息为参数呼叫适当的窗口消息处理程序。

在这两个函数之间是TranslateMessage函数，它将按键消息转换为字符消息。如果消息为WM_KEYDOWN或者WM_SYSKEYDOWN，并且按键与位移状态相组合产生一个字符，则TranslateMessage把字符消息放入消息队列中。此字符消息将是GetMessage从消息队列中得到的按键消息之后的下一个消息。

四类字符消息

字符消息可以分为四类，如表6-9所示。

表6-9

	字符	死字符
非系统字符	WM_CHAR	WM_DEADCHAR
系统字符	WM_SYSCHAR	WM_SYSDEADCHAR

WM_CHAR和WM_DEADCHAR消息是从WM_KEYDOWN得到的；而WM_SYSCHAR和WM_SYSDEADCHAR消息是从WM_SYSKEYDOWN消息得到的（我将简要地讨论一下什么是死字符）。

有一个好消息：在大多数情况下，Windows程序会忽略除WM_CHAR之外的任何消息。伴随四个字符消息的lParam参数与产生字符代码消息的按键消息之lParam参数相同。不过，参数wParam不是虚拟键码。实际上，它是ANSI或Unicode字符代码。

这些字符消息是我们将文字传递给窗口消息处理程序时遇到的第一个消息。它们不是唯一的消息，其它消息伴随以0结尾的整个字符串。窗口消息处理程序是如何知道该字符是8位的ANSI字符还是16位的Unicode宽字符呢？很简单：任何与您用RegisterClassA（RegisterClass的ANSI版）注册的窗口类别相联系的窗口消息处理程序，都会获得含有ANSI字符代码的消息。如果窗口消息处理程序用RegisterClassW（RegisterClass的宽字符版）注册，那么传递给窗口消息处理程序的消息就带有Unicode字符代码。如果程序用RegisterClass注册窗口类别，那么在UNICODE标识符被定义时就呼叫RegisterClassW，否则呼叫RegisterClassA。

除非在程序写作的时候混合了ANSI和Unicode的函数与窗口消息处理程序，用WM_CHAR消息（及其它三种字符消息）说明的字符代码将是：

(TCHAR) wParam

同一个窗口消息处理程序可能会用到两个窗口类别，一个用RegisterClassA注册，而另一个用RegisterClassW注册。也就是说，窗口消息处理程序可能会获得一些ANSI字符代码消息和一些Unicode字符代码消息。如果您的窗口消息处理程序需要晓得目前窗口是否处理Unicode消息，则它可以呼叫：

fUnicode = IsWindowUnicode (hwnd) ;

如果hwnd的窗口消息处理程序获得Unicode消息，那么变量fUnicode将为TRUE，这表示窗口是用RegisterClassW注册的窗口类别。

消息顺序

因为TranslateMessage函数从WM_KEYDOWN和WM_SYSKEYDOWN消息产生了字符消息，所以字符消息是夹在按键消息之间传递给窗口消息处理程序的。例如，如果Caps Lock未打开，而使用者按下再释放A键，则窗口消息处理程序将接收到如表6-10所示的三个消息：

表6-10

消息	按键或者代码
WM_KEYDOWN	「A」的虚拟键码（0x41）
WM_CHAR	「a」的字符代码（0x61）
WM_KEYUP	「A」的虚拟键码（0x41）

如果您按下Shift键，再按下A键，然后释放A键，再释放Shift键，就会输入大写的A，而窗口消息处理程序会接收到五个消息，如表6-11所示：

表6-11

消息	按键或者代码
WM_KEYDOWN	虚拟键码VK_SHIFT （0x10）
WM_KEYDOWN	「A」的虚拟键码（0x41）
WM_CHAR	「A」的字符代码（0x41）
WM_KEYUP	「A」的虚拟键码（0x41）
WM_KEYUP	虚拟键码VK_SHIFT（0x10）

Shift键本身不产生字符消息。

如果使用者按住A键，以使自动重复产生一系列的按键，那么对每条WM_KEYDOWN消息，都会得到一条字符消息，如表6-12所示：

表6-12

消息	按键或者代码
WM_KEYDOWN	「A」的虚拟键码（0x41）
WM_CHAR	「a」的字符代码（0x61）
WM_KEYDOWN	「A」的虚拟键码（0x41）
WM_CHAR	「a」的字符代码（0x61）
WM_KEYDOWN	「A」的虚拟键码（0x41）
WM_CHAR	「a」的字符代码（0x61）
WM_KEYDOWN	「A」的虚拟键码（0x41）
WM_CHAR	「a」的字符代码（0x61）
WM_KEYUP	「A」的虚拟键码（0x41）

如果某些WM_KEYDOWN消息的重复计数大于1，那么相应的WM_CHAR消息将具有同样的重复计数。

组合使用Ctrl键与字母键会产生从0x01（Ctrl-A）到0x1A（Ctrl-Z）的ASCII控制代码，其中的某些控制代码也可以由表6-13列出的键产生：

表6-13

按键	字符代码	产生方法	ANSI C控制字符
Backspace	0x08	Ctrl-H	\b
Tab	0x09	Ctrl-I	\t
Ctrl-Enter	0x0A	Ctrl-J	\n
Enter	0x0D	Ctrl-M	\r
Esc	0x1B	Ctrl-[	

最右列给出了在ANSI C中定义的控制字符，它们用于描述这些键的字符代码。

有时Windows程序将Ctrl与字母键的组合用作菜单快捷键（我将在第十章讨论），此时，不会将字母键转换成字符消息。

处理控制字符

处理按键和字符消息的基本规则是：如果需要读取输入到窗口的键盘字符，那么您可以处理WM_CHAR消息。如果需要读取光标键、功能键、Delete、Insert、Shift、Ctrl以及Alt键，那么您可以处理WM_KEYDOWN消息。

但是Tab键怎么办？Enter、Backspace和Escape键又怎么办？传统上，这些键都产生表6-13列出的ASCII控制字符。但是在Windows中，它们也产生虚拟键码。这些键应该在处理WM_CHAR或者在处理WM_KEYDOWN期间处理吗？

经过10年的考虑（回顾这些年来我写过的Windows程序代码），我更喜欢将Tab、Enter、Backspace和Escape键处理成控制字符，而不是虚拟键。我通常这样处理WM_CHAR：

```
case WM_CHAR:
  //其它行程序
  switch (wParam)
  {
  case '\b': // backspace
    //其它行程序
    break ;
  case '\t': // tab
```

```
  //其它行程序
  break ;
case '\n': // linefeed
  //其它行程序
  break ;
case '\r': // carriage return
  //其它行程序
  break ;
default: // character codes
  //其它行程序
  break ;
}
return 0 ;
```

死字符消息

Windows程序经常忽略WM_DEADCHAR和WM_SYSDEADCHAR消息，但您应该明确地知道死字符是什么，以及它们工作的方式。

在某些非U.S.英语键盘上，有些键用于给字母加上音调。因为它们本身不产生字符，所以称之为「死键」。例如，使用德语键盘时，对于U.S.键盘上的+/=键，德语键盘的对应位置就是一个死键，未按下Shift键时它用于标识锐音，按下Shift键时则用于标识抑音。

当使用者按下这个死键时，窗口消息处理程序接收到一个wParam等于音调本身的ASCII或者Unicode代码的WM_DEADCHAR消息。当使用者再按下可以带有此音调的字母键（例如A键）时，窗口消息处理程序会接收到WM_CHAR消息，其中wParam等于带有音调的字母「a」的ANSI代码。

因此，使用者程序不需要处理WM_DEADCHAR消息，原因是WM_CHAR消息已含有程序所需要的所有信息。Windows的做法甚至还设计了内部错误处理。如果在死键之后跟有不能带此音调符号的字母（例如「s」），那么窗口消息处理程序将在一行接收到两条WM_CHAR消息 – 前一个消息的wParam等于音调符号本身的ASCII代码（与传递到WM_DEADCHAR消息的wParam值相同），第二个消息的wParam等于字母s的ASCII代码。

当然，要感受这种做法的运作方式，最好的方法就是实际操作。您必须加载使用死键的外语键盘，例如前面讲过的德语键盘。您可以这样设定：在「控制台」中选择「键盘」，然后选择「语系」页面标签。然后您需要一个应用程序，该程序可以显示它接收的每一个键盘消息的详细信息。下面的KEYVIEW1就是这样的程序。

键盘消息和字符集

本章剩下的范例程序有缺陷。它们不能在所有版本的Windows下都正常执行。这些缺陷不是特意引过程序代码中的；事实上，您也许永远不会遇到这些缺陷。只有在不同的键盘语言和键盘布局间切换，以及在多字节字符集的远东版Windows下执行程序时，这些问题才会出现 – 所以我不愿将它们称为「错误」。

不过，如果程序使用Unicode编译并在Windows NT下执行，那么程序会执行得更好。我在第二章提到过这个问题，并且展示了Unicode对简化棘手的国际化问题的重要性。

KEYVIEW1程序

了解键盘国际化问题的第一步，就是检查Windows传递给窗口消息处理程序的键盘内容和字符消息。程序6-2所示的KEYVIEW1会对此有所帮助。该程序在显示区域显示Windows向窗口消息处理程序发送的8种不同键盘消息的全部信息。

程序6-2 KEYVIEW1

KEYVIEW1.C

```
/*------------------------------------------------------------------------
KEYVIEW1.C --Displays Keyboard and Character Messages
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("KeyView1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Keyboard Message Viewer #1"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxClientMax, cyClientMax, cxClient, cyClient, cxChar, cyChar ;
  static int cLinesMax, cLines ;
  static PMSG pmsg ;
  static RECT rectScroll ;
  static TCHAR szTop[] = TEXT ("Message Key Char ")
    TEXT ("Repeat Scan Ext ALT Prev Tran") ;
  static TCHAR szUnd[] = TEXT ("_______ ___ ____ ")
    TEXT ("______ ____ ___ ___ ____ ____") ;

  static TCHAR * szFormat[2] = {
    TEXT ("%-13s %3d %-15s%c%6u %4d %3s %3s %4s %4s"),
      TEXT ("%-13s 0x%04X%1s%c %6u %4d %3s %3s %4s %4s") } ;
    static TCHAR * szYes = TEXT ("Yes") ;
    static TCHAR * szNo = TEXT ("No") ;
    static TCHAR * szDown = TEXT ("Down") ;
    static TCHAR * szUp = TEXT ("Up") ;
```

```
static TCHAR * szMessage [] = {
  TEXT ("WM_KEYDOWN"), TEXT ("WM_KEYUP"),
    TEXT ("WM_CHAR"), TEXT ("WM_DEADCHAR"),
    TEXT ("WM_SYSKEYDOWN"),TEXT ("WM_SYSKEYUP"),
    TEXT ("WM_SYSCHAR"), TEXT ("WM_SYSDEADCHAR") } ;
  HDC hdc ;
  int i, iType ;
  PAINTSTRUCT ps ;
  TCHAR szBuffer[128], szKeyName [32] ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_CREATE:
  case WM_DISPLAYCHANGE:
    // Get maximum size of client area
    cxClientMax = GetSystemMetrics (SM_CXMAXIMIZED) ;
    cyClientMax = GetSystemMetrics (SM_CYMAXIMIZED) ;

    // Get character size for fixed-pitch font
    hdc = GetDC (hwnd) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cyChar = tm.tmHeight ;

    ReleaseDC (hwnd, hdc) ;
    // Allocate memory for display lines
    if (pmsg)
      free (pmsg) ;
    cLinesMax = cyClientMax / cyChar ;
    pmsg = (MSG*)malloc (cLinesMax * sizeof (MSG)) ;
    cLines = 0 ;
    // fall through
  case WM_SIZE:
    if (message == WM_SIZE)
    {
      cxClient = LOWORD (lParam) ;
      cyClient = HIWORD (lParam) ;
    }
    // Calculate scrolling rectangle
    rectScroll.left = 0 ;
    rectScroll.right = cxClient ;
    rectScroll.top = cyChar ;
    rectScroll.bottom = cyChar * (cyClient / cyChar) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_KEYDOWN:
  case WM_KEYUP:
  case WM_CHAR:
  case WM_DEADCHAR:
  case WM_SYSKEYDOWN:
  case WM_SYSKEYUP:
  case WM_SYSCHAR:
  case WM_SYSDEADCHAR:
    // Rearrange storage array
    for (i = cLinesMax - 1 ; i > 0 ; i--)
    {
      pmsg[i] = pmsg[i - 1] ;
    }
    // Store new message
    pmsg[0].hwnd = hwnd ;
    pmsg[0].message = message ;
    pmsg[0].wParam = wParam ;
    pmsg[0].lParam = lParam ;

    cLines = min (cLines + 1, cLinesMax) ;
```

```
      // Scroll up the display
      ScrollWindow (hwnd, 0, -cyChar, &rectScroll, &rectScroll) ;
      break ; // i.e., call DefWindowProc so Sys messages work
    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;
      SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
      SetBkMode (hdc, TRANSPARENT) ;
      TextOut (hdc, 0, 0, szTop, lstrlen (szTop)) ;
      TextOut (hdc, 0, 0, szUnd, lstrlen (szUnd)) ;

      for (i = 0 ; i < min (cLines, cyClient / cyChar - 1) ; i++)
      {
        iType = pmsg[i].message == WM_CHAR ||
          pmsg[i].message == WM_SYSCHAR ||
          pmsg[i].message == WM_DEADCHAR ||
          pmsg[i].message == WM_SYSDEADCHAR ;

        GetKeyNameText (pmsg[i].lParam, szKeyName,
          sizeof (szKeyName) / sizeof (TCHAR)) ;

        TextOut (hdc, 0, (cyClient / cyChar - 1 - i) * cyChar, szBuffer,
          wsprintf (szBuffer, szFormat [iType],
          szMessage [pmsg[i].message - WM_KEYFIRST],
          pmsg[i].wParam,
          (PTSTR) (iType ? TEXT (" ") : szKeyName),
          (TCHAR) (iType ? pmsg[i].wParam : ' '),
          LOWORD (pmsg[i].lParam),
          HIWORD (pmsg[i].lParam) & 0xFF,
          0x01000000 & pmsg[i].lParam ? szYes : szNo,
          0x20000000 & pmsg[i].lParam ? szYes : szNo,
          0x40000000 & pmsg[i].lParam ? szDown : szUp,
          0x80000000 & pmsg[i].lParam ? szUp : szDown)) ;
      }
      EndPaint (hwnd, &ps) ;
      return 0 ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

KEYVIEW1显示窗口消息处理程序接收到的每次按键和字符消息的内容，并将这些消息储存在一个MSG结构的数组中。该数组的大小依据最大化窗口的大小和等宽的系统字体。如果使用者在程序执行时调整了视讯显示的大小（在这种情况下KEYVIEW1接收WM_DISPLAYCHANGE消息），将重新分配此数组。KEYVIEW1使用标准C的malloc函数为数组配置内存。

图6-2给出了在键入「Windows」之后KEYVIEW1的屏幕显示。第一列显示了键盘消息；第二列在键名称的前面显示了按键消息的虚拟键代码，此代码是经由GetKeyNameText函数取得的；第三列（标注为「Char」）在字符本身的后面显示字符消息的十六进制字符代码。其余六列显示了lParam消息参数中六个字段的状态。

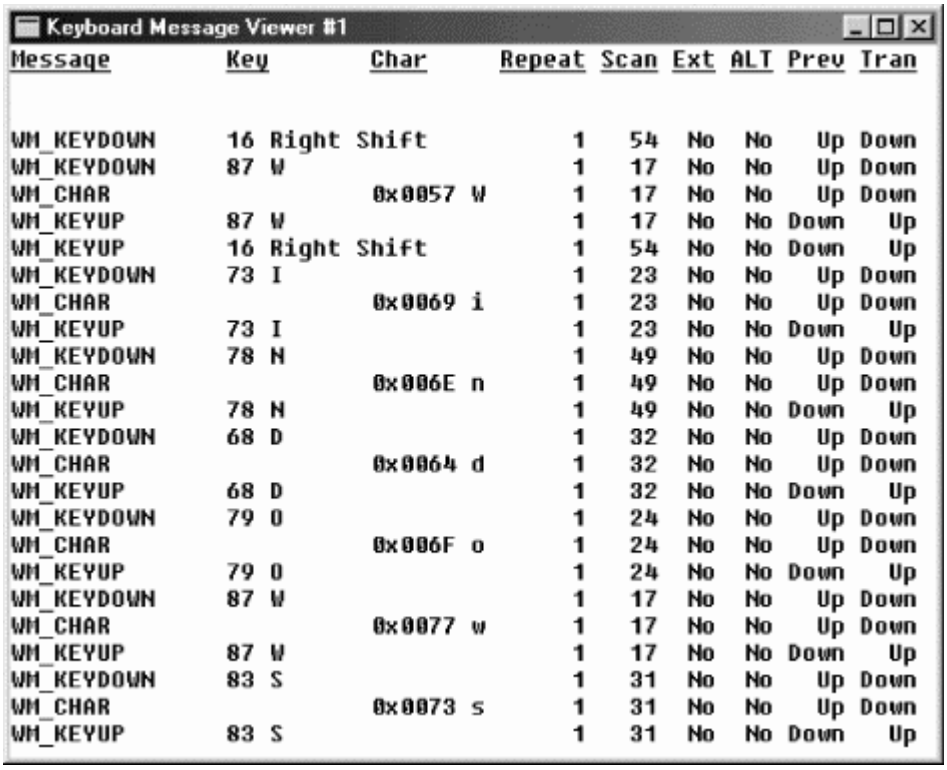


图6-2 KEYVIEW1的屏幕显示

为便于以分行的方式显示此信息，KEYVIEW1使用了等宽字体。与前一章所讨论的一样，这需要呼叫GetStockObject和SelectObject：

```
SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
```

KEYVIEW1在显示区域上部画了一个标题以确定分成九行。此列文字带有底线。虽然可以建立一种带底线的字体，但这里使用了另一种方法。我定义了两个字符串变量szTop（有文字）和szUnd（有底线），并在WM_PAINT消息处理期间将它们同时显示在窗口顶部的同一位置。通常，Windows以一种「不透明」的方式显示文字，也就是说显示字符时Windows将擦除字符背景区。这将导致第二个字符串（szUnd）擦除掉前一个（szTop）。要防止这一现象的发生，可将设备内容切换到「透明」模式：

```
SetBkMode (hdc, TRANSPARENT) ;
```

这种加底线的方法只有在使用等宽字体时才可行。否则，底线字符将无法与显现在底线上面的字符等宽。

外语键盘问题

如果您执行美国英语版本的Windows，那么您可安装不同的键盘布局，并输入外语。可以在 **控制台**的**键盘**中安装外语键盘布局。选择 **语系**页面标签，按下**新增** 键。要查看死键的工作方式，您可能想安装「德语」键盘。此外，我还要讨论「俄语」和「希腊语」的键盘布局，因此您也可安装这些键盘布局。如果在「键盘」显示的列表中找不到「俄语」和「希腊语」的键盘布局，则需要安装多语系支持：从「控制台」中选择 **新增/删除**程序，然后选择 **Windows安装程序**页面卷标，确认选中 **多语系支持**复选框。在任何情况下，这些变更都需要原始的Windows光盘。

安装完其它键盘布局后，您将在工作列右侧的通知区看到一个带有两个字母代码的蓝色框。如果内定的是英语，那么这两个字母是「EN」。单击此图标，将得到所有已安装键盘布局的列表。从中单击需要的键盘布局即可更改目前活动程序的键盘。此改变只影响目前活动的程序。

现在开始进行实验。不使用UNICODE标识符定义来编译KEYVIEW1程序（在本书附带的光盘中，非Unicode版本的KEYVIEW1程序位于RELEASE子目录）。在美国英语版本的Windows下执行该程序，并输入字符『abcde』。 WM_CHAR消息与您所期望的一样：ASCII字符代码0x61、0x62、0x63、0x64和0x65以及字母a、b、c、d和e。

现在，KEYVIEW1还在执行，选择德语键盘布局。按下=键然后输入一个元音（a、e、i、o或者u）。=键将产生一个WM_DEADCHAR消息，元音产生一个WM_CHAR消息和（单独的）字符代码0xE1、0xE9、0xED、0xF3、0xFA和字符á、é、í、ó或ú。这就是死键的工作方式。

现在选择希腊键盘布局。输入『abcde』，您会得到什么？您将得到WM_CHAR消息和字符代码0xE1、0xE2、0xF8、0xE4、0xE5和字符á、狻 ii 、?#160;和滩 T 谖饮镉行┬址　荒苷　废允尽 D 训滥　挥Ω玫玫较＠白帜副碇械淖帜嘎穑?/p>

现在切换到俄语键盘并重新输入『abcde』。现在您得到WM_CHAR消息和字符代码0xF4、0xE8、0xF1、0xE2和0xF3，以及字符簟viiiā Ⅰ 、?#160;和ó。而且，还是有些字母不能正常显示。您应从斯拉夫字母表中得到这些字母。

问题在于：您已经切换键盘以产生不同的字符代码，但您还没有将此切换通知GDI，好让GDI能选择适当的符号来显示解释这些字符代码。

如果您非常勇敢，还有可用的备用PC，并且是专业或全球版Microsoft Developer Network（MSDN）的订阅户，那么您也许想安装（例如）希腊版的Windows，您还可以把那四种键盘布局（英语、希腊语、德语和俄语）安装上去。现在执行KEYLOOK1，切换到英语键盘布局，然后输入『abcde』。您应得到ASCII字符代码0x61、0x62、0x63、0x64和0x65以及字符a、b、c、d和e（并且您可以放心：即使在希腊版，ASCII还是正常通行的）。

在希腊版的Windows中，切换到希腊键盘布局并输入『abcde』。您将得到WM_CHAR消息和字符代码0xE1、0xE2、0xF8、0xE4和0xE5。这与您在安装希腊键盘布局的英语版Windows中得到的字符代码相同。但现在显示的字符是 、 、 、 和 。这些确实是小写的希腊字母alpha、beta、psi、delta和epsilon（gamma怎么了？是这样，如果使用希腊版的Windows，那么您将使用键帽上带有希腊字母的键盘。与英语c相对应的键正好是psi。gamma由与英语g相对应的键产生。您可在Nadine Kano编写的《Developing International Software for Windows 95 and Windows NT》的第587页看到完整的希腊字母表）。

继续在希腊版的Windows下运行KEYVIEW1，切换到德语键盘布局。输入『=』键，然后依次输入a、e、i、o和u。您将得到WM_CHAR消息和字符代码0xE1、0xE9、0xED、0xF3和0xFA。这些字符代码与安装德语键盘布局的英语版Windows中的一样。不过，显示的字符却是 、 、 、 和ï，而不是正确的á、é、í、ó和ú。

现在切换到俄语键盘并输入『abcde』。您会得到字符代码0xF4、0xE8、0xF1、0xE2和0xF3，这与安装俄语键盘的英语版Windows中得到的一样。不过，显示的字符是 、 、 、 和 ，而不是斯拉夫字母表中的字母。

您还可安装俄语版的Windows。现在您可以猜到，英语和俄语键盘都可以工作，而德语和希腊语则不行。

现在，如果您真的很勇敢，您还可安装日语版的Windows并执行KEYVIEW1。如果再依美国键盘输入，那么您将输入英语文字，一切似乎都正常。不过，如果切换到德语、希腊语或者俄语键盘布局，并且试著作上述介绍的任何练习，您将看到以点显示的字符。如果输入大写的字母 – 无论是带重音符号的德语字母、希腊语字母还是俄语字母 – 您将看到这些字母显示为日语中用于拼写外来语的片假名。您也许对输入片假名感兴趣，但那不是德语、希腊语或者俄语。

远东版本的Windows包括一个称作「输入法编辑器」(IME)的实用程序，该程序显示为浮动的工具列，它允许您用标准键盘输入象形文字，即汉语、日语和朝鲜语中使用的复杂字符。一般来说，输入一组字母后，组成的字符将显示在另一个浮动窗口内。然后按 Enter键，合成的字符代码就发送到了活动窗口(即KEYVIEW1)。KEYVIEW1几乎没什么响应 – WM_CHAR消息带来的字符代码大于128，但这些代码没有意义(Nadine Kano的书中有许多关于使用IME的内容)。

这时，我们已经看到了许多KEYLOOK1显示错误字符的例子 – 当执行安装了俄语或希腊语键盘布局的英语版Windows时，当执行安装了俄语或德语键盘布局的希腊版Windows时，以及执行安装了德语、俄语或者希腊语键盘布局的俄语版Windows时，都是这样。我们也看到了从日语版Windows的输入法编辑器输入字符时的错误显示。

字符集和字体

KEYLOOK1的问题是字体问题。用于在屏幕上显示字符的字体和键盘接收的字符代码不一致。因此，让我们看一下字体。

我将在第十七章进行详细讨论，Windows支持三类字体 – 点阵字体、向量字体和(从Windows 3.1开始的) TrueType字体。

事实上向量字体已经过时了。这些字体中的字符由简单的线段组成，但这些线段没有定义填入区域。向量字体可以较好地缩放到任意大小，但字符通常看上去有些单薄。

TrueType字体是定义了填入区域的文字轮廓字体。TrueType字体可缩放；而且该字符的定义包括「提示」，以消除可能带来的文字不可见或者不可读的圆整问题。使用TrueType字体，Windows就真正实现了WYSIWYG(「所见即所得」)，即文字在视讯显示器显示与打印机输出完全一致。

在点阵字体中，每个字符都定义为与视讯显示器上的像素对应的位点阵。点阵字体可拉伸到较大的尺寸，但看上去带有锯齿。点阵字体通常被设计成方便在视讯显示器上阅读的字体。因此，Windows中的标题栏、菜单、按钮和对话框的显示文字都使用点阵字体。

在内定的设备内容下获得的点阵字体称为系统字体。您可通过呼叫带有SYSTEM_FONT标识符的GetStockObject函数来获得字体句柄。KEYVIEW1程序选择使用SYSTEM_FIXED_FONT表示的等宽系统字体。GetStockObject函数的另一个选项是OEM_FIXED_FONT。

这三种字体有(各自的)字体名称 – System、FixedSys和Terminal。程序可以在CreateFont或者CreateFontIndirect函数呼叫中使用字体名称来指定字体。这三种字体储存在两组放在Windows目录内的FONTS子目录下的三个文件中。Windows使用哪一组文件取决于「控制台」里的「显示器」是选择显示「小字体」还是「大字体」(亦即，您希望Windows假定视讯显示器是96 dpi的分辨率还是120 dpi的分辨率)。表6-14总结了所有的情况：

表6-14

GetStockObject 标识符	字体名称	小字体文件	大字体文件
SYSTEM_FONT	System	VGASYS.FON	8514SYS.FON
SYSTEM_FIXED_FONT	FixedSys	VGAFIX.FON	8514FIX.FON
OEM_FIXED_FONT	Terminal	VGAOEM.FON	8514OEM.FON

在文件名称中，「VGA」指的是视频图形数组(Video Graphics Array)，IBM在1987年推出的显示卡。这是IBM第一块可显示640×480像素大小的PC显示卡。如果在「控制台」的「显示器」中选择了「小字体」(表示您希望Windows假定视讯显示的分辨率为96 dpi)，则Windows使用的这三种字体文件名将以「VGA」开头。如果选择了「大字体」(表示您希望分辨率为120 dpi)，

Windows使用的文件名将以「8514」开头。8514是IBM在1987年推出的另一种显示卡，它的最大显示尺寸为1024×768。

Windows不希望您看到这些文件。这些文件的属性设定为系统和隐藏，如果用Windows Explorer来查看FONTS子目录的内容，您是不会看到它们的，即使选择了查看系统和隐藏文件也不行。从开始菜单选择「寻找」选项来寻找文件名满足 *.FON限定条件的文件。这时，您可以双击文件名来查看字体字符是些什么。

对于许多标准控件和使用者接口组件，Windows不使用系统字体。相反地，使用名称为MS Sans Serif的字体（「MS」代表Microsoft）。这也是一种点阵字体。文件（名为SSERIFE.FON）包含依据96 dpi视讯显示器的字体，点值为8、10、12、14、18和24。您可在GetStockObject函数中使用DEFAULT_GUI_FONT标识符来得到该字体。Windows使用的点值取决于「控制台」的「显示」中选择的显示分辨率。

到目前为止，我已提到四种标识符，利用这四种标识符，您可以用GetStockObject来获得用于设备内容的字体。还有三种其它字体标识符：ANSI_FIXED_FONT、ANSI_VAR_FONT和DEVICE_DEFAULT_FONT。为了开始处理键盘和字符显示问题，让我们先看一下Windows中的所有备用字体。显示这些字体的程序是STOKFONT，如程序6-3所示。

程序6-3 STOKFONT

STOKFONT.C

```
/*------------------------------------------------------------------------
STOKFONT.C -- Stock Font Objects
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("StokFont") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Stock Fonts"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static struct
  {
    int idStockFont ;
    TCHAR * szStockFont ;
  }
  stockfont [] = { OEM_FIXED_FONT, "OEM_FIXED_FONT",
    ANSI_FIXED_FONT, "ANSI_FIXED_FONT",
    ANSI_VAR_FONT, "ANSI_VAR_FONT",
    SYSTEM_FONT, "SYSTEM_FONT",
    DEVICE_DEFAULT_FONT,"DEVICE_DEFAULT_FONT",
    SYSTEM_FIXED_FONT, "SYSTEM_FIXED_FONT",
    DEFAULT_GUI_FONT, "DEFAULT_GUI_FONT" } ;

  static int iFont, cFonts = sizeof stockfont / sizeof stockfont[0] ;
  HDC hdc ;
  int i, x, y, cxGrid, cyGrid ;
  PAINTSTRUCT ps ;
  TCHAR szFaceName [LF_FACESIZE], szBuffer [LF_FACESIZE + 64] ;
  TEXTMETRIC tm ;
  switch (message)
  {
  case WM_CREATE:
    SetScrollRange (hwnd, SB_VERT, 0, cFonts - 1, TRUE) ;
    return 0 ;
  case WM_DISPLAYCHANGE:
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_VSCROLL:
    switch (LOWORD (wParam))
    {
    case SB_TOP: iFont = 0 ; break ;
    case SB_BOTTOM: iFont = cFonts - 1 ; break ;
    case SB_LINEUP:
    case SB_PAGEUP: iFont -= 1 ; break ;
    case SB_LINEDOWN:
    case SB_PAGEDOWN: iFont += 1 ; break ;
    case SB_THUMBPOSITION:iFont = HIWORD (wParam) ; break ;
    }
    iFont = max (0, min (cFonts - 1, iFont)) ;
    SetScrollPos (hwnd, SB_VERT, iFont, TRUE) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
    case WM_KEYDOWN:
      switch (wParam)
      {
      case VK_HOME: SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ; break ;
      case VK_END: SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ; break ;
      case VK_PRIOR:
      case VK_LEFT:
      case VK_UP: SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ; break ;
      case VK_NEXT:
      case VK_RIGHT:
      case VK_DOWN: SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0) ; break ;
      }
      return 0 ;
      case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        SelectObject (hdc, GetStockObject (stockfont[iFont].idStockFont)) ;
```

```
      GetTextFace (hdc, LF_FACESIZE, szFaceName) ;
      GetTextMetrics (hdc, &tm) ;
      cxGrid = max (3 * tm.tmAveCharWidth, 2 * tm.tmMaxCharWidth) ;
      cyGrid = tm.tmHeight + 3 ;

      TextOut (hdc, 0, 0, szBuffer,
        wsprintf ( szBuffer, TEXT (" %s: Face Name = %s, CharSet = %i"),
        stockfont[iFont].szStockFont,
        szFaceName, tm.tmCharSet)) ;

      SetTextAlign (hdc, TA_TOP | TA_CENTER) ;
      // vertical and horizontal lines
      for (i = 0 ; i < 17 ; i++)
      {
        MoveToEx (hdc, (i + 2) * cxGrid, 2 * cyGrid, NULL) ;
        LineTo (hdc, (i + 2) * cxGrid, 19 * cyGrid) ;

        MoveToEx (hdc, cxGrid, (i + 3) * cyGrid, NULL) ;
        LineTo (hdc, 18 * cxGrid, (i + 3) * cyGrid) ;
      }
      // vertical and horizontal headings

      for (i = 0 ; i < 16 ; i++)
      {
        TextOut (hdc, (2 * i + 5) * cxGrid / 2, 2 * cyGrid + 2, szBuffer,
          wsprintf (szBuffer, TEXT ("%X-"), i)) ;

        TextOut (hdc, 3 * cxGrid / 2, (i + 3) * cyGrid + 2, szBuffer,
          wsprintf (szBuffer, TEXT ("-%X"), i)) ;
      }
      // characters

      for (y = 0 ; y < 16 ; y++)
        for (x = 0 ; x < 16 ; x++)
        {
          TextOut (hdc, (2 * x + 5) * cxGrid / 2,
            (y + 3) * cyGrid + 2, szBuffer,
            wsprintf (szBuffer, TEXT ("%c"), 16 * x + y)) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这个程序相当简单。它使用滚动条和光标移动键让您选择显示七种备用字体之一。该程序在一个网格中显示一种字体的256个字符。顶部的标题和网格的左侧显示字符代码的十六进制值。

在显示区域的顶部，STOKFONT用GetStockObject函数显示用于选择字体的标识符。它还显示由GetTextFace函数得到的字体样式名称和TEXTMETRIC结构的tmCharSet字段。这个「字符集标识符」对理解Windows如何处理外语版本的Windows是非常重要的。

如果在美国英语版本的Windows中执行STOKFONT，那么您看到的第一个画面将显示使用OEM_FIXED_FONT标识符呼叫GetStockObject函数得到的字体。如图6-3所示。

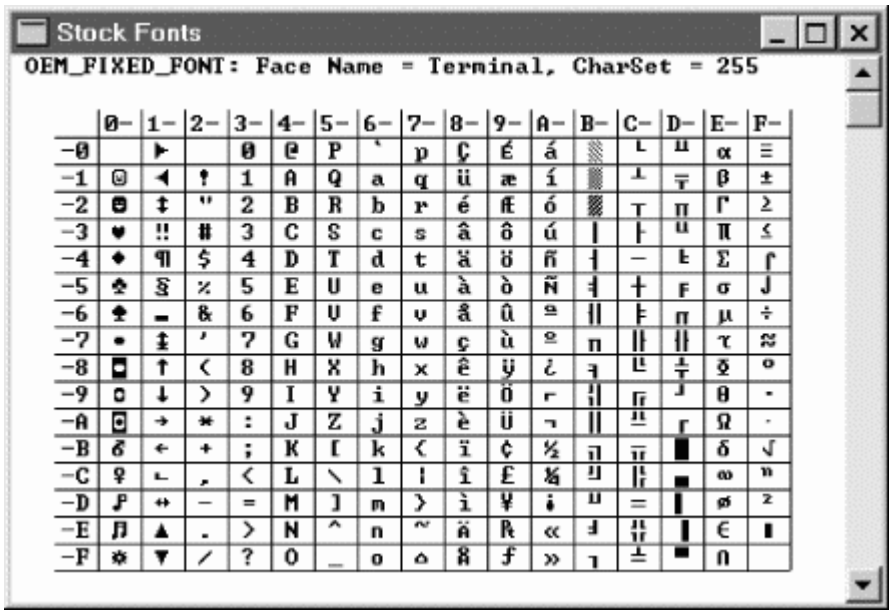


图6-3 美国版Windows中的OEM_FIXED_FONT

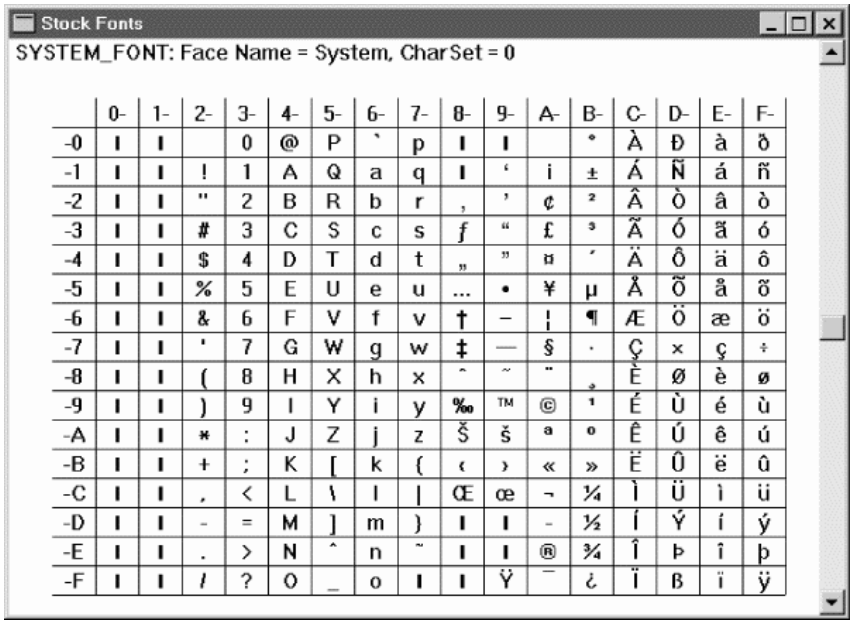
在本字符集中（与本章其它部分一样），您将看到一些ASCII。但请记住ASCII是7位代码，它定义了从代码0x20到0x7E的可显示字符。到IBM开发出IBM PC原型机时，8位字节代码已被稳固地建立起来，因此可使用全8位代码作为字符代码。IBM决定使用一系列由线和方块组成的字符、带重音字母、希腊字母、数学符号和一些其它字符来扩展ASCII字符集。许多文字模式的MS-DOS程序在其屏幕显示中都使用绘图字符，并且许多MS-DOS程序都在文件中使用了一些扩展字符。

这个特殊的字符集给Windows最初的开发者带来了一个问题。一方面，因为Windows有完整的图形程序设计语言，所以线和方块字元在Windows中不需要。因此，这些字符使用的48个代码最好用于许多西欧语言所需要的附带重音字母。另一方面，IBM字符集定义了一个无法完全忽略的标准。

因此，Windows最初的开发者决定支持IBM字符集，但将其重要性降低到第二位 – 它们大多用于在窗口中执行的旧MS-DOS应用程序，和需要使用由MS-DOS应用程序建立文件的Windows程序。Windows应用程序不使用IBM字符集，并且随着时间的推移，其重要性日渐衰退。然而，如果需要，您还是可以使用。在此环境下，「OEM」指的就是「IBM」。

（您应知道外语版本的Windows不必支持与美国英语版相同的OEM字符集。其它国家有其自己的MS-DOS字符集。这是个独立的问题，就不在本书中讨论了。）

因为IBM字符集被认为不适合Windows，于是选择了另一种扩展字符集。此字符集称作「ANSI字符集」，由美国国家标准协会（American National Standards Institute）制定，但它实际上是ISO（International Standards Organization，国际标准化组织）标准，也就是ISO标准8859。它还称为Latin 1、Western European、或者代码页1252。图6-4显示了ANSI字符集的一个版本 – 美国英语版Windows的系统字体。



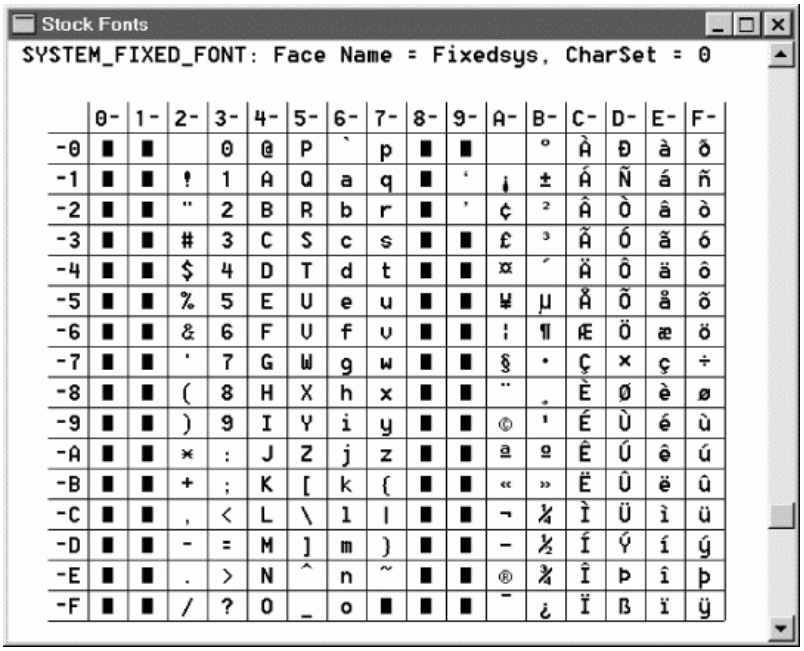
	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	▮	▮		0	@	P	`	p	▮	▮		°	À	Ð	à	ð
-1	▮	▮	!	1	A	Q	a	q	▮	‘	¡	±	Á	Ñ	á	ñ
-2	▮	▮	"	2	B	R	b	r	‚	’	¢	²	Â	Ò	â	ò
-3	▮	▮	#	3	C	S	c	s	ƒ	“	£	³	Ã	Ó	ã	ó
-4	▮	▮	$	4	D	T	d	t	„	”	¤	´	Ä	Ô	ä	ô
-5	▮	▮	%	5	E	U	e	u	…	•	¥	µ	Å	Õ	å	õ
-6	▮	▮	&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	æ	ö
-7	▮	▮	'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
-8	▮	▮	(	8	H	X	h	x	ˆ	˜	¨	¸	È	Ø	è	ø
-9	▮	▮	)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
-A	▮	▮	*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
-B	▮	▮	+	;	K	[	k	{	‹	›	«	»	Ë	Û	ë	û
-C	▮	▮	,	<	L	\	l	|	Œ	œ	¬	¼	Ì	Ü	ì	ü
-D	▮	▮	-	=	M	]	m	}	▮	▮	-	½	Í	Ý	í	ý
-E	▮	▮	.	>	N	^	n	~	▮	▮	®	¾	Î	Þ	î	þ
-F	▮	▮	/	?	O	_	o	▮	▮	Ÿ	¯	¿	Ï	ß	ï	ÿ

图6-4 美国版Windows中的SYSTEM_FONT

粗的垂直条表示这些字符代码没有定义。注意，代码0x20到0x7E还是ASCII。此外，ASCII控制字符（0x00到0x1F以及0x7F）并不是可显示字符。它们本应如此。

代码0xC0到0xFF使得ANSI字符集对外语版Windows来说非常重要。这些代码提供64个在西欧语言中普遍使用的字符。字符0xA0，看起来像空格，但实际上定义为非断开空格，例如「WW II」中的空格。

之所以说这是ANSI字符集的「一个版本」，是因为存在代码0x80到0x9F的字符。等宽的系统字体只包括其中的两个字符，如图6-5所示。



	0-	1-	2-	3-	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	▮	▮		0	@	P	`	p	▮	▮		°	À	Ð	à	ð
-1	▮	▮	!	1	A	Q	a	q	▮	‘	¡	±	Á	Ñ	á	ñ
-2	▮	▮	"	2	B	R	b	r	▮	’	¢	²	Â	Ò	â	ò
-3	▮	▮	#	3	C	S	c	s	▮	▮	£	³	Ã	Ó	ã	ó
-4	▮	▮	$	4	D	T	d	t	▮	▮	¤	´	Ä	Ô	ä	ô
-5	▮	▮	%	5	E	U	e	u	▮	▮	¥	µ	Å	Õ	å	õ
-6	▮	▮	&	6	F	U	f	u	▮	▮	¦	¶	Æ	Ö	æ	ö
-7	▮	▮	'	7	G	W	g	w	▮	▮	§	·	Ç	×	ç	÷
-8	▮	▮	(	8	H	X	h	x	▮	▮	¨	¸	È	Ø	è	ø
-9	▮	▮	)	9	I	Y	i	y	▮	▮	©	¹	É	Ù	é	ù
-A	▮	▮	*	:	J	Z	j	z	▮	▮	ª	º	Ê	Ú	ê	ú
-B	▮	▮	+	;	K	[	k	{	▮	▮	«	»	Ë	Û	ë	û
-C	▮	▮	,	<	L	\	l	|	▮	▮	¬	¼	Ì	Ü	ì	ü
-D	▮	▮	-	=	M	]	m	}	▮	▮	-	½	Í	Ý	í	ý
-E	▮	▮	.	>	N	^	n	~	▮	▮	®	¾	Î	Þ	î	þ
-F	▮	▮	/	?	O	_	o	▮	▮	▮	¯	¿	Ï	ß	ï	ÿ

图6-5 美国版Windows中的SYSTEM_FIXED_FONT

在Unicode中，代码0x0000到0x007F与ASCII相同，代码0x0080到0x009F复制了0x0000到0x001F的控制字符，代码0x00A0到0x00FF与Windows中使用的ANSI字符集相同。

如果执行德语版的Windows，那么当您用SYSTEM_FONT或者SYSTEM_FIXED_FONT标识符来呼叫GetStockObject函数时会得到同样的ANSI字符集。其它西欧版Windows也是如此。ANSI字符集中含有这些语言所需要的所有字符。

不过，当您执行希腊版的Windows时，内定的字符集就改变了。相反地，SYSTEM_FONT如图6-6所示。

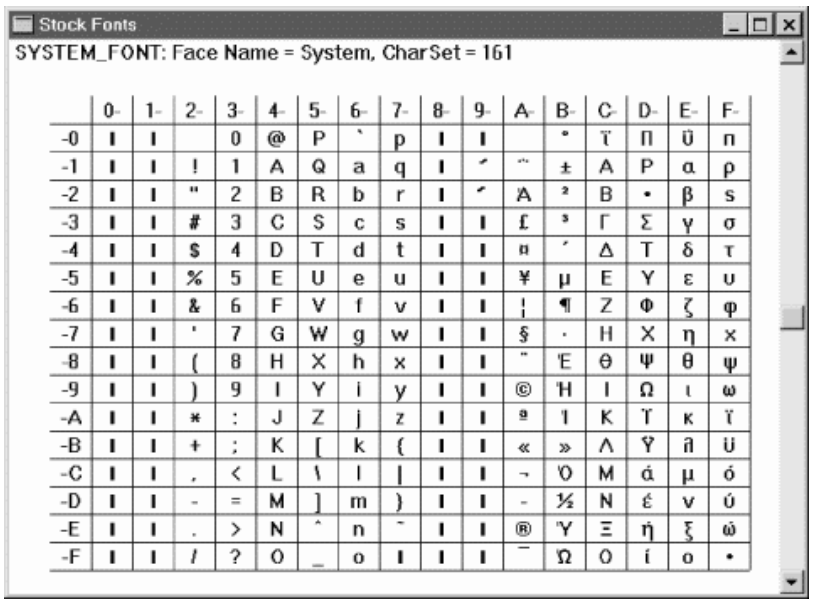


图6-6 希腊版Windows中的SYSTEM_FONT

SYSTEM_FIXED_FONT有同样的字符。注意从0xC0到0xFF的代码。这些代码包含希腊字母表中的大写字母和小写字母。当您执行俄语版Windows时，内定的字符集如图6-7所示。

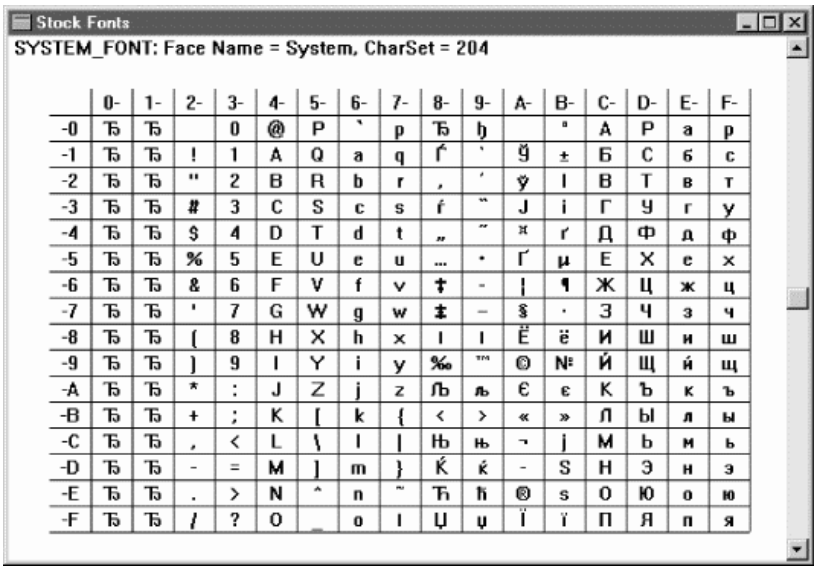


图6-7 俄语版Windows中的SYSTEM_FONT

此外， 注意斯拉夫字母表中的大写和小写字母占用了代码0xC0和0xFF。

图6-8显示了日语版Windows的SYSTEM_FONT。从0xA5到0xDF的字符都是片假名字母表的一部分。

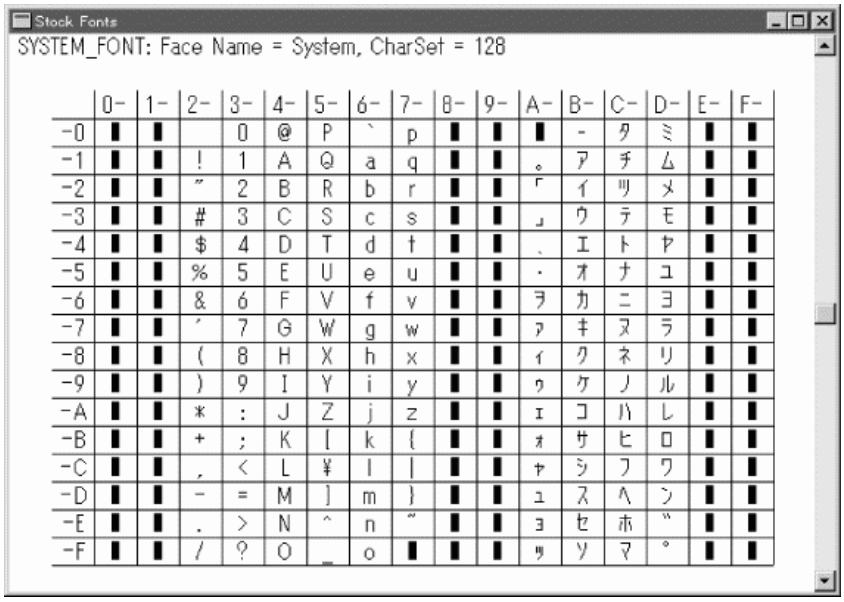


图6-8 日语版Windows中的SYSTEM_FONT

图6-8所示的日文系统字体不同于前面显示的那些，因为它实际上是双字节字符集（DBCS），称为「Shift-JIS」（「JIS」代表日本工业标准，Japanese Industrial Standard）。从0x81到0x9F以及从0xE0到0xFF的大多数字符代码实际上只是双字节代码的第一个字节，其第二个字节通常在0x40到0xFC的范围内（关于这些代码的完整表格，请参见Nadine Kano书中的附录G）。

现在，我们就可以看看KEYVIEW1中的问题在哪里：如果您安装了希腊键盘布局并键入『abcde』，不考虑执行的Windows版本，Windows将产生WM_CHAR消息和字符代码0xE1、0xE2、0xF8、0xE4和0xE5。但只有执行带有希腊系统字体的希腊版Windows时，这些字符代码才能与 、 、 、 和 相对应。

如果您安装了俄语键盘布局并敲入『abcde』，不考虑所使用的Windows版本，Windows将产生WM_CHAR消息和字符代码0xF4、0xE8、0xF1、0xE2和0xF3。但只有在使用俄语版Windows或者使用斯拉夫字母表的其它语言版，并且使用斯拉夫系统字体时，这些字符代码才会与字符ф、и、с、в和у相对应。

如果您安装了德语键盘布局并按下=键（或者位于同一位置的键），然后按下a、e、i、o或者u键，不考虑使用的Windows版本，Windows将产生WM_CHAR消息和字符代码0xE1、0xE9、0xED、0xF3和0xFA。只有执行西欧版或者美国版的Windows时，也就是说有西欧系统字体，这些字符代码才会和字符amp;nbsp;á、é、í、ó和ú相对应。

如果安装了美国英语键盘布局，则您可在键盘上键入任何字符，Windows将产生WM_CHAR消息以及与字符正确匹配的字符代码。

Unicode怎么样?

我在第二章谈到过Windows NT支持的Unicode有助于为国际市场程序写作。让我们编译一下定义了UNICODE标识符的KEYVIEW1，并在不同版本的Windows NT下执行（在本书附带的光盘中，Unicode版的KEYVIEW1位于DEBUG目录中）。

如果程序编译时定义了UNICODE标识符，则「KeyView1」窗口类别就用RegisterClassW函数注册，而不是RegisterClassA函数。这意味着任何带有字符或文字数据的消息传递给WndProc时都将使用16位字符而不是8位字符。特别是WM_CHAR消息，将传递16位字符代码而不是8位字符代码。

请在美国英语版的Windows NT下执行Unicode版的KEYVIEW1。这里假定您已经安装了至少三种我们试验过的键盘布局 – 即德语、希腊语和俄语。

使用美国英语版的Windows NT，并安装了英语或者德语的键盘布局，Unicode版的KEYVIEW1在工作时将与非Unicode版相同。它将接收相同的字符代码（所有0xFF或者更低的值），并显示同样正确的字符。这是因为最初的256个Unicode字符与Windows中使用的ANSI字符集相同。

现在切换到希腊键盘布局，并键入『abcde』。WM_CHAR消息将含有Unicode字符代码0x03B1、 0x03B2、0x03C8、 0x03B4和0x03B5。注意，我们先看到的字符代码值比0xFF高。这些Unicode字符代码与希腊字母 、 、 、d和 相对应。不过，所有这五个字符都显示为方块！这是因为SYSTEM_FIXED_FONT只含有256个字符。

现在切换到俄语键盘布局，并键入『abcde』。KEYVIEW1显示WM_CHAR消息和Unicode字符代码0x0444、0x0438、0x0441、0x0432和0x0443，这些字符对应于斯拉夫字母ф、и、с、в和у。不过，所有这五个字母也显示为实心方块。

简言之，非Unicode版的KEYVIEW1显示错误字符的地方，Unicode版的KEYVIEW1就显示实心方块，以表示目前的字体没有那种特殊字符。虽然我不愿说Unicode版的KEYVIEW1是非Unicode版的改进，但事实确实如此。非Unicode版显示错误字符，而Unicode版不会这样。

Unicode和非Unicode版KEYVIEW1的不同之处主要在两个方面。

首先，WM_CHAR消息伴随一个16位字符代码，而不是8位字符代码。在非Unicode版本的KEYVIEW1中，8位字符代码的含义取决于目前活动的键盘布局。如果来自德语键盘，则0xE1代码表示á，如果来自希腊语键盘则代表 ，如果来自俄语键盘则代表 。在Unicode版本程序中，16位字符代码的含义很明确：a字符是0x00E1， 字符是0x03B1，而 字符是0x0431。

第二，Unicode的TextOutW函数显示的字符依据16位字符代码，而不是非Unicode的TextOutA函数的8位字符代码。因为这些16位字符代码含义明确，GDI可以确定目前在设备内容中选择的字体是否可显示每个字符。

在美国英语版Windows NT下执行Unicode版的KEYVIEW1多少让人感到有些迷惑，因为它所显示的就好像GDI只显示了0x0000到0x00FF之间的字符代码，而没有显示高于0x00FF的代码。也就是说，只是在字符代码和系统字体中256个字符之间简单的一对一映射。

然而，如果安装了希腊或者俄语版的Windows NT，您将发现情况就大不一样了。例如，如果安装了希腊版的Windows NT，则美国英语、德语、希腊语和俄语键盘将会产生与美国英语版Windows NT同样的Unicode字符代码。不过，希腊版的Windows NT将不显示德语重音字符或者俄语字符，因为这些字符并不在希腊系统字体中。同样，俄语版的Windows NT也不显示德语重音字符或者希腊字符，因为这些字符也不在俄语系统字体中。

其中，Unicode版的KEYVIEW1的区别在日语版Windows NT下更具戏剧性。您从IME输入日文字符，这些字符可以正确显示。唯一的问题是格式：因为日文字符通常看起来非常复杂，它们的显示宽度是其它字符的两倍。

TrueType 和大字体

我们使用的点阵字体（在日文版Windows中带有附加字体）最多包括256个字符。这是我们所希望的，因为当假定字符代码是8位时，点阵字体文件的格式就跟早期Windows时代的样子一样了。这就是为什么当我们使用SYSTEM_FONT或者SYSTEM_FIXED_FONT时，某些语言中一些字符总不能正确显示（日本系统字体有点不同，因为它是双字节字符集；大多数字符实际上保存在TrueType集合文件中，文件扩展名是.TTC）。

TrueType字体包含的字符可以多于256个。并不是所有TrueType字体中的字符都多于256个，但Windows 98和Windows NT中的字体包含多于256个字符。或者，安装了多语系支持后，TrueType字体中也包含多于256个字符。在「 **控制台**」的「**新增 /删除程序**」中，单击「**Windows安装程序**」页面卷标，并确保选中了「 **多语系支持**」。这个多语系支持包括五个字符集：波罗的海语系、中欧语系、斯拉夫语系、希腊语系和土耳其语系。波罗的海语系字符集用于爱沙尼亚语、拉脱维亚语和立陶宛语。中欧字符集用于阿尔巴尼亚语、捷克语、克罗地亚语、匈牙利语、波兰语、罗马尼亚语、斯洛伐克语和斯洛文尼亚语。斯拉夫字符集用于保加利亚语、白俄罗斯语、俄语、塞尔维亚语和乌克兰语。

Windows 98中的TrueType字体支持这五种字符集，再加上西欧（ANSI）字符集，西欧字符集实际上用于其它所有语言，但远东语言（汉语、日语和朝鲜语）除外。支持多种字符集的TrueType字体有时也称为「大字体」。在这种情况下的「大」并不是指字符的大小，而是指数量。

即使在非Unicode程序中也可利用大字体，这意味着可以用大字体显示几种不同字母表中的字符。然而，为了要将得到的字体选进设备内容，还需要GetStockObject以外的函数。

函数CreateFont和CreateFontIndirect建立了一种逻辑字体，这与CreatePen建立逻辑画笔以及CreateBrush建立逻辑画刷的方式类似。CreateFont用14个参数描述要建立的字体。CreateFontIndirect只有一个参数，但该参数是指向LOGFONT结构的指针。LOGFONT结构有14个字段，分别对应于CreateFont函数的参数。我将在第十七章详细讨论这些函数。现在，让我们看一下CreateFont函数，但我们只注意其中两个参数，其它参数都设定为0。

如果需要等宽字体（就像KEYVIEW1程序中使用的），将CreateFont的第13个参数设定为FIXED_PITCH。如果需要非内定字符集的字体（这也是我们所需要的），将CreateFont的第9个参数设定为某个「字符集ID」。此字符集ID将是WINGDI.H中定义的下列值之一。我已给出注释，指出和这些字符集相关的代码页：

#define ANSI_CHARSET	0	// 1252 Latin 1 (ANSI)
#define DEFAULT_CHARSET	1	
#define SYMBOL_CHARSET	2	
#define MAC_CHARSET	77	
#define SHIFTJIS_CHARSET	128	// 932 (DBCS，日本)
#define HANGEUL_CHARSET	129	// 949 (DBCS，韩文)
#define HANGUL_CHARSET	129	//""
#define JOHAB_CHARSET	130	// 1361 (DBCS，韩文)
#define GB2312_CHARSET	134	// 936 (DBCS，简体中文)
#define CHINESEBIG5_CHARSET	136	// 950 (DBCS，繁体中文)
#define GREEK_CHARSET	161	// 1253希腊文
#define TURKISH_CHARSET	162	// 1254 Latin 5 (土耳其文)
#define	163	// 1258越南文

VIETNAMESE_CHARSET		
#define HEBREW_CHARSET	177	// 1255希伯来文
#define ARABIC_CHARSET	178	// 1256阿拉伯文
#define BALTIC_CHARSET	186	// 1257波罗的海字集
#define RUSSIAN_CHARSET	204	// 1251俄文（斯拉夫语系）
#define THAI_CHARSET	222	// 874泰文
#define EASTEUROPE_CHARSET	238	// 1250 Latin 2 (中欧语系)
#define OEM_CHARSET	255	// 地区自订

为什么Windows对同一个字符集有两个不同的ID：字符集ID和代码页ID？这只是Windows中的一种怪癖。注意，字符集ID只需要1字节的储存空间，这是LOGFONT结构中字符集字段的大小（试回忆Windows 1.0时期，内存和储存空间有限，每个字节都必须斤斤计较）。注意，有许多不同的MS-DOS代码页用于其它国家，但只有一种字符集ID – OEM_CHARSET – 用于MS-DOS字符集。

您还会注意到，这些字符集的值与STOKFONT程序最上头的「CharSet」值一致。在美国英语版Windows中，我们看到常备字体的字符集ID是0 （ANSI_CHARSET）和255（OEM_CHARSET）。希腊版Windows中的是161（GREEK_CHARSET），在俄语版中的是204（RUSSIAN_CHARSET），在日语版中是128（SHIFTJIS_CHARSET）。

在上面的代码中，DBCS代表双字节字符集，用于远东版的Windows。其它版的Windows不支持DBCS字体，因此不能使用那些字符集ID。

CreateFont传回HFONT值 – 逻辑字体的句柄。您可以使用SelectObject将此字体选进设备内容。实际上，您必须呼叫DeleteObject来删除您建立的所有逻辑字体。

大字体解决方案的其它部分是WM_INPUTLANGCHANGE消息。一旦您使用桌面下端的弹出式菜单来改变键盘布局，Windows都会向您的窗口消息处理程序发送WM_INPUTLANGCHANGE消息。wParam消息参数是新键盘布局的字符集ID。

程序6-4所示的KEYVIEW2程序实作了键盘布局改变时改变字体的逻辑。

程序6-4 KEYVIEW2

KEYVIEW2.C

```
/*------------------------------------------------------------------------
KEYVIEW2.C -- Displays Keyboard and Character Messages
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("KeyView2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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= (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName= NULL ;
  wndclass.lpszClassName= szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Keyboard Message Viewer #2"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static DWORD dwCharSet = DEFAULT_CHARSET ;
  static int cxClientMax, cyClientMax, cxClient, cyClient, cxChar, cyChar ;
  static int cLinesMax, cLines ;
  static PMSG pmsg ;
  static RECT rectScroll ;
  static TCHAR szTop[] = TEXT ("Message Key Char ")
    TEXT ("Repeat Scan Ext ALT Prev Tran") ;
  static TCHAR szUnd[] = TEXT ("_______ ___ ____ ")
    TEXT ("______ ____ ___ ___ ____ ____") ;

  static TCHAR * szFormat[2] = {
    TEXT ("%-13s %3d %-15s%c%6u %4d %3s %3s %4s %4s"),
      TEXT ("%-13s 0x%04X%1s%c %6u %4d %3s %3s %4s %4s") } ;

    static TCHAR * szYes = TEXT ("Yes") ;
    static TCHAR * szNo = TEXT ("No") ;
    static TCHAR * szDown= TEXT ("Down") ;
    static TCHAR * szUp = TEXT ("Up") ;

    static TCHAR * szMessage [] = {
      TEXT ("WM_KEYDOWN"), TEXT ("WM_KEYUP"),
        TEXT ("WM_CHAR"), TEXT ("WM_DEADCHAR"),
        TEXT ("WM_SYSKEYDOWN"), TEXT ("WM_SYSKEYUP"),
        TEXT ("WM_SYSCHAR"), TEXT ("WM_SYSDEADCHAR") } ;
      HDC hdc ;
      int i, iType ;
      PAINTSTRUCT ps ;
      TCHAR szBuffer[128], szKeyName [32] ;
      TEXTMETRIC tm ;

      switch (message)
      {
      case WM_INPUTLANGCHANGE:
        dwCharSet = wParam ;
        // fall through
      case WM_CREATE:
```

```
    case WM_DISPLAYCHANGE:
      // Get maximum size of client area
      cxClientMax = GetSystemMetrics (SM_CXMAXIMIZED) ;
      cyClientMax = GetSystemMetrics (SM_CYMAXIMIZED) ;

      // Get character size for fixed-pitch font
      hdc = GetDC (hwnd) ;
      SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
        dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
      GetTextMetrics (hdc, &tm) ;
      cxChar = tm.tmAveCharWidth ;
      cyChar = tm.tmHeight ;

      DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
      ReleaseDC (hwnd, hdc) ;

      // Allocate memory for display lines
      if (pmsg)
        free (pmsg) ;
      cLinesMax = cyClientMax / cyChar ;
      pmsg = (MSG*)malloc (cLinesMax * sizeof (MSG)) ;
      cLines = 0 ;
      // fall through
    case WM_SIZE:
      if (message == WM_SIZE)
      {
        cxClient = LOWORD (lParam) ;
        cyClient = HIWORD (lParam) ;
      }
      // Calculate scrolling rectangle

      rectScroll.left = 0 ;
      rectScroll.right = cxClient ;
      rectScroll.top = cyChar ;
      rectScroll.bottom = cyChar * (cyClient / cyChar) ;

      InvalidateRect (hwnd, NULL, TRUE) ;

      if (message == WM_INPUTLANGCHANGE)
        return TRUE ;
      return 0 ;
    case WM_KEYDOWN:
    case WM_KEYUP:
    case WM_CHAR:
    case WM_DEADCHAR:
    case WM_SYSKEYDOWN:
    case WM_SYSKEYUP:
    case WM_SYSCHAR:
    case WM_SYSDEADCHAR:
      // Rearrange storage array
      for (i = cLinesMax - 1 ; i > 0 ; i--)
      {
        pmsg[i] = pmsg[i - 1] ;
      }
      // Store new message
      pmsg[0].hwnd = hwnd ;
      pmsg[0].message = message ;
      pmsg[0].wParam = wParam ;
      pmsg[0].lParam = lParam ;

      cLines = min (cLines + 1, cLinesMax) ;
      // Scroll up the display
      ScrollWindow (hwnd, 0, -cyChar, &rectScroll, &rectScroll) ;
      break ; // ie, call DefWindowProc so Sys messages work

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
```

```
        dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
      SetBkMode (hdc, TRANSPARENT) ;
      TextOut (hdc, 0, 0, szTop, lstrlen (szTop)) ;
      TextOut (hdc, 0, 0, szUnd, lstrlen (szUnd)) ;

      for (i = 0 ; i < min (cLines, cyClient / cyChar - 1) ; i++)
      {
        iType =pmsg[i].message == WM_CHAR ||
          pmsg[i].message == WM_SYSCHAR ||
          pmsg[i].message == WM_DEADCHAR ||
          pmsg[i].message == WM_SYSDEADCHAR ;

        GetKeyNameText (pmsg[i].lParam, szKeyName,
          sizeof (szKeyName) / sizeof (TCHAR)) ;

        TextOut (hdc, 0, (cyClient / cyChar - 1 - i) * cyChar, szBuffer,
          wsprintf ( szBuffer, szFormat [iType],
          szMessage [pmsg[i].message -
          WM_KEYFIRST],
          pmsg[i].wParam,
          (PTSTR) (iType ? TEXT (" ") : szKeyName),
          (TCHAR) (iType ? pmsg[i].wParam : ' '),
          LOWORD (pmsg[i].lParam),
          HIWORD (pmsg[i].lParam) & 0xFF,
          0x01000000 & pmsg[i].lParam ? szYes : szNo,
          0x20000000 & pmsg[i].lParam ? szYes : szNo,
          0x40000000 & pmsg[i].lParam ? szDown : szUp,
          0x80000000 & pmsg[i].lParam ? szUp : szDown));
      }
      DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

注意，键盘输入语言改变后，KEYVIEW2就清除画面并重新分配储存空间。这样做有两个原因：第一，因为KEYVIEW2并不是某种字体专用的，当输入语言改变时字体文字的大小也会改变。程序需要根据新字符大小重新计算某些变量。第二，在接收每个字符消息时，KEYVIEW2并不有效地保留字符集ID。因此，如果键盘输入语言改变了，而且KEYVIEW2需要重画显示区域时，所有的字符将用新字体显示。

第十七章将详细讨论字体和字符集。如果您想深入研究国际化问题，可以在/Platform SDK/Windows Base Services/International Features找到需要的文件，还有许多基础信息则位于/Platform SDK/Windows Base Services/General Library/String Manipulation。

插入符号（不是光标）

当您往程序中输入文字时，通常有一个底线、竖条或者方框来指示输入的下一个字符将出现在屏幕上的位置。这个标志通常称为「光标」，但是在Windows下写程序，您必须改变这个习惯。在Windows中，它称为「插入符号」。「光标」是指表示鼠标位置的那个位图图像。

插入符号函数

主要有五个插入符号函数：

CreateCaret　建立与窗口有关的插入符号

SetCaretPos 在窗口中设定插入符号的位置
ShowCaret 显示插入符号
HideCaret 隐藏插入符号
DestroyCaret 撤消插入符号

另外还有取得插入符号目前位置（GetCaretPos）和取得以及设定插入符号闪烁时间（GetCaretBlinkTime和SetCaretBlinkTime）的函数。

在Windows中，插入符号定义为水平线、与字符大小相同的方框，或者与字符同高的竖线。如果使用调和字体，例如Windows内定的系统字体，则推荐使用竖线插入符号。因为调和字体中的字符没有固定大小，水平线或方框不能设定为字符的大小。

如果程序中需要插入符号，那么您不应该简单地在窗口消息处理程序的WM_CREATE消息处理期间建立它，然后在WM_DESTROY消息处理期间撤消。其原因显而易见：一个消息队列只能支持一个插入符号。因此，如果您的程序有多个窗口，那么各个窗口必须有效地共享相同的插入符号。

其实，它并不像听起来那么多限制。您再想想就会发现，只有在窗口有输入焦点时，窗口内显示插入符号才有意义。事实上，闪烁的插入符号只是一种视觉提示：您可以在程序中输入文字。因为任何时候都只有一个窗口拥有输入焦点，所以多个窗口同时都有闪烁的插入符号是没有意义的。

通过处理WM_SETFOCUS和WM_KILLFOCUS消息，程序就可以确定它是否有输入焦点。正如名称所暗示的，窗口消息处理程序在有输入焦点的时候接收到WM_SETFOCUS消息，失去输入焦点的时候接收到WM_KILLFOCUS消息。这些消息成对出现：窗口消息处理程序在接收到WM_KILLFOCUS消息之前将一直接收到WM_SETFOCUS消息，并且在窗口打开期间，此窗口总是接收到相同数量的WM_SETFOCUS和WM_KILLFOCUS消息。

使用插入符号的主要规则很简单：窗口消息处理程序在WM_SETFOCUS消息处理期间呼叫CreateCaret，在WM_KILLFOCUS消息处理期间呼叫DestroyCaret。

这里还有几条其它规则：插入符号刚建立时是隐蔽的。如果想使插入符号可见，那么您在呼叫CreateCaret之后，窗口消息处理程序还必须呼叫ShowCaret。另外，当窗口消息处理程序处理一条非WM_PAINT消息而且希望在窗口内绘制某些东西时，它必须呼叫HideCaret隐藏插入符号。在绘制完毕后，再呼叫ShowCaret显示插入符号。HideCaret的影响具有累积效果，如果多次呼叫HideCaret而不呼叫ShowCaret，那么只有呼叫ShowCaret相同次数时，才能看到插入符号。

TYPER程序

程序6-5所示的TYPER程序使用了本章讨论的所有内容，您可以认为TYPER是一个相当简单的文字编辑器。在窗口中，您可以输入字符，用光标移动键（也可以称为插入符号移动键）来移动光标（I型标），按下Escape键清除窗口的内容等。缩放窗口、改变键盘输入语言时都会清除窗口的内容。本程序没有卷动，没有文字寻找和定位功能，不能储存文件，没有拼写检查，但它确实是写作一个文字编辑器的开始。

程序6-5 TYPER

TYPER.C

```
/*------------------------------------------------------------------------
TYPER.C -- Typing Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

#define BUFFER(x,y) *(pBuffer + y * cxBuffer + x)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
```

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Typer") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Typing Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static DWORD dwCharSet = DEFAULT_CHARSET ;
  static int cxChar, cyChar, cxClient, cyClient, cxBuffer, cyBuffer,
    xCaret, yCaret ;
  static TCHAR *pBuffer = NULL ;
  HDC hdc ;
  int x, y, i ;
  PAINTSTRUCT ps ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_INPUTLANGCHANGE:
    dwCharSet = wParam ;
    // fall through
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
      dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;

    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cyChar = tm.tmHeight ;

    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    ReleaseDC (hwnd, hdc) ;
```

```
  // fall through
case WM_SIZE:
  // obtain window size in pixels
  if (message == WM_SIZE)
  {
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
  }
  // calculate window size in characters
  cxBuffer = max (1, cxClient / cxChar) ;
  cyBuffer = max (1, cyClient / cyChar) ;

  // allocate memory for buffer and clear it
  if (pBuffer != NULL)
    free (pBuffer) ;

  pBuffer = (TCHAR *) malloc (cxBuffer * cyBuffer * sizeof (TCHAR)) ;

  for (y = 0 ; y < cyBuffer ; y++)
    for (x = 0 ; x < cxBuffer ; x++)
      BUFFER(x,y) = ' ' ;
    // set caret to upper left corner

    xCaret = 0 ;
    yCaret = 0 ;

    if (hwnd == GetFocus ())
      SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
case WM_SETFOCUS:
  // create and show the caret
  CreateCaret (hwnd, NULL, cxChar, cyChar) ;
  SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;
  ShowCaret (hwnd) ;
  return 0 ;
case WM_KILLFOCUS:
  // hide and destroy the caret
  HideCaret (hwnd) ;
  DestroyCaret () ;
  return 0 ;
case WM_KEYDOWN:
  switch (wParam)
  {
  case VK_HOME:
    xCaret = 0 ;
    break ;
  case VK_END:
    xCaret = cxBuffer - 1 ;
    break ;
  case VK_PRIOR:
    yCaret = 0 ;
    break ;
  case VK_NEXT:
    yCaret = cyBuffer - 1 ;
    break ;
  case VK_LEFT:
    xCaret = max (xCaret - 1, 0) ;
    break ;
  case VK_RIGHT:
    xCaret = min (xCaret + 1, cxBuffer - 1) ;
    break ;
  case VK_UP:
    yCaret = max (yCaret - 1, 0) ;
    break ;
  case VK_DOWN:
    yCaret = min (yCaret + 1, cyBuffer - 1) ;
    break ;
```

```
    case VK_DELETE:
      for (x = xCaret ; x < cxBuffer - 1 ; x++)
        BUFFER (x, yCaret) = BUFFER (x + 1, yCaret) ;
      BUFFER (cxBuffer - 1, yCaret) = ' ' ;
      HideCaret (hwnd) ;
      hdc = GetDC (hwnd) ;
      SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
        dwCharSet, 0, 0, 0,FIXED_PITCH, NULL)) ;
      TextOut (hdc, xCaret * cxChar, yCaret * cyChar,
        & BUFFER (xCaret, yCaret),
        cxBuffer - xCaret) ;
      DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
      ReleaseDC (hwnd, hdc) ;
      ShowCaret (hwnd) ;
      break ;
    }
    SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;
    return 0 ;
    case WM_CHAR:
      for (i = 0 ; i < (int) LOWORD (lParam) ; i++)
      {
        switch (wParam)
        {
        case '\b': // backspace
          if (xCaret > 0)
          {
            xCaret-- ;
            SendMessage (hwnd, WM_KEYDOWN, VK_DELETE, 1) ;
          }
          break ;
        case '\t': // tab
          do
          {
            SendMessage (hwnd, WM_CHAR, ' ', 1) ;
          }
          while (xCaret % 8 != 0) ;
          break ;
        case '\n': // line feed
          if (++yCaret == cyBuffer)
            yCaret = 0 ;
          break ;
        case '\r': // carriage return
          xCaret = 0 ;
          if (++yCaret == cyBuffer)
            yCaret = 0 ;
          break ;
        case '\x1B': // escape
          for (y = 0 ; y < cyBuffer ; y++)
            for (x = 0 ; x < cxBuffer ; x++)
              BUFFER (x, y) = ' ' ;
            xCaret = 0 ;
            yCaret = 0 ;
            InvalidateRect (hwnd, NULL, FALSE) ;
            break ;
        default: // character codes
          BUFFER (xCaret, yCaret) = (TCHAR) wParam ;

          HideCaret (hwnd) ;
          hdc = GetDC (hwnd) ;
          SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
            dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
          TextOut (hdc, xCaret * cxChar, yCaret * cyChar,
            & BUFFER (xCaret, yCaret), 1) ;
          DeleteObject (
            SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
          ReleaseDC (hwnd, hdc) ;
          ShowCaret (hwnd) ;

          if (++xCaret == cxBuffer)
```

```
        {
          xCaret = 0 ;
          if (++yCaret == cyBuffer)
            yCaret = 0 ;
        }
        break ;
      }
    }

    SetCaretPos (xCaret * cxChar, yCaret * cyChar) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, CreateFont (0, 0, 0, 0, 0, 0, 0, 0,
      dwCharSet, 0, 0, 0, FIXED_PITCH, NULL)) ;
    for (y = 0 ; y < cyBuffer ; y++)
      TextOut (hdc, 0, y * cyChar, & BUFFER(0,y), cxBuffer) ;
    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

为了简单起见，TYPER程序使用一种等宽字体，因为编写处理调和字体的文字编辑器要困难得多。程序在好几个地方取得设备内容：在WM_CREATE消息处理期间，在WM_KEYDOWN消息处理期间，在WM_CHAR消息处理期间以及在WM_PAINT消息处理期间，每次都通过GetStockObject和SelectObject呼叫来选择等宽字体。

在WM_SIZE消息处理期间，TYPER计算窗口的字符宽度和高度并把值保存在cxBuffer和cyBuffer变量中，然后使用malloc分配缓冲区以保存在窗口内输入的所有字符。注意，缓冲区的字节大小取决于cxBuffer、cyBuffer和sizeof（TCHAR），它可以是1或2，这依赖于程序是以8位的字符处理还是以Unicode方式编译的。

xCaret和yCaret变量保存插入符号位置。在WM_SETFOCUS消息处理期间，TYPER呼叫CreateCaret来建立与字符有相同宽度和高度的插入符号，呼叫SetCaretPos来设定插入符号的位置，呼叫ShowCaret使插入符号可见。在WM_KILLFOCUS消息处理期间，TYPER呼叫HideCaret和DestroyCaret。

对WM_KEYDOWN的处理大多要涉及光标移动键。Home和End把插入符号送至一行的开始和末尾处，Page Up和Page Down把插入符号送至窗口的顶端和底部，箭头的用法不变。对Delete键，TYPER将缓冲区中从插入符号之后的那个位置开始到行尾的所有内容向前移动，并在行尾显示空格。

WM_CHAR处理Backspace、Tab、Linefeed（Ctrl-Enter）、Enter、Escape和字符键。注意，在处理WM_CHAR消息时（假设使用者输入的每个字符都非常重要），我使用了lParam中的重复计数；而在处理WM_KEYDOWN消息时却不这么作（避免有害的重复卷动）。对Backspace和Tab的处理由于使用了SendMessage函数而得到简化，Backspace与Delete做法相仿，而Tab则如同输入了若干个空格。

前面我已经提到过，在非WM_PAINT消息处理期间，如果要在窗口中绘制内容，则应该隐蔽光标。TYPER为Delete键处理WM_KEYDOWN消息和为字符键处理WM_CHAR消息时即是如此。在这两种情况下，TYPER改变缓冲区中的内容，然后在窗口中绘制一个或者多个新字符。

虽然TYPER使用了与KEYVIEW2相同的做法以在字符集之间切换（就像使用者切换键盘布局一

样），但对于远东版的Windows，它还是不能正常工作。TYPER不允许使用两倍宽度的字符。此问题将在第十七章讨论，那时我们将详细讨论字体与文字输出。

第七章 鼠标

鼠标是有一个或多个键的定位设备。虽然也可以使用诸如触摸画面和光笔之类的输入设备，但是只有鼠标以及常用在膝上型计算机上的轨迹球等才是渗透了PC市场的唯一输入设备。

情况并非总是如此。当然，Windows的早期开发人员认为他们不应该要求使用者为了执行其产品而必须买只鼠标。因此，他们将鼠标作为一种选择性的附加设备，而为Windows中的所有操作以及applet提供一种键盘接口（例如，查看Windows小算盘程序的在线说明信息，可以看到每个按钮都提供了一个同等功效的键盘操作方式）。第三方软件开发人员使用键盘接口来提供与鼠标操作相同的功能，这本书以前的版本也是这么做的。

理论上来说，现在的Windows需要鼠标。至少，一些消息框是这样讲的。当然，您也可以拔下鼠标，而且Windows仍然可以执行良好（只有消息框会提示您没有连接鼠标）。试图不用鼠标来使用Windows就像用脚趾来弹钢琴一样（至少在最初的一段时间里是这样），但您依然可以这样做。正因为如此，我还是喜欢为鼠标功能提供键盘操作。打字员尤其喜欢让他们的手保持在键盘上，并且我认为每个人都有在杂乱的桌上找不到鼠标，或者鼠标移动不灵敏的经验。使用键盘通常不需要花费更多的精力和努力，并且为喜欢使用键盘的人提供更多的功能。

我们通常认为，键盘便于输入和操作文字数据，而鼠标则便于画图和操作图形对象。实际上，本章大多数的范例程序都画了一些图形，并且用到了我们在第五章所学到的知识。

鼠标基础

Windows 98能支持单键、双键或者三键鼠标，也可以使用摇杆或者光笔来仿真单键鼠标。早期，由于许多使用者都有单键鼠标，所以Windows应用程序总是避免使用双键或三键鼠标。不过，由于双键鼠标已经成为事实上的标准，因此不使用第二个键的传统已经不再合理了。当然，第二个鼠标按键是用于启动一个「快捷菜单」，亦即出现在普通菜单列之外的窗口中菜单，或者用于特殊的拖曳操作（拖曳将在后面加以解释）。然而，程序不能依赖双键鼠标。

理论上，您可以用我们的老朋友GetSystemMetrics函数来确认鼠标是否存在：

fMouse = GetSystemMetrics (SM_MOUSEPRESENT) ;

如果已经安装了鼠标，fMouse将传回TRUE（非0）；如果没有安装，则传回0。然而，在Windows 98中，不论鼠标是否安装，此函数都将传回TRUE 。在Microsoft Windows NT中，它可以正常工作。

要确定所安装鼠标其上按键的个数，可使用

cButtons = GetSystemMetrics (SM_CMOUSEBUTTONS) ;

如果没有安装鼠标，那么函数将传回0。然而，在Windows 98下，如果没有安装鼠标，此函数将传回2。

习惯用左手的使用者可以使用Windows的「控制台」来切换鼠标按键。虽然应用程序可以通过在GetSystemMetrics中使用SM_SWAPBUTTON参数来确定是否进行了这种切换，但通常没有这个必要。由食指触发的键被认为是左键，即使事实上是位于鼠标的右边。不过，在一个教育训练程序中，您可能想在屏幕上画一个鼠标，在这种情况下，您可能想知道鼠标按键是否被切换过了。

您可以在「控制台」中设定鼠标的其它参数，例如双击速度。从Windows应用程序，通过使用SystemParametersInfo函数可以设定或获得此项信息。

一些简单的定义

当Windows使用者移动鼠标时，Windows在显示器上移动一个称为「鼠标光标」的小位图。鼠标光标有一个指向显示器上精确位置的单像素「热点」。当我提到鼠标光标在屏幕上的位置时，指的是热点的位置。

Windows支持几种预先定义的鼠标光标，程序可以使用这些光标。最常见的是称为IDC_ARROW的斜箭头（在WINUSER.H中定义）。热点在箭头的顶端。IDC_CROSS光标（在本章后面的BLOKOUT程序中有用到）的热点在十字交叉线的中心。IDC_WAIT光标是一个沙漏，通常用于指示程序正在执行。程序写作者也可以设计自己的光标。我们将在第十章学习设计方法。在定义窗口类别结构时指定特定窗口的内定光标，例如：

```
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
```

下面是一些描述鼠标按键动作的术语：

Clicking按下并放开一个鼠标按键。

Double-clicking快速按下并放开鼠标按键两次。

Dragging按住鼠标按键并移动鼠标。

对三键鼠标来说，三个键分别称为左键、中键、右键。在Windows表头文件中定义的与鼠标有关的标识符使用缩写LBUTTON、MBUTTON和RBUTTON。双键鼠标只有左键与右键，单键鼠标只有一个左键。

鼠标(Mouse)的复数

现在，为了展现我的勇气，我将面对输入设备最难辩的争论话题：什么是「mouse」的复数。虽然每个人都知道多只啮齿动物称为mice，似乎没有人对该如何称呼多个输入设备有最后的答案。不管「mice」或「mouse」听起来都不对劲。我惯常参考的《American Heritage Dictionary of the English Language》第三版则只字未提。

《Wired style：Principles of English Usage in the Digital Age》（HardWired, 1996）指出「mouse」比较好，以避免与啮齿动物搞混。在1964发明鼠标的Doug Engelbart对此争议也帮不上忙。我曾经问过他mouse的复数是什么，他说我不知道。

最后，高权威的Microsoft Manual of Style for Technical Publications告诉我们「避免使用复数mice。假如你必须提到多只mouse，使用mouse devices」。这听起来像是在逃避问题，但当一切听起来都不对劲时，它确实是个明智的忠告了。事实上，大部分需要mouse复数的句子都能重新修改来避开。例如，试着说"People use the almost as much as keyboard"，而不是"Pople use mice almost as much as keyboards"。

显示区域鼠标消息

在前一章中您已经看到，Windows只把键盘消息发送给拥有输入焦点的窗口。鼠标消息与此不同：只要鼠标跨越窗口或者在某窗口中按下鼠标按键，那么窗口消息处理程序就会收到鼠标消息，而不管该窗口是否活动或者是否拥有输入焦点。Windows为鼠标定义了21种消息，不过，其中有11个消息和显示区域无关（下面称之为「非显示区域」消息），Windows程序经常忽略这些消息。

当鼠标移过窗口的显示区域时，窗口消息处理程序收到WM_MOUSEMOVE消息。当在窗口的显示区域中按下或者释放一个鼠标按键时，窗口消息处理程序会接收到下面这些消息：

表7-1

键	按下	释放	按下(双键)
左	WM_LBUTTONDOWN	WM_LBUTTONUP	WM_LBUTTONDBLCLK
中	WM_MBUTTONDOWN	WM_MBUTTONUP	WM_MBUTTONDBLCLK
右	WM_RBUTTONDOWN	WM_RBUTTONUP	WM_RBUTTONDBLCLK

只有对三键鼠标，窗口消息处理程序才会收到MBUTTON消息；只有对双键或者三键鼠标，才会接收到RBUTTON消息。只有当定义的窗口类别能接收DBLCLK（双击）消息，窗口消息处理程序才能接收到这些消息（请参见本章中「双击鼠标按键」一节）。

对于所有这些消息来说，其lParam值均含有鼠标的位置：低字组为x坐标，高字组为y坐标，这两个坐标是相对于窗口显示区域左上角的位置。您可以用LOWORD和HIWORD宏来提取这些值：

x = LOWORD (lParam) ;

y = HIWORD (lParam) ;

wParam的值指示鼠标按键以及Shift和Ctrl键的状态。您可以使用表头文件WINUSER.H中定义的位屏蔽来测试wParam。MK前缀代表「鼠标按键」。

MK_LBUTTON	按下左键
MK_MBUTTON	按下中键
MK_RBUTTON	按下右键
MK_SHIFT	按下Shift键
MK_CONTROL	按下Ctrl键

例如，如果收到了WM_LBUTTONDOWN消息，而且值

wparam & MK_SHIFT

是TRUE（非0），您就知道当左键按下时也按下了Shift键。

当您把鼠标移过窗口的显示区域时，Windows并不为鼠标的每个可能的像素位置都产生一个WM_MOUSEMOVE消息。您的程序接收到WM_MOUSEMOVE消息的次数，依赖于鼠标硬件，以及您的窗口消息处理程序在处理鼠标移动消息时的速度。换句话说，Windows不能用未处理的WM_MOUSEMOVE消息来填入消息队列。当您执行下面将描述的CONNECT程序时，您将会更了解WM_MOUSEMOVE消息处理的速率。

如果您在非活动窗口的显示区域中按下鼠标左键，Windows将把活动窗口改为在其中按下鼠标按键的窗口，然后把WM_LBUTTONDOWN消息送到该窗口消息处理程序。当窗口消息处理程序得到WM_LBUTTONDOWN消息时，您的程序就可以安全地假定该窗口是活动化的了。不过，您的窗口消息处理程序可能在未接收到WM_LBUTTONDOWN消息的情况下先接收到了WM_LBUTTONUP的消息。如果在一个窗口中按下鼠标按键，然后移动到使用者窗口释放它，就会出现这种情况。类似的情况，当鼠标按键在另一个窗口中被释放时，窗口消息处理程序只能接收到WM_LBUTTONDOWN消息，而没有相应的WM_LBUTTONUP消息。

这些规则有两个例外：

窗口消息处理程序可以「拦截鼠标」并且连续地接收鼠标消息，即使此时鼠标在该窗口显示区域之外。您将在本章的后面学习如何拦截鼠标。

如果正在显示一个系统模态消息框或者系统模态对话框，那么其它程序就不能接收鼠标消息。当系统模态消息框或者对话框活动时，禁止切换到其它窗口或者程序。一个显示系统模态消息框的例子，是当您关闭Windows时。

简单的鼠标处理：一个例子

程序7-1中所示的CONNECT程序能作一些简单的鼠标处理，使您对Windows如何向您的程序发送鼠标消息有一些体会。

程序7-1 CONNECT

CONNECT.C

```
/*------------------------------------------------------------------------
CONNECT.C -- Connect-the-Dots Mouse Demo Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define MAXPOINTS 1000
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Connect") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Connect-the-Points Mouse Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }

  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
```

```
  static POINT pt[MAXPOINTS] ;
  static int iCount ;
  HDC hdc ;
  in i, j ;
  PAINTSTRUCT ps ;
  switch (message)
  {
  case WM_LBUTTONDOWN:
    iCount = 0 ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_MOUSEMOVE:
    if (wParam & MK_LBUTTON && iCount < 1000)
    {
      pt[iCount ].x = LOWORD (lParam) ;
      pt[iCount++].y = HIWORD (lParam) ;

      hdc = GetDC (hwnd) ;
      SetPixel (hdc, LOWORD (lParam), HIWORD (lParam), 0) ;
      ReleaseDC (hwnd, hdc) ;
    }
    return 0 ;
  case WM_LBUTTONUP:
    InvalidateRect (hwnd, NULL, FALSE) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;
    for (i = 0 ; i < iCount - 1 ; i++)
      for (j = i + 1 ; j < iCount ; j++)
      {
        MoveToEx (hdc, pt[i].x, pt[i].y, NULL) ;
        LineTo (hdc, pt[j].x, pt[j].y) ;
      }

      ShowCursor (FALSE) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CONNECT处理三个鼠标消息：

WM_LBUTTONDOWNCONNECT 清除显示区域。

WM_MOUSEMOVE如果按下左键，那么CONNECT就在显示区域中的鼠标位置处绘制一个黑点，并保存该坐标。

WM_LBUTTONUP CONNECT把显示区域中绘制的点与其它每个点连接起来。有时会产生一个漂亮的图形，有时则会是黑鸦鸦的一团糟（见图7-1）。

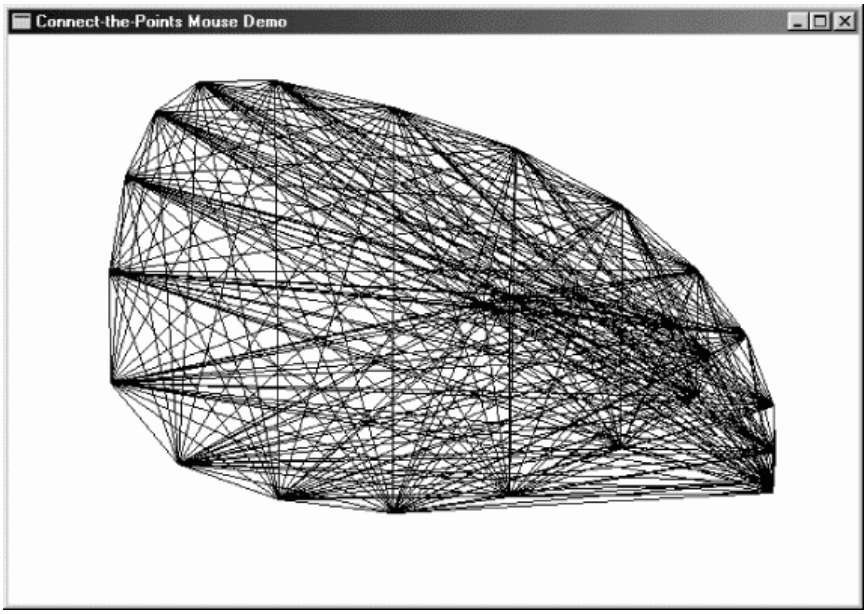


图7-1 CONNECT的屏幕显示

CONNECT的使用方法：把鼠标光标移动到显示区域中，按下左键，移动一下位置，释放左键。对几个构成曲线的点，CONNECT能处理得很好，方法是按住左键，快速移动鼠标，这样就可以绘制出该曲线图案。

CONNECT使用了三个简单的图形设备接口(GDI)函数，我在第五章讨论过这些函数。当鼠标左键按下时，SetPixel为每个WM_MOUSEMOVE消息绘制一个黑像素（对于高分辨率的显示器，像素几乎看不见）。画直线需要MoveToEx和LineTo函数。

如果您在释放鼠标按键之前把鼠标光标移到显示区域之外，那么CONNECT就不会连接这些点，因为它没有收到WM_LBUTTONUP消息。如果您把鼠标移回显示区域内并按下左键，那么CONNECT将清除显示区域。如果想在显示区域外释放左键后还继续进行画图，那么可以在显示区域外按下鼠标再移回显示区域中。

CONNECT最多可以保存1000个点。设点数为P，则CONNECT画的线数就等于P × (P - 1) / 2。如果有1000个点，则要绘制50万条直线，大约需要几分钟才能画完(时间的长短取决于您的硬设备)。由于Windows 98是一种优先权式多任务环境，因此您可以在这一段时间切换到别的程序中。但是，当程序正在忙的时候，您将无法对CONNECT程序做任何事（诸如移动或者缩放等）。在第二十章中，我们将讨论解决这一问题的方法。

因为CONNECT可能会花一些时间来绘制直线，因此在处理WM_PAINT消息时它将切换到沙漏光标，然后再恢复原状。这要求使用两个现有光标来呼叫SetCursor。CONNECT还呼叫两次ShowCursor，一次用TRUE参数，另一次用FALSE参数。我将在本章的后面，「使用键盘仿真鼠标」一节中更详细地讨论这些呼叫。

有时，我们使用「跟踪」这个词代表程序处理鼠标移动的方法。但是，跟踪并不意味着，程序在窗口消息处理程序中的某个循环里，不断跟随鼠标在显示器上的运动。实际上，窗口消息处理程序处理每条鼠标消息，然后迅速退出。

处理Shift键

当CONNECT接收到一个WM_MOUSEMOVE消息时，它把wParam和MK_LBUTTON进行位与（AND）运算，来确定是否按下了左键。wParam也可以用于确定Shift键的状态。例如，如果处理必须依赖于Shift和Ctrl键的状态，那么您可以使用如下所示的方法：

```
if (wParam & MK_SHIFT)
{
  if (wParam & MK_CONTROL)
  {
    //按下了Shift和Ctrl键
  }
  else
  {
    //按下了Shift键
  }
}
else
{
  if (wParam & MK_CONTROL)
  {
    //按下了Ctrl键
  }
  else
  {
    //Shift和Ctrl键均未按下
  }
}
```

如果您想在程序中同时使用左右键，同时如果您还希望只有单键鼠标的使用者也能使用您的程序，那么您可以这样来写作程序：Shift与左键的组合使用等效于右键。在这种情况下，对鼠标按键的处理可以采用如下所示的方法：

```
case WM_LBUTTONDOWN:
  if (!(wParam & MK_SHIFT))
  {
    //处理左键
    return 0 ;
  }
  // Fall through
case WM_RBUTTONDOWN:
  //处理右键
  return 0 ;
```

Windows函数GetKeyState（在第六章中介绍过）可以使用虚拟键码VK_LBUTTON、VK_RBUTTON、VK_MBUTTON、VK_SHIFT和VK_CONTROL来传回鼠标按键与Shift键的状态。如果GetKeyState传回负值，则说明已按下了鼠标按键或者Shift键。因为GetKeyState传回目前正在处理的鼠标按键或者Shift键的状态，所以全部状态信息与相应的消息都是同步的。但是，正如不能把GetKeyState用于尚未按下的键一样，您也不能为尚未按下的鼠标按键呼叫GetKeyState。请不要这样做：

while (GetKeyState (VK_LBUTTON) >= 0) ; // WRONG !!!

只有在您呼叫GetKeyState期间处理消息时，而左键已经按下，才会报告键已经按下的消息。

双击鼠标按键

双击鼠标按键是指在短时间内单击两次。要确定为双击，则这两次单击必须发生在其相距的实际位置十分接近的状况下（内定范围是一个平均系统字体字符的宽，半个字符的高），并且发生在指定的时间间隔（称为「双击速度」）内。您可以在「控制台」中改变时间间隔。

如果希望您的窗口消息处理程序能够收到双按键的鼠标消息，那么在呼叫RegisterClass初始化窗口类别结构时，必须在窗口风格中包含CS_DBLCLKS标识符：

wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS ;

如果在窗口风格中未包含CS_DBLCLKS，而使用者在短时间内双击了鼠标按键，那么窗口消息处理程序会接收到下面这些消息：

WM_LBUTTONDOWN

WM_LBUTTONUP

WM_LBUTTONDOWN

WM_LBUTTONUP

窗口消息处理程序可能在这些键的消息之前还收到了其它消息。如果您想实作自己的双击处理，那么您可以使用Windows函数GetMessageTime取得WM_LBUTTONDOWN消息之间的相对时间。第八章将更详细地讨论这个函数。

如果您的窗口类别风格中包含了CS_DBLCLKS，那么双击时窗口消息处理程序将收到如下消息：

WM_LBUTTONDOWN

WM_LBUTTONUP

WM_LBUTTONDBLCLK

WM_LBUTTONUP

WM_LBUTTONDBLCLK消息简单地替换了第二个WM_LBUTTONDOWN消息。

如果双击中的第一次键操作完成单击的功能，那么双击这一消息是很容易处理的。第二次按键（WM_LBUTTONDBLCLK消息）则完成第一次按键以外的事情。例如，看看Windows Explorer中是如何用鼠标来操作文件列表的。按一次键将选中文件，Windows Explorer用反白显示列指出被选择文件的位置。双击则实作两个功能：第一次是单击那个选中文件；第二次则指向Windows Explorer以打开该文件。执行方式相当简单，如果双击中的第一次按键不执行单击功能，那么鼠标处理方式会变得非常复杂。

非显示区域鼠标消息

在窗口的显示区域内移动或按下鼠标按键时，将产生10种消息。如果鼠标在窗口的显示区域之外但还在窗口内，Windows就给窗口消息处理程序发送一条「非显示区域」鼠标消息。窗口非显示区域包括标题栏、菜单和窗口滚动条。

通常，您不需要处理非显示区域鼠标消息，而是将这些消息传给DefWindowProc，从而使Windows执行系统功能。就这方面来说，非显示区域鼠标消息类似于系统键盘消息WM_SYSKEYDOWN、WM_SYSKEYUP和WM_SYSCHAR。

非显示区域鼠标消息几乎完全与显示区域鼠标消息相对应。消息中含有字母「NC」以表示是非显示区域消息。如果鼠标在窗口的非显示区域中移动，那么窗口消息处理程序会接收到WM_NCMOUSEMOVE消息。鼠标按键产生如表7-2所示的消息。

表7-2

键	按下	释放	按下（双击）
左	WM_NCLBUTTONDOWN	WM_NCLBUTTONUP	WM_NCLBUTTONDBLCLK
中	WM_NCMBUTTONDOWN	WM_NCMBUTTONUP	WM_NCMBUTTONDBLCLK
右	WM_NCRBUTTONDOWN	WM_NCRBUTTONUP	WM_NCRBUTTONDBLCLK

对非显示区域鼠标消息，wParam和lParam参数与显示区域鼠标消息的wParam和lParam参

数不同。wParam参数指明移动或者按鼠标按键的非显示区域。它设定为WINUSER.H中定义的以HT开头的标识符之一（HT表示「命中测试」）。

lParam参数的低位word为x坐标，高位word为y坐标，但是，它们是屏幕坐标，而不是像显示区域鼠标消息那样指的是显示区域坐标。对屏幕坐标，显示器左上角的x和y的值为0。当往右移时x的值增加，往下移时y的值增加（见图7-2）。

您可以用两个Windows函数将屏幕坐标转换为显示区域坐标或者反之：

ScreenToClient (hwnd, &pt) ;

ClientToScreen (hwnd, &pt) ;

这里pt是POINT结构。这两个函数转换了保存在结构中的值，而且没有保留以前的值。注意，如果屏幕坐标点在窗口显示区域的上面或者左边，显示区域坐标x或y值就是负值。

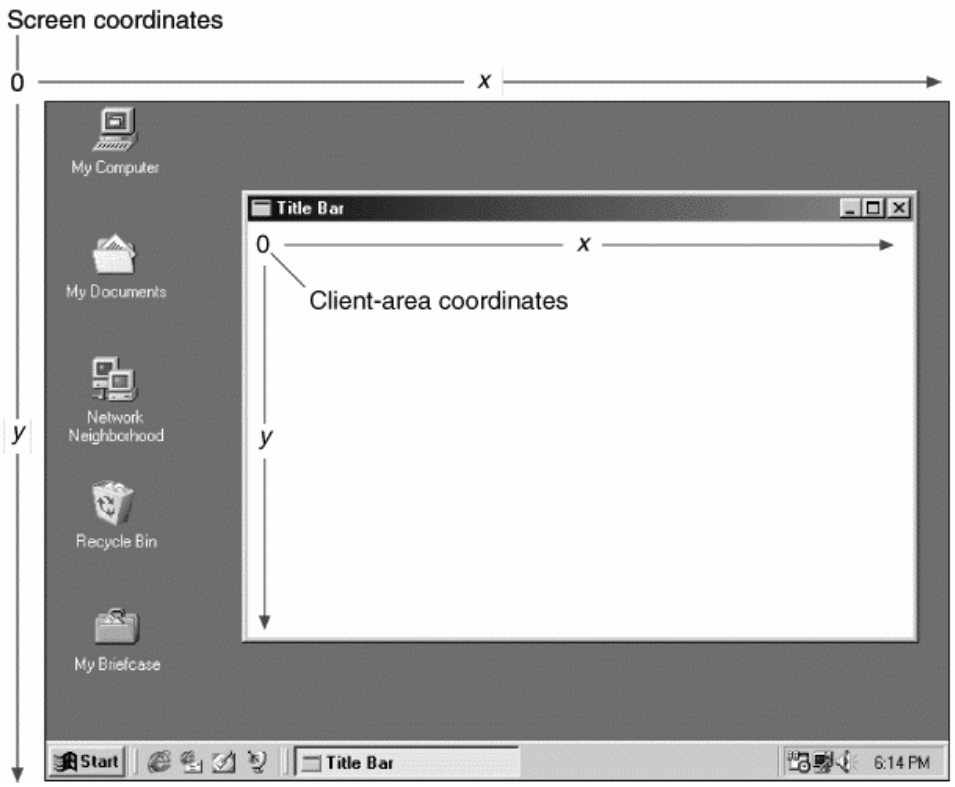


图7-2 屏幕坐标与客户显示区域坐标

命中测试消息

如果您数一下，就可以知道我们已经介绍了21个鼠标消息中的20个，最后一个消息是WM_NCHITTEST，它代表「非显示区域命中测试」。此消息优先于所有其它的显示区域和非显示区域鼠标消息。lParam参数含有鼠标位置的x和y屏幕坐标，wParam参数没有用。

Windows应用程序通常把这个消息传送给DefWindowProc，然后Windows用WM_NCHITTEST消息产生与鼠标位置相关的所有其它鼠标消息。对于非显示区域鼠标消息，在处理WM_NCHITTEST时，从DefWindowProc传回的值将成为鼠标消息中的wParam参数，这个值可以是任意非显示区域鼠标消息的wParam值再加上以下内容：

HTCLIENT	显示区域
HTNOWHERE	不在窗口中

HTTRANSPARENT	窗口由另一个窗口覆盖
HTERROR	使DefWindowProc产生警示用的哔声

如果DefWindowProc在其处理WM_NCHITTEST消息后传回HTCLIENT，那么Windows将把屏幕坐标转换为显示区域坐标并产生显示区域鼠标消息。

如果您还记得我们如何通过拦截WM_SYSKEYDOWN消息来停用所有的系统键盘功能，那么您可能会想我们可否通过拦截鼠标消息完成类似的事情。完全可以！只要您在窗口消息处理程序中包含以下几条叙述：

```
case WM_NCHITTEST:
return (LRESULT) HTNOWHERE ;
```

就可以有效地禁用您窗口中的所有显示区域和非显示区域鼠标消息。这样一来，当鼠标在您的窗口（包括系统菜单图标、缩放按钮以及关闭按钮）中时，鼠标按键将会失效。

从消息产生消息

Windows用WM_NCHITTEST消息产生所有其它鼠标消息，这种由消息引出其它消息的想法在Windows中是很普遍的。让我们来举个例子。您知道，如果您在一个Windows程序的系统菜单图标上双击一下，那么程序将会终止。双击产生一系列的WM_NCHITTEST消息。由于鼠标定位在系统菜单图标上，因此DefWindowProc将传回HTSYSMENU的值，并且Windows把wParam等于HTSYSMENU的WM_NCLBUTTONDBLCLK消息放在消息队列中。

窗口消息处理程序通常把鼠标消息传递给DefWindowProc，当DefWindowProc接收到wParam参数等于HTSYSMENU的WM_NCLBUTTONDBLCLK消息时，它就把wParam参数等于SC_CLOSE的WM_SYSCOMMAND消息放入消息队列中（这个WM_SYSCOMMAND消息是在使用者从系统菜单中选择「Close」时产生的）。同样地，窗口消息处理程序也把这个消息传给DefWindowProc。DefWindowProc通过给窗口消息处理程序发送WM_CLOSE消息来处理该消息。

如果一个程序在终止之前要求来自使用者的确认，那么窗口消息处理程序就需要拦截WM_CLOSE，否则，DefWindowProc呼叫DestroyWindow函数来处理WM_CLOSE。除了其它处理，DestroyWindow还给窗口消息处理程序发送一个WM_DESTROY消息。窗口消息处理程序通常用下列程序代码来处理WM_DESTROY消息：

```
caseWM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
```

PostQuitMessage使得Windows把WM_QUIT消息放入消息队列中，此消息永远不会到达窗口消息处理程序，因为它使GetMessage传回0，并终止消息循环，从而也终止了程序。

程序中的命中测试

我在前面讨论了Windows Explorer如何响应鼠标的单击和双击。显然，程序（或者更精确的说，如同Windows Explorer般使用list view control）必须确定使用者鼠标所指向的是哪一个文件。

这叫做「命中测试」。正如DefWindowProc在处理WM_NCHITTEST消息时做一些命中测试一样，窗口消息处理程序经常必须在显示区域中进行一些命中测试。一般来说，命中测试中会使用x

和y坐标值，它们由传到窗口消息处理程序的鼠标消息的lParam参数给出。

一个假想的例子

有这样一个例子。假设您的程序需要显示几列按字母排列的文件。通常，您可以使用list view control，他会帮您由于要做全部的命中测试工作。但我们假设您由于某种原因而不能使用，这时就需要自己来做了。让我们假定文件名保存在称为szFileNames的已排序字符串指针数组中。

让我们也假定文件列表开始于显示区域的顶端，显示区域为cxClient像素宽，cyClient像素高，每列为cxColWidth像素宽，每个字符高度为cyChar像素高。那么每栏可填入的文件数就是：

iNumInCol = cyClient / cyChar ;

接收到一个鼠标单击消息后，您就能从lParam获得cxMouse和cyMouse坐标。然后可以用下面的公式来计算使用者所指的是哪一列的文件名：

iColumn = cxMouse / cxColWidth ;

相对于列顶端的文件名位置为：

iFromTop = cyMouse / cyChar ;

现在您就可以计算szFileNames数组的下标：

iIndex = iColumn * iNumInCol + iFromTop ;

如果iIndex超过了数组中的文件数，则表示使用者是在显示器的空白区域内按鼠标按键。

在许多情况下，命中测试要比本例更加复杂。在显示一幅包含许多小图形的图像时，您必须决定要显示的每个小图形的坐标。在命中计算中，您必须从坐标找到对象。但这将在使用不确定字体大小的字处理程序中变得非常凌乱，因为您必须找到字符在字符串中的位置。

范例程序

程序7-2所示的CHECKER1程序展示了一些简单的命中测试，此程序把显示区域分为5×5的25个矩形。如果您在某个矩形中按下鼠标按键，那么在该矩形中将出现一个「X」。如果您再按一次，那么「X」将被删除。

程序7-2 CHECKER1

CHECKER1.C

```
/*------------------------------------------------------------------------
CHECKER1.C -- Mouse Hit-Test Demo Program No. 1
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define DIVISIONS 5
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Checker1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Checker1 Mouse Hit-Test Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message,
                          WPARAM wParam,LPARAM lParam)
{
  static BOOL fState[DIVISIONS][DIVISIONS] ;
  static int cxBlock, cyBlock ;
  HDC hdc ;
  int x, y ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_SIZE :
    cxBlock = LOWORD (lParam) / DIVISIONS ;
    cyBlock = HIWORD (lParam) / DIVISIONS ;
    return 0 ;

  case WM_LBUTTONDOWN :
    x = LOWORD (lParam) / cxBlock ;
    y = HIWORD (lParam) / cyBlock ;

    if (x < DIVISIONS && y < DIVISIONS)
    {
      fState [x][y] ^= 1 ;
      rect.left = x * cxBlock ;
      rect.top = y * cyBlock ;
      rect.right = (x + 1) * cxBlock ;
      rect.bottom = (y + 1) * cyBlock ;

      InvalidateRect (hwnd, &rect, FALSE) ;
    }
    else
      MessageBeep (0) ;
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    for (x = 0 ; x < DIVISIONS ; x++)
      for (y = 0 ; y < DIVISIONS ; y++)
      {
        Rectangle (hdc, x * cxBlock, y * cyBlock,
          (x + 1) * cxBlock, (y + 1) * cyBlock) ;
```

```
      if (fState [x][y])
      {
        MoveToEx (hdc, x * cxBlock, y * cyBlock, NULL) ;
        LineTo(hdc, (x+1) * cxBlock, (y+1) * cyBlock) ;
        MoveToEx (hdc, x * cxBlock, (y+1) * cyBlock, NULL) ;
        LineTo (hdc, (x+1) * cxBlock, y * cyBlock) ;
      }
    }
    EndPaint (hwnd,&ps);
    return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图7-3是CHECKER1的显示。程序画的25个矩形的宽度和高度均相同。这些宽度和高度保存在cxBlock和cyBlock中，当显示区域大小发生改变时，将重新对这些值进行计算。WM_LBUTTONDOWN处理过程使用鼠标坐标来确定在哪个矩形中按下了键，它在fState数组中标志目前矩形的状态，并使该矩形区域失效，从而产生WM_PAINT消息。

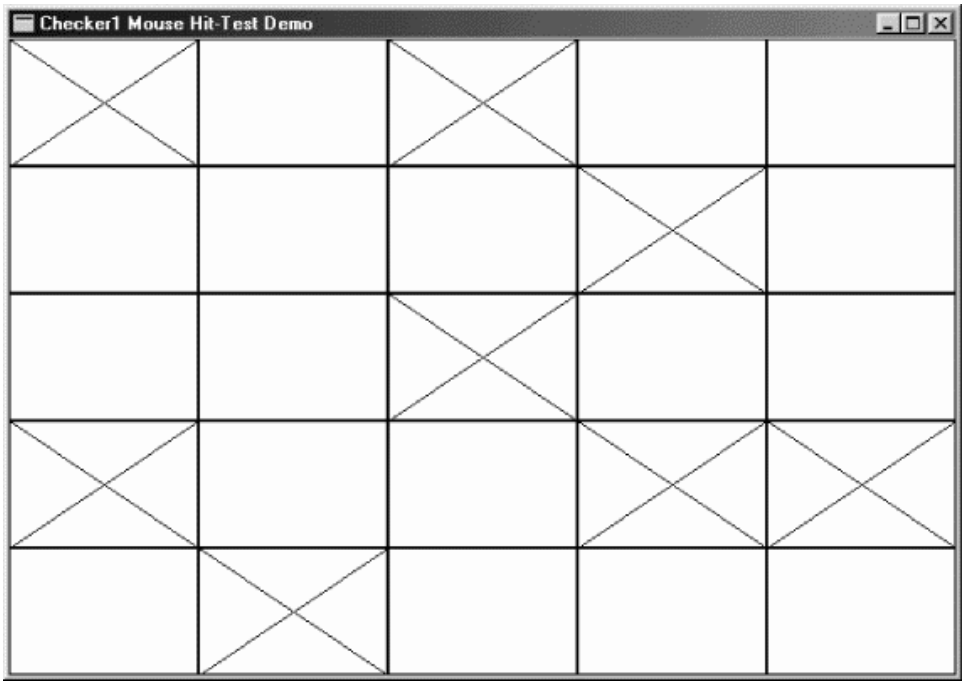


图7-3 CHECKER1的屏幕显示

如果显示区域的宽度和高度不能被5整除，那么在显示区域的左边和下边将有一小条区域不能被矩形所覆盖。对于错误情况，CHECKER1通过呼叫MessageBeep响应此区域中的鼠标按键操作。

当CHECKER1收到WM_PAINT消息时，它通过GDI的Rectangle函数来重新绘制显示区域。如果设定了fState值，那么CHECKER1将使用MoveToEx和LineTo函数来绘制两条直线。在处理WM_PAINT期间，CHECKER1在重新绘制之前并不检查每个矩形区域的有效性，尽管它可以这样做。检查有效性的一种方法是在循环中为每个矩形块建立RECT结构（使用与WM_LBUTTONDOWN处理程序中相同的公式），并使用IntersectRect函数检查它是否与无效矩形（ps.rcPaint）相交。

使用键盘仿真鼠标

CHECKER1只能在装有鼠标情况下才可执行。下面我们在程序中加入键盘接口，就如同第六章

中对SYSMETS程序所做的那样。不过，即使在一个使用鼠标光标作为指向用途的程序中加入键盘接口，我们还是必须处理鼠标光标的移动和显示问题。

即使没有安装鼠标，Windows仍然可以显示一个鼠标光标。Windows为这个光标保存了一个「显示计数」。如果安装了鼠标，显示计数会被初始化为0；否则，显示计数会被初始化为-1。只有在显示计数非负时才显示鼠标光标。要增加显示计数，您可以呼叫：

ShowCursor (TRUE) ;

要减少显示计数，可以呼叫：

ShowCursor (FALSE) ;

您在使用ShowCursor之前，不需要确定是否安装了鼠标。如果您想显示鼠标光标，而不管鼠标存在与否，那么只需呼叫ShowCursor来增加显示计数。增加一次显示计数之后，如果没有安装鼠标则减少它以隐藏光标，如果安装了鼠标，则保留其显示。

即使没有安装鼠标，Windows也保留了鼠标目前的位置。如果没有安装鼠标，而您又显示鼠标光标，光标就可能出现在显示器的任意位置，直到您确实移动了它。要获得光标的位置，可以呼叫：

GetCursorPos (&pt) ;

其中pt是POINT结构。函数使用鼠标的x和y坐标来填入POINT字段。要设定光标位置，可以使用：

SetCursorPos (x, y) ;

在这两种情况下，x和y都是屏幕坐标，而不是显示区域坐标（这是很明显的，因为这些函数没有要求hwnd参数）。前面已经提到过，呼叫ScreenToClient和ClientToScreen就能做到屏幕坐标与客户坐标的相互转换。

如果您在处理鼠标消息并转换显示区域坐标时呼叫GetCursorPos ，这些坐标可能与鼠标消息的lParam参数中的坐标稍微有些不同。从GetCursorPos传回的坐标表示鼠标目前的位置。lParam中的坐标则是产生消息时鼠标的位置。

您或许想写一个键盘处理程序：使用键盘方向键来移动鼠标光标，使用Spacebar和Enter键来仿真鼠标按键。您肯定不希望每次按键只是将鼠标光标移动一个像素，如果这样做，当要把鼠标光标从显示器的一边移动到另一边时，会使用者在很长一段时间内都要按住同一个方向键。

如果您需要实作鼠标光标的键盘接口，并保持光标的精确定位能力，那么您可以采用下面的方式来处理按键消息：当按下方向键时，一开始鼠标光标移动较慢，但随后会加快。您也许还记得WM_KEYDOWN消息中的lParam参数标志着按键消息是否是重复活动的结果，这就是此参数的一个重要应用。

在CHECKER中加入键盘接口

程序7-3所示的CHECKER2程序，除了包括键盘接口外，和CHECKER1是一样的，您可以使用左、右、上和下方向键在25个矩形之间移动光标。Home键把光标移动到矩形的左上角， End键把光标移动到矩形的右下角。Spacebar和Enter键都能切换X标记。

程序7-3 CHECKER2

CHECKER2.C

```
/*------------------------------------------------------------------------
CHECKER2.C -- Mouse Hit-Test Demo Program No. 2
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
```

```
#include <windows.h>

#define DIVISIONS 5

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Checker2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow ( szAppName, TEXT ("Checker2 Mouse Hit-Test Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL fState[DIVISIONS][DIVISIONS] ;
  static int cxBlock, cyBlock ;
  HDC hdc ;
  int x, y ;
  PAINTSTRUCT ps ;
  POINT point ;
  RECT rect ;

  switch (message)
  {
  case WM_SIZE :
    cxBlock = LOWORD (lParam) / DIVISIONS ;
    cyBlock = HIWORD (lParam) / DIVISIONS ;
    return 0 ;
  case WM_SETFOCUS :
    ShowCursor (TRUE) ;
    return 0 ;
  case WM_KILLFOCUS :
    ShowCursor (FALSE) ;
```

```
    return 0 ;
  case WM_KEYDOWN :
    GetCursorPos (&point) ;
    ScreenToClient (hwnd, &point) ;
    x = max (0, min (DIVISIONS - 1, point.x / cxBlock)) ;
    y = max (0, min (DIVISIONS - 1, point.y / cyBlock)) ;

    switch (wParam)
    {
    case VK_UP :
      y-- ;
      break ;
    case VK_DOWN :
      y++ ;
      break ;
    case VK_LEFT :
      x-- ;
      break ;
    case VK_RIGHT :
      x++ ;
      break ;
    case VK_HOME :
      x = y = 0 ;
      break ;
    case VK_END :
      x = y = DIVISIONS - 1 ;
      break ;
    case VK_RETURN :
    case VK_SPACE :
      SendMessage (hwnd, WM_LBUTTONDOWN, MK_LBUTTON,
        MAKELONG (x * cxBlock, y * cyBlock)) ;
      break ;
    }
    x = (x + DIVISIONS) % DIVISIONS ;
    y = (y + DIVISIONS) % DIVISIONS ;

    point.x = x * cxBlock + cxBlock / 2 ;
    point.y = y * cyBlock + cyBlock / 2 ;

    ClientToScreen (hwnd, &point) ;
    SetCursorPos (point.x, point.y) ;
    return 0 ;
    case WM_LBUTTONDOWN :
      x = LOWORD (lParam) / cxBlock ;
      y = HIWORD (lParam) / cyBlock ;

      if (x < DIVISIONS && y < DIVISIONS)
      {
        fState[x][y] ^= 1 ;

        rect.left = x * cxBlock ;
        rect.top = y * cyBlock ;
        rect.right = (x + 1) * cxBlock ;
        rect.bottom = (y + 1) * cyBlock ;

        InvalidateRect (hwnd, &rect, FALSE) ;
      }
      else
        MessageBeep (0) ;
      return 0 ;
    case WM_PAINT :
      hdc = BeginPaint (hwnd, &ps) ;
      for (x = 0 ; x < DIVISIONS ; x++)
        for (y = 0 ; y < DIVISIONS ; y++)
        {
          Rectangle (hdc, x * cxBlock, y * cyBlock,
            (x + 1) * cxBlock, (y + 1) * cyBlock) ;
          if (fState [x][y])
          {
```

```
            MoveToEx (hdc, x *cxBlock, y *cyBlock, NULL) ;
            LineTo (hdc, (x+1)*cxBlock, (y+1)*cyBlock) ;
            MoveToEx (hdc, x *cxBlock, (y+1)*cyBlock, NULL) ;
            LineTo (hdc, (x+1)*cxBlock, y *cyBlock) ;
          }
        }
        EndPaint (hwnd, &ps) ;
        return 0 ;
    case WM_DESTROY :
      PostQuitMessage (0) ;
      return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CHECKER2中的WM_KEYDOWN的处理方式决定光标的位置（用GetCursorPos），把屏幕坐标转换为显示区域坐标（用ScreenToClient），并用矩形方块的宽度和高度来除这个坐标。这会产生指示矩形位置的x和y值（5×5数组）。当按下一个键时，鼠标光标可能在或不在显示区域中，所以x和y必须经过min和max宏处理以保证它们的范围是0到4之间。

对方向键，CHECKER2近似地增加或减少x和y。如果是Enter键或Spacebar键，那么CHECKER2使用SendMessage把WM_LBUTTONDOWN消息发送给它自身。这种技术类似于在第六章SYSMETS程序中把键盘接口加到窗口滚动条时所使用的方法。WM_KEYDOWN的处理方式是通过计算指向矩形中心的显示区域坐标，再用ClientToScreen转换成屏幕坐标，然后用SetCursorPos设定光标位置来实作的。

将子窗口用于命中测试

有些程序（例如，Windows的「画图」程序），把显示区域划分为几个小的逻辑区域。「画图」程序在其左边有一个由图标组成的工具菜单区，在底部有颜色菜单区。在这两个区做命中测试的时候，「画图」必须在使用者选中菜单项之前记住菜单的位置。

不过，也可能不需要这么做。实际上，画风经由使用子窗口简化了菜单的绘制和命中测试。子窗口把整个矩形区域划分为几个更小的矩形区，每个子窗口有自己的窗口句柄、窗口消息处理程序和显示区域，每个窗口消息处理程序接收只适用于它的子窗口的鼠标消息。鼠标消息中的lParam参数含有相当于该子窗口显示区域左上角的坐标，而不是其父窗口（那是「画图」的主应用程序窗口）显示区域左上角的坐标。

以这种方式使用子窗口有助于程序的结构化和模块化。如果子窗口使用不同的窗口类别，那么每个子窗口都有它自己的窗口消息处理程序。不同的窗口也可以定义不同的背景颜色和不同的内定光标。在第九章中，我将看到「子窗口控件」－滚动条、按钮和编辑方块等预先定义的子窗口。现在，我们说明在CHECKER程序中是如何使用子窗口的。

CHECKER中的子窗口

程序7-4所示的CHECKER3程序，这一版本建立了25个处理鼠标单击的子窗口。它没有键盘接口，但是可以按本章后面的CHECKER4程序范例的方法添加。

程序7-4 CHECKER3

CHECKER3.C

```
/*------------------------------------------------------------------------
CHECKER3.C -- Mouse Hit-Test Demo Program No. 3
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
```

```
#define DIVISIONS 5

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ChildWndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szChildClass[] = TEXT ("Checker3_Child") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Checker3") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  wndclass.lpfnWndProc = ChildWndProc ;
  wndclass.cbWndExtra = sizeof (long) ;
  wndclass.hIcon = NULL ;
  wndclass.lpszClassName = szChildClass ;

  RegisterClass (&wndclass) ;
  hwnd = CreateWindow (szAppName, TEXT ("Checker3 Mouse Hit-Test Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HWND hwndChild[DIVISIONS][DIVISIONS] ;
  int cxBlock, cyBlock, x, y ;

  switch (message)
  {
  case WM_CREATE :
    for (x = 0 ; x < DIVISIONS ; x++)
      for (y = 0 ; y < DIVISIONS ; y++)
        hwndChild[x][y] = CreateWindow (szChildClass, NULL,
        WS_CHILDWINDOW | WS_VISIBLE,
        0, 0, 0, 0,
        hwnd, (HMENU) (y << 8 | x),
        (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
        NULL) ;
```

```
    return 0 ;
  case WM_SIZE :
    cxBlock = LOWORD (lParam) / DIVISIONS ;
    cyBlock = HIWORD (lParam) / DIVISIONS ;
    for (x = 0 ; x < DIVISIONS ; x++)
      for (y = 0 ; y < DIVISIONS ; y++)
        MoveWindow ( hwndChild[x][y],
        x * cxBlock, y * cyBlock,
        cxBlock, cyBlock, TRUE) ;
      return 0 ;
  case WM_LBUTTONDOWN :
    MessageBeep (0) ;
    return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ChildWndProc (HWND hwnd, UINT message,
                          WPARAM wParam, LPARAM lParam)
{
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE :
    SetWindowLong (hwnd, 0, 0) ; // on/off flag
    return 0 ;
  case WM_LBUTTONDOWN :
    SetWindowLong (hwnd, 0, 1 ^ GetWindowLong (hwnd, 0)) ;
    InvalidateRect (hwnd, NULL, FALSE) ;
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    GetClientRect (hwnd, &rect) ;
    Rectangle (hdc, 0, 0, rect.right, rect.bottom) ;

    if (GetWindowLong (hwnd, 0))
    {
      MoveToEx (hdc, 0, 0, NULL) ;
      LineTo (hdc, rect.right, rect.bottom) ;
      MoveToEx (hdc, 0, rect.bottom, NULL) ;
      LineTo (hdc, rect.right, 0) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CHECKER3有两个窗口消息处理程序WndProc和ChildWndProc。WndProc还是主（或父）窗口的窗口消息处理程序。ChildWndProc是针对25个子窗口的窗口消息处理程序。这两个窗口消息处理程序都必须定义为CALLBACK函数。

因为窗口消息处理程序与特定的窗口类别结构相关联，该窗口类别结构由Windows呼叫RegisterClass函数来注册，CHECKER3需要两个窗口类别。第一个窗口类别用于主窗口，名为「Checker3」。第二个窗口类别名为「Checker3_Child」。当然，您不必选择像这样有意义的名字。

CHECKER3在WinMain函数中注册了这两个窗口类别。注册完常规的窗口类别之后，CHECKER3只是简单地重新使用wndclass结构中的大多数的字段来注册Checker3_Child类别。无

论如何，有四个字段根据子窗口类别而设定为不同的值：

pfnWndProc字段设定为ChildWndProc，子窗口类别的窗口消息处理程序。

cbWndExtra字段设定为4字节，或者更确切地用sizeof (long)。该字段告诉Windows在其为依据此窗口类别的窗口保留的内部结构中，预留了4字节额外的空间。您能使用此空间来保存每个窗口的可能有所不同的信息。

因为像CHECKER3中的子窗口不需要图标，所以hIcon字段设定为NULL 。

pszClassName字段设定为「Checker3_Child」，是类别的名称。

通常，在WinMain中，CreateWindow呼叫建立依据Checker3类别的主窗口。然而，当WndProc收到WM_CREATE消息后，它呼叫CreateWindow 25次以建立25个Checker3_Child类别的子窗口。表7-3是在WinMain中CreateWindow呼叫的参数，与在建立25个子窗口的WndProc中CreateWindow呼叫的参数间的比较。

表7-3

参数	主窗口	子窗口
窗口类别	「Checker3」	「Checker3_Child」
窗口标题	「Checker3...」	NULL
窗口样式	WS_OVERLAPPEDWINDOW	WS_CHILDWINDOW| WS_VISIBLE
水平位置	CW_USEDEFAULT	0
垂直位置	CW_USEDEFAULT	0
宽度	CW_USEDEFAULT	0
高度	CW_USEDEFAULT	0
父窗口句柄	NULL	hwnd
菜单句柄/子ID	NULL	(HMENU) (y << 8 | x)
执行实体句柄	hInstance	(HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE)
额外参数	NULL	NULL

一般情况下，子窗口要求有关位置和大小的参数，但是在CHECKER3中的子窗口由WndProc确定位置和大小。对于主窗口，因为它本身就是父窗口，所以它的父窗口句柄是NULL。当使用CreateWindow呼叫来建立一个子窗口时，就需要父窗口句柄了。

主窗口没有菜单，因此参数是NULL。对于子窗口，相同位置的参数称为子ID（或子窗口ID）。这是唯一代表子窗口的数字。像我们在第十一章将看到的一样，在处理对话框的子窗口控件时，子ID显得更为重要。对于CHECKER3来说，我只是简单地将子ID设定为一个数值，该数值是每个子窗口在5×5的主窗口中的x和y位置的组合。

CreateWindow函数需要一个执行实体句柄。在WinMain中，执行实体句柄可以很容易地取得，因为它是WinMain的一个参数。在建立子窗口时， CHECKER3必须用GetWindowLong来从Windows为窗口保留的结构中取得hInstance值（相对于GetWindowLong，我也能将hInstance的值保存到整体变量，并直接使用它）。

每一个子窗口都在hwndChild数组中保存了不同的窗口句柄。当WndProc接收到一个WM_SIZE消息后，它将为这25个子窗口呼叫MoveWindow。MoveWindow的参数表示子窗口左

上角相对于父窗口显示区域的坐标、子窗口的宽度和高度以及子窗口是否需要重画。

现在让我们看一下ChildWndProc。此窗口消息处理程序为所有这25个子窗口处理消息。ChildWndProc的hwnd参数是子窗口接收消息的句柄。当ChildWndProc处理WM_CREATE消息时（因为有25个子窗口，所以要发生25次），它用SetWindowWord在窗口结构保留的额外区域中储存一个0值（通过在定义窗口类别时使用的cbWndExtra来保留的空间）。ChildWndProc用此值来恢复目前矩形的状态（有X或没有X）。在子窗口中单击时，WM_LBUTTONDOWN处理例程简单地修改这个整数值（从0到1，或从1到0），并使整个子窗口无效。此区域是被单击的矩形。WM_PAINT的处理很简单，因为它所绘制的矩形与显示区域一样大。

因为CHECKER3的C原始码文件和.EXE文件比CHECKER1的大（更不用说程序的说明了），我不会试着告诉你说CHECKER3比CHECKER1更简单。但请注意，我们没有做任何的鼠标命中测试！我们所要的，就是知道CHECKER3中是否有个子窗口得到了命中窗口的WM_LBUTTONDOWN消息。

子窗口和键盘

为CHECKER3添加键盘接口就像CHECKER系列构想中的最后一步。但在这样做的时候，可能有更适当的做法。在CHECKER2中，鼠标光标的位置决定按下Spacebar键时哪个区域将获得标记符号。当我们处理子窗口时，我们能从对话框功能中获得提示。在对话框中，带有闪烁的插入符号或点划的矩形的子窗口表示它有输入焦点（当然也可以用键盘进行定位）。

我们不需要把Windows内部已有的对话框处理方式重新写过，我只是要告诉您大致上应该如何在应用程序中仿真对话框。研究过程中，您会发现这样一件事：父窗口和子窗口可能要共享同键盘消息处理。按下Spacebar键和Enter键时，子窗口将锁定复选标记。按下方向键时，父窗口将在子窗口之间移动输入焦点。实际上，当您在子窗口上单击时，情况会有些复杂，这时是父窗口而不是子窗口获得输入焦点。

CHECKER4.C如程序7-5所示。

程序7-5 CHECKER4

CHECKER4.C

```
/*--------------------------------------------------------------------------
CHECKER4.C -- Mouse Hit-Test Demo Program No. 4
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#define DIVISIONS 5

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ChildWndProc (HWND, UINT, WPARAM, LPARAM) ;
int idFocus = 0 ;
TCHAR szChildClass[] = TEXT ("Checker4_Child") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Checker4") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  wndclass.lpfnWndProc = ChildWndProc ;
  wndclass.cbWndExtra = sizeof (long) ;
  wndclass.hIcon = NULL ;
  wndclass.lpszClassName = szChildClass ;

  RegisterClass (&wndclass) ;
  hwnd = CreateWindow (szAppName, TEXT ("Checker4 Mouse Hit-Test Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HWND hwndChild[DIVISIONS][DIVISIONS] ;
  int cxBlock, cyBlock, x, y ;

  switch (message)
  {
  case WM_CREATE :
    for (x = 0 ; x < DIVISIONS ; x++)
      for (y = 0 ; y < DIVISIONS ; y++)
        hwndChild[x][y] = CreateWindow (szChildClass, NULL,
        WS_CHILDWINDOW | WS_VISIBLE,
        0, 0, 0, 0,
        hwnd, (HMENU) (y << 8 | x),
        (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
        NULL) ;
      return 0 ;
  case WM_SIZE :
    cxBlock = LOWORD (lParam) / DIVISIONS ;
    cyBlock = HIWORD (lParam) / DIVISIONS ;

    for (x = 0 ; x < DIVISIONS ; x++)
      for (y = 0 ; y < DIVISIONS ; y++)
        MoveWindow ( hwndChild[x][y],
        x * cxBlock, y * cyBlock,
        cxBlock, cyBlock, TRUE) ;
      return 0 ;
  case WM_LBUTTONDOWN :
    MessageBeep (0) ;
    return 0 ;
    // On set-focus message, set focus to child window
  case WM_SETFOCUS:
    SetFocus (GetDlgItem (hwnd, idFocus)) ;
    return 0 ;
    // On key-down message, possibly change the focus window
```

```
  case WM_KEYDOWN:
    x = idFocus & 0xFF ;
    y = idFocus >> 8 ;
    switch (wParam)
    {
    case VK_UP: y-- ; break ;
    case VK_DOWN: y++ ; break ;
    case VK_LEFT: x-- ; break ;
    case VK_RIGHT: x++ ; break ;
    case VK_HOME: x = y = 0 ; break ;
    case VK_END: x = y = DIVISIONS - 1 ; break ;
    default: return 0 ;
    }

    x = (x + DIVISIONS) % DIVISIONS ;
    y = (y + DIVISIONS) % DIVISIONS ;

    idFocus = y << 8 | x ;
    SetFocus (GetDlgItem (hwnd, idFocus)) ;
    return 0 ;
    case WM_DESTROY :
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ChildWndProc (HWND hwnd, UINT message,
                               WPARAM wParam, LPARAM lParam)
{
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;
  switch (message)
  {
  case WM_CREATE :
    SetWindowLong (hwnd, 0, 0) ; // on/off flag
    return 0 ;
  case WM_KEYDOWN:
    // Send most key presses to the parent window
    if (wParam != VK_RETURN && wParam != VK_SPACE)
    {
      SendMessage (GetParent (hwnd), message, wParam, lParam) ;
      return 0 ;
    }
    // For Return and Space, fall through to toggle the square
  case WM_LBUTTONDOWN :
    SetWindowLong (hwnd, 0, 1 ^ GetWindowLong (hwnd, 0)) ;
    SetFocus (hwnd) ;
    InvalidateRect (hwnd, NULL, FALSE) ;
    return 0 ;
    // For focus messages, invalidate the window for repaint
  case WM_SETFOCUS:
    idFocus = GetWindowLong (hwnd, GWL_ID) ;
    // Fall through
  case WM_KILLFOCUS:
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    GetClientRect (hwnd, &rect) ;
    Rectangle (hdc, 0, 0, rect.right, rect.bottom) ;
    // Draw the "x" mark
    if (GetWindowLong (hwnd, 0))
    {
      MoveToEx (hdc, 0, 0, NULL) ;
      LineTo (hdc, rect.right, rect.bottom) ;
      MoveToEx (hdc, 0, rect.bottom, NULL) ;
```

```
      LineTo (hdc, rect.right, 0) ;
    }
    // Draw the "focus" rectangle
    if (hwnd == GetFocus ())
    {
      rect.left += rect.right / 10 ;
      rect.right -= rect.left ;
      rect.top += rect.bottom / 10 ;
      rect.bottom -= rect.top ;

      SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
      SelectObject (hdc, CreatePen (PS_DASH, 0, 0)) ;
      Rectangle (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
      DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

您应该能回忆起每一个子窗口有唯一的子窗口ID，该ID在呼叫CreateWindow建立窗口时定义。在CHECKER3中，此ID是矩形的x和y位置的组合。一个程序可以通过下面的呼叫来获得一个特定子窗口的子窗口ID：

idChild = GetWindowLong (hwndChild, GWL_ID) ;

下面的函数也有同样的功能：

idChild = GetDlgCtrlID (hwndChild) ;

正如函数名称所表示的，它主要用于对话框和控制窗口。如果您知道父窗口的句柄和子窗口ID，此函数也可以获得子窗口的句柄：

hwndChild = GetDlgItem (hwndParent, idChild) ;

在CHECKER4中，整体变量idFocus用于保存目前输入焦点窗口的子窗口ID。我在前面说过，当您在子窗口上面单击鼠标时，它们不会自动获得输入焦点。因此，CHECKER4中的父窗口将通过呼叫下面的函数来处理WM_SETFOCUS消息：

SetFocus (GetDlgItem (hwnd, idFocus)) ;

这样设定一个子窗口为输入焦点。

ChildWndProc处理WM_SETFOCUS和WM_KILLFOCUS消息。对于WM_SETFOCUS，它将保存在整体变量idFocus中接收输入焦点的子窗口ID。对于这两种消息，窗口是无效的，并产生一个WM_PAINT消息。如果WM_PAINT消息画出了有输入焦点的子窗口，则它将用PS_DASH画笔的风格画一个矩形以表示此窗口有输入焦点。

ChildWndProc也处理WM_KEYDOWN消息。对于除了Spacebar和Enter键以外的其它消息，WM_KEYDOWN 都将给父窗口发送消息。另外，窗口消息处理程序也处理类似WM_LBUTTONDOWN消息的消息。

处理方向移动键是父窗口的事情。在风格相似的CHECKER2中，此程序可获得有输入焦点的子窗口的x和y坐标，并根据按下的特定方向键来改变它们。然后通过呼叫SetFocus将输入焦点设定给新的子窗口。

拦截鼠标

一个窗口消息处理程序通常只在鼠标光标位于窗口的显示区域，或非显示区域上时才接收鼠标消息。一个程序也可能需要在鼠标位于窗口外时接收鼠标消息。如果是这样，程序可以自行「拦截」鼠标。别害怕，这么做没什么大不了的。

设计矩形

为了说明拦截鼠标的必要性，请让我们看一下BLOKOUT1程序（如程序7-6所示）。此程序看起来达到了一定的功能，但它却有十分严重的缺陷。

程序7-6 BLOKOUT1

BLOKOUT1.C

```
/*------------------------------------------------------------------------
BLOKOUT1.C -- Mouse Button Demo Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("BlokOut1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Mouse Button Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
```

```
void DrawBoxOutline (HWND hwnd, POINT ptBeg, POINT ptEnd)
{
  HDC hdc ;
  hdc = GetDC (hwnd) ;
  SetROP2 (hdc, R2_NOT) ;
  SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
  Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x, ptEnd.y) ;
  ReleaseDC (hwnd, hdc) ;
}
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL fBlocking, fValidBox ;
  static POINT ptBeg, ptEnd, ptBoxBeg, ptBoxEnd ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  switch (message)
  {
  case WM_LBUTTONDOWN :
    ptBeg.x = ptEnd.x = LOWORD (lParam) ;
    ptBeg.y = ptEnd.y = HIWORD (lParam) ;

    DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
    SetCursor (LoadCursor (NULL, IDC_CROSS)) ;
    fBlocking = TRUE ;
    return 0 ;

  case WM_MOUSEMOVE :
    if (fBlocking)
    {
      SetCursor (LoadCursor (NULL, IDC_CROSS)) ;

      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
      ptEnd.x = LOWORD (lParam) ;
      ptEnd.y = HIWORD (lParam) ;
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
    }
    return 0 ;

  case WM_LBUTTONUP :
    if (fBlocking)
    {
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
      ptBoxBeg = ptBeg ;
      ptBoxEnd.x = LOWORD (lParam) ;
      ptBoxEnd.y = HIWORD (lParam) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
      fBlocking = FALSE ;
      fValidBox = TRUE ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

  case WM_CHAR :
    if (fBlocking & wParam == '\x1B') // i.e., Escape
    {
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
      fBlocking = FALSE ;
    }
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    if (fValidBox)
    {
      SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
      Rectangle ( hdc, ptBoxBeg.x, ptBoxBeg.y,
        ptBoxEnd.x, ptBoxEnd.y) ;
    }
```

```
    if (fBlocking)
    {
      SetROP2 (hdc, R2_NOT) ;
      SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
      Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x, ptEnd.y) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

此程序展示了一些，它可以实作在Windows的「画图」程序中的东西。由按下鼠标左键开始确定矩形的一角，然后拖动鼠标。程序将画一个矩形的轮廓，其相对位置是鼠标目前的位置。当您释放鼠标后，程序将填入这个矩形。图7-4显示了一个已经画完的矩形和另一个正在画的矩形。

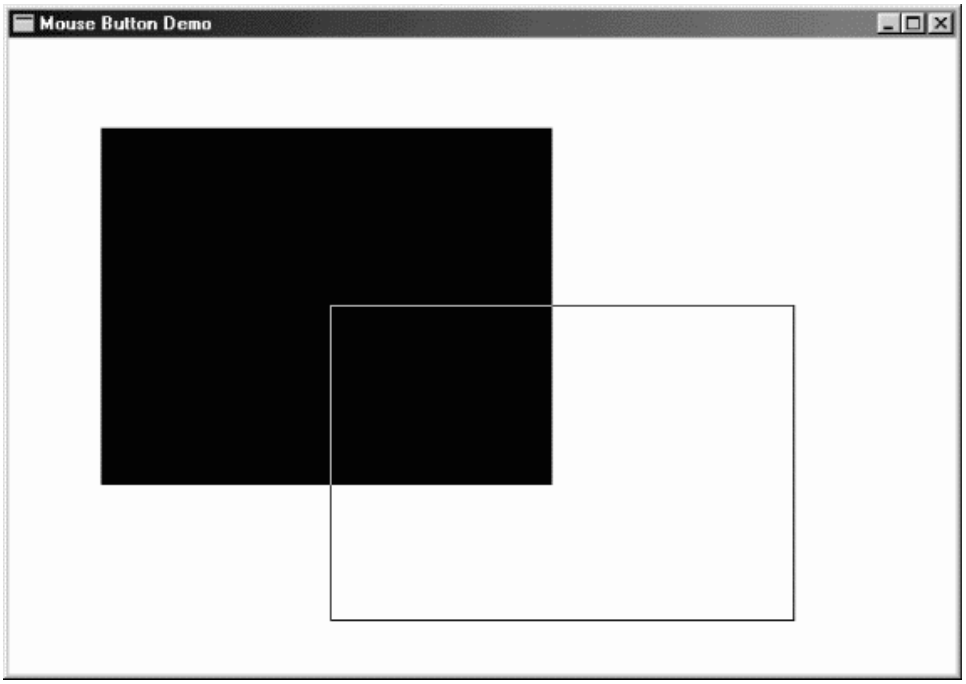


图7-4 BLOKOUT1的屏幕显示

那么，问题在哪里呢?

请试一试下面的操作：在BLOKOUT1的显示区域按下鼠标的左键，然后将光标移出窗口。程序将停止接收WM_MOUSEMOVE消息。现在释放按钮，BLOKOUT1将不再获得WM_BUTTONUP消息，因为光标在显示区域以外。然后将光标移回BLOKOUT1的显示区域，窗口消息处理程序仍然认为按钮处于按下状态。

这样做并不好，因为程序不知道发生了什么事情。

拦截的解决方案

BLOKOUT1显示了一些常见的程序功能，但它的程序代码显然有缺陷。这种问题就是要使用鼠标拦截来对付。如果使用者正在拖曳鼠标，那么当鼠标短时间内被拖出窗口时应该没有什么大问题，程序应该仍然控制着鼠标。

拦截鼠标要比放置一个老鼠夹子容易一些，您只要呼叫：

SetCapture (hwnd) ;

在这个函数呼叫之后，Windows将所有鼠标消息发给窗口句柄为hwnd的窗口消息处理程序。之后收到鼠标消息都是以显示区域消息的型态出现，即使鼠标正在窗口的非显示区域。lParam参数将指示鼠标在显示区域坐标中的位置。不过，当鼠标位于显示区域的左边或者上方时，这些x和y坐标可以是负的。当您想释放鼠标时，呼叫：

ReleaseCapture () ;

从而使处理恢复正常。

在32位的Windows中，鼠标拦截要比在以前的Windows版本中有多一些限制。特别是，如果鼠标被拦截，而鼠标按键目前并未被按下，并且鼠标光标移到了另一个窗口上，那么将不是由拦截鼠标的那个窗口，而是由光标下面的窗口来接收鼠标消息。对于防止一个程序在拦截鼠标之后不释放它而引起整个系统的混乱，这是必要的。

换句话说，只有当鼠标按键在您的显示区域中被按下时才拦截鼠标；当鼠标按键被释放时，才释放鼠标拦截。

BLOKOUT2程序

展示鼠标拦截的BLOKOUT2程序如程序7-7所示。

程序7-7 BLOKOUT2

BLOKOUT2.C

```
/*------------------------------------------------------------------------
BLOKOUT2.C -- Mouse Button & Capture Demo Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("BlokOut2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Mouse Button & Capture Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
```

```
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void DrawBoxOutline (HWND hwnd, POINT ptBeg, POINT ptEnd)
{
  HDC hdc ;
  hdc = GetDC (hwnd) ;
  SetROP2 (hdc, R2_NOT) ;
  SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
  Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x, ptEnd.y) ;

  ReleaseDC (hwnd, hdc) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL fBlocking, fValidBox ;
  static POINT ptBeg, ptEnd, ptBoxBeg, ptBoxEnd ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  switch (message)
  {
  case WM_LBUTTONDOWN :
    ptBeg.x = ptEnd.x = LOWORD (lParam) ;
    ptBeg.y = ptEnd.y = HIWORD (lParam) ;

    DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
    SetCapture (hwnd) ;
    SetCursor (LoadCursor (NULL, IDC_CROSS)) ;
    fBlocking = TRUE ;
    return 0 ;
  case WM_MOUSEMOVE :
    if (fBlocking)
    {
      SetCursor (LoadCursor (NULL, IDC_CROSS)) ;
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
      ptEnd.x = LOWORD (lParam) ;
      ptEnd.y = HIWORD (lParam) ;
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
    }
    return 0 ;
  case WM_LBUTTONUP :
    if (fBlocking)
    {
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
      ptBoxBeg = ptBeg ;
      ptBoxEnd.x = LOWORD (lParam) ;
      ptBoxEnd.y = HIWORD (lParam) ;
      ReleaseCapture () ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
      fBlocking = FALSE ;
      fValidBox = TRUE ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;
  case WM_CHAR :
    if (fBlocking & wParam == '\x1B') // i.e., Escape
    {
      DrawBoxOutline (hwnd, ptBeg, ptEnd) ;
```

```
      ReleaseCapture () ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
      fBlocking = FALSE ;
    }
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    if (fValidBox)
    {
      SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
      Rectangle (hdc, ptBoxBeg.x, ptBoxBeg.y,
        ptBoxEnd.x, ptBoxEnd.y) ;
    }
    if (fBlocking)
    {
      SetROP2 (hdc, R2_NOT) ;
      SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
      Rectangle (hdc, ptBeg.x, ptBeg.y, ptEnd.x, ptEnd.y) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

BLOKOUT2程序和BLOKOUT1程序一样，只是多了三行新程序代码：在WM_LBUTTONDOWN消息处理期间呼叫SetCapture，而在WM_LBUTTONDOWN和WM_CHAR消息处理期间呼叫ReleaseCapture。检查画出窗口：使窗口小于屏幕大小，开始在显示区域画出一块矩形，然后将鼠标光标移出显示区域的右边或下边，最后释放鼠标按键。程序将获得整个矩形的坐标。但是需要扩大窗口才能看清楚它。

拦截鼠标并非只适用于那些古怪的应用程序。如果您需要鼠标按键在显示区域按下时都能够追踪WM_MOUSEMOVE消息，并直到鼠标按键被释放为止，那么您就应该拦截鼠标。这样将简化您的程序，同时又符合使用者的期望。

鼠标滑轮

与传统的鼠标相比，Microsoft IntelliMouse的特点是在两个键之间多了一个小滑轮。您可以按下这个滑轮，这时它的功能相当于鼠标按键的中键；或者您也可以用食指来转动它，这会产生一条特殊的消息，叫做WM_MOUSEWHEEL。使用鼠标滑轮的程序通过滚动或放大文件来响应此消息。它最初听起来像一个不必要的隐藏机关，但我必须承认，我很快就习惯于使用鼠标滑轮来滚动Microsoft Word和Microsoft Internet Explorer了。

我不想讨论鼠标滑轮的所有使用方法。实际上，我只是想告诉您如何在现有的程序（例如程序SYSMETS4）中添加鼠标滑轮处理程序，以便在显示区域中卷动数据。最终的SYSMETS程序如程序7-8所示。

程序7-8 SYSMETS4

SYSMETS.C

```
/*------------------------------------------------------------------------
SYSMETS.C -- Final System Metrics Display Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
```

```
#include "sysmets.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("SysMets") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Get System Metrics"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int cxChar, cxCaps, cyChar, cxClient, cyClient, iMaxWidth ;
  static int iDeltaPerLine, iAccumDelta ; // for mouse wheel logic
  HDC hdc ;
  int i, x, y, iVertPos, iHorzPos, iPaintBeg, iPaintEnd ;
  PAINTSTRUCT ps ;
  SCROLLINFO si ;
  TCHAR szBuffer[10] ;
  TEXTMETRIC tm ;
  ULONG ulScrollLines ; // for mouse wheel logic
  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
    cxChar = tm.tmAveCharWidth ;
    cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2 ;
    cyChar = tm.tmHeight + tm.tmExternalLeading ;
    ReleaseDC (hwnd, hdc) ;
    // Save the width of the three columns
    iMaxWidth = 40 * cxChar + 22 * cxCaps ;
    // Fall through for mouse wheel information
  case WM_SETTINGCHANGE:
    SystemParametersInfo (SPI_GETWHEELSCROLLLINES, 0, &ulScrollLines, 0) ;
```

```
  // ulScrollLines usually equals 3 or 0 (for no scrolling)
  // WHEEL_DELTA equals 120, so iDeltaPerLine will be 40
  if (ulScrollLines)
    iDeltaPerLine = WHEEL_DELTA / ulScrollLines ;
  else
    iDeltaPerLine = 0 ;
  return 0 ;
case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  // Set vertical scroll bar range and page size
  si.cbSize = sizeof (si) ;
  si.fMask = SIF_RANGE | SIF_PAGE ;
  si.nMin = 0 ;
  si.nMax = NUMLINES - 1 ;
  si.nPage = cyClient / cyChar ;
  SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
  // Set horizontal scroll bar range and page size

  si.cbSize = sizeof (si) ;
  si.fMask = SIF_RANGE | SIF_PAGE ;
  si.nMin = 0 ;
  si.nMax = 2 + iMaxWidth / cxChar ;
  si.nPage = cxClient / cxChar ;
  SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
  return 0 ;
case WM_VSCROLL:
  // Get all the vertical scroll bar information
  si.cbSize = sizeof (si) ;
  si.fMask = SIF_ALL ;
  GetScrollInfo (hwnd, SB_VERT, &si) ;
  // Save the position for comparison later on
  iVertPos = si.nPos ;
  switch (LOWORD (wParam))
  {
  case SB_TOP:
    si.nPos = si.nMin ;
    break ;
  case SB_BOTTOM:
    si.nPos = si.nMax ;
    break ;
  case SB_LINEUP:
    si.nPos -= 1 ;
    break ;
  case SB_LINEDOWN:
    si.nPos += 1 ;
    break ;
  case SB_PAGEUP:
    si.nPos -= si.nPage ;
    break ;
  case SB_PAGEDOWN:
    si.nPos += si.nPage ;
    break ;
  case SB_THUMBTRACK:
    si.nPos = si.nTrackPos ;
    break ;
  default:
    break ;
  }
  // Set the position and then retrieve it. Due to adjustments
  // by Windows it may not be the same as the value set.
  si.fMask = SIF_POS ;
  SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
  GetScrollInfo (hwnd, SB_VERT, &si) ;
  // If the position has changed, scroll the window and update it
  if (si.nPos != iVertPos)
  {
    ScrollWindow (hwnd, 0, cyChar * (iVertPos - si.nPos),
      NULL, NULL) ;
```

```
    UpdateWindow (hwnd) ;
   }
   return 0 ;
   case WM_HSCROLL:
    // Get all the vertical scroll bar information
    si.cbSize = sizeof (si) ;
    si.fMask = SIF_ALL ;
    // Save the position for comparison later on
    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    iHorzPos = si.nPos ;
    switch (LOWORD (wParam))
    {
    case SB_LINELEFT:
      si.nPos -= 1 ;
      break ;
    case SB_LINERIGHT:
      si.nPos += 1 ;
      break ;
    case SB_PAGELEFT:
      si.nPos -= si.nPage ;
      break ;
    case SB_PAGERIGHT:
      si.nPos += si.nPage ;
      break ;
    case SB_THUMBPOSITION:
      si.nPos = si.nTrackPos ;
      break ;
    default:
      break ;
    }
    // Set the position and then retrieve it. Due to adjustments
    // by Windows it may not be the same as the value set.
    si.fMask = SIF_POS ;
    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    // If the position has changed, scroll the window
    if (si.nPos != iHorzPos)
    {
      ScrollWindow (hwnd, cxChar * (iHorzPos - si.nPos), 0,
        NULL, NULL) ;
    }
    return 0 ;
    case WM_KEYDOWN :
      switch (wParam)
      {
      case VK_HOME :
        SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0) ;
        break ;
      case VK_END :
        SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0) ;
        break ;
      case VK_PRIOR :
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0) ;
        break ;
      case VK_NEXT :
        SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0) ;
        break ;
      case VK_UP :
        SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;
        break ;
      case VK_DOWN :
        SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0) ;
        break ;
      case VK_LEFT :
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEUP, 0) ;
        break ;
      case VK_RIGHT :
        SendMessage (hwnd, WM_HSCROLL, SB_PAGEDOWN, 0) ;
        break ;
```

```
      }
      return 0 ;
      case WM_MOUSEWHEEL:
        if (iDeltaPerLine == 0)
          break ;
        iAccumDelta += (short) HIWORD (wParam) ; // 120 or -120
        while (iAccumDelta >= iDeltaPerLine)
        {
          SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0) ;
          iAccumDelta -= iDeltaPerLine ;
        }
        while (iAccumDelta <= -iDeltaPerLine)
        {
          SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0) ;
          iAccumDelta += iDeltaPerLine ;
        }
        return 0 ;
      case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;
        // Get vertical scroll bar position
        si.cbSize = sizeof (si) ;
        si.fMask = SIF_POS ;
        GetScrollInfo (hwnd, SB_VERT, &si) ;
        iVertPos = si.nPos ;
        // Get horizontal scroll bar position

        GetScrollInfo (hwnd, SB_HORZ, &si) ;
        iHorzPos = si.nPos ;
        // Find painting limits

        iPaintBeg = max (0, iVertPos + ps.rcPaint.top / cyChar) ;
        iPaintEnd = min (NUMLINES - 1,
          iVertPos + ps.rcPaint.bottom / cyChar) ;

        for (i = iPaintBeg ; i <= iPaintEnd ; i++)
        {
          x = cxChar * (1 - iHorzPos) ;
          y = cyChar * (i - iVertPos) ;

          TextOut ( hdc, x, y,
            sysmetrics[i].szLabel,
            lstrlen (sysmetrics[i].szLabel)) ;

          TextOut ( hdc, x + 22 * cxCaps, y,
            sysmetrics[i].szDesc,
            lstrlen (sysmetrics[i].szDesc)) ;

          SetTextAlign (hdc, TA_RIGHT | TA_TOP) ;

          TextOut ( hdc, x + 22 * cxCaps + 40 * cxChar, y, szBuffer,
            wsprintf (szBuffer, TEXT ("%5d"),
            GetSystemMetrics (sysmetrics[i].iIndex))) ;
          SetTextAlign (hdc, TA_LEFT | TA_TOP) ;
        }

        EndPaint (hwnd, &ps) ;
        return 0 ;
      case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

转动滑轮会导致Windows在有输入焦点的窗口（不是鼠标光标下面的窗口）产生WM_MOUSEWHEEL消息。与平常一样，lParam将获得鼠标的位置，当然坐标是相对于屏幕左上角的，而不是显示区域的。另外，wParam的低字组包含一系列的旗标，用于表示鼠标按键、Shift与Ctrl键的状态。

新的信息保存在wParam的高字组。其中有一个「delta」值，该值目前可以是120或-120，这取决于滑轮的向前转动（也就是说，向鼠标的前面，即带有按钮与电缆的一端）还是向后转动。值120或-120表示文件将分别向上或向下卷动三行。这里的构想是，以后版本的鼠标滑轮能有比现在的鼠标产生更精确的移动速度信息，并且用delta值，例如40和-40，来产生WM_MOUSEWHEEL消息。这些值能使文件只向上或向下卷动一行。

为使程序能在一般化环境执行，SYSMETS将在WM_CREATE和WM_SETTINGCHANGE消息处理时，以SPI_GETWHEELSCROLLLINES作为参数来呼叫SystemParametersInfo。此值说明WHEEL_DELTA的delta值将滚动多少行，WHEEL_DELTA在WINUSER.H中定义。WHEEL_DELTA等于120，并且，在内定情况下SystemParametersInfo传回3，因此与卷动一行相联系的delta值就是40。SYSMETS将此值保存在iDeltaPerLine。

在WM_MOUSEWHEEL消息处理期间，SYSMETS将delta值给静态变量iAccumDelta。然后，如果iAccumDelta大于或等于iDeltaPerLine（或者是小于或等于-iDeltaPerLin），SYSMETS用SB_LINEUP或SB_LINEDOWN值产生WM_VSCROLL消息。对于每一个WM_VSCROLL消息，iAccumDelta由iDeltaPerLine增加（或减少）。此代码允许delta值大于、小于或等于滚动一行所需要的delta值。

下面还有

还有一个引人注目的鼠标问题：建立自订鼠标光标。我将在第十章，与其它Windows资源一起讨论此问题。

第八章 定时器

Microsoft Windows定时器是一种输入设备，它周期性地在每经过一个指定的时间间隔后就通知应用程序一次。您的程序将时间间隔告诉Windows，例如「每10秒钟通知我一声」，然后Windows给您的程序发送周期性发生的WM_TIMER消息以表示时间到了。

初看之下，Windows定时器似乎不如键盘和鼠标设备重要，而且对许多应用程序来说确实如此。但是，定时器比您可能认为的要重要得多，它不只用于计时程序，比如出现在工具列中的Windows时钟和这一章中的两个时钟程序。下面是Windows定时器的其它应用，有些可能并不那么明显：

多任务虽然Windows 98是一个优先权式的多任务环境，但有时候如果程序尽快将控制传回给Windows效率会更高。如果一个程序必须进行大量的处理，那么它可以将作业分成小块，每接收到一个WM_TIMER消息处理一块（我将在第二十章中对此做更多的讨论）。

维护更新过的状态报告程序可以利用定时器来显示持续变化信息的「实时」更新，比如关于系统资源的变化或某个任务的进展情况。

实作「自动储存」功能定时器提示Windows程序在指定的时间过去后把使用者的工作储存到磁盘上。

终止程序展示版本的执行一些程序的展示版本被设计成在其开始后，多长时间结束，比如说，30分钟。如果时间已到，那么定时器就会通知应用程序。

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。利用定时器可以消除由于微处理器速度不同而造成的不一致。

多媒体播放CD声音、声音或音乐的程序通常在背景播放声音数据。一个程序可以使用定时器来周期性地检查已播放了多少声音数据，并据此协调屏幕上的视觉信息。

另一项应用可以保证程序在退出窗口消息处理程序后，能够重新得到控制。在大多数时情况下，程序不能够知道何时下一个消息会到来。

定时器入门

您可以通过呼叫SetTimer函数为您的Windows程序分配一个定时器。SetTimer有一个时间间隔范围为1毫秒到4,294,967,295毫秒（将近50天）的整数型态参数，这个值指示Windows每隔多久时间给您的程序发送WM_TIMER消息。例如，如果间隔为1000毫秒，那么Windows将每秒给程序发送一个WM_TIMER消息。

当您的程序用完定时器时，它呼叫KillTimer函数来停止定时器消息。在处理WM_TIMER消息时，您可以通过呼叫KillTimer函数来编写一个「限用一次」的定时器。KillTimer呼叫清除消息队列中尚未被处理的WM_TIMER消息，从而使程序在呼叫KillTimer之后就不会再接收到WM_TIMER消息。

系统和定时器

Windows定时器是PC硬件和ROM BIOS架构下之定时器一种相对简单的扩充。回到Windows以前的MS-DOS程序写作环境下，应用程序能够通过拦截者称为timer tick的BIOS中断来实作时钟或定时器。一些为MS-DOS编写的程序自己拦截这个硬件中断以实作时钟和定时器。这些中断每54.915毫秒产生一次，或者大约每秒18.2次。这是原始的IBM PC的微处理器时脉值4.772720 MHz

被218所除而得出的结果。

Windows应用程序不拦截BIOS中断，相反地，Windows本身处理硬件中断，这样应用程序就不必进行处理。对于目前拥有定时器的每个程序，Windows储存一个每次硬件timer tick减少的计数。当这个计数减到0时，Windows在应用程序消息队列中放置一个WM_TIMER消息，并将计数重置为其最初值。

因为Windows应用程序从正常的消息队列中取得WM_TIMER消息，所以您的程序在进行其它处理时不必担心WM_TIMER消息会意外中断了程序。在这方面，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把这些事件翻译为规律、结构化和顺序化的消息。

在Windows 98中，定时器与其下的PC定时器一样具有55毫秒的分辨率。在Microsoft Windows NT中，定时器的分辨率为10毫秒。

Windows应用程序不能以高于这些分辨率的频率（在Windows 98下，每秒18.2次，在Windows NT下，每秒大约100次）接收WM_TIMER消息。在SetTimer呼叫中指定的时间间隔总是截尾后tick数的整数倍。例如，1000毫秒的间隔除以54.925毫秒，得到18.207个tick，截尾后是18个tick，它实际上是989毫秒。对每个小于55毫秒的间隔，每个tick都会产生一个WM_TIMER消息。

定时器消息不是异步的

因为定时器使用硬件定时器中断，程序写作者有时会误解，认为他们的程序会异步地被中断来处理WM_TIMER消息。

然而，WM_TIMER消息并不是异步的。WM_TIMER消息放在正常的消息队列之中，和其它消息排列在一起，因此，如果在SetTimer呼叫中指定间隔为1000毫秒，那么不能保证程序每1000毫秒或者989毫秒就会收到一个WM_TIMER消息。如果其它程序的执行事件超过一秒，在此期间内，您的程序将收不到任何WM_TIMER消息。您可以使用本章的程序来展示这一点。事实上，Windows对WM_TIMER消息的处理非常类似于对WM_PAINT消息的处理，这两个消息都是低优先级的，程序只有在消息队列中没有其它消息时才接收它们。

WM_TIMER还在另一方面和WM_PAINT相似：Windows不能持续向消息队列中放入多个WM_TIMER消息，而是将多余的WM_TIMER消息组合成一个消息。因此，应用程序不会一次收到多个这样的消息，尽管可能在短时间内得到两个WM_TIMER消息。应用程序不能确定这种处理方式所导致的WM_TIMER消息「遗漏」的数目。

这样，WM_TIMER消息仅仅在需要更新时才提示程序，程序本身不能经由统计WM_TIMER消息的数目来计时（在本章后面，我们将编写两个每秒更新一次的时钟程序，并可以看到如何做到这一点）。

为了方便起见，下面在讨论时钟时，我将使用「每秒得到一次WM_TIMER消息」这样的叙述，但是请记住，这些消息并非精确的tick中断。

定时器的使用：三种方法

如果您需要在整个程序执行期间都使用定时器，那么您将得从WinMain函数中或者在处理WM_CREATE消息时呼叫SetTimer，并在退出WinMain或响应WM_DESTROY消息时呼叫KillTimer。根据呼叫SetTimer时使用的参数，可以下列三种方法之一使用定时器。

方法一

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正常窗口消息处

理程序中，SetTimer呼叫如下所示：

SetTimer (hwnd, 1, uiMsecInterval, NULL) ;

第一个参数是其窗口消息处理程序将接收WM_TIMER消息的窗口句柄。第二个参数是定时器ID，它是一个非0数值，在整个例子中假定为1。第三个参数是一个32位无正负号整数，以毫秒为单位指定一个时间间隔，一个60,000的值将使Windows每分钟发送一次WM_TIMER消息。

您可以通过呼叫

KillTimer (hwnd, 1) ;

在任何时刻停止WM_TIMER消息（即使正在处理WM_TIMER消息）。此函数的第二个参数是SetTimer呼叫中所用的同一个定时器ID。在终止程序之前，您应该响应WM_DESTROY消息停止任何活动的定时器。

当您的窗口消息处理程序收到一个WM_TIMER消息时，wParam参数等于定时器的ID值（上述情形为1），lParam参数为0。如果需要设定多个定时器，那么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消息处理程序的WM_TIMER消息的不同而不同。为了使程序更具有可读性，您可以使用#define叙述定义不同的定时器ID：

#define TIMER_SEC 1

#define TIMER_MIN 2

然后您可以使用两个SetTimer呼叫来设定两个定时器：

SetTimer (hwnd, TIMER_SEC, 1000, NULL) ;

SetTimer (hwnd, TIMER_MIN, 60000, NULL) ;

WM_TIMER的处理如下所示：

```
case WM_TIMER:
  switch (wParam)
  {
  case TIMER_SEC:
    //每秒一次的处理
    break ;
  case TIMER_MIN:
    //每分钟一次的处理
    break ;
  }
  return 0 ;
```

如果您想将一个已经存在的定时器设定为不同的时间间隔，您可以简单地用不同的时间值再次呼叫SetTimer。在时钟程序里，如果显示秒或不显示秒是可以选择的，您就可以这样做，只需简单地将时间间隔在1000毫秒和60 000毫秒间切换就可以了。

程序8-1显示了一个使用定时器的简单程序，名为BEEPER1，定时器的时间间隔设定为1秒。当它收到WM_TIMER消息时，它将显示区域的颜色由蓝色变为红色或由红色变为蓝色，并通过呼叫MessageBeep函数发出响声。（虽然MessageBeep通常用于MessageBox，但它确实是一个全功能的鸣叫函数。在有声卡的PC机上，一般可以使用不同的MB_ICON参数作为MessageBeep的一个参数以用于MessageBox，来播放使用者在「控制台」的「声音」程序中选择的不同声音）。

BEEPER1在窗口消息处理程序处理WM_CREATE消息时设定定时器。在处理WM_TIMER消息处理期间，BEEPER1呼叫MessageBeep，翻转bFlipFlop的值并使窗口无效以产生WM_PAINT消息。在处理WM_PAINT消息处理期间，BEEPER1通过呼叫GetClientRect获得窗口大小的RECT结构，并通过呼叫FillRect改变窗口的颜色。

程序8-1 BEEPER1

BEEPER1.C

```
/*------------------------------------------------------------------------
BEEPER1.C -- Timer Demo Program No. 1
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

#define ID_TIMER 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Beeper1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Beeper1 Timer Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL fFlipFlop = FALSE ;
  HBRUSH hBrush ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rc ;

  switch (message)
  {
  case WM_CREATE:
    SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
    return 0 ;
  case WM_TIMER :
```

```
    MessageBeep (-1) ;
    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) ;
}
```

因为BEEPER1每次收到WM_TIMER消息时，都用颜色的变换显示出来，所以您可以通过呼叫BEEPER1来查看WM_TIMER消息的性质，并完成Windows内部的一些其它操作。

例如，首先呼叫**控制台**的 **显示器**程序，选择**效果**，确定 **拖曳时显示窗口内容**复选框没有被选中。现在，试着移动或者缩放BEEPER1窗口，这将导致程序进入「模态消息循环」。Windows通过在内部消息而非您程序的消息循环中拦截所有消息，来禁止对移动或者缩放操作的任何干扰。通过此循环到达程序窗口的大多数消息都被丢弃，这就是BEEPER1停止蜂鸣的原因。当完成了移动与缩放之后，您将会注意到BEEPER1不能取得它所丢弃的所有WM_TIMER消息，尽管前两个消息的间隔可能少于1秒。

在「拖曳时显示窗口内容」复选框被选中时，Windows中，的模态消息循环会试图给您的窗口消息处理程序传递一些丢失的消息。这样做有时工作得很好，有时却不行。

方法二

设定定时器的第一种方法是把WM_TIMER消息发送到通常的窗口消息处理程序，而第二种方法是让Windows直接将定时器消息发送给您程序的另一个函数。

接收这些定时器消息的函数被称为「callback」函数，这是一个在您的程序之中但是由Windows呼叫的函数。您先告诉Windows此函数的地址，然后Windows呼叫此函数。这看起来也很熟悉，因为程序的窗口消息处理程序实际上也是一种callback函数。当注册窗口类别时，要将函数的地址告诉Windows，当发送消息给程序时，Windows会呼叫此函数。

SetTimer并非是唯一使用callback函数的Windows函数。CreateDialog和DialogBox函数（将在第十一章中介绍）使用callback函数处理对话框中的消息；有几个Windows函数（EnumChildWindow、EnumFonts、EnumObjects、EnumProps和EnumWindow）把列举信息传递给callback函数；还有几个不那么常用的函数（GrayString、LineDDA和SetWindowHookEx）也要求callback函数。

像窗口消息处理程序一样，callback函数也必须定义为CALLBACK，因为它是由Windows从程序的程序代码段呼叫的。callback函数的参数和callback函数的传回值取决于callback函数的目的。跟定时器有关的callback函数中，输入参数与窗口消息处理程序的输入参数一样。定时器callback函数不向Windows传回值。

我们把以下的callback函数称为TimerProc（您能够选择与其它一些用语不会发生冲突的任何名称），它只处理WM_TIMER消息：

```
VOID CALLBACK TimerProc ( HWND hwnd, UINT message,
```

```
                    UINT iTimerID, DWORD dwTime)
{
  //处理WM_TIMER消息
}
```

TimerProc的参数hwnd是在呼叫SetTimer时指定的窗口句柄。Windows只把WM_TIMER消息送给TimerProc，因此消息参数总是等于WM_TIMER。iTimerID值是定时器ID，dwTimer值是与从GetTickCount函数的传回值相容的值。这是自Windows启动后所经过的毫秒数。

在BEEPER1中已经看到过，用第一种方法设定定时器时要求下面格式的SetTimer呼叫：

SetTimer (hwnd, iTimerID, iMsecInterval, NULL) ;

您使用callback函数处理WM_TIMER消息时，SetTimer的第四个参数由callback函数的地址取代，如下所示：

SetTimer (hwnd, iTimerID, iMsecInterval, TimerProc) ;

我们来看看一些范例程序代码，这样您就会了解这些东西是如何组合在一起的。在功能上，除了Windows发送一个定时器消息给TimerProc而非WndProc之外，程序8-2所示的BEEPER2程序与BEEPER1是相同的。注意，TimerProc和WndProc一起被声明在程序的开始处。

程序8-2　BEEPER2

BEEPER2.C

```
/*------------------------------------------------------------------------
BEEPER2.C -- Timer Demo Program No. 2
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
VOID CALLBACK TimerProc (HWND, UINT, UINT, DWORD ) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static char szAppName[] = "Beeper2" ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, "Beeper2 Timer Demo",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
```

```
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_CREATE:
    SetTimer (hwnd, ID_TIMER, 1000, TimerProc) ;
    return 0 ;
  case WM_DESTROY:
    KillTimer (hwnd, ID_TIMER) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

VOID CALLBACK TimerProc (HWND hwnd, UINT message, UINT iTimerID, DWORD dwTime)
{
  static BOOL fFlipFlop = FALSE ;
  HBRUSH hBrush ;
  HDC hdc ;
  RECT rc ;

  MessageBeep (-1) ;
  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) ;
}
```

方法三

设定定时器的第三种方法类似于第二种方法，只是传递给SetTimer的hwnd参数被设定为NULL，并且第二个参数（通常为定时器ID）被忽略了，最后，此函数传回定时器ID：

iTimerID = SetTimer (NULL, 0, wMsecInterval, TimerProc) ;

如果没有可用的定时器，那么从SetTimer传回的iTimerID值将为NULL。

KillTimer的第一个参数（通常是窗口句柄）也必须为NULL，定时器ID必须是SetTimer的传回值：

KillTimer (NULL, iTimerID) ;

传递给TimerProc定时器函数的hwnd参数也必须是NULL。这种设定定时器的方法很少被使用。如果在您的程序在不同时刻有一系列的SetTimer呼叫，而又不希望追踪您已经用过了那些定时器ID，那么使用此方法是很方便的。

既然您已经知道了如何使用Windows定时器，就可以开始讨论一些有用的定时器程序了。

定时器用于时钟

时钟是定时器最明显的应用，因此让我们来看看两个时钟，一个数字时钟，一个模拟时钟。

建立数字时钟

程序8-3所示的DIGCLOCK程序，使用类似LED的7个显示方块显示了目前的时间。

程序8-3　DIGCLOCK

DIGCLOCK.C

```
/*------------------------------------------------------------------------
DIGCLOCK.C -- Digital Clock
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("DigClock") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Digital Clock"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void DisplayDigit (HDC hdc, int iNumber)
{
  static BOOL fSevenSegment [10][7] = {
```

```
    1, 1, 1, 0, 1, 1, 1, // 0
      0, 0, 1, 0, 0, 1, 0, // 1
      1, 0, 1, 1, 1, 0, 1, // 2
      1, 0, 1, 1, 0, 1, 1, // 3
      0, 1, 1, 1, 0, 1, 0, // 4
      1, 1, 0, 1, 0, 1, 1, // 5
      1, 1, 0, 1, 1, 1, 1, // 6
      1, 0, 1, 0, 0, 1, 0, // 7
      1, 1, 1, 1, 1, 1, 1, // 8
      1, 1, 1, 1, 0, 1, 1 } ; // 9
    static POINT ptSegment [7][6] = {
      7, 6, 11, 2, 31, 2, 35, 6, 31, 10, 11, 10,
        6, 7, 10, 11, 10, 31, 6, 35, 2, 31, 2, 11,
        36, 7, 40, 11, 40, 31, 36, 35, 32, 31, 32, 11,
        7 , 36, 11, 32, 31, 32, 35, 36, 31, 40, 11, 40,
        6 , 37, 10, 41, 10, 61, 6, 65, 2, 61, 2, 41,
        36, 37, 40, 41, 40, 61, 36, 65, 32, 61, 32, 41,
        7 , 66, 11, 62, 31, 62, 35, 66, 31, 70, 11, 70 } ;
      int iSeg ;
      for (iSeg = 0 ; iSeg < 7 ; iSeg++)
        if (fSevenSegment [iNumber][iSeg])
          Polygon (hdc, ptSegment [iSeg], 6) ;
}

void DisplayTwoDigits (HDC hdc, int iNumber, BOOL fSuppress)
{
  if (!fSuppress || (iNumber / 10 != 0))
    DisplayDigit (hdc, iNumber / 10) ;
  OffsetWindowOrgEx (hdc, -42, 0, NULL) ;
  DisplayDigit (hdc, iNumber % 10) ;
  OffsetWindowOrgEx (hdc, -42, 0, NULL) ;
}

void DisplayColon (HDC hdc)
{
  POINT ptColon [2][4] = { 2, 21, 6, 17, 10, 21, 6, 25,
    2,51, 6, 47, 10, 51, 6, 55 } ;

  Polygon (hdc, ptColon [0], 4) ;
  Polygon (hdc, ptColon [1], 4) ;

  OffsetWindowOrgEx (hdc, -12, 0, NULL) ;
}

void DisplayTime (HDC hdc, BOOL f24Hour, BOOL fSuppress)
{
  SYSTEMTIME st ;
  GetLocalTime (&st) ;
  if (f24Hour)
    DisplayTwoDigits (hdc, st.wHour, fSuppress) ;
  else
    DisplayTwoDigits (hdc, (st.wHour %= 12) ? st.wHour : 12, fSuppress) ;
  DisplayColon (hdc) ;
  DisplayTwoDigits (hdc, st.wMinute, FALSE) ;
  DisplayColon (hdc) ;
  DisplayTwoDigits (hdc, st.wSecond, FALSE) ;
}
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL f24Hour, fSuppress ;
  static HBRUSH hBrushRed ;
  static int cxClient, cyClient ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  TCHAR szBuffer [2] ;

  switch (message)
  {
  case WM_CREATE:
```

```
    hBrushRed = CreateSolidBrush (RGB (255, 0, 0)) ;
    SetTimer (hwnd, ID_TIMER, 1000, NULL) ;// fall through
  case WM_SETTINGCHANGE:
    GetLocaleInfo (LOCALE_USER_DEFAULT, LOCALE_ITIME, szBuffer, 2) ;
    f24Hour = (szBuffer[0] == '1') ;
    GetLocaleInfo (LOCALE_USER_DEFAULT, LOCALE_ITLZERO, szBuffer, 2) ;
    fSuppress = (szBuffer[0] == '0') ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_TIMER:
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SetMapMode (hdc, MM_ISOTROPIC) ;
    SetWindowExtEx (hdc, 276, 72, NULL) ;
    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

    SetWindowOrgEx (hdc, 138, 36, NULL) ;
    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
    SelectObject (hdc, GetStockObject (NULL_PEN)) ;
    SelectObject (hdc, hBrushRed) ;
    DisplayTime (hdc, f24Hour, fSuppress) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    KillTimer (hwnd, ID_TIMER) ;
    DeleteObject (hBrushRed) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIGCLOCK窗口如图8-1所示。

图8-1 DIGCLOCK的屏幕显示

虽然，在图8-1中您看不到时钟的数字是红色的。DIGCLOCK的窗口消息处理程序在处理WM_CREATE消息处理期间建立了一个红色的画刷并在处理WM_DESTROY消息处理期间清除它。WM_CREATE消息也为DIGCLOCK设定了一个一秒的定时器，该定时器在处理WM_DESTROY消息

处理期间被终止（待会将讨论对GetLocaleInfo的呼叫）。

在收到WM_TIMER消息后，DIGCLOCK的窗口过程调用InvalidateRect简单地使整个窗口无效。这不是最佳方法，因为每秒整个窗口都要被擦除和重画，有时会引起显示器的闪烁。依据目前的时间使窗口需要更新的部分无效是最好的解决方法。然而，在逻辑上这样做的确很复杂。

在处理WM_TIMER消息处理期间使窗口无效会迫使所有程序的真正活动转入WM_PAINT。DIGCLOCK在WM_PAINT消息一开始将映像方式设定为MM_ISOTROPIC。这样，DIGCLOCK将使用水平方向和垂直方向相等的轴。这些轴（由SetWindowExtEx呼叫设定）是水平276个单位，垂直72个单位。当然，这些轴定得有点太随意了，但它们是按照时钟数字元的大小和间距安排的。

DIGCLOCK将窗口原点设定为（138,36），这是窗口范围的中心；将视埠原点设定为（cxClient / 2,cyClient / 2）。这意味着时钟的显示位于DIGCLOCK显示区域的中心，但是该DIGCLOCK也可以使用在显示屏左上角的原点（0, 0）的轴。

然后WM_PAINT将目前画刷设定为之前建立的红画刷，将目前画笔设定为NULL_PEN，并呼叫DIGCLOCK中的函数DisplayTime。

取得目前时间

DisplayTime函数开始呼叫Windows函数GetLocalTime，它带有一个的SYSTEMTIME结构的参数，在WINBASE.H中定义为：

```
typedef struct _SYSTEMTIME
{
  WORD wYear ;
  WORD wMonth ;
  WORD wDayOfWeek ;
  WORD wDay ;
  WORD wHour ;
  WORD wMinute ;
  WORD wSecond ;
  WORD wMilliseconds ;
}
SYSTEMTIME, * PSYSTEMTIME ;
```

很明显，SYSTEMTIME结构包含日期和时间。月份由1开始递增（也就是说，一月是1），星期由0开始递增（星期天是0）。wDay成员是本月目前的日子，也是由1开始递增的。

SYSTEMTIME主要用于GetLocalTime和GetSystemTime函数。GetSystemTime函数传回目前的世界时间（Coordinated Universal Time，UTC），大概与英国格林威治时间相同。GetLocalTime函数传回当地时间，依据计算机所在的时区。这些值的精确度完全决定于使用者所调整的时间精确度以及是否指定了正确的时区。可以双击工作列的时间显示来检查计算机上的时区设定。第二十三章会有一个程序，能够通过Internet精确地设定时间。

Windows还有SetLocalTime和SetSystemTime函数，以及在/Platform SDK/Windows Base Services/General Library/Time中说明的其它与时间有关的函数。

显示数字和冒号

如果DIGCLOCK使用一种仿真7段显示的字体将会简单一些。否则，它就得使用Polygon函数做所有的工作。

DIGCLOCK中的DisplayDigit函数定义了两个数组。fSevenSegment数组有7个BOOL值，用于从0到9的每个十进制数。这些值指出了哪一段需要显示（为1），哪一段不需要显示（为0）。在这个数组中，7段由上到下、由左到右排序。7段中的每个段都是一个6边的多边形。ptSegment数组是一个POINT结构的数组，指出了7个段中每个点的图形坐标。每个数字由下列程序代码画出：

```
for (iSeg = 0 ; iSeg < 7 ; iSeg++)
  if ( fSevenSegment [iNumber][iSeg])
    Polygon (hdc, ptSegment [iSeg], 6) ;
```

类似地（但更简单），DisplayColon函数在小时与分钟、分钟与秒之间画一个冒号。数字是42个单位宽，冒号是12个单位宽，因此6个数字与2个冒号，总宽度是276个单位，SetWindowExtEx呼叫中使用了这个大小。

回到DisplayTime函数，原点位于最左数字位置的左上角。DisplayTime呼叫DisplayTwoDigits，DisplayTwoDigits呼叫DisplayDigit两次，并且在每次呼叫OffsetWindowOrgEx后，将窗口原点向右移动42个单位。类似地，DisplayColon函数在画完冒号后，将窗口原点向右移动12个单位。用这种方法，不管对象出现在窗口内的哪个地方，函数对数字和冒号都使用同样的坐标。

这个程序的其它技巧是以12小时或24小时的格式显示时间以及当最左边的小时数字为0时不显示它。

国际化

尽管像DIGCLOCK这样显示时间是非常简单的，但是要显示复杂的日期和时间还是要依赖Windows的国际化支持。格式化日期和时间的最简单的方法是呼叫GetDateFormat和GetTimeFormat函数。这些函数在/Platform SDK/Windows Base Services/General Library/String Manipulation/String Manipulation Reference/String Manipulation Functions中有记载，但是它们在/Platform SDK/Windows Base Services/International Features/National Language Support中进行了说明。这些函数接受SYSTEMTIME结构并且依据使用者在「控制台」的「区域设定」 程序中所做的选择而将日期和时间格式化。

DIGCLOCK不能使用GetDateFormat函数，因为它只知道显示数字和冒号，然而，DIGCLOCK应该能够根据使用者的参数选择来显示12小时或24小时的格式，并禁止（或不禁止）开头的小时数字。您可以从GetLocaleInfo函数中取得这种信息。虽然GetLocaleInfo在/Platform SDK/Windows Base Services/General Library/String Manipulation/String Manipulation Reference/String Manipulation Functions中有记载，但是这个函数使用的标识符在/Platform SDK/Windows Base Services/International Features/National Language Support/National Language Support Constants中有说明。

DIGCLOCK在处理WM_CREATE消息时，最初呼叫GetLocaleInfo两次，第一次使用LOCALE_ITIME标识符（确定使用的是12小时还是24小时格式），然后使用LOCALE_ITLZERO标识符（在小时显示中禁止前面显示0）。GetLocaleInfo函数在字符串中传回所有的信息，但是在大多数情况下把字符串转变为整数并不是非常容易。DIGCLOCK把字符串储存在两个静态变量中并把它们传递给DisplayTime函数。

如果使用者更改了任何系统设定，则会将WM_SETTINGCHANGE消息传送给所有的应用程序。DIGCLOCK通过再次呼叫GetLocaleInfo处理这个消息。以这种方式，您可以在「控制台」的「区域设定」 程序中进行不同的设定来实验一下。

在理论上，DIGCLOCK也应该使用LOCALE_STIME标识符呼叫GetLocaleInfo。这会传回使用者为时间的小时、分钟和秒等单个部分选择的字符。因为DIGCLOCK被设定为仅显示冒号，所以不管选择了什么，都会得到冒号。要指出时间是A.M.或P.M.，应用程序可以使用带有LOCALE_S1159和LOCALE_S2359标识符的GetLocaleInfo函数。这些标识符使程序获得适合于使用者国家/地区和语言的字符串。

我们也可以让DIGCLOCK处理WM_TIMECHANGE消息，这样它将系统时间与日期发生变化的消息通知应用程序。DIGCLOCK因WM_TIMER消息而每秒更新一次，实际上没有必要这样作，对

WM_TIMECHANGE消息的处理使得每分钟更新一次的时钟变得更为合理。

建立模拟时钟

模拟时钟不必关心国际化问题，但是由于图形所引起的复杂性却抵消了这种简化。为了正确地产生时钟，您需要知道一些三角函数。CLOCK如程序8-4所示。

程序8-4 CLOCK

CLOCK.C

```
/*--------------------------------------------------------------------------
CLOCK.C -- Analog Clock Program
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>

#define ID_TIMER 1
#define TWOPI (2 * 3.14159)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Clock") ;
  HWND hwnd;
  MSG msg;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("Program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Analog Clock"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void SetIsotropic (HDC hdc, int cxClient, int cyClient)
{
  SetMapMode (hdc, MM_ISOTROPIC) ;
  SetWindowExtEx (hdc, 1000, 1000, NULL) ;
```

```
  SetViewportExtEx (hdc, cxClient / 2, -cyClient / 2, NULL) ;
  SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;
}

void RotatePoint (POINT pt[], int iNum, int iAngle)
{
  int i ;
  POINT ptTemp ;

  for (i = 0 ; i < iNum ; i++)
  {
    ptTemp.x = (int) (pt[i].x * cos (TWOPI * iAngle / 360) +
      pt[i].y * sin (TWOPI * iAngle / 360)) ;

    ptTemp.y = (int) (pt[i].y * cos (TWOPI * iAngle / 360) -
      pt[i].x * sin (TWOPI * iAngle / 360)) ;

    pt[i] = ptTemp ;
  }
}

void DrawClock (HDC hdc)
{
  int iAngle ;
  POINT pt[3] ;
  for (iAngle = 0 ; iAngle < 360 ; iAngle += 6)
  {
    pt[0].x = 0 ;
    pt[0].y = 900 ;

    RotatePoint (pt, 1, iAngle) ;

    pt[2].x = pt[2].y = iAngle % 5 ? 33 : 100 ;

    pt[0].x -= pt[2].x / 2 ;
    pt[0].y -= pt[2].y / 2 ;

    pt[1].x = pt[0].x + pt[2].x ;
    pt[1].y = pt[0].y + pt[2].y ;

    SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
    Ellipse (hdc, pt[0].x, pt[0].y, pt[1].x, pt[1].y) ;
  }
}

void DrawHands (HDC hdc, SYSTEMTIME * pst, BOOL fChange)
{
  static POINT pt[3][5] ={0, -150, 100, 0, 0, 600, -100, 0, 0, -150,
    0, -200, 50, 0, 0, 800, -50, 0, 0,-200,
    0,0, 0, 0, 0, 0, 0, 0, 0, 800 } ;
  int i, iAngle[3] ;
  POINT ptTemp[3][5] ;

  iAngle[0] = (pst->wHour * 30) % 360 + pst->wMinute / 2 ;
  iAngle[1] = pst->wMinute * 6 ;
  iAngle[2] = pst->wSecond * 6 ;

  memcpy (ptTemp, pt, sizeof (pt)) ;
  for (i = fChange ? 0 : 2 ; i < 3 ; i++)
  {
    RotatePoint (ptTemp[i], 5, iAngle[i]) ;
    Polyline (hdc, ptTemp[i], 5) ;
  }
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int cxClient, cyClient ;
  static SYSTEMTIME stPrevious ;
```

```
  BOOL fChange ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  SYSTEMTIME st ;

  switch (message)
  {
  case WM_CREATE :
    SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
    GetLocalTime (&st) ;
    stPrevious = st ;
    return 0 ;
  case WM_SIZE :
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_TIMER :
    GetLocalTime (&st) ;
    fChange = st.wHour != stPrevious.wHour ||
      st.wMinute != stPrevious.wMinute ;

    hdc = GetDC (hwnd) ;
    SetIsotropic (hdc, cxClient, cyClient) ;
    SelectObject (hdc, GetStockObject (WHITE_PEN)) ;
    DrawHands (hdc, &stPrevious, fChange) ;

    SelectObject (hdc, GetStockObject (BLACK_PEN)) ;
    DrawHands (hdc, &st, TRUE) ;
    ReleaseDC (hwnd, hdc) ;

    stPrevious = st ;
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;
    SetIsotropic (hdc, cxClient, cyClient) ;
    DrawClock (hdc) ;
    DrawHands (hdc, &stPrevious, TRUE) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY :
    KillTimer (hwnd, ID_TIMER) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CLOCK屏幕显示如图8-2。

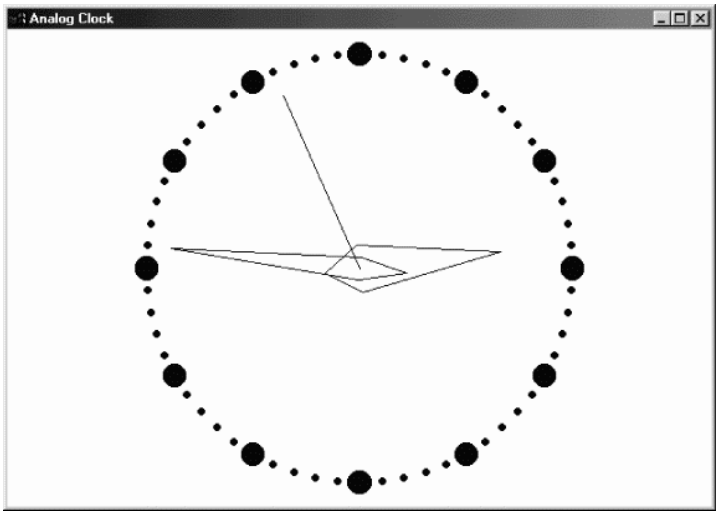


图8-2 CLOCK的屏幕显示

等方向性(isotropic)映像对于这样的应用来说是理想的，CLOCK.C中的SetIsotropic函数负责设定此模式。在呼叫SetMapMode之后，SetIsotropic将窗口范围设定为1000，并将视端口范围设定为显示区域的一半宽度和显示区域的负的一半高度。视端口原点被设定为显示区域的中心。我在第五章中讨论过，这将建立一个笛卡儿坐标系，其点(0,0)位于显示区域的中心，在所有方向上的范围都是1000。

RotatePoint函数是用到三角函数的地方，此函式的三个参数分别是一个或者多个点的数组、数组中点的个数以及以度为单位的旋转角度。函式以原点为中心按顺时针方向（这对一个时钟正合适）旋转这些点。例如，如果传给函式的点是(0,100) – 即12:00的位置 – 而角度为90度，那么该点将被变换为(100,0) – 即3:00。它使用下列公式来做到这一点：

x' = x * cos (a) + y * sin (a)

y' = y * cos (a) - x * sin (a)

RotatePoint函数在绘制时钟表面的点和表针时都是有用的，我们将马上看到这一点。

DrawClock函数绘制60个时钟表面的点，从顶部(12:00)开始，其中每个点离原点900单位，因此第一个点位于(0,900)，此后的每个点按顺时针依次增加6度。这些点中的12个直径为100个单位；其余的为33个单位。使用Ellipse函数来画点。

DrawHands函数绘制时钟的时针、分针和秒针。定义表针轮廓（当它们垂直向上时的形状）的坐标存放在一个POINT结构的数组中。根据时间，这些坐标使用RotatePoint函数进行旋转，并用Windows的Polyline函数进行显示。注意时针和分针只有当传递给DrawHands的bChange参数为TRUE时才被显示。当程序更新时钟的表针时，大多数情况下时针和分针不需要重画。

现在让我们将注意力转到窗口消息处理程序。在WM_CREATE消息处理期间，窗口消息处理程序取得目前时间并将它存放在名为dtPrevious的变量中，这个变量将在以后被用于确定时针或者分针从上次更新以来是否改变过。

第一次绘制时钟是在第一个WM_PAINT消息处理期间，这只不过是依次呼叫SetIsotropic、DrawClock和DrawHands，后者的bChange参数被设定为TRUE。

在WM_TIMER消息处理期间，WndProc首先取得新的时间并确定是否需要重新绘制时针和分针。如果需要，则使用一个白色画笔和上一次时间绘制所有的表针，从而有效地擦除它们。否则，只对秒针使用白色画笔进行擦除，然后，再使用一个黑色画笔绘制所有的表针。

以定时器进行状态报告

本章的最后一个程序是我在第五章提到过的。它是一个使用GetPixel函数的好例子。

WHATCLR （见程序8-5）显示了鼠标光标下目前像素的RGB颜色。

程序8-5 WHATCLR

WHATCLR.C

```
/*------------------------------------------------------------------------
WHATCLR.C -- Displays Color Under Cursor
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
void FindWindowSize (int *, int *) ;
```

```
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("WhatClr") ;
  HWND hwnd ;
  int cxWindow, cyWindow ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  FindWindowSize (&cxWindow, &cyWindow) ;
  hwnd = CreateWindow (szAppName, TEXT ("What Color"),
    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_BORDER,
    CW_USEDEFAULT, CW_USEDEFAULT,
    cxWindow, cyWindow,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void FindWindowSize (int * pcxWindow, int * pcyWindow)
{
  HDC hdcScreen ;
  TEXTMETRIC tm ;

  hdcScreen = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
  GetTextMetrics (hdcScreen, &tm) ;
  DeleteDC (hdcScreen) ;

  * pcxWindow = 2 * GetSystemMetrics (SM_CXBORDER) +
    12 * tm.tmAveCharWidth ;
  * pcyWindow = 2 * GetSystemMetrics (SM_CYBORDER) +
    GetSystemMetrics (SM_CYCAPTION) +
    2 * tm.tmHeight ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static COLORREF cr, crLast ;
  static HDC hdcScreen ;
  HDC hdc ;
  PAINTSTRUCT ps ;
```

```
  POINT pt ;
  RECT rc ;
  TCHAR szBuffer [16] ;

  switch (message)
  {
  case WM_CREATE:
    hdcScreen = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
    SetTimer (hwnd, ID_TIMER, 100, NULL) ;
    return 0 ;
  case WM_TIMER:
    GetCursorPos (&pt) ;
    cr = GetPixel (hdcScreen, pt.x, pt.y) ;
    SetPixel (hdcScreen, pt.x, pt.y, 0) ;

    if (cr != crLast)
    {
      crLast = cr ;
      InvalidateRect (hwnd, NULL, FALSE) ;
    }
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    GetClientRect (hwnd, &rc) ;
    wsprintf (szBuffer, TEXT (" %02X %02X %02X "),
      GetRValue (cr), GetGValue (cr), GetBValue (cr)) ;

    DrawText (hdc, szBuffer, -1, &rc,
      DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    DeleteDC (hdcScreen) ;
    KillTimer (hwnd, ID_TIMER) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

WHATCLR在WinMain中做了一点与以往不同的事。因为WHATCLR的窗口只需要显示十六进制RGB值那么大，所以它在CreateWindow函数中使用WS_BORDER窗口样式建立了一个不能改变大小的窗口。要计算窗口的大小，WHATCLR通过先呼叫CreateIC再呼叫GetSystemMetrics以取得用于视讯显示的设备内容信息。计算好的窗口宽度和高度值被传递给CreateWindow。

WHATCLR的窗口消息处理程序在处理WM_CREATE消息处理期间，呼叫CreateDC建立了用于整个视讯显示的设备内容。这个设备内容在程序的生命周期内都有效。在处理WM_TIMER消息处理期间，程序取得目前鼠标光标位置的像素。在处理WM_PAINT消息处理期间显示RGB颜色。

您可能想知道，从CreateDC函数中取得的设备内容句柄是否能让您在屏幕的任意位置显示一些东西，而不光只是取得像素颜色。答案是可以的，一般而言，让一个应用程序在另一个程控的画面区域上画图是不好的，但在某些特殊情况下，这可能会非常有用。

第九章 子窗口控件

回忆第七章的CHECKER程序。这些程序显示了矩形网格。当您在一个矩形中按下鼠标按键时，该程序就画一个x；如果您再按一次鼠标按键，那么x就消失。虽然这个程序的CHECKER1和CHECKER2版本只使用一个主窗口，但CHECKER3版本却为每个矩形使用一个子窗口。这些矩形由一个叫做ChildProc的独立窗口消息处理程序维护。

如果有必要，无论矩形是否被选中，都可以给ChildProc增加一种向其父窗口消息处理程序(WndProc)发送消息的手段。通过呼叫GetParent，子窗口消息处理程序能确定其父窗口的窗口句柄：

hwndParent = GetParent (hwnd) ;

其中，hwnd是子窗口的窗口句柄。它可以向其父窗口消息处理程序发送消息：

SendMessage (hwndParent, message, wParam, lParam) ;

那么message应该设定为什么呢？您可以随意地设定，数值大小可以与WM_USER相同或更大，这些数字代表和预先定义的WM_ 消息不冲突的消息。也许对这个消息，子窗口可以将wParam设定为它的子窗口ID。如果在该子窗口单击，那么lParam可以被设为1；如果未在该子窗口上单击，那么lParam将被设为0。这是处理方式的一种选择。

事实上，这是在建立一个「子窗口控件」。当子窗口的状态改变时，子窗口处理鼠标和键盘消息并通知父窗口。使用这种方法，子窗口就变成了其父窗口的高阶输入设备。它将与自己在屏幕上的图形外观相应的处理，对使用者输入的响应以及在发生重要的输入事件时通知另一个窗口的方法给封装起来。

虽然您可以建立自己的子窗口控件，但是也可以利用一些预先定义的窗口类别（和窗口消息处理程序）来建立标准的子窗口控件，您一定在别的Windows程序中看到过这些控件。这些控件采用的形式有：按钮、复选框、编辑方块、清单方块、下拉式清单方块、字符串卷标和卷动列。例如，如果想在您的电子表格程序的某个角落放置一个标有「Recalculate」的按钮，那么您可以通过呼叫CreateWindow来建立这个按钮。您不必担心鼠标操作、按钮显示操作或按下该按钮时的自动闪烁操作，这些是由Windows内部完成的。您所要做的只是拦截WM_COMMAND消息 – 当按钮被按下时，它通过这一消息通知您的窗口消息处理程序。真的这样简单吗？是的，一点也没错。

子窗口控件在对话框中最常用。在第十一章中您将会看到，子窗口控件的位置和尺寸，是在范例程序的资源描述叙述中的对话框模板里定义的。但是，您也可以使用预先定义的，在普通窗口显示区域里的子窗口控件。您可以呼叫一次CreateWindow来建立一个子窗口，并通过呼叫MoveWindow来调整子窗口的位置和尺寸。父窗口消息处理程序向子窗口控件发送消息，子窗口控件向父窗口消息处理程序传回消息。

在建立普通窗口时，首先定义窗口类别，并使用RegisterClass将其注册到Windows中，然后用CreateWindow命令依据该窗口类别建立一个普通窗口，从第三章开始，我们就是这么做的。但是，当您使用预先定义的某个控件时，不必为子窗口注册窗口类别，窗口类别已经存在于Windows之中，并且有一个预先定义的名字。您只需在CreateWindow中把它们用作窗口类别参数。CreateWindow中的窗口样式参数准确地定义了子窗口控件的外形和功能。Windows内建了处理发送给依据这些窗口类别建立的子窗口消息的窗口消息处理程序。

直接在您的窗口上使用子窗口控件完成某些任务，这些任务的层次低于在对话框中使用子窗口控件所要求的层次。这里，对话框管理器在您的程序和控件之间增加一个隔离层。值得一提的，您可能会发现在您的窗口上建立的子窗口控件，没有利用Tab键或方向键将输入焦点从一个控件移动到另一个控件的内部功能。子窗口控件能够获得输入焦点，但是获得后，它将不能把输入焦点传回给父窗口。这就是本章要解决的问题。

Windows程序设计的文件在两个地方讨论了子窗口控件：首先是，简单的常用控件，我们可以在/Platform SDK/User Interface Services/Controls的文件所描述的无数对话框中看到。这些子窗口包括按钮（其中包括复选框的单选按钮）、静态控件（例如文字卷标）、编辑方块（您可以在此编辑一行或多行文字）、卷动列、清单方块和下拉式清单方块。除下拉式清单方块以外，在Windows 1.0中就包括了这些控件。这部分的Windows文件还包括Rich Text文字编辑控件，它与编辑方块相似，但还允许编辑不同字体与样式的格式化文字，以及桌面应用工具列。

相对于「常用控件」，还有一些神秘的特殊控件。这些控件在/Platform SDK/User Interface Services/Shell and Common Controls/Common Controls描述。本章不讨论常用控件，但它们将出现在本书的其它部分。在这部分的Windows文件中，很容易找到您想从别的Windows应用程序中应用到您自己的应用程序里头那些部分信息。

按钮类别

下面我们将通过叫做BTNLOOK（「button look」）的程序来开始介绍按钮窗口类别，如程序9-1所示。BTNLOOK建立10个子窗口按钮控件，每个控件对应一个标准的按钮样式，因此共有10种标准按钮样式。

程序9-1 BTNLOOK

BTNLOOK.C

```
/*------------------------------------------------------------------------
BTNLOOK.C -- Button Look Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
struct
{
  int iStyle ;
  TCHAR * szText ;
}
button[] =
{
  BS_PUSHBUTTON, TEXT ("PUSHBUTTON"),
    BS_DEFPUSHBUTTON, TEXT ("DEFPUSHBUTTON"),
    BS_CHECKBOX, TEXT ("CHECKBOX"),
    BS_AUTOCHECKBOX, TEXT ("AUTOCHECKBOX"),
    BS_RADIOBUTTON, TEXT ("RADIOBUTTON"),
    BS_3STATE, TEXT ("3STATE"),
    BS_AUTO3STATE, TEXT ("AUTO3STATE"),
    BS_GROUPBOX, TEXT ("GROUPBOX"),
    BS_AUTORADIOBUTTON, TEXT ("AUTORADIO"),
    BS_OWNERDRAW, TEXT ("OWNERDRAW")
} ;

#define NUM (sizeof button / sizeof button[0])
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("BtnLook") ;
```

```
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Button Look"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndButton[NUM] ;
  static RECT rect ;
  static TCHAR szTop[] = TEXT ("message wParam lParam"),
    szUnd[] = TEXT ("_______ ______ ______"),
    szFormat[] = TEXT ("%-16s%04X-%04X %04X-%04X"),
    szBuffer[50] ;
  static int cxChar, cyChar ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  int i ;

  switch (message)
  {
  case WM_CREATE :
    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    for (i = 0 ; i < NUM ; i++)
      hwndButton[i] =CreateWindow ( TEXT("button"),button[i].szText,
      WS_CHILD | WS_VISIBLE | button[i].iStyle,
      cxChar, cyChar * (1 + 2 * i),
      20 * cxChar, 7 * cyChar / 4,
      hwnd, (HMENU) 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, lstrlen (szTop)) ;
    TextOut (hdc, 24 * cxChar, cyChar, szUnd, lstrlen (szUnd)) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DRAWITEM :
  case WM_COMMAND :
    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,
      message == WM_DRAWITEM ? TEXT ("WM_DRAWITEM") :
    TEXT ("WM_COMMAND"),
      HIWORD (wParam), LOWORD (wParam),
      HIWORD (lParam), LOWORD (lParam))) ;

    ReleaseDC (hwnd, hdc) ;
    ValidateRect (hwnd, &rect) ;
    break ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

单击按钮时，按钮就给父窗口消息处理程序发送一个WM_COMMAND消息，也就是我们所熟悉的WndProc。BTNLOOK的WndProc将该消息的wParam参数和lParam参数显示在显示区域的右边，如图9-1所示。

具有BS_OWNERDRAW样式的按钮在窗口上显示为一个背景阴影，因为这种样式的按钮是由程序来负责绘制的。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型态结构的指针。在BTNLOOK中，这些消息也同样被显示。我将在本章的后面更详细地讨论这种拥有者绘制（owner draw）按钮。

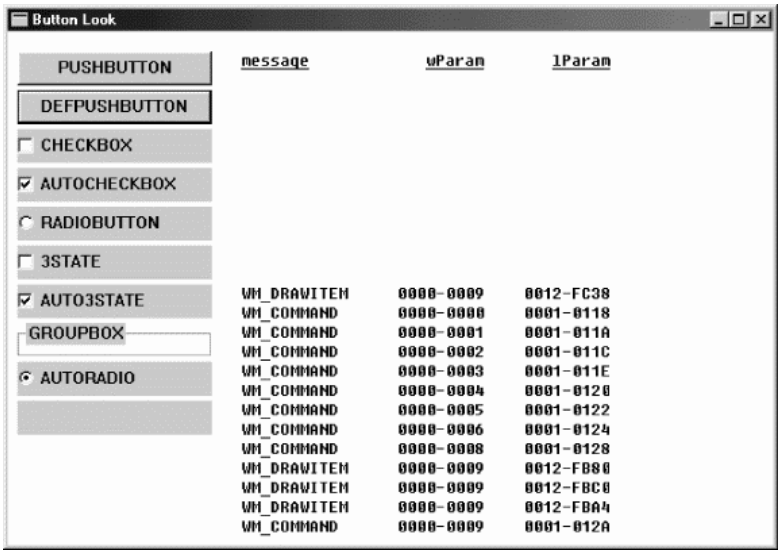


图9-1 BTNLOOK的屏幕显示

建立子窗口

BTNLOOK定义了一个叫做button的结构，它包括了按钮窗口样式和描述性字符串，它们对应于10个按钮型态，所有按钮窗口样式都以字母「BS」开头，它表示「按钮样式」。10个按钮子窗口是在WndProc中处理WM_CREATE消息的过程中使用一个for循环建立的。CreateWindow呼叫使用下面这些参数：

Class name（类别名称）	TEXT ("button")
Window text（窗口文字）	button[i].szText
Window style（窗口样式）	WS_CHILD | WS_VISIBLE | button[i].iStyle
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）	(HMENU) i
Instance handle（执行实体句柄）	((LPCREATESTRUCT) lParam) -> hInstance
Extra parameters（附加参数）	NULL

类别名称参数是预先定义的名字。窗口样式使用WS_CHILD、WS_VISIBLE以及在button结构中定义的10个按钮样式之一（BS_PUSHBUTTON、BS_DEFPUSHBUTTON等等）。窗口文字参数（对于普通窗口来说，它是显示在标题栏中的文字）将在每个按钮上显示出来。我简单地使用标识按钮样式文字的x位置和y位置参数，说明子窗口左上角相对于父窗口显示区域左上角的位置。宽度和高度参数规定了每个子窗口的宽度和高度。请注意，我用的是GetDialogBaseUnits函数来获得内定字体字符的宽度和高度。这是对话框用来获得文字尺寸的函数。此函数传回一个32位的值，其中低字组表示宽度，高字组表示高度。由于GetDialogBaseUnits传回的值与从GetTextMetrics获得的值大致上相同，但GetDialogBaseUnits有时使用起来会更方便些，而且能够与对话框控件更好地保持一致。

对每个子窗口，它的子窗口ID参数应该各不相同。在处理来自子窗口的WM_COMMAND消息时，ID帮助您的窗口消息处理程序识别出相应的子窗口。注意子窗口ID是作为CreateWindow的一个参数传递的，该参数通常用于指定程序的菜单，因此子窗口ID必须被强制转换为HMENU。

CreateWindow呼叫的执行实体句柄看起来有点奇怪，但是它利用了如下的事实，亦即在处理WM_CREATE消息的过程中，lParam实际上是指向CREATESTRUCT （「建立结构」）结构的指针，该结构有一个hInstance成员。所以将lParam转换成指向CREATESTRUCT结构的一个指针，并取出hInstance。

（有些Windows程序使用名为hInst的整体变量，使窗口消息处理程序能存取WinMain中的执行实体句柄。在WinMain中，您只需在建立主窗口之前设定：

hInst = hInstance ;

在第七章中的CHECKER3程序中，我们曾用GetWindowLong取得执行实体句柄：

GetWindowLong (hwnd, GWL_HINSTANCE)

这几种方法都是正确的。）

在呼叫CreateWindow之后，我们不必再为这些子窗口做任何事情，由Windows中的按钮窗口消息处理程序负责维护它们，并处理所有的重画工作（BS_OWNERDRAW样式的按钮例外，它要求程序绘制它，这些将在后面加以讨论）。在程序终止时，如果父窗口已经被清除，那么Windows将清除这些子窗口。

子窗口向父窗口发消息

当您执行BTNLOOK时，将看到在显示区域的左边会显示出不同的按钮型态。我在前面已经提到过，用鼠标单击按钮时，子窗口控件就向其父窗口发送一个WM_COMMAND消息。BTNLOOK拦截WM_COMMAND消息并显示wParam和lParam的值，它们的含义如下：

LOWORD (wParam)	子窗口ID
HIWORD (wParam)	通知码
lParam	子窗口句柄

如果您正在移植16位Windows程序，那么要注意改变这些消息参数以容纳32位的句柄。

子窗口ID是在建立子窗口时传递给CreateWindow的值。在BTNLOOK中，这些ID被显示在显示区域中，并使用0到9分别标识10个按钮。子窗口句柄是Windows从CreateWindow传回的值。

通知码更详细表示了消息的含义。按钮通知码的可能值在Windows表头文件中定义如下：

表9-1

按钮通知码标识符	值
BN_CLICKED	0
BN_PAINT	1
BN_HILITE or BN_PUSHED	2
BN_UNHILITE or BN_UNPUSHED	3
BN_DISABLE	4
BN_DOUBLECLICKED or BN_DBLCLK	5
BN_SETFOCUS	6
BN_KILLFOCUS	7

实际上，您不会看到这些按钮值中的大多数。从1到4的通知码是用于一种叫做BS_USERBUTTON的已不再使用的按钮的（它已经由BS_OWNERDRAW和另一种不同的通知方式所替换）。通知码6到7只有当按钮样式包括标识BS_NOTIFY才发送。通知码5只对BS_RADIOBUTTON、BS_AUTORADIOBUTTON和BS_OWNERDRAW按钮发送，或者当按钮样式中包括BS_NOTIFY时，也为其它按钮发送。

您会注意到，在用鼠标单击按钮时，该按钮文字的周围会有虚线。这表示该按钮拥有了输入焦点，所有键盘输入都将传送给子窗口按钮控件，而不是传送给主窗口。但是，当该按钮控件拥有输入焦点时，它将忽略所有的键盘输入，除了Spacebar键例外，此时Spacebar键与鼠标具有相同的效果。

父窗口向子窗口发送消息

虽然BTNLOOK中没有显示这一事实，但是父窗口消息处理程序也能向子窗口控件发送消息。这些消息包括以前缀WM开头的许多消息。另外，在WINUSER.H中还定义了8个按钮说明消息；前缀BM表示「按钮消息」。这些按钮消息如下表所示：

表9-2

按钮消息	值
BM_GETCHECK	0x00F0
BM_SETCHECK	0x00F1
BM_GETSTATE	0x00F2
BM_SETSTATE	0x00F3
BM_SETSTYLE	0x00F4
BM_CLICK	0x00F5
BM_GETIMAGE	0x00F6
BM_SETIMAGE	0x00F7

BM_GETCHECK和BM_SETCHECK消息由父窗口发送给子窗口控件，以取得或者设定复选框和单选按钮的选中标记。BM_GETSTATE和BM_SETSTATE消息表示按钮处于正常状态还是（鼠标或Spacebar键按下时的）「按下」状态。我们将在讨论按钮的每种型态时，看到这些消息是如何起作用的。BM_SETSTYLE消息允许您在按钮建立之后改变按钮样式。

每个子窗口控件都具有一个在其兄弟中唯一的窗口句柄和ID值。对于句柄和ID这两者，知道其中的一个您就可以获得另一个。如果您知道子窗口控件的窗口句柄，那么您可以用下面的叙述来获得ID：

```
id = GetWindowLong (hwndChild, GWL_ID) ;
```

第七章的CHECKER3程序曾用此函数（与SetWindowLong一起）来维护注册窗口类别时保留的特殊区域的数据。在建立子窗口时，Windows保留了GWL_ID标识符存取的数据。您也可以使用：

```
id = GetDlgCtrlID (hwndChild) ;
```

虽然函数中的「Dlg」部分指的是对话框，但实际上这是一个通用的函数。

知道ID和父窗口句柄，您就能获得子窗口句柄：

```
hwndChild = GetDlgItem (hwndParent, id) ;
```

按键

在BTNLOOK中显示的前两个按钮是「压入」按钮。按钮是一个矩形，包括了CreateWindow呼叫中窗口文字参数所指定的文字。该矩形占用了在CreateWindow或者MoveWindow呼叫中给出的全部高度和宽度，而文字在矩形的中心。

按键控件主要用来触发一个立即响应的动作，而不保留任何形式的开/关指示。两种型态的按钮控件有两种窗口样式，分别叫做BS_PUSHBUTTON和BS_DEFPUSHBUTTON，BS_DEFPUSHBUTTON中的「DEF」代表「内定」。当用来设计对话框时，BS_PUSHBUTTON控件和BS_DEFPUSHBUTTON控件的作用不同。但是当用作子窗口控件时，两种型态的按钮作用相同，尽管BS_DEFPUSHBUTTON的边框要粗一些。

当按钮的高度为文字字符高度的7/4倍时，按钮的外观看起来最好，其中文字字符由BTNLOOK使用；而按钮的宽度至少调节到文字的宽度再加上两个字符的宽度。

当鼠标光标在按钮中时，按下鼠标按键将使按钮用三维阴影重画自己，就好像真的被按下一样。

放开鼠标按键时，就恢复按钮的原貌，并向父窗口发送一个WM_COMMAND消息和BN_CLICKED通知码。与其它按钮型态相似，当按钮拥有输入焦点时，在文字的周围就有虚线，按下及释放Spacebar键与按下及释放鼠标按键具有相同的效果。

您可以通过给窗口发送BM_SETSTATE消息来仿真按钮闪动。以下的操作将导致按钮被按下：

SendMessage (hwndButton, BM_SETSTATE, 1, 0) ;

下面的呼叫使按钮恢复正常：

SendMessage (hwndButton, BM_SETSTATE, 0, 0) ;

hwndButton窗口句柄是从CreateWindow呼叫传回的值。

您也可以向按键发送BM_GETSTATE消息，子窗口控件传回按钮目前的状态：如果按钮被按下，则传回TRUE；如果按钮处于正常状态，则传回FALSE。但是，绝大多数应用并不需要这一消息。因为按钮不保留任何开/关信息，所以BM_SETCHECK消息和BM_GETCHECK消息不会被用到。

复选框

复选框是一个文字方块，文字通常出现在复选框的右边（如果您在建立按钮时指定了BS_LEFTTEXT样式，那么文字会出现在左边；您也许将用BS_RIGHT直接调整文字来组合此样式）。复选框通常用于允许使用者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：单击框一次将显示勾选标记，再次单击清除勾选标记。

复选框最常用的两种样式是BS_CHECKBOX和BS_AUTOCHECKBOX。在使用BS_CHECKBOX时，您需要自己向该控件发送BM_SETCHECK消息来设定勾选标记。wParam参数设1时设定勾选标记，设0时清除勾选标记。通过向该控件发送BM_GETCHECK消息，您可以得到该复选框的目前状态。在处理来自控件的WM_COMMAND消息时，您可以用如下的指令来翻转X标记：

```
SendMessage ((HWND) lParam, BM_SETCHECK, (WPARAM)
          !SendMessage ((HWND) lParam, BM_GETCHECK, 0, 0), 0) ;
```

注意第二个SendMessage呼叫前面的运算子「!」，其中lParam是在WM_COMMAND消息中传给使用者窗口消息处理程序的子窗口句柄。如果您以后又想知道按钮的状态，那么可以向它发送另一条BM_GETCHECK消息；您也可以将目前状态储存在您的窗口消息处理程序中的一个静态变量里，或者向它发送BM_SETCHECK消息来初始化带勾选标记的BS_CHECKBOX复选框：

SendMessage (hwndButton, BM_SETCHECK, 1, 0) ;

对BS_AUTOCHECKBOX样式，按钮自己触发勾选标记的开和关，所以您的窗口消息处理程序可以忽略WM_COMMAND消息。当您需要按钮目前的状态时，可以向控件发送BM_GETCHECK消息：

iCheck = (int) SendMessage (hwndButton, BM_GETCHECK, 0, 0) ;

如果该按钮被选中，则iCheck的值为TRUE或者非零数；如果按钮末被选中，则iCheck的值为FALSE或0。

其余两种复选框样式是BS_3STATE和BS_AUTO3STATE，正如它们名字所暗示的，这两种样式能显示第三种状态 – 复选框内是灰色 – 它出现在向控件发送wParam等于2的WM_SETCHECK消息时。灰色是向使用者表示此框不能被选本章的或者禁止使用。

复选框沿矩形的左边框对齐，并集中在呼叫CreateWindow时规定的矩形的顶边和底边之间，在该矩形内的任何地方按下鼠标都会向其父窗口发送一个WM_COMMAND消息。复选框的最小高度是一个字符的高度，最小宽度是文字中的字符数加2。

单选按钮

单选按钮的名称在一列按钮的后面，这些按钮就像汽车上的收音机一样。汽车收音机上的每一个按钮都对应一种收音状态，而且一次只能有一个按钮被按下。在对话框中，单选按钮组常常用来表示相互排斥的选项。与复选框不同，单选按钮的工作与开关不一样，也就是说，当第二次按单选按钮时，它的状态会保持不变。

单选按钮的形状是一个圆圈，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经被选中。单选按钮有窗口样式BS_RADIOBUTTON或BS_AUTORADIOBUTTON两种，但是后者只用于对话框。

当您收到来自单选按钮的WM_COMMAND消息时，应该向它发送wParam等于1的BM_SETCHECK消息来显示其选中状态：

SendMessage (hwndButton, BM_SETCHECK, 1, 0) ;

对同组中的其它所有单选按钮，您可以通过向它们发送wParam等于0的BM_SETCHECK消息来显示其未选中状态：

SendMessage (hwndButton, BM_SETCHECK, 0, 0) ;

分组方块

分组方块即样式为BS_GROUPBOX的选择框，它是按钮类中的特例，既不处理鼠标输入和键盘输入，也不向其父窗口发送WM_COMMAND消息。分组方块是一个矩形框，分组方块标题在其顶部显示。分组方块常用来包含其它的按钮控件。

改变按钮文字

您可以通过SetWindowText来改变按钮（或者其它任何窗口）内的文字：

SetWindowText (hwnd, pszString) ;

其中hwnd是欲改变窗口的句柄，pszString是一个指向以null为终结的字符串指针。对于一般的窗口来说，这个文字是标题栏的文字；对于按钮控件来说，它是随着该按钮显示的文字。

您也可以取得窗口目前的文字：

iLength = GetWindowText (hwnd, pszBuffer, iMaxLength) ;

iMaxLength指定复制到pszBuffer指向的缓冲区中的最大字符数。该函数传回复制的字符数。您可以首先通过下面的呼叫来获得特定文字的长度：

iLength = GetWindowTextLength (hwnd) ;

可见的和启用的按钮

为了接收鼠标和键盘输入，子窗口必须是可见的（被显示）和被启用的。当窗口是可见的而未被启用时，那么窗口将以灰色而非黑色显示文字。

如果在建立子窗口时，您没有将WS_VISIBLE包含在窗口类别中，那么直到呼叫ShowWindow时子窗口才会被显示出来：

ShowWindow (hwndChild, SW_SHOWNORMAL) ;

如果您将WS_VISIBLE包含在窗口类别中，就没有必要呼叫ShowWindow。但是，您可以通过呼叫ShowWindow将子窗口隐藏起来：

ShowWindow (hwndChild, SW_HIDE) ;

您可以通过下面的呼叫来确定子窗口是否可见：

IsWindowVisible (hwndChild) ;

您也可以使子窗口被启用或者不被启用。在内定情况下，窗口是被启用的。您可以通过下面的呼叫使窗口不被启用：

EnableWindow (hwndChild, FALSE) ;

对于按钮控件，这具有使按钮字符串变成灰色的作用。按钮将不再对鼠标输入和键盘输入做出响应，这是表示按钮选项目前不可用的最好方法。

您可以通过下面的呼叫使子窗口再次被启用：

EnableWindow (hwndChild, TRUE) ;

您还可以使用下面的呼叫来确定子窗口是否被启用：

IsWindowEnabled (hwndChild) ;

按钮和输入焦点

我在本章前面已经提到过，当用鼠标单击按钮、复选框、单选框和拥有者绘制按钮时，它们接收到输入焦点。这些控件使用文字周围的虚线来表示它拥有了输入焦点。当子窗口控件得到输入焦点时，其父窗口就失去了输入焦点；所有的键盘输入都进入子窗口控件，而不会进入父窗口中。但是，子窗口控件只对Spacebar键作出回应，此时Spacebar键的作用就如同鼠标按键一样。这种情形导致了一个明显的问题：您的程序失去了对键盘处理的控件。让我们看看我们对此能做一些什么。

我在第六章中已经提到过，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（例如一个子窗口控件）时，它首先给正在失去输入焦点的窗口发送一个WM_KILLFOCUS消息，wParam参数是接收输入焦点的窗口的句柄。然后，Windows向正在接收输入焦点的窗口发送一个WM_SETFOCUS消息，同时wParam是还在失去输入焦点的窗口的句柄（在这两种情况中，wParam值可能为NULL，它表示没有窗口拥有或者正在接收输入焦点）。

通过处理WM_KILLFOCUS消息，父窗口可以阻止子窗口控件获得输入焦点。假定数组hwndChild包含了所有子窗口的窗口句柄（它们是在呼叫CreateWindow来建立窗口的时候储存到数组中的）。 NUM是子窗口的数目：

```
case WM_KILLFOCUS :
  for ( i = 0 ; i < NUM ; i++)
    if (hwndChild [i] == (HWND) wParam)
    {
      SetFocus (hwnd) ;
      break ;
    }
    return 0 ;
```

在这段程序代码中，当父窗口获知它正在失去输入焦点，而让它的某个子窗口得到输入焦点时，它将呼叫SetFocus来重新取得输入焦点。

下面是可达到相同目的、但更为简单（但不太直观）的方法：

```
case WM_KILLFOCUS :
  if (hwnd == GetParent ((HWND) wParam))
    SetFocus (hwnd) ;
  return 0 ;
```

但是，这两种方法都有缺点：它们阻止按钮对Spacebar键作出响应，因为该按钮总是得不到输入焦点。一个更好的方法是使按钮得到输入焦点，也能让使用者用Tab键从一个按钮转移到另一

个按钮。这听起来似乎不太可能，在本章的后面，我们将要说明在COLORS1程序中如何用「窗口子类别化」技术来实作这种方法。

控件与颜色

您可以在图9-1中看到，许多按钮的显示看起来并不正确。按键还好，但是其它按钮却带有一个本不应该在那里的一个矩形灰色背景。这是因为这些按钮本来是为对话框中的显示而设计的，而在Windows 98中，对话框有一个灰色的表面。我们的窗口有一个白色的表面，这是因为我们在WNDCLASS结构中就是这样定义的。

wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;

我们已经这么做了，因为我们经常在显示区域中显示文字，而GDI使用在内定设备内容中定义的文字颜色和背景颜色，它们总是黑色和白色。为了使这些按钮更加美观一些，我们必须要改变显示区域的颜色使之和按钮的背景颜色一致，所以要以某种方法将按钮的背景颜色改为白色。

解决此问题的第一步，是理解Windows对「系统颜色」的使用。

系统颜色

Windows保留了29种系统颜色以供各种显示使用。您可以使用GetSysColor和SetSysColors来获得和设定这些颜色。在Windows表头文件中定义的标识符规定了系统颜色。使用SetSysColors设定的系统颜色只在目前Windows对话过程中有效。

借助Windows「控制台」程序的「显示器」部分，您可以改变一些（但不是全部）系统颜色。若是Microsoft Windows NT，选中的颜色会储存在系统登录中；若是Microsoft Windows 98，则储存在WIN.INI文件中。系统登录和WIN.INI文件都为这29种系统颜色使用了关键词(与GetSysColor和SetSysColors的标识符不同)，在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。下表说明了这29种系统颜色是如何在GetSysColor、SetSysColors以及WIN.INI关键词中用常数来标识的。这张表是按照COLOR_ 常数值（从0开始到28结束）顺序排列的:

表9-3

GetSysColor 和 SetSysColors	系统登录键或WIN.INI标识符	内定的RGB值
COLOR_SCROLLBAR	Scrollbar	C0-C0-C0
COLOR_BACKGROUND	Background	00-80-80
COLOR_ACTIVECAPTION	ActiveTitle	00-00-80
COLOR_INACTIVECAPTION	InactiveTitle	80-80-80
COLOR_MENU	Menu	C0-C0-C0
COLOR_WINDOW	Window	FF-FF-FF
COLOR_WINDOWFRAME	WindowFrame	00-00-00
COLOR_MENUTEXT	MenuText	C0-C0-C0
COLOR_WINDOWTEXT	WindowText	00-00-00
COLOR_CAPTIONTEXT	TitleText	FF-FF-FF
COLOR_ACTIVEBORDER	ActiveBorder	C0-C0-C0
COLOR_INACTIVEBORDER	InactiveBorder	C0-C0-C0

COLOR_APPWORKSPACE	AppWorkspace	80-80-80
COLOR_HIGHLIGHT	Highlight	00-00-80
COLOR_HIGHLIGHTTEXT	HighlightText	FF-FF-FF
COLOR_BTNFACE	ButtonFace	C0-C0-C0
COLOR_BTNSHADOW	ButtonShadow	80-80-80
COLOR_GRAYTEXT	GrayText	80-80-80
COLOR_BTNTEXT	ButtonText	00-00-00
COLOR_INACTIVECAPTIONTEXT	InactiveTitleText	C0-C0-C0
COLOR_BTNHIGHLIGHT	ButtonHighlight	FF-FF-FF
COLOR_3DDKSHADOW	ButtonDkShadow	00-00-00
COLOR_3DLIGHT	ButtonLight	C0-C0-C0
COLOR_INFOTEXT	InfoText	00-00-00
COLOR_INFOBK	InfoWindow	FF-FF-FF
[no identifier; use value 25]	ButtonAlternateFace	B8-B4-B8
COLOR_HOTLIGHT	HotTrackingColor	00-00-FF
COLOR_GRADIENTACTIVECAPTION	GradientActiveTitle	00-00-80
COLOR_GRADIENTINACTIVECAPTION	GradientInactiveTitle	80-80-80

这29种颜色的默认值是由显示驱动程序提供的，在不同的机器上可能略有不同。

坏消息：虽然这些颜色中有许多似乎都可以从颜色常数名称上了解其代表意义（例如，COLOR_BACKGROUND是所有窗口后面的桌面区域颜色），在最近版本的Windows中系统颜色的使用变得非常混乱。以前，Windows在视觉上要比今天简单得多。实际上，在Windows 3.0以前，只定义了前13种系统颜色。但随着使用看起来越来越难以控制的立体外观，相对应地也需要更多的系统颜色。

按钮颜色

对需要多种颜色的每一个按钮来说，这个问题更加地明显。COLOR_BTNFACE被用于按键主要的表面颜色，以及其它按钮主要的背景颜色（这也是用于对话框和消息框的系统颜色）。COLOR_BTNSHADOW被建议用作按键右下边、以及复选框内部和单选按钮圆点的阴影。对于按键，COLOR_BTNTEXT被用作文字颜色；而对于其它的按钮，则使用COLOR_WINDOWTEXT作为文字颜色。还有其它几种系统颜色用于按钮设计的各个部分。

因此，如果您想在我们的显示区域表面显示按钮，那么一种避免颜色冲突的方法便是屈服于这些系统颜色。首先，在定义窗口类别时使用COLOR_BTNFACE作为您显示区域的背景颜色：

```
wndclass.hbrBackground = (HBRUSH) (COLOR_BTNFACE + 1) ;
```

您可以在BTNLOOK程序中尝试这种方法。当WNDCLASS结构中的hbrBackground值是这个值时，Windows会明白这实际上指的是一种系统颜色而非一个实际的句柄。Windows要求当您在WNDCLASS结构的hbrBackground栏中指定这些标识符时加上1，这样做的目的是防止其值为NULL，而没有任何其它目的。如果您的在程序执行过程中，系统颜色恰好发生了变化，那么显示区域将变得无效，而Windows将使用新的COLOR_BTNFACE值。但是现在我们又引发了另一个问题。

当您使用TextOut显示文字时，Windows使用的是在设备内容中为背景颜色（它擦除文字后的背景）和文字颜色定义的值，其默认值为白色（背景）和黑色（文字），而不管系统颜色和窗口类别结构中的hbrBackground字段为何值。所以，您需要使用SetTextColor和SetBkColor将文字和文字背景的颜色改变为系统颜色。您可以在获得设备内容句柄之后这么做：

SetBkColor (hdc, GetSysColor (COLOR_BTNFACE)) ;

SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT)) ;

这样，显示区域背景、文字背景和文字的颜色都与按钮的颜色一致了。但是，如果当您的程序执行时，使用者改变了系统颜色，您可能要改变文字背景颜色和文字颜色。这时您可以使用下面的程序代码：

```
case WM_SYSCOLORCHANGE:
  InvalidateRect (hwnd, NULL, TRUE) ;
  break ;
```

WM_CTLCOLORBTN消息

在这边已经看到了如何将显示区域的颜色和文字颜色调节成按钮的背景颜色。我们是否可以将程序中按钮的颜色调节为我们喜欢的颜色呢？理论上没有问题，但在实际中请别这样做。用SetSysColors来改变按钮的外观可能不是您想做的，这会影响目前在Windows下执行的所有程序，这也是使用者不太喜欢的。

更好的方法（同样也只是理论上）是处理WM_CTLCOLORBTN消息，这是当子窗口即将为其显示区域着色时，由按钮控件发送给其父窗口消息处理程序的一个消息。父窗口可以利用这个机会来改变子窗口消息处理程序将用来着色的颜色（在Windows的16位版本中，一个称为WM_CTLCOLOR的消息被用于所有的控件，现在针对每种型态的标准控件，分别代之以不同的消息）。

当父窗口消息处理程序收到WM_CTLCOLORBTN消息时，wParam消息参数是按钮的设备内容句柄，lParam是按钮的窗口句柄。当父窗口消息处理程序得到这个消息时，按钮控件已经获得了它的设备内容。当您的窗口消息处理程序处理一个WM_CTLCOLORBTN消息时，您必须完成以下三个动作：

使用SetTextColor选择设定一种文字颜色。

使用SetBkColor选择设定一种文字背景颜色。

将一个画刷句柄传回给子窗口。

理论上，子窗口使用该画刷来着色背景。当不再需要这个画刷时，您应该负责清除它。

下面是使用WM_CTLCOLORBTN的问题所在：只有按键和拥有者绘制按钮才给其父窗口发送WM_CTLCOLORBTN，而只有拥有者绘制按钮才会响应父窗口消息处理程序对消息的处理，而使用画刷来着色背景。这基本上是没有意义的，因为无论怎样都是由父窗口来负责绘制拥有者绘制按钮。

在本章后面，我们将说明，在某些情况下，一些类似于WM_CTLCOLORBTN但适用于其它型态控件的消息将更为有用。

拥有者绘制按钮

如果您想对按钮的所有可见部分实行全面控制，而不想被键盘和鼠标消息处理所干扰，那么您可以建立BS_OWNERDRAW样式的按钮，如程序9-2所展示的那样。

程序9-2 OWNDRAW

OWNDRAW.C

```
/*---------------------------------------------------------------------
OWNDRAW.C -- Owner-Draw Button Demo Program
(c) Charles Petzold, 1996
---------------------------------------------------------------------*/
#include <windows.h>

#define ID_SMALLER 1
#define ID_LARGER 2
#define BTN_WIDTH ( 8 * cxChar)
#define BTN_HEIGHT ( 4 * cyChar)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HINSTANCE hInst ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("OwnDraw") ;
  MSG msg ;
  HWND hwnd ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Owner-Draw Button Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  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)) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HWND hwndSmaller, hwndLarger ;
  static int cxClient, cyClient, cxChar, cyChar ;
  int cx, cy ;
```

```
LPDRAWITEMSTRUCT pdis ;
POINT pt[3] ;
RECT rc ;

switch (message)
{
case WM_CREATE :
  cxChar = LOWORD (GetDialogBaseUnits ()) ;
  cyChar = HIWORD (GetDialogBaseUnits ()) ;
  // Create the owner-draw pushbuttons

  hwndSmaller = CreateWindow (TEXT ("button"), TEXT (""),
    WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
    0, 0, BTN_WIDTH, BTN_HEIGHT,
    hwnd, (HMENU) ID_SMALLER, hInst, NULL) ;

  hwndLarger = CreateWindow (TEXT ("button"), TEXT (""),
    WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,
    0, 0, BTN_WIDTH, BTN_HEIGHT,
    hwnd, (HMENU) ID_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 ID_SMALLER :
    rc.left += cxClient / 20 ;
    rc.right -= cxClient / 20 ;
    rc.top += cyClient / 20 ;
    rc.bottom -= cyClient / 20 ;
    break ;
  case ID_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 :
    pdis = (LPDRAWITEMSTRUCT) lParam ;
    // Fill area with white and frame it black

    FillRect (pdis->hDC, &pdis->rcItem,
      (HBRUSH) GetStockObject (WHITE_BRUSH)) ;
    FrameRect ( pdis->hDC, &pdis->rcItem,
      ( HBRUSH) GetStockObject (BLACK_BRUSH)) ;
    // Draw inward and outward black triangles
    cx = pdis->rcItem.right - pdis->rcItem.left ;
    cy = pdis->rcItem.bottom - pdis->rcItem.top ;

    switch (pdis->CtlID)
    {
```

```
    case ID_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 (pdis->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 (pdis->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 (pdis->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 (pdis->hDC, pt) ;
      break ;
    case ID_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 (pdis->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 (pdis->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 (pdis->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 (pdis->hDC, pt) ;
      break ;
    }

    // Invert the rectangle if the button is selected

    if (pdis->itemState & ODS_SELECTED)
      InvertRect (pdis->hDC, &pdis->rcItem) ;
    // Draw a focus rectangle if the button has the focus

    if (pdis->itemState & ODS_FOCUS)
    {
      pdis->rcItem.left += cx / 16 ;
      pdis->rcItem.top += cy / 16 ;
      pdis->rcItem.right -= cx / 16 ;
      pdis->rcItem.bottom-= cy / 16 ;

      DrawFocusRect (pdis->hDC, &pdis->rcItem) ;
    }
    return 0 ;
    case WM_DESTROY :
      PostQuitMessage (0) ;
      return 0 ;
```

```
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

该程序在其显示区域的中央包含了两个按钮，如图9-2所示。左边的按钮有四个三角形指向按钮的中央，按下该按钮时，窗口的尺寸将缩小10%。右边的按钮有四个向外指的三角形，按下此按钮时，窗口的尺寸将增大10%。

如果您只需要在按钮中显示图标或位图，您可以用BS_ICON或BS_BITMAP样式，并用BM_SETIMAGE消息设定位图。但是，对于BS_OWNERDRAW样式的按钮，它允许完全自由地绘制按钮。

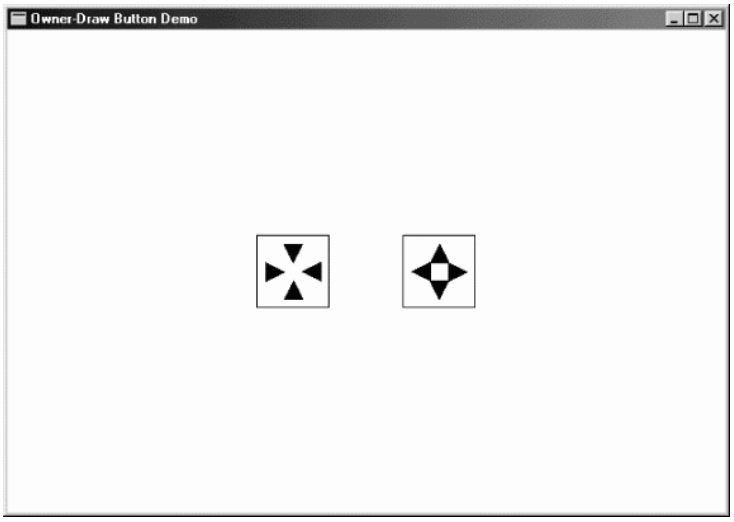


图9-2 OWNDRAW的屏幕显示

在处理WM_CREATE消息处理期间，OWNDRAW建立了两个BS_OWNERDRAW样式的按钮；按钮的宽度是系统字体的8倍，高度是系统字体的4倍（在使用预先定义好的位图绘制按钮时，这些尺寸在VGA上建立的按钮为64像素宽64像素高，知道这些数据将非常有用）。这些按钮尚未就定位，在处理WM_SIZE消息处理期间，通过呼叫MoveWindow函数，OWNDRAW将按钮位置放在显示区域的中心。

按下这些按钮时，它们就会产生WM_COMMAND消息。为了处理这些WM_COMMAND消息，OWNDRAW呼叫GetWindowRect，将整个窗口（不只是显示区域）的位置和尺寸存放在RECT（矩形）结构中，这个位置是相对于屏幕的。然后，根据按下的是左边还是右边的按钮，OWNDRAW调节这个矩形结构的各个字段值。程序再通过呼叫MoveWindow来重新确定位置和尺寸。这将产生另一个WM_SIZE消息，按钮被重新定位在显示区域的中央。

如果这是程序所做的全部处理，那么这完全可以，只不过按钮是不可见的。使用BS_OWNERDRAW样式建立的按钮会在需要重新着色的任何时候都向它的父窗口发送一个WM_DRAWITEM消息。这出现在以下几种情况中：当按钮被建立时，当按钮被按下或被放开时，当按钮得到或者失去输入焦点时，以及当按钮需要重新着色的任何时候。

在处理WM_DRAWITEM消息处理期间，lParam消息参数是指向型态DRAWITEMSTRUCT结构的指针，OWNDRAW程序将这个指针储存在pdis变量中，这个结构包含了画该按钮时程序所必需的消息（这个结构也可以让自绘清单方块和菜单使用）。对按钮而言非常重要的结构字段有hDC（按钮的设备内容）、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，或者按钮是否拥有输入焦点）。

呼叫FillRect用白色画刷抹掉按钮的内面，呼叫FrameRect在按钮的周围画上黑框，由此

OWNDRAW便启动了WM_DRAWITEM处理过程。然后，通过呼叫Polygon，OWNDRAW在按钮上画出4个黑色实心的三角形。这是一般的情形。

如果按钮目前被按下，那么DRAWITEMSTRUCT的itemState字段中的某位将被设为1。您可以使用ODS_SELECTED常数来测试这些位。如果这些位被设立，那么OWNDRAW将通过呼叫InvertRect将按钮翻转为相反的颜色。如果按钮拥有输入焦点，那么itemState的ODS_FOCUS位将被设立。在这种情况下，OWNDRAW通过呼叫DrawFocusRect，在按钮的边界内画一个虚线的矩形。

在使用拥有者绘制按钮时，应该注意以下几个方面：Windows获得设备内容并将其作为DRAWITEMSTRUCT结构的一个字段。保持设备内容处于您找到它时所处的状态，任何被选进设备内容的GDI对象都必需被释放。另外，当心不要在定义按钮边界的矩形外面进行绘制。

静态类别

在CreateWindow函数中指定窗口类别为「static」，您就可以建立静态文字的子窗口控件。这些子窗口非常「文静」。它既不接收鼠标或键盘输入，也不向父窗口发送WM_COMMAND消息。

当您在静态子窗口上移动或者按下鼠标时，这个子窗口将拦截WM_NCHITTEST消息并将HTTRANSPARENT的值传回给Windows，这将使Windows向其下层窗口，通常是它的父窗口，发送相同的WM_NCHITTEST消息。父窗口常常将该消息传递给DefWindowProc，在这里，它被转换为显示区域的鼠标消息。

前六个静态窗口样式只简单地在子窗口的显示区域内画一个矩形或者边框。在下表的上部，「RECT」静态样式（左列）是填入图样的矩形样式；三个「FRAME」样式（右列）是没有填入图样的矩形轮廓：

SS_BLACKRECT	SS_BLACKFRAME
SS_GRAYRECT	SS_GRAYFRAME
SS_WHITERECT	SS_WHITEFRAME

「BLACK」、「GRAY」、「WHITE」并不意味着黑、灰和白色，这些颜色是由系统颜色决定的，如表9-4所示。

表9-4

静态控件	**系统颜色**
BLACK	COLOR_3DDKSHADOW
GRAY	COLOR_BTNSHADOW
WHITE	COLOR_BTNHIGHLIGHT

对这些样式，CreateWindow呼叫中的窗口文字字段被忽略。矩形的左上角开始于x位置坐标和y位置坐标，这些坐标都相对于父窗口。您也可以使用SS_ETCHEDHORZ、SS_ETCHEDVERT或者SS_ETCHEDFRAME ，采用灰色和白色建立一个形似阴影的边框。

静态类别也包括了三种文字样式：SS_LEFT、SS_RIGHT和SS_CENTER。它们建立左对齐、置右对齐和居中文字。文字在CreateWindow呼叫的窗口文字参数中给出，并且在以后可以用SetWindowText来改变它。当静态控件的窗口消息处理程序显示文字时，它使用DrawText函数以及DT_WORDBREAK、DT_NOCLIP和DT_EXPANDTABS参数。文字在子窗口的矩形内可以按文字进

行换行。

这三种文字样式子窗口的背景通常为COLOR_BTNFACE，而文字本身是COLOR_WINDOWTEXT。在拦截WM_CTLCOLORSTATIC消息时，您可以通过呼叫SetTextColor来改变文字颜色，通过SetBkColor来改变背景颜色，并传回背景画刷句柄。后面的COLORS1程序展示了这一点。

最后，静态类别还包括了窗口样式SS_ICON和SS_USERITEM，但是当它们被用作子窗口控件时却没有任何意义。我们在讨论对话框时还要提及它们。

滚动条类别

我在第四章首次讨论了滚动条，也讨论了「窗口滚动条」和「滚动条控件」之间的一些区别。SYSMETS程序使用窗口滚动条，它出现在窗口的右边和底部。您可以在建立窗口时通过将标识符WS_VSCROLL、WS_HSCROLL或者两者都包含在窗口样式中，让窗口加上滚动条。现在我们准备建立一些滚动条控件，它们是能在父窗口的显示区域的任何地方出现的子窗口。您可以使用预先定义的窗口类别「scrollbar」以及两个滚动条样式SBS_VERT和SBS_HORZ中的一个来建立子窗口滚动条控件。

与按钮控件（以及将在后面讨论的编辑和清单方块控件）不同，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。在处理卷动消息时，您可以通过lParam参数来区分窗口滚动条与滚动条控件。对子窗口滚动条其值为0，对于滚动条控件其值为滚动条窗口句柄。对窗口滚动条和滚动条控件来说，wParam参数的高字组和低字组的含义相同。

虽然窗口滚动条有固定的宽度，Windows使用CreateWindow呼叫中（或者在后面的MoveWindow呼叫中）给定的矩形尺寸来确定滚动条控件的尺寸。您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。

如果您想建立与窗口滚动条尺寸相同的滚动条控件，那么可以使用GetSystemMetrics取得水平滚动条的高度：

GetSystemMetrics (SM_CYHSCROLL) ;

或者垂直滚动条的宽度：

GetSystemMetrics (SM_CXVSCROLL) ;

根据Windows文件，滚动条窗样式标识符SBS_LEFTALIGN、SBS_RIGHTALIGN、SBS_TOPALIGN和SBS_BOTTOMALIGN给出滚动条的标准尺寸，但是这些样式只在对话框中对滚动条有效。

对窗口滚动条，您可以使用同样的呼叫来建立滚动条控件的范围和位置：

```
SetScrollRange (hwndScroll, SB_CTL, iMin, iMax, bRedraw) ;
SetScrollPos (hwndScroll, SB_CTL, iPos, bRedraw) ;
SetScrollInfo (hwndScroll, SB_CTL, &si, bRedraw) ;
```

其区别在于：窗口滚动条将父窗口的句柄作为第一个参数，并且以SB_VERT或者SB_HORZ作为第二个参数。

令人吃惊的是，名为COLOR_SCROLLBAR的系统颜色不再用于滚动条。两端的按钮和小方块的颜色由COLOR_BTNFACE、COLOR_BTNHILIGHT、COLOR_BTNSHADOW、COLOR_BTNTEXT（用于小箭头）、COLOR_DKSHADOW和COLOR_BTNLIGHT决定。两端按钮之间区域的颜色由COLOR_BTNFACE和COLOR_BTNHIGHLIGHT决定。

如果您拦截了WM_CTLCOLORSCROLLBAR消息，那么可以在消息处理中传回画刷以取代该颜色。让我们来试一下。

COLORS1程序

为了解滚动条和静态子窗口的一些用法 – 也为了深入了解颜色 – 我们将使用COLORS1程序，如程序9-3所示。COLORS1在显示区域的左半部显示三种滚动条，并分别标以「Red」、「 Green」和「Blue」。当您挪动滚动条时，显示区域的右半部将变为三种原色混合而成的合成色，三种原色的数值显示在三个滚动条的下面。

程序9-3 COLORS1

COLORS1.C

```
/*------------------------------------------------------------------------
COLORS1.C -- Colors Using Scroll Bars
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ScrollProc(HWND, UINT, WPARAM, LPARAM) ;

int idFocus ;
WNDPROC OldScroll[3] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Colors1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = CreateSolidBrush (0) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Color Scroll"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
```

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static COLORREF crPrim[3] = { RGB (255, 0, 0), RGB (0, 255, 0),
    RGB (0, 0, 255) } ;
  static HBRUSH hBrush[3], hBrushStatic ;
  static HWND hwndScroll[3], hwndLabel[3], hwndValue[3], hwndRect ;
  static int color[3], cyChar ;
  static RECT rcColor ;
  static TCHAR *szColorLabel[] = { TEXT ("Red"), TEXT ("Green"),
    TEXT ("Blue") } ;
  HINSTANCE hInstance ;
  int i, cxClient, cyClient ;
  TCHAR szBuffer[10] ;

  switch (message)
  {
  case WM_CREATE :
    hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;
    // Create the white-rectangle window against which the
    // scroll bars will be positioned. The child window ID is 9.

    hwndRect = CreateWindow (TEXT ("static"), NULL,
      WS_CHILD | WS_VISIBLE | SS_WHITERECT,
      0, 0, 0, 0,
      hwnd, (HMENU) 9, hInstance, NULL) ;
    for (i = 0 ; i < 3 ; i++)
    {
      // The three scroll bars have IDs 0, 1, and 2, with
      // scroll bar ranges from 0 through 255.
      hwndScroll[i] = CreateWindow (TEXT ("scrollbar"), NULL,
        WS_CHILD | WS_VISIBLE |
        WS_TABSTOP | SBS_VERT,
        0, 0, 0, 0,
        hwnd, (HMENU) i, hInstance, NULL) ;

      SetScrollRange (hwndScroll[i], SB_CTL, 0, 255, FALSE) ;
      SetScrollPos (hwndScroll[i], SB_CTL, 0, FALSE) ;
      // The three color-name labels have IDs 3, 4, and 5,
      // and text strings "Red", "Green", and "Blue".

      hwndLabel [i] = CreateWindow (TEXT ("static"), szColorLabel[i],
        WS_CHILD | WS_VISIBLE | SS_CENTER,
        0, 0, 0, 0,
        hwnd, (HMENU) (i + 3),
        hInstance, NULL) ;
      // The three color-value text fields have IDs 6, 7,
      // and 8, and initial text strings of "0".
      hwndValue [i] = CreateWindow (TEXT ("static"), TEXT ("0"),
        WS_CHILD | WS_VISIBLE | SS_CENTER,
        0, 0, 0, 0,
        hwnd, (HMENU) (i + 6),
        hInstance, NULL) ;

      OldScroll[i] = (WNDPROC) SetWindowLong (hwndScroll[i],
        GWL_WNDPROC, (LONG) ScrollProc) ;
      hBrush[i] = CreateSolidBrush (crPrim[i]) ;
    }

    hBrushStatic = CreateSolidBrush (
      GetSysColor (COLOR_BTNHIGHLIGHT)) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    return 0 ;
  case WM_SIZE :
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    SetRect (&rcColor, cxClient / 2, 0, cxClient, cyClient) ;
    MoveWindow (hwndRect, 0, 0, cxClient / 2, cyClient, TRUE) ;

    for (i = 0 ; i < 3 ; i++)
```

```
    {
      MoveWindow (hwndScroll[i],
        (2 * i + 1) * cxClient / 14, 2 * cyChar,
        cxClient / 14, cyClient - 4 * cyChar, TRUE) ;

      MoveWindow (hwndLabel[i],
        (4 * i + 1) * cxClient / 28, cyChar / 2,
        cxClient / 7, cyChar, TRUE);

      MoveWindow (hwndValue[i],
        (4 * i + 1) * cxClient / 28,
        cyClient - 3 * cyChar / 2,
        cxClient / 7, cyChar, TRUE) ;
    }
    SetFocus (hwnd) ;
    return 0 ;
  case WM_SETFOCUS :
    SetFocus (hwndScroll[idFocus]) ;
    return 0 ;
  case WM_VSCROLL :
    i = GetWindowLong ((HWND) lParam, GWL_ID) ;
    switch (LOWORD (wParam))
    {
    case SB_PAGEDOWN :
      color[i] += 15 ;
      // fall through
    case SB_LINEDOWN :
      color[i] = min (255, color[i] + 1) ;
      break ;
    case SB_PAGEUP :
      color[i] -= 15 ;
      // fall through
    case SB_LINEUP :
      color[i] = max (0, color[i] - 1) ;
      break ;
    case SB_TOP :
      color[i] = 0 ;
      break ;
    case SB_BOTTOM :
      color[i] = 255 ;
      break ;
    case SB_THUMBPOSITION :
    case SB_THUMBTRACK :
      color[i] = HIWORD (wParam) ;
      break ;
    default :
      break ;
    }
    SetScrollPos (hwndScroll[i], SB_CTL, color[i], TRUE) ;
    wsprintf (szBuffer, TEXT ("%i"), color[i]) ;
    SetWindowText (hwndValue[i], szBuffer) ;

    DeleteObject ((HBRUSH)
      SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
      CreateSolidBrush (RGB (color[0], color[1], color[2])))) ;

    InvalidateRect (hwnd, &rcColor, TRUE) ;
    return 0 ;
    case WM_CTLCOLORSCROLLBAR :
      i = GetWindowLong ((HWND) lParam, GWL_ID) ;
      return (LRESULT) hBrush[i] ;
    case WM_CTLCOLORSTATIC :
      i = GetWindowLong ((HWND) lParam, GWL_ID) ;

      if (i >= 3 && i <= 8) // static text controls
      {
        SetTextColor ((HDC) wParam, crPrim[i % 3]) ;
        SetBkColor ((HDC) wParam, GetSysColor (COLOR_BTNHIGHLIGHT));
        return (LRESULT) hBrushStatic ;
```

```
     }
     break ;
   case WM_SYSCOLORCHANGE :
     DeleteObject (hBrushStatic) ;
     hBrushStatic = CreateSolidBrush (GetSysColor(COLOR_BTNHIGHLIGHT)) ;
     return 0 ;
   case WM_DESTROY :
     DeleteObject ((HBRUSH)
       SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
       GetStockObject (WHITE_BRUSH))) ;

     for (i = 0 ; i < 3 ; i++)
       DeleteObject (hBrush[i]) ;

     DeleteObject (hBrushStatic) ;
     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ScrollProc (HWND hwnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
  int id = GetWindowLong (hwnd, GWL_ID) ;
  switch (message)
  {
  case WM_KEYDOWN :
    if (wParam == VK_TAB)
      SetFocus (GetDlgItem (GetParent (hwnd),
      (id + (GetKeyState (VK_SHIFT) < 0 ? 2 : 1)) % 3)) ;
    break ;
  case WM_SETFOCUS :
    idFocus = id ;
    break ;
  }
  return CallWindowProc (OldScroll[id], hwnd, message, wParam,lParam) ;
}
```

COLORS1利用子窗口进行工作，该程序使用10个子窗口控件：3个滚动条、6个静态文字窗口和1个静态矩形框。COLORS1拦截WM_CTLCOLORSCROLLBAR消息来给红、绿、蓝3个滚动条的内部着色，并拦截WM_CTLCOLORSTATIC消息来着色静态文字。

您可以使用鼠标或者键盘来挪动滚动条，从而利用COLORS1作为一种实验颜色显示的开发工具，为您自己的Windows程序选择漂亮的颜色（或者，您可能更喜欢难看的颜色）。COLORS1的显示如图9-3所示。不幸的是，这些颜色在印表纸上被显示为不同深浅的灰色。

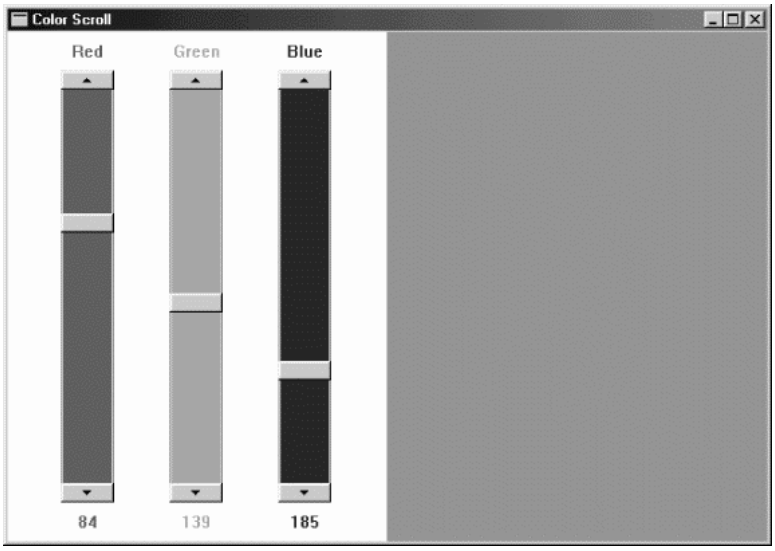


图9-3 COLORS1的屏幕显示

COLORS1不处理WM_PAINT消息，所有的工作几乎都是由子窗口完成的。

显示区域右半部显示的颜色实际上是窗口的背景颜色。SS_WHITERECT样式的静态子窗口显示在显示区域的左半部。三个滚动条是SBS_VERT样式的子窗口控件，它们被定位在SS_WHITERECT子窗口的顶部。另外六个SS_CENTER样式（居中文字）的静态子窗口提供卷标和颜色值。COLORS1在WinMain函数中用CreateWindow建立它的普通重迭式窗口和10个子窗口。SS_WHITERECT和SS_CENTER静态窗口使用窗口类别「static」；三个滚动条使用窗口类别「scrollbar」。

CreateWindow呼叫中的x位置、y位置、宽度和高度参数最初设为0，因为位置和大小都取决于显示区域的尺寸，而它目前尚未确定。COLORS1的窗口消息处理程序在接收到WM_SIZE消息时，就使用MoveWindow给10个子窗口重新确定大小。所以，每当您对COLORS1窗口进行缩放时，滚动条的尺寸就会按比例变化。

当WndProc窗口消息处理程序收到WM_VSCROLL消息时，lParam参数的高字组就是子窗口的句柄。我们可以使用GetWindowWord来得到子窗口的ID：

```
i = GetWindowLong ((HWND) lParam, GWL_ID) ;
```

对于这三个滚动条，我们已经按习惯将其ID设为0、1、2，所以WndProc能区别出是哪个滚动条在产生消息。

由于子窗口的句柄在建立时就被储存在数组中，所以WndProc就能对相对应的滚动条消息进行处理，并通过呼叫SetScrollPos来设定相对应的新值：

```
SetScrollPos (hwndScroll[i], SB_CTL, color[i], TRUE) ;
```

WndProc也改变滚动条底部子窗口的文字：

```
wsprintf (szBuffer, TEXT ("%i"), color[I]) ;

SetWindowText (hwndValue[i], szBuffer) ;
```

自动键盘接口

滚动条控件也能处理键盘输入，但是只有在拥有输入焦点时才行。下表说明怎样将键盘光标键转变为卷动消息：

表9-5

光标键	卷动消息的wParam值
Home	SB_TOP
End	SB_BOTTOM
Page Up	SB_PAGEUP
Page Down	SB_PAGEDOWN
左或上	SB_LINEUP
右或下	SB_LINEDOWN

事实上，SB_TOP和SB_BOTTOM卷动消息只能用键盘产生。在使用鼠标按动卷动列时，如果想使该卷动列获得输入焦点，那么您必须将WS_TABSTOP标识符包含到CreateWindow呼叫的窗口类别参数中。当滚动条拥有输入焦点时，在该滚动条的小方框上将显示一个闪烁的灰色块。

为了给滚动条提供全面的键盘接口，还需要另外一些工作。首先，WndProc窗口消息处理程

序必须使滚动条拥有输入焦点，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。WndProc给其中一个滚动条设定输入焦点。

SetFocus (hwndScroll[idFocus]) ;

其中idFocus是一个整体变量。

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，因为一旦某个滚动条拥有了输入焦点，它就处理所有的键盘输入，但滚动条只关心光标键，而忽略Tab键。解决这一两难处境的方法是「窗口子类别化」。我们将用它来给COLORS1增加使用Tab键从一个滚动条跳到另一个滚动条的功能。

窗口子类别化（Window Subclassing）

滚动条控件的窗口消息处理程序是Windows内部的。但是，将GWL_WNDPROC标识符作为参数来呼叫GetWindowLong，您就可以得到这个窗口消息处理程序的地址。另外，您可以呼叫SetWindowLong给该滚动条设定一个新的窗口消息处理程序，这个技术叫做「窗口子类别化」，非常有用。它能让您给现存的窗口消息处理程序设定「挂勾」，以便在自己的程序中处理一些消息，同时将其它所有消息传递给旧的窗口消息处理程序。

在COLORS1中对卷动消息进行初步处理的窗口消息处理程序叫做ScrollProc，它在COLORS1.C文件的尾部。由于ScrollProc是COLORS1中的函数，而Windows将呼叫COLORS1，所以ScrollProc必须被定义为callback函数。

对三个滚动条中的每一个，COLORS1使用SetWindowLong来设定新的滚动条窗口消息处理程序的地址，并取得现存滚动条窗口消息处理程序的地址：

OldScroll[i] = (WNDPROC) SetWindowLong (hwndScroll[i], GWL_WNDPROC,

(LONG) ScrollProc)) ;

现在，函数ScrollProc得到了Windows发送到COLORS1中三个滚动条（当然不是其它程序中的滚动条）的滚动条窗口消息处理程序的全部消息。ScrollProc窗口消息处理程序在接收到Tab或者Shift-Tab键时，就将输入焦点改变到下一个（或者上一个）滚动条。它使用CallWindowProc呼叫旧的滚动条窗口消息处理程序。

给背景着色

当COLORS1定义它的窗口类别时，也为其显示区域背景定义了一个实心的黑色画刷：

wndclass.hbrBackground = CreateSolidBrush (0) ;

当您改变COLORS1的滚动条设定时，程序必须建立一个新的画刷，并将该新画刷句柄放入窗口类别结构中。如同使用GetWindowLong和SetWindowLong能得到并设定滚动条窗口消息处理程序一样，用GetClassWord和SetClassWord能得到这个画刷的句柄。

您可以建立新的画刷并将其句柄插入窗口类别结构中，然后删除旧的画刷：

```
DeleteObject ((HBRUSH)
           SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
           CreateSolidBrush (RGB (color[0], color[1], color[2])))) ;
```

Windows下一次重新为窗口的背景着色时，将使用这个新画刷。为了强迫Windows抹掉背景，我们将使整个显示区域无效：

InvalidateRect (hwnd, &rcColor, 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在窗口消息处理程序的消息队列中放进一个WM_PAINT消息。由于WM_PAINT消息的优先等级比较低，所以，如果您还在使用鼠标或者光标键移动滚动条的话，这个消息将不会立即被处理。如果您想在颜色改变之后使该窗口立即变成最新的（目前的），那么您可以在InvalidateRect之后增加下面的叙述：

UpdateWindow (hwnd) ;

但这会使得键盘和鼠标处理变慢。

COLORS1中的WndProc函数不处理WM_PAINT消息，而是将其传给DefWindowProc。Windows对WM_PAINT消息的内定处理只是呼叫BeginPaint和EndPaint使窗口生效。因为在InvalidateRect呼叫中已经指定背景要被抹掉，所以BeginPaint呼叫使Windows发出一个WM_ERASEBKGND（删除背景）消息，WndProc也将忽略这个消息。Windows用窗口类别中指定的画刷将显示区域的背景抹去，这样就处理了这个消息。

在终止以前进行清除总是一个好主意，因此在处理WM_DESTROY消息处理期间，再一次呼叫DeleteObject：

```
DeleteObject ((HBRUSH)
           SetClassLong (hwnd, GCL_HBRBACKGROUND,
           (LONG) GetStockObject (WHITE_BRUSH))) ;
```

给滚动条和静态文字着色

在COLORS1中，三个滚动条的内部和六个文字字段中的文字着色为红、绿和蓝色。滚动条的着色是通过处理WM_CTLCOLORSCROLLBAR消息来完成的。

在WndProc中，我们为画刷定义了一个由三个句柄组成的静态数组：

static HBRUSH hBrush [3] ;

在处理WM_CREATE期间，我们建立三个画刷：

for (I = 0 ; I < 3 ; I++)

hBrush[0] = CreateSolidBrush (crPrim [I]) ;

其中crPrim数组中包含三种原色的RGB值。在WM_CTLCOLORSCROLLBAR处理期间窗口消息处理程序传回这三画刷中的一个：

```
case WM_CTLCOLORSCROLLBAR:
  i = GetWindowLong ((HWND) lParam, GWL_ID) ;
  return (LRESULT) hBrush [i] ;
```

在处理WM_DESTROY消息的过程中，这些画刷必须被删除：

for (i = 0 ; i < 3 ; i++)

DeleteObject (hBrush [i])) ;

同样地，静态文字字段中的文字是在处理WM_CTLCOLORSTATIC消息中呼叫SetTextColor来着色的。文字背景用SetBkColor函数设定为系统颜色COLOR_BTNHIGHLIGHT，这导致文字背景颜色和滚动条与文字后面的静态矩形控件的颜色一样。对于静态文字控件，这种文字背景颜色只用于字符串中每个字符后面的矩形，而不会用于整个控件窗口。为了实作这一点，窗口消息处理程序还必须传回COLOR_BTNHIGHLIGHT颜色画刷的句柄。这个画刷被称为hBrushStatic，它在WM_CREATE消息处理期间建立，在WM_DESTROY消息处理期间清除。

在WM_CREATE消息处理期间依据COLOR_BTNHIGHLIGHT颜色建立画刷，并且在执行期间使

用这一画刷时，我们遇到了一个小问题。如果程序在执行期间改变了COLOR_BTNHIGHLIGHT颜色，那么静态矩形的颜色将发生变化，并且文字背景的颜色也会变化，但是文字窗口控件的整个背景将保持原有的COLOR_BTNHIGHLIGHT颜色。

为了解决这个问题，COLORS1也简单地通过使用新颜色重新建立hBrushStatic来处理WM_SYSCOLORCHANGE消息。

编辑类别

在某些方面，编辑类别是最简单的预先定义窗口类别；在另一方面，它又是最复杂的窗口类别。当您使用类别名称「edit」建立子窗口时，您根据CreateWindow呼叫中的x位置、y位置、宽度和高度这些参数定义了一个矩形。此矩形含有可编辑文字。当子窗口控件拥有输入焦点时，您可以输入文字，移动光标，使用鼠标或者Shift键与一个光标键来选取部分文字，按Ctrl-X来删除所选文字或按Ctrl-C来复制所选文字、并送到剪贴簿上，按Ctrl-V键插入剪贴簿上的文字。

编辑控件的最简单的应用之一是作为单行输入区域。但是编辑控件并不仅限于单行，这一点我将在程序9-4 POPPAD1中说明。和我们在这本书中所遇到的各种其它问题一样，POPPAD程序将逐步增强以使用菜单、对话框(加载与储存文件)和打印。最后的版本将是一个简单而完整的文字编辑器，且其程序代码将非常简洁。

程序9-4 POPPAD1

POPPAD1.C

```
/*---------------------------------------------------------------------------
POPPAD1.C -- Popup Editor using child window edit box
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#define ID_EDIT 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

TCHAR szAppName[] = TEXT ("PopPad1") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
```

```
  hwnd = CreateWindow (szAppName, szAppName,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HWND hwndEdit ;
  switch (message)
  {
  case WM_CREATE :
    hwndEdit = CreateWindow (TEXT ("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, (HMENU) ID_EDIT,
      ((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 (LOWORD (wParam) == ID_EDIT)
      if (HIWORD (wParam) == EN_ERRSPACE ||
        HIWORD (wParam) == EN_MAXTEXT)
        MessageBox (hwnd, TEXT ("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是一个多行编辑器（只是没有文件I/O），其C语言原始码不到100行（不过，有一个缺陷，即预先定义的多行编辑控件只限于30,000字符的文字）。您可以看到，POPPAD1本身并没有做多少工作，预先定义的编辑控件完成了许多工作，这样，您可以知道，无需额外的程序时编辑控件能做些什么。

编辑类别样式

如前面所提到的，在CreateWindow呼叫中将「edit」作为窗口类别建立了一个编辑控件，窗口样式是WS_CHILD加上几个选项。如同在静态子窗口控件中一样，编辑控件中的文字可以置左对齐、置右对齐或者居中，您使用窗口样式ES_LEFT、ES_RIGHT和ES_CENTER来指定这些格式。

内定状态下，编辑控件是单行的。您使用ES_MULTILINE窗口样式可以建立多行编辑控件。对于单行编辑控件，您一般只可以在编辑控件矩形的尾部输入文字。要建立一个自动水平卷动的编辑控件，您可以采用样式ES_AUTOHSCROLL。对一个多行编辑控件，文字会自动跳行，除非使用ES_AUTOHSCROLL样式。在这种情况下，您必须按Enter键来开始新的一行。您还可以便用样式

ES_AUTOVSCROLL来将垂直滚动条包括在多行编辑控件中。

当您在多行编辑控件中包括这些卷动样式时，也许还想给编辑控件增加卷动列。要做到这些，可以对非子窗口使用同一窗口样式标识符WS_HSCROLL和WS_VSCROLL。内定状态下，编辑控件没有边界，利用样式WS_BORDER则可以增加边界。

当您在编辑控件中选择文字时，Windows将选择的文字反白显示。但是当编辑控件失去输入焦点时，被选择的文字将不再被加亮。如果希望在编辑控件没有输入焦点时被选择的文字仍然被加亮，您可以使用样式ES_NOHIDESEL。

在POPPAD1建立其编辑控件时，CreateWindow呼叫依如下形式给出样式：

```
WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
        WS_BORDER | ES_LEFT | ES_MULTILINE |
    ES_AUTOHSCROLL | ES_AUTOVSCROLL
```

在POPPAD1中，编辑控件的大小是后来当WndProc接收到WM_SIZE消息时通过呼叫MoveWindow来定义的。编辑控件的尺寸被简单地设定为主窗口的尺寸：

```
MoveWindow (hwndEdit, 0, 0, LOWORD (lParam),
                            HIWORD (lParam), TRUE) ;
```

对于单行编辑控件，控件的高度必须可以容纳一个字符。如果编辑控件有边界（大多数都有），那么使用一个字符高度的1.5倍（包括外部间距）。

编辑控件通知

编辑控件给父窗口消息处理程序发送WM_COMMAND消息，对按钮控件来说，wParam和lParam变量的含义是相同的：

LOWORD (wParam)	子窗口ID
HIWORD (wParam)	通知码
lParam	子窗口句柄

通知码如下所示：

EN_SETFOCUS	编辑控件已经获得输入焦点
EN_KILLFOCUS	编辑控件已经失去输入焦点
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	编辑控件执行已经超出中间
EN_MAXTEXT	编辑控件在插入时执行超出空间
EN_HSCROLL	编辑控件的水平滚动条已经被按下
EN_VSCROLL	编辑控件的垂直滚动条已经被按下

POPPAD1只拦截EN_ERRSPACE和EN_MAXTEXT通知码，并显示一个消息框。

使用编辑控件

如果在您的主窗口上使用了几个单行编辑控件，那么您需要将窗口子类别化以便把输入焦点从一个控件转移到另一个控件。您可以通过拦截Tab键和Shift-Tab键来完成这种移动，非常像

COLORS1中所做的（窗口子类别化的另一个例子在后面的HEAD程序中说明）。如何处理Enter键取决于您，可以像Tab键那样使用，也可以当成给程序的信号，表示所有的编辑字段都准备好了。

如果您想在编辑区中插入文字，那么可以使用SetWindowText来做到。将文字从编辑控件中取出涉及了GetWindowTextLength和GetWindowText，我们将在POPPAD程序的修订版本中看到这些操作的实例。

发送给编辑控件的消息

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POPPAD修订版本中用到，所以这里不解说所有用SendMessage发送给编辑控件的消息，只概要地说明一下。

这些消息允许您剪下、复制或者清除目前被选择的文字。使用者使用鼠标或者Shift键加上光标控件键来选择文字并进行上面的操作，这样，在编辑控件中选中的文字将被加亮：

```
SendMessage (hwndEdit, WM_CUT, 0, 0) ;
SendMessage (hwndEdit, WM_COPY, 0, 0) ;
SendMessage (hwndEdit, WM_CLEAR, 0, 0) ;
```

WM_CUT将目前选择的文字从编辑控件中移走，并将其发送到剪贴簿中；WM_COPY将选择的文字复制到剪贴簿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，但是不向剪贴簿中发送。

您也可以将剪贴簿上的文字插入到编辑控件中的光标位置：

```
SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
```

您可以取得目前选择的起始位置和末尾位置：

```
SendMessage (hwndEdit, EM_GETSEL, (WPARAM) &iStart,
                                  (LPARAM) &iEnd) ;
```

结束位置实际上是最后一个选择字符的位置加1。

您可以选择文字：

```
SendMessage (hwndEdit, EM_SETSEL, iStart, iEnd) ;
```

您还可以使用别的文字来置换目前的选择内容：

```
SendMessage (hwndEdit, EM_REPLACESEL, 0, (LPARAM) szString) ;
```

对多行编辑控件，您可以取得行数：

```
iCount = SendMessage (hwndEdit, EM_GETLINECOUNT, 0, 0) ;
```

对任何特定的行，您可以取得距离编辑缓冲区文字开头的偏移量：

```
iOffset = SendMessage (hwndEdit, EM_LINEINDEX, iLine, 0) ;
```

行数从0开始计算，iLine值为-1时传回包含光标所在行的偏移量。您可以取得行的长度：

```
iLength = SendMessage (hwndEdit, EM_LINELENGTH, iLine, 0) ;
```

并将行本身复制到一个缓冲区中：

```
iLength = SendMessage (hwndEdit, EM_GETLINE, iLine, (LPARAM) szBuffer) ;
```

清单方块类别

我在本章讨论的最后一个预先定义子窗口控件是清单方块。一个清单方块是字符串的集合，这些字符串是一个矩形中可以卷动显示的清单。－程序通过向清单方块窗口消息处理程序发送消息，可以在清单中增加或者删除字符串。当清单方块中的某项被选择时，清单方块控件就向其父窗口发送WM_COMMAND消息，父窗口也就可以确定选择的是哪一项。

一个清单方块可以是单选的，也可以是多选的，后者允许使用者从清单方块中选择多个项目。当清单方块拥有输入焦点时，其中项目的周围显示有虚线。在清单方块中，光标位置并不指明被选择的项目。被选择的项目被加亮显示，并且是反白显示的。

在单项选择的清单方块中，使用者按Spacebar键就可以选择光标所在位置的项目。方向键移动光标和目前选择指示，并且能够滚动清单方块的内容。Page Up和Page Down键也能滚动清单方块，但它移动的是光标而不是选择指示。按字母键能将光标和选择指示移到以此字母开头的第一个（或下一个）选项。也可以使用鼠标在要选择的项目上单击或者双击来选择它。

在多项选择清单方块中，Spacebar键可以切换光标所在位置的项目的选择状态（如果该项已经被选择，则取消选择）。如同在单项选择清单方块中一样，方向键取消前面选择过的项目，并且移动光标和选择指示。但是，Ctrl键和方向键能够在移动光标的同时不移动选择，Shift键加方向键能扩展一个选择。

在多项选择清单方块中，单击或者双击鼠标按键能取消之前所有的选择，而选择被点中的项目。但是，如果在鼠标点中某一项的同时也按下Shift键，则只能切换该项的选择状态，而不会改变任何其它项的选择状态。

清单方块样式

当您使用CreateWindow建立清单方块子窗口时，您应该将「listbox」作为窗口类别，将WS_CHILD作为窗口样式。但是，这个内定清单方块样式不向其父窗口发送WM_COMMAND消息，这样一来，程序必须向清单方块询问其中的项目的选择状态（借助于发送给清单方块控件的消息）。所以，清单方块控件通常都包括清单方块样式标识符LBS_NOTIFY，它允许父窗口接收来自清单方块的WM_COMMAND消息。如果您希望清单方块控件对清单方块中的项目进行排序，那么您可以使用另一种常用的样式LBS_SORT。

内定情况下，清单方块是单项选择的。多项选择的清单方块相当少。如果您想建立一个多项选择清单方块，那么您可以使用样式LBS_MULTIPLESEL。通常，当给有滚动条的清单方块增加新项目时，清单方块本身会自己重画。您可以通过将样式LBS_NOREDRAW包含进去来防止这种现象。但是您也许不想使用这种样式，这时可以使用WM_SETREDRAW消息来暂时防止清单方块控件重新画过，我将在稍后讨论WM_SETREDRAW消息。

内定状态下，清单方块窗口消息处理程序只显示列表项目，它的周围没有任何边界。您可以使用窗口样式标识符WS_BORDER来加上边界。另外，您可以使用窗口样式标识符WS_VSCROLL来增加垂直滚动条，以便用鼠标来滚动条表项目。

Windows表头文件定义了一个清单方块样式，叫做LBS_STANDARD，它包含了最常用的样式，其定义如下：

(LBS_NOTIFY | LBS_SORT | WS_VSCROLL | WS_BORDER)

您也可以采用WS_SIZEBOX和WS_CAPTION标识符，但是这两个标识符允许您重新定义清单方块的大小，也允许您在清单方块父窗口的显示区域中移动清单方块。

清单方块的宽度应该能够容纳最长字符串的宽度加上滚动条的宽度。您可以使用：

GetSystemMetrics (SM_CXVSCROLL) ;

来获得垂直滚动条的宽度。您用一个字符的高度乘以想要在视端口中显示的项目数来计算出清单方块的高度。

将字符串放入清单方块

建立清单方块之后，下一步是将字符串放入其中，您可以通过呼叫SendMessage为清单方块窗口消息处理程序发送消息来做到这一点。字符串通常通过以0开始计数的索引数来引用，其中0对应于最顶上的项目。在下面的例子中，hwndList是子窗口清单方块控件的句柄，而iIndex是索引值。在使用SendMessage传递字符串的情况下，lParam参数是指向以null字符结尾字符串的指针。

在大多数例子中，当窗口消息处理程序储存的清单方块内容超过了可用内存空间时，SendMessage将传回LB_ERRSPACE（定义为-2）。如果是因为其它原因而出错，那么SendMessage将传回LB_ERR（-1）。如果操作成功，那么SendMessage将传回LB_OKAY（0）。您可以通过测试SendMessage的非零值来判断这两种错误。

如果您采用LBS_SORT样式（或者如果您在清单方块中按照想要呈现的顺序排列字符串），那么填入清单方块最简单的方法是借助LB_ADDSTRING消息：

SendMessage (hwndList, LB_ADDSTRING, 0, (LPARAM) szString) ;

如果您没有采用LBS_SORT，那么可以使用LB_INSERTSTRING指定一个索引值，将字符串插入到清单方块中：

SendMessage (hwndList, LB_INSERTSTRING, iIndex, (LPARAM) szString) ;

例如，如果iIndex等于4，那么szString将变为索引值为4的字符串 – 从顶头开始算起的第5个字符串（因为是从0开始计数的），位于这个点后面的所有字符串都将向后推移。索引值为-1时，将字符串增加在最后。您可以对样式为LBS_SORT的清单方块使用LB_INSERTSTRING，但是这个清单方块的内容不能被重新排序（您也可以使用LB_DIR消息将字符串插入到清单方块中，这将在本章的最后进行讨论）。

您可以在指定索引值的同时使用LB_DELETESTRING参数，这就可以从清单方块中删除字符串：

SendMessage (hwndList, LB_DELETESTRING, iIndex, 0) ;

您可以使用LB_RESETCONTENT清除清单方块中的内容：

SendMessage (hwndList, LB_RESETCONTENT, 0, 0) ;

当在清单方块中增加或者删除字符串时，清单方块窗口消息处理程序将更新显示。如果您有许多字符串需要增加或者删除，那么您也许希望暂时阻止这一动作，其方法是关掉控件的重画旗标：

SendMessage (hwndList, WM_SETREDRAW, FALSE, 0) ;

当您完成后，可以再打开重画旗标：

SendMessage (hwndList, WM_SETREDRAW, TRUE, 0) ;

使用LBS_NOREDRAW样式建立的清单方块开始时其重画旗标是关闭的。

选择和取得项

SendMessage完成了下面所描述的任务之后，通常传回一个值。如果出错，那么这个值将被设定为LB_ERR（定义为-1）。

当清单方块中放入一些项目之后，您可以弄清楚清单方块中有多少项目：

iCount = SendMessage (hwndList, LB_GETCOUNT, 0, 0) ;

其它一些呼叫对单项选择清单方块和多项选择清单方块是不同的。让我们先来看看单项选择清单方块。

通常，您让使用者在清单方块中选择条目。但是如果您想加亮显示一个内定选择，则可以使用：

SendMessage (hwndList, LB_SETCURSEL, iIndex, 0) ;

将iParam设定为-1则取消所有选择。

您也可以根据项目的第一个字母来选择：

iIndex = SendMessage (hwndList, LB_SELECTSTRING, iIndex,

(LPARAM) szSearchString) ;

在SendMessage呼叫中将iIndex作为iParam参数时，iIndex是索引，可以根据它搜索其开头字符与szSearchString相匹配的项目。iIndex的值等于-1时从头开始搜索，SendMessage传回被选中项目的索引。如果没有开头字符与szSearchString相匹配的项目时，SendMessage传回LB_ERR。

当您得到来自清单方块的WM_COMMAND消息时（或者在任何其它时候），您可以使用LB_GETCURSEL来确定目前选项的索引：

iIndex = SendMessage (hwndList, LB_GETCURSEL, 0, 0) ;

如果没有项目被选中，那么从呼叫中传回的iIndex值为LB_ERR。

您可以确定清单方块中字符串的长度：

iLength = SendMessage (hwndList, LB_GETTEXTLEN, iIndex, 0) ;

并可以将某项目复制到文字缓冲区中：

```
iLength = SendMessage ( hwndList, LB_GETTEXT, iIndex,
                  (LPARAM) szBuffer) ;
```

在这两种情况下，从呼叫传回的iLength值是字符串的长度。对以NULL字符终结的字符串长度来说，szBuffer数组必须够大。您也许想用LB_GETTEXTLEN先分配一些局部内存来存放字符串。

对于一个多项选择清单方块，您不能使用LB_SETCURSEL、LB_GETCURSEL或者LB_SELECTSTRING，但是您可以使用LB_SETSEL来设定某特定项目的选择状态，而不影响有可能被选择的其它项：

SendMessage (hwndList, LB_SETSEL, wParam, iIndex) ;

wParam参数不为0时，选择并加亮某一项目；wParam为0时，取消选择。如果wParam等于-1，那么将选择所有项目或者取消所有被选中的项目。您可以如下确定某特定项目的选择状态：

iSelect = SendMessage (hwndList, LB_GETSEL, iIndex, 0) ;

其中，如果由iIndex指定的项目被选中，iSelect被设为非0，否则被设为0。

接收来自清单方块的消息

当使用者用鼠标单击清单方块时，清单方块将接收输入焦点。下面的操作可以使父窗口将输入焦点转交给清单方块控件：

SetFocus (hwndList) ;

当清单方块拥有输入焦点时，光标移动键、字母键和Spacebar键都可以用来在该清单方块中

选择某项。

清单方块控件向其父窗口发送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：

LOWORD (wParam)	子窗口ID
HIWORD (wParam)	通知码
lParam	子窗口句柄

通知码及其值如下所示：

LBN_ERRSPACE	-2
LBN_SELCHANGE	1
LBN_DBLCLK	2
LBN_SELCANCEL	3
LBN_SETFOCUS	4
LBN_KILLFOCUS	5

只有清单方块窗口样式包括LBS_NOTIFY时，清单方块控件才会向父窗口发送LBN_SELCHANGE和LBN_DBLCLK。

LBN_ERRSPACE表示清单方块已经超出执行空间。LBN_SELCHANGE表示目前选择已经被改变。这些消息出现在下列情况下：使用者在清单方块中移动加亮的项目时，使用者使用Spacebar键切换选择状态或者使用鼠标单击某项时。LBN_DBLCLK说明某项目已经被鼠标双击（LBN_SELCHANGE和LBN_DBLCLK通知码的值表示鼠标按下的次数）。

根据应用的需要，您也许要使用LBN_SELCHANGE或LBN_DBLCLK，也许二者都要使用。您的程序会收到许多LBN_SELCHANGE消息，但是LBN_DBLCLK消息只有当使用者双击鼠标时才会出现。如果您的程序使用双击，那么您需要提供一个复制LBN_DBLCLK的键盘接口。

一个简单的清单方块应用程序

既然您知道了如何建立清单方块，如何使用文字项目填入清单方块，如何接收来自清单方块的控件以及如何取得字符串，现在是到了写一个应用程序的时候了。如程序9-5中所示，ENVIRON程序在显示区域中使用清单方块来显示目前操作系统环境变量（例如PATH和WINDIR）。当您选择一个环境变量时，其内容将显示在显示区域的顶部。

程序9-5 ENVIRON

ENVIRON.C

```
/*------------------------------------------------------------------------
ENVIRON.C -- Environment List Box
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_LIST 1
#define ID_TEXT 2

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Environ") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
```

```
  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 = (HBRUSH) (COLOR_WINDOW + 1) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Environment List Box"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void FillListBox (HWND hwndList)
{
  int iLength ;
  TCHAR * pVarBlock, * pVarBeg, * pVarEnd, * pVarName ;
  pVarBlock = GetEnvironmentStrings () ; // Get pointer to environment block

  while (*pVarBlock)
  {
    if (*pVarBlock != '=') // Skip variable names beginning with '='
    {
      pVarBeg = pVarBlock ; // Beginning of variable name
      while (*pVarBlock++ != '=') ; // Scan until '='
      pVarEnd = pVarBlock - 1 ; // Points to '=' sign
      iLength = pVarEnd - pVarBeg ; // Length of variable name

      // Allocate memory for the variable name and terminating
      // zero. Copy the variable name and append a zero.
      pVarName = (TCHAR*)calloc (iLength + 1, sizeof (TCHAR)) ;
      CopyMemory (pVarName, pVarBeg, iLength * sizeof (TCHAR)) ;
      pVarName[iLength] = '\0' ;
      // Put the variable name in the list box and free memory.
      SendMessage (hwndList, LB_ADDSTRING, 0, (LPARAM) pVarName) ;
      free (pVarName) ;
    }
    while (*pVarBlock++ != '\0') ; // Scan until terminating zero
  }
  FreeEnvironmentStrings (pVarBlock) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HWND hwndList, hwndText ;
  int iIndex, iLength, cxChar, cyChar ;
  TCHAR * pVarName, * pVarValue ;
```

```
  switch (message)
  {
  case WM_CREATE :
    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    // Create listbox and static text windows.
    hwndList = CreateWindow (TEXT ("listbox"), NULL,
      WS_CHILD | WS_VISIBLE | LBS_STANDARD,
      cxChar, cyChar * 3,
      cxChar * 16 + GetSystemMetrics (SM_CXVSCROLL),
      cyChar * 5,
      hwnd, (HMENU) ID_LIST,
      (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
      NULL) ;

    hwndText = CreateWindow (TEXT ("static"), NULL,
      WS_CHILD | WS_VISIBLE | SS_LEFT,
      cxChar, cyChar,
      GetSystemMetrics (SM_CXSCREEN), cyChar,
      hwnd, (HMENU) ID_TEXT,
      (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
      NULL) ;

    FillListBox (hwndList) ;
    return 0 ;
  case WM_SETFOCUS :
    SetFocus (hwndList) ;
    return 0 ;
  case WM_COMMAND :
    if (LOWORD (wParam) == ID_LIST && HIWORD (wParam) == LBN_SELCHANGE)
    {
      // Get current selection.
      iIndex = SendMessage (hwndList, LB_GETCURSEL, 0, 0) ;
      iLength = SendMessage (hwndList, LB_GETTEXTLEN, iIndex, 0) + 1 ;
      pVarName = (TCHAR*)calloc (iLength, sizeof (TCHAR)) ;
      SendMessage (hwndList, LB_GETTEXT, iIndex, (LPARAM) pVarName) ;

      // Get environment string.
      iLength = GetEnvironmentVariable (pVarName, NULL, 0) ;
      pVarValue = (TCHAR*)calloc (iLength, sizeof (TCHAR)) ;
      GetEnvironmentVariable (pVarName, pVarValue, iLength) ;
      // Show it in window.

      SetWindowText (hwndText, pVarValue) ;
      free (pVarName) ;
      free (pVarValue) ;
    }
    return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

ENVIRON建立两个子窗口：一个是LBS_STANDARD样式的清单方块，另一个是SS_LEFT样式（置左对齐文字）的静态窗口。ENVIRON使用函数GetEnvironmentStrings来获得一个指标，该指标指向存有全部环境变量名及其值的内存区块。ENVIRON用FillListBox函数来分析此内存区块，并使用LB_ADDSTRING消息来指定清单方块窗口消息处理程序将每个字符串放入清单方块中。

当您执行ENVIRON时，可以使用鼠标或者键盘来选择环境变量。每次您改变选择时，清单方块都会给其父窗口WndProc发送一个WM_COMMAND消息。当WndProc收到WM_COMMAND消息时，它就检查wParam的低字组是否为ID_LIST（清单方块的子窗口ID）和wParam的高字组（通知码）是否等于LBN_SELCHANGE。如果是的，那么它就使用LB_GETCURSEL消息来获得选中项目

的索引，并使用LB_GETTEXT来获得外部环境变量名的字符串本身。ENVIRON程序使用C语言函数GetEnvironmentVariable来获得与变量相对应的环境字符串，使用SetWindowText将该字符串传递到静态子窗口控件中，这个静态子窗口控件被用来显示文字。

文件列表

我将最好的留在最后：LB_DIR，这是功能最强的清单方块消息。它用文件目录列表填入清单方块，并且可以选择将子目录和有效的磁盘驱动器也包括进来：

SendMessage (hwndList, LB_DIR, iAttr, (LPARAM) szFileSpec) ;

使用文件属性码

iAttr参数是文件属性代码，其最低字节是文件属性代码，该代码可以是表9-6数据的组合：

表9-6

iAttr	值	属性
DDL_READWRITE	0x0000	普通文件
DDL_READONLY	0x0001	只读文件
DDL_HIDDEN	0x0002	隐藏文件
DDL_SYSTEM	0x0004	系统文件
DDL_DIRECTORY	0x0010	子目录
DDL_ARCHIVE	0x0020	归档位设立的档案

高字节提供了一些对所要求项目的附加控制：

表9-7

iAttr	值	属性
DDL_DRIVES	0x4000	包括磁盘驱动器句柄
DDL_EXCLUSIVE	0x8000	互斥搜索

前缀DDL表示「对话目录列表」。

当LB_DIR消息的iAttr值为DDL_READWRITE时，清单方块列出普通文件、只读文件和归档位设立的档案。当值为DDL_DIRECTORY时，清单方块除了列出上述文件之外，还列出子目录，目录位于中括号之内。当值为DDL_DRIVES | DDL_DIRECTORY时，那么列表将扩展到包括所有有效的磁盘驱动器，而磁盘驱动器句柄显示在虚线之间。

将iAttr的最高位设立就可以只列出符合条件的文件，而不包括其它文件。例如，对Windows的文件备份程序，也许您只想列出最后一次备份后修改过的文件，这种文件的归档位设立，因此您可以使用DDL_EXCLUSIVE | DDL_ARCHIVE。

文件列表的排序

lParam参数是指向文件指定字符串如「*.*」的指针，这个文件指定字符串不影响清单方块中的子目录。

您也许希望给列有文件清单的清单方块使用LBS_SORT消息。清单方块首先列出符合文件指定要求的文件，再（可选择）列出子目录名。列出的第一个子目录名将采用下面的格式：

[..]

这一个「两个点」的子目录项允许使用者向根目录回溯一层（在根目录下列出文件名时此项目不会出现）。最后，具体的子目录名称采用下面的形式：

[SUBDIR]

再来是以下列形式列出的有效磁盘驱动器（也是可选择的）：

[-A-]

Windows的head程序

UNIX中有一个著名的实用程序叫做head，它显示文件开始的几行。让我们使用清单方块为Windows编写一个类似的程序。如程序9-6所示，HEAD将所有文件和子目录列在清单方块中。您可以挑选某个被选择的文件来显示，方法是在该文件上使用鼠标双击或者使用Enter键按下要选的文件。您也可以使用这两种方法之一来改变子目录。这个程序在HEAD窗口显示区域的右边，从文件的开头开始显示，它最多能够显示8 KB的内容。

程序9-6 HEAD

HEAD.C

```
/*------------------------------------------------------------------------
HEAD.C -- Displays beginning (head) of file
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_LIST 1
#define ID_TEXT 2

#define MAXREAD 8192
#define DIRATTR (DDL_READWRITE | DDL_READONLY | DDL_HIDDEN | DDL_SYSTEM | \
DDL_DIRECTORY | DDL_ARCHIVE | DDL_DRIVES)
#define DTFLAGS (DT_WORDBREAK | DT_EXPANDTABS | DT_NOCLIP |DT_NOPREFIX)
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK ListProc (HWND, UINT, WPARAM, LPARAM) ;

WNDPROC OldList ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("head") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) (COLOR_BTNFACE + 1) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("head"),
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
```

```
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL bValidFile ;
  static BYTE buffer[MAXREAD] ;
  static HWND hwndList, hwndText ;
  static RECT rect ;
  static TCHAR szFile[MAX_PATH + 1] ;
  HANDLE hFile ;
  HDC hdc ;
  int i, cxChar, cyChar ;
  PAINTSTRUCT ps ;
  TCHAR szBuffer[MAX_PATH + 1] ;
  switch (message)
  {
  case WM_CREATE :
    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    rect.left = 20 * cxChar ;
    rect.top = 3 * cyChar ;

    hwndList = CreateWindow (TEXT ("listbox"), NULL,
      WS_CHILDWINDOW | WS_VISIBLE | LBS_STANDARD,
      cxChar, cyChar * 3,
      cxChar * 13 + GetSystemMetrics (SM_CXVSCROLL),
      cyChar * 10,
      hwnd, (HMENU) ID_LIST,
      (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
      NULL) ;
    GetCurrentDirectory (MAX_PATH + 1, szBuffer) ;

    hwndText = CreateWindow (TEXT ("static"), szBuffer,
      WS_CHILDWINDOW | WS_VISIBLE | SS_LEFT,
      cxChar, cyChar, cxChar * MAX_PATH, cyChar,
      hwnd, (HMENU) ID_TEXT,
      (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE),
      NULL) ;

    OldList = (WNDPROC) SetWindowLong (hwndList, GWL_WNDPROC,
      (LPARAM) ListProc) ;
    SendMessage (hwndList, LB_DIR, DIRATTR, (LPARAM) TEXT ("*.*")) ;
    return 0 ;
caseWM_SIZE :
    rect.right = LOWORD (lParam) ;
    rect.bottom = HIWORD (lParam) ;
    return 0 ;
  case WM_SETFOCUS :
    SetFocus (hwndList) ;
    return 0 ;
  case WM_COMMAND :
    if (LOWORD (wParam) == ID_LIST && HIWORD (wParam) == LBN_DBLCLK)
    {
      if (LB_ERR == (i = SendMessage (hwndList, LB_GETCURSEL, 0, 0)))
        break ;
      SendMessage (hwndList, LB_GETTEXT, i, (LPARAM) szBuffer) ;

      if (INVALID_HANDLE_VALUE != (hFile = CreateFile (szBuffer,
        GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, 0, NULL)))
      {
```

```
      CloseHandle (hFile) ;
      bValidFile = TRUE ;
      lstrcpy (szFile, szBuffer) ;
      GetCurrentDirectory (MAX_PATH + 1, szBuffer) ;

      if (szBuffer [lstrlen (szBuffer) - 1] != '\\')
        lstrcat (szBuffer, TEXT ("\\")) ;
      SetWindowText (hwndText, lstrcat (szBuffer, szFile)) ;
    }
    else
    {
      bValidFile = FALSE ;
      szBuffer [lstrlen (szBuffer) - 1] = '\0' ;

      // If setting the directory doesn't work, maybe it's
      // a drive change, so try that.
      if (!SetCurrentDirectory (szBuffer + 1))
      {
        szBuffer [3] = ':' ;
        szBuffer [4] = '\0' ;
        SetCurrentDirectory (szBuffer + 2) ;
      }
      // Get the new directory name and fill the list box.

      GetCurrentDirectory (MAX_PATH + 1, szBuffer) ;
      SetWindowText (hwndText, szBuffer) ;
      SendMessage (hwndList, LB_RESETCONTENT, 0, 0) ;
      SendMessage (hwndList, LB_DIR, DIRATTR,
        (LPARAM) TEXT ("*.*")) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
  }
  return 0 ;
 case WM_PAINT :
  if (!bValidFile)
    break ;
  if (INVALID_HANDLE_VALUE == (hFile = CreateFile (szFile,
    GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL)))
  {
    bValidFile = FALSE ;
    break ;
  }
  ReadFile (hFile, buffer, MAXREAD, &i, NULL) ;
  CloseHandle (hFile) ;
  // i now equals the number of bytes in buffer.
  // Commence getting a device context for displaying text.
  hdc = BeginPaint (hwnd, &ps) ;
  SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
  SetTextColor (hdc, GetSysColor (COLOR_BTNTEXT)) ;
  SetBkColor (hdc, GetSysColor (COLOR_BTNFACE)) ;
  // Assume the file is ASCII

  DrawTextA (hdc, buffer, i, &rect, DTFLAGS) ;
  EndPaint (hwnd, &ps) ;
  return 0 ;
 case WM_DESTROY :
  PostQuitMessage (0) ;
  return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK ListProc (HWND hwnd, UINT message,
                      WPARAM wParam, LPARAM lParam)
{
 if (message == WM_KEYDOWN && wParam == VK_RETURN)
  SendMessage (GetParent (hwnd), WM_COMMAND,
  MAKELONG (1, LBN_DBLCLK), (LPARAM) hwnd) ;
 return CallWindowProc (OldList, hwnd, message, wParam, lParam) ;
```

```
}
```

在ENVIRON中，当我们选择一个环境变量时 – 无论是使用鼠标还是键盘 – 程序都将显示一个环境字符串。但是，如果我们在HEAD中使用这种选择显示方法，那么程序响应会很慢，这是因为在清单方块中移动选择时，程序仍然要不断地打开和关闭文件。然而，HEAD要求文件或者子目录被双击，从而引起一些问题，这是因为清单方块控件没有鼠标双击的自动键盘接口。前面讲过，如果可能，应该尽量提供键盘接口。

解决的方法是什么呢？当然是窗口子类别化。HEAD中的清单方块子类则函数叫做ListProc，它寻找wParam参数等于VK_RETURN的WM_KEYDOWN消息，并给其父窗口发送一条带有LBN_DBLCLK通知码的WM_COMMAND消息。在WndProc中，对WM_COMMAND的处理使用了Windows函数的CreateFile来检查清单方块中的选择。如果CreateFile传回一个错误信息，则表示该选择不是文件，而可能是一个子目录。然后HEAD使用SetCurrentDirectory来改变这个子目录。如果SetCurrentDirectory不能执行，程序将假定使用者已经选择了一个磁盘驱动器句柄。改变磁盘驱动器也需要呼叫SetCurrentDirectory，作为该函数参数的字符串则为是选择字符串中拿掉开头的斜线，并加上一个冒号。它向清单方块发送一条LB_RESETCONTENT消息来清除其中的内容，再发送一条LB_DIR消息，使用新子目录中的文件来填入清单方块。

WndProc中的WM_PAINT消息是用Windows的CreateFile函数来打开文件的，这将传回一个文件句柄，该句柄可以传递给Windows的ReadFile和CloseHandle函数。

现在，在本章中，我们第一次碰到这个问题：Unicode。我们所希望最完美的方式大概就是让操作系统辨认文本文件的种类，使ReadFile能将ASCII文件转换成Unicode文字，或者将Unicode文件转换成ASCII文字。但现实并非如此完美。ReadFile的功能只是读取文件中未经转换的字节，也就是说，DrawTextA（在编译好的可执行档中没有定义UNICODE标识符）会把文字解释为ASCII，而DrawTextW（Unicode版）会假设文字是Unicode的。

因此程序真正应该做的是去判别文件所包含的是ASCII文字还是Unicode文字，然后再恰当地呼叫DrawTextA或者DrawTextW。实际上，HEAD采用一个比较简单的方式，它只呼叫了DrawTextA。

第十章 菜单及其它资源

大多数Windows程序都包含一个自订的图标，Windows将该图标显示在应用程序窗口标题栏的左上角。当程序被列在「开始」菜单中，被显示在屏幕底部的工作列中，被列在Windows Explorer中，或者作为快捷方式显示在桌面上时，Windows也显示该程序的图标。有些程序 – 大部分是像小画家一类的图形绘制工具 – 也使用自订鼠标光标来表示程序的不同操作。还有许多Windows程序使用菜单和对话框。菜单、对话框加上滚动条，这是标准Windows使用者接口的卖点。

图标、光标、菜单和对话框都是相互关联的，它们是Windows的全部资源型态。资源即数据，它们被储存在程序的.EXE文件中，但是它们并非驻留在程序的数据区域中。也就是说，资源不能从程序原始码中定义的变量直接存取，Windows提供函数直接或间接地把它们加载内存以备使用。我们已经遇到了两个这样的函数，即LoadIcon和LoadCursor，它们出现在范例程序，定义窗口类别结构的内容设定叙述中。它们从Windows中加载二进制图标和光标映象，并传回该图标或光标的句柄。在本章中，我们先建立自己的图标，它会从程序自己的.EXE文件中载入。

在本书中，我们将讨论这些资源：

- 图标
- 光标
- 字符串
- 自订资源
- 菜单
- 键盘快捷键
- 对话框
- 位图

前六个资源在本章讨论，对话框在第十一章讨论，而位图在第十四章讨论。

图标、光标、字符串和自订资源

使用资源的好处之一，在于程序的许多组件能够连结编译进程序的.EXE文件中。如果没有资源这一个概念，如图标图像之类的二进制文件可能会存放在单独的文件中，.EXE会把它读入内存中使用。或者图标不得不在程序中以字节数组的形式定义（这样就无法看到实际的图标图像了）。作为资源，图标储存在开发者计算机上可单独编辑的文件中，但在编译程序中被连结编译进.EXE文件中。

将图标添加到程序

将资源添加到程序中需要Visual C++ Developer Studio的一些附加功能。对于图标来说，可以使用「Image Editor」（也称为「Graphics Editor」）来绘制图标的图像。该图像被储存在扩展名为.ICO的图标文件中。Developer Studio还产生一个资源描述文件（扩展名为.RC的文件，有时

也称作资源定义文件），它列出了程序的所有资源和一个让程序引用资源的表头文件（RESOURCE.H）。

因此，您可以看到这些新文件是如何组织在一起的，让我们以建立名为ICONDEMO的新项目开始。像往常一样，在Developer Studio中从File菜单中选择New，然后依次选择 项目页面标签和Win32 Application。在Project Name栏中键入ICONDEMO并单击OK。这时，Developer Studio建立了用于支持工作区和项目的五个文件。这些文件包括文本文件ICONDEMO.DSW、ICONDEMO.DSP和ICONDEMO.MAK（假设当您从 Tools菜单选择Open后，在显示的 Open对话框中，从Build页面标签中选中 Export makefile when saving project file）。现在，让我们像通常那样所做的建立C原始码文件。从 File菜单上选择New，选择Files页面标签，并单击 C++Source File。在File Name栏中键入ICONDEMO.C并单击OK。此时，Developer Studio就建立了一个空的ICONDEMO.C文件。键入程序10-1中的程序，或选择 Insert菜单，然后选择File As Text选项，从本书附上的光盘中复制原始码。

程序10-1 ICONDEMO

ICONDEMO.C

```
/*------------------------------------------------------------------------
ICONDEMO.C -- Icon Demonstration Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  TCHAR szAppName[] = TEXT ("IconDemo") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Icon Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
```

```
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HICON hIcon ;
  static int cxIcon, cyIcon, cxClient, cyClient ;
  HDC hdc ;
  HINSTANCE hInstance ;
  PAINTSTRUCT ps ;
  int x, y ;

  switch (message)
  {
  case WM_CREATE :
    hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
    hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
    cxIcon = GetSystemMetrics (SM_CXICON) ;
    cyIcon = GetSystemMetrics (SM_CYICON) ;
    return 0 ;
  case WM_SIZE :
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    for (y = 0 ; y < cyClient ; y += cyIcon)
      for (x = 0 ; x < cxClient ; x += cxIcon)
        DrawIcon (hdc, x, y, hIcon) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;
  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

如果您试着编译该程序，因为在程序开头引用的RESOURCE.H文件并不存在，所以会产生错误。然而，您不必直接建立RESOURCE.H文件，而是由Developer Studio为您建立一个。

您可以通过将资源描述文件添加到项目中来做到这一点。从「File」菜单中选择「New」，选择「Files」页面标签，单击「Resource Script」，在「File Name」栏中键入「ICONDEMO」，单击OK。此时，Developer Studio会建立两个文本文件：ICONDEMO.RC（资源描述文件）和RESOURCE.H（允许C原始码文件和资源描述文件引用相同的已定义标识符）。不必直接编辑这两个档案，只要让Developer Studio来维护它们就可以。如果您想查看资源描述文件和RESOURCE.H而不希望对Developer Studio产生干扰，可以用记事本打开它们。除非您对所做的动作很有把握，否则不要轻易地更改它们。请记住，只有在您下达明确的操作命令或重新编译项目时，Developer Studio才会储存这些文件的新版本。

资源描述文件是文本文件。它包括这些资源的可用文字形式表达的描述，例如菜单和对话框。资源描述文件也包括对非文字资源的二进制档案的引用，例如图标和自订的鼠标光标。

现在，已经存在RESOURCE.H文件，您可以试着重新编译一下ICONDEMO。现在会出现一条错误消息，指出IDI_ICON还没被定义。这个标识符第一次出现在下面的叙述中：

wndclass.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;

在本书前面的程序中，这个叙述是由下面的叙述代替的：

wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;

之所以改变叙述，是因为以前我们为应用程序使用的是标准的图标，而这里我们的目的是使用自订图标。

那么让我们建立一个图标吧！在Developer Studio的「File View」窗口中，您会看到两个文件 – ICONDEMO.C和ICONDEMO.RC。您开启CONDEMO.C后，就可以编辑原始码。开启ICONDEMO.RC后，就可以把资源添加到文件中或编辑已存在的资源。要添加图标的话，请从「 Insert」菜单上选择「Resource」选择您想添加的资源，也就是图标，然后再按下「 New」按钮。

现在呈现的是一个空白的32×32像素的图标，您可以在其中填入颜色。您会看到带有一组绘图工具和可用颜色的浮动工具列。注意颜色工具列中包括两个与颜色无关的选项，这两种颜色选项有时被称为「屏幕颜色」跟「反屏幕颜色」。当一个像素在着色时选择了「屏幕颜色」时，它实际上是透明的。不管图标在什么表面上显示，图标未着色的部分会显示出底色。这样我们就可以建立非矩形的图标。

双击围绕图标的区域，会出现「Icon Properties」对话框，该对话框使您能够更改图标的ID和文件名称。Developer Studio可能已经将ID设定为IDI_ICON1，将它改为IDI_ICON，这样ICONDEMO就可以引用图标（前缀IDI代表「图标的ID」）。同样地，将文件名改为ICONDEMO.ICO。

现在选择一种有特色的颜色（如红色）并在图标上画一个大的B（代表BIG），请注意不必像图10-1那么整齐。

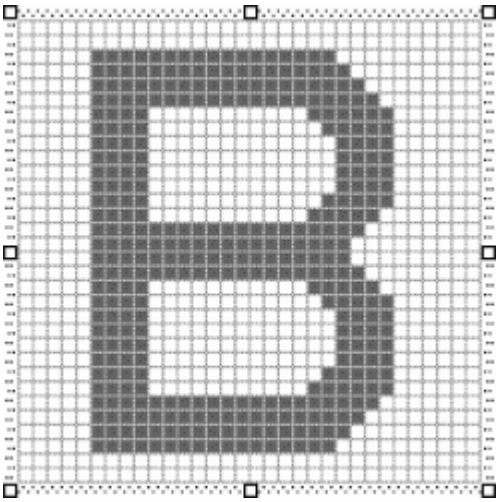


图10-1 显示在Developer Studio中的标准（32×32）ICONDEMO文件

此时程序应该能够编译并执行得很好了。Developer Studio将在ICONDEMO.RC资源描述文件中划一条横线，表示下面是带有标识符（IDI_ICON）的图标文件（ICONDEMO.ICO）。RESOURCE.H表头文件中会包含IDI_ICON标识符的定义。

Developer Studio通过资源编译器RC.EXE编译资源。文字资源描述文件被转化为二进制形式，也就是具有扩展名.RES的文件。然后，该已编译的资源文件随同.OBJ和.LIB文件一起在LINK步骤中被指定连结。这就是资源被添加到最后产生出来的.EXE文件中的方式。

当您执行ICONDEMO时，程序图标显示在标题栏的左上角和工作列中。如果您将程序添加到「开始」菜单中，或在桌面上放置快捷方式，您也会在那儿看到该图标。

ICONDEMO也在显示区域水平和垂直地重复显示该图标。程序使用叙述

hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;

取得图标的句柄。使用叙述

cxIcon = GetSystemMetrics (SM_CXICON) ;

cyIcon = GetSystemMetrics (SM_CYICON) ;

取得图标的大小。然后，程序通过多次呼叫

DrawIcon (hdc, x, y, hIcon) ;

显示图标，其中x和y是被显示图标其左上角的坐标。

在目前使用的大多数视讯显示卡上，带有SM_CXICON和SM_CYICON索引的GetSystemMetrics会回报图标的大小为32×32像素。这是我们在Developer Studio中建立的图标大小，它也是图标出现在桌面上和显示在ICONDEMO程序显示区域的大小。然而，这个大小并非显示在程序的标题栏或工作列中的图标大小。小图标的大小可以由带有SM_CXSMSIZE和SM_CYSMSIZE索引的GetSystemMetrics获得（第一个SM表示「system metrics（系统度量）」，被包含的SM表示「small（小）」）。对于目前使用的大多数显示卡来说，小图标的大小为16×16像素。

这会产生问题。当Windows将32×32的图标缩小为16×16的图标时，必需减少像素的行和列。这样，对于某些比较复杂的图标，就会失真。因此，我们应该为那些图像缩小就会变形的图标建立特殊的16×16像素的图标。在Developer Studio中图标图像的上面是标识为「Device」的下拉式清单方块，在它的右边有一个按钮，按下该按钮会弹出「New Icon Image」对话框，此时选择「Small（16×16）」。现在您可以画另一个图标。如图10-2所示，画一个「S」（表示「小」）。

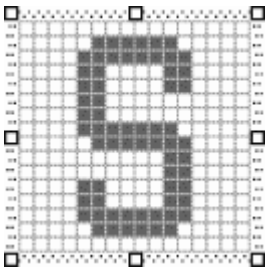


图10-2 在Developer Studio中显示的小（16×16）ICONDEMO文件

在该程序中您不必做任何事情。第二个图标图像被储存在相同的ICONDEMO.ICO文件中，并以相同的IDI_ICON标识符引用。在适当的时候，Windows会自动使用该较小的图标，例如在标题栏或工作列中。当在桌面上显示快捷方式，以及程序呼叫DrawIcon装饰显示区域时，Windows会使用大图标。

在掌握这些知识之后，让我们看一看使用图标的详细情况。

取得图标句柄

如果您仔细阅读ICONDEMO.RC和RESOURCE.H文件，会看到由Developer Studio产生用于维护文件的一些标记。然而，当编译资源描述文件时，只有少数几行是重要的。这些从ICONDEMO.RC和RESOURCE.H文件中摘录下来的关键部分被列在程序10-2中。

ICONDEMO.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
/////////////////////////////////////////////////////////////////////////////
// Icon
IDI_ICON ICON DISCARDABLE "icondemo.ico"
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by IconDemo.rc
#define IDI_ICON 101
```

程序10-2 ICONDEMO.RC和RESOURCE.H文件的摘录

程序10-2所显示的ICONDEMO.RC和RESOURCE.H文件与您在普通的文字编辑器中手动建立的很相似，80年代的Windows程序写作者就是这样做的。唯一不同的是AFXRES.H，它是个表头文件，包含了在建立由机器产生的MFC项目时由Developer Studio使用的常用标识符。在本书中，我们不会用到AFXRES.H。

ICONDEMO.RC中的这行

IDI_ICON ICON DISCARDABLE "icondemo.ico"

是资源描述文件的ICON叙述。该图标有一个数值标识符IDI_ICON，等于101。由Developer Studio添加的DISCARDABLE关键词指出，必要时Windows可以从内存中丢弃图标，以获得额外的空间。之后不需要程序任何特定的操作，Windows就能够重新加载图标。DISCARDABLE属性是内定的，不需要指定。只有在名称和目录路径包含空格时，Developer Studio才将文件名加上引号。

当资源编译程序将编译的资源储存在ICONDEMO.RES中，并且由连结程序将资源添加到ICONDEMO.EXE中以后，该资源就可以经由一个资源型态（RT_ICON）和一个标识符（IDI_ICON或101）来标识。程序可以通过呼叫LoadIcon函数取得此图标的句柄：

hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;

请注意ICONDEMO在两个地方呼叫这个函数，一次在定义窗口类别时，另一次在窗口消息处理程序中取得图标的句柄用于绘制。LoadIcon传回HICON型态的值，它是图标的句柄。

LoadIcon的第一个参数，是指出资源来自哪个文件的执行实体句柄。使用hInstance表示它来自程序自己的.EXE文件。LoadIcon的第二个参数实际上被定义为指向字符串的指针。待会将会看到，可以使用字符串而不是用数值标识符标识资源。宏MAKEINTRESOURCE（把整数转换成资源字符串）生成指向非数字的指针，如下所示：

#define MAKEINTRESOURCE(i) (LPTSTR) ((DWORD) ((WORD) (i)))

LoadIcon知道，如果第二个参数的高字组为0，那么低字组就为图标的数值标识符。图标的标识符必须为16位值。

本书前面的范例程序使用了预先定义的图标：

LoadIcon (NULL, IDI_APPLICATION) ;

hInstance参数被设定为NULL，因此Windows知道这是预先定义的图标。IDI_APPLICATION也在WINUSER.H中用MAKEINTRESOURCE定义：

#define IDI_APPLICATION MAKEINTRESOURCE(32512)

LoadIcon的第二个参数带来了一个有趣的问题：图标的标识符能可以为字符串吗？答案是可以。方法如下：在 Developer Studio 中，在 ICONDEMO 项目的文件列表上，选择 IDONDEMO.RC。您会看到顶端为「IconDemo Resource」的树状结构，然后是资源型态「Icon」，再下来是「IDI_ICON」。如果用鼠标右键单击图标标识符，并从菜单上选择「 Properties」，您就能改变ID。实际上，您可以把名称放在引号内将其更改为字符串。我用这种方法指定资源名称，并在本书的其它地方也使用该方法。

我喜欢为图标（以及一些其它资源）使用文字名称，因为名称可以是程序的名称。例如，假定

文件被命名为MYPROG。如果您使用「Icon Properties」对话框将图标的ID指定为「MyProg」(包括引号)，资源描述文件将包含下列叙述：

MYPROG ICON DISCARDABLE myprog.ico

然而，在RESOURCE.H中并没有#define叙述，来指出MYPROG是数值标识符。资源描述文件将假定MYPROG是字符串标识符。

在C程序中，使用LoadIcon函数来取得图标句柄。您可能已经有了表示程序名的字符串：

static TCHAR szAppName [] = TEXT ("MyProg") ;

这意味着程序可以使用叙述：

hIcon = LoadIcon (hInstance, szAppName) ;

来加载图标，这比宏MAKEINTRESOURCE更清晰一些。

但是如果您确实想用数字来命名，那么您可以用数字代替标识符或字符串。在「Icon Properties」对话框中，在ID栏中输入数字。资源描述文件将有一个类似下面的ICON叙述：

125 ICON DISCARDABLE myprog.ico

可以使用两种方法之一引用图标。明显易读的方式是：

hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (125)) ;

另一个不易阅读的方式是：

hIcon = LoadIcon (hInstance, TEXT ("#125")) ;

Windows识别初始字符#作为ASCII形式中字符数值的开头。

在程序中使用图标

虽然Windows以几种方式用图标来代表程序，但是许多Windows程序仅在用WNDCLASS结构和RegisterClass定义窗口类别时指定一个图标。如我们所看到的，这样作用得很好，尤其当图标文件包含标准和较小的图像大小时，更是如此。Windows在显示图标图像时，它会在图标文件中选择最合适的图像大小。

RegisterClass有一个改进版本叫做RegisterClassEx，它使用名为WNDCLASSEX的结构。WNDCLASSEX有两个附加的字段：cbSize和hIconSm。cbSize字段指出了WNDCLASSEX结构的大小，假设hIconSm被设定为小图标的图标句柄。这样，在WNDCLASSEX结构中，您可以设定与两个图标文件相关的两个图标句柄 – 一个用于标准图标，一个用于小图标。

有这种必要吗？没有。正如我们看到的，Windows已经从单个图标文件中提取了大小合适的图标图像。RegisterClassEx似乎没有RegisterClass聪明。如果hIconSm字段使用了包含多个图像的图标文件，则只有第一个图像能被利用。它可能是标准大小的图标，使用时才被缩小。RegisterClassEx似乎是为了使用多个图标图像而设计的，每个图像只包含一种图标大小。因为现在可以将多个图标大小包括在同一个图标文件中，所以我建议使用WNDCLASS和RegisterClass。

如果您想在程序执行的时候，动态地更改程序的图标，可以使用SetClassLong来达到目的。例如，如果您有与标识符IDI_ALTICON相关的第二个图标文件，则您可以使用以下的叙述将其切换到那个图标：

```
SetClassLong (hwnd, GCL_HICON,
          LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ALTICON))) ;
```

如果不想储存程序图标的句柄，但要使用DrawIcon函数在别处显示它，可以使用

GetClassLong获得句柄。例如：

DrawIcon (hdc, x, y, GetClassLong (hwnd, GCL_HICON)) ;

在Windows文件的某些部分，LoadIcon被称为「过时的」，并推荐使用LoadImage（LoadIcon在/Platform SDK/User Interface Services/Resources/Icons中说明，LoadImage在/Platform SDK/User Interface Services/Resources/Resources中说明）。当然LoadImage更为灵活，但它没有LoadIcon简单。您会注意到，在ICONDEMO中对同一个图标呼叫了LoadIcon两次。这不会产生问题，也没有使用额外的内存。LoadIcon是取得句柄但不需要清除句柄的少数几个函数之一。实际上有一个DestroyIcon函数，但它与CreateIcon、CreateIconIndirect和CreateIconFromResource连在一起使用。这些函数使程序能够动态地建立图标图像。

使用自订光标

在程序中使用自订的鼠标光标与使用自订的图标相似，只是大多数程序写作者总是使用Windows提供的光标。自订光标一般为单色，大小为32×32像素。在Developer Studio中建立光标与建立图标的方法相同（从「Insert」菜单上选择「 Resource」，然后单击「Cursor」），但不要忘记定义热点。

可以在对象类别定义中设定自订光标，叙述为：

wndclass.hCursor = LoadCursor (hInstance, MAKEINTRESOURCE (IDC_CURSOR)) ;

如果光标用文字名称定义，则为：

wndclass.hCursor = LoadCursor (hInstance, szCursor) ;

每当鼠标位于根据这个类别建立的窗口上时，就会显示与IDC_CURSOR或szCursor相对应的鼠标光标。

如果使用了子窗口，那么您可能希望光标随着所在窗口的不同而有所区别。如果程序为这些子窗口定义了窗口类别，就可以在每个窗口类别中适当地设定hCursor字段，让每个窗口类别使用不同的光标。如果使用了预先定义的子窗口控件，就可以使用以下方法改变窗口类别的hCursor字段：

SetClassLong (hwndChild, GCL_HCURSOR,

LoadCursor (hInstance, TEXT ("childcursor")) ;

如果您将显示区域划分为较小的逻辑区域而不使用子窗口，就可以使用SetCursor来改变鼠标光标：

SetCursor (hCursor) ;

在处理WM_MOUSEMOVE消息处理期间，您应该呼叫SetCursor；否则，当光标移动时，Windows将使用窗口类别中定义的光标来重画光标。文件指出，如果没有改变光标，则SetCursor速度将会很快。

字符串资源

把字符串当成资源的观念一开始可能令人觉得诡异。因为我们在使用原始码中定义为变量的一般字符串时，并没有碰到任何问题。

字符串资源主要是为了让程序转换成其它语言时更为方便。正如后面两章中将看到的一样，菜单和对话框也是资源描述文件的一部分。如果使用字符串资源而不是将字符串直接放入原始码中，那么程序所使用的所有文字将在同一文件 – 资源描述文件中。如果转换了资源描述文件中的文字，那么建立程序的另一种语言版本所需做的一切就是重新连结程序。这种方法比重新组织原始码安全

得多（然而，除了下一个范例程序，我在本书的其它程序中不使用字符串表，原因是字符串表使程序代码看起来更为模糊和复杂）。

您可以在「Insert」菜单中选择「Resource」，再选择「 String Table」，建立一个字符串表。字符串会显示在屏幕右边的列表中。通过双击字符串就可以选中它。针对每个字符串，您可以指定标识符和字符串的内容。

在资源描述中，字符串显示在一个多行的叙述中，如下所示：

```
STRINGTABLE DISCARDABLE
BEGIN
IDS_STRING1, "character string 1"
IDS_STRING2, "character string 2"
//其它字符串定义
END
```

如果您在替早期版本的Windows写程序，并在文字编辑器中手动建立这个字符串表（用Developer Studio来做这件事当然更容易得多了），您可以用左右大括号代替BEGIN和END叙述。

资源描述可以包含多个字符串表，但是每个ID必须唯一表示一个字符串。每个字符串占一行，最多4097个字符。\t可以作为制表符，\n则作为linefeed字符号。DrawText和MessageBox函数能够识别这些控制符号。

您的程序可以使用LoadString呼叫把字符串复制到程序数据段的缓冲区中：

LoadString (hInstance, id, szBuffer, iMaxLength) ;

参数id是ID，它加在资源描述文件中每个字符串的前面；szBuffer是指向接收字符串的字符数组的指针；iMaxLength是送入szBuffer中的最大字符数。函数传回字符串中的字符数。

每个字符串前面的ID一般是定义在表头文件中的宏标识符。许多Windows程序写作者使用前缀IDS_ 来表示字符串的ID。有时，文件名称或其它信息需要在字符串显示时插入到字符串中。在这种情况下，您可以将C的格式化字符放入字符串，并把它用于wsprintf中作为一个格式化字符串。

所有资源文字 – 包括字符串表中的文字 – 以Unicode格式储存在.RES编译资源文件以及最终的.EXE文件中。LoadStringW函数直接加载Unicode文字。LoadStringA函数（仅在Windows 98下有效）完成由Unicode到本地代码页的文字转换。

让我们来看一个程序，它使用三个字符串，在消息框中显示三条错误信息。RESOURCE.H表头文件为这些信息定义了三个标识符：

```
#define IDS_FILENOTFOUND 1
#define IDS_FILETOOBIG 2
#define IDS_FILEREADONLY 3
```

资源描述文件具有此字符串表：

```
STRINGTABLE
BEGIN
IDS_FILENOTFOUND, "File %s not found."
IDS_FILETOOBIG, "File %s too large to edit."
IDS_FILEREADONLY, "File %s is read-only."
END
```

C原始码文件也包含这个表头文件，并定义了一个显示消息框的函数（我假定szAppName是一个包含程序名称的整体变量）。

```
OkMessage (HWND hwnd, int iErrorNumber, TCHAR *szFileName)
{
  TCHAR szFormat [40] ;
  TCHAR szBuffer [60] ;
  LoadString (hInst, iErrorNumber, szFormat, 40) ;
  wsprintf (szBuffer, szFormat, szFilename) ;
```

```
  return MessageBox ( hwnd, szBuffer, szAppName,
    MB_OK | MB_ICONEXCLAMATION) ;
}
```

为了显示包含「file not found」信息的消息框，程序呼叫：

OkMessage (hwnd, IDS_FILENOTFOUND, szFileName) ;

自订的资源

Windows也定义了「自订资源」，这又称为「使用者定义的资源」(使用者就是您 – 程序写作者，而不是那个使用您程序的幸运者)。自订资源让连结.EXE文件中的各种数据更为方便，对取得程序中的数据也是如此。资料可以是您需要的任何格式。程序用于存取自订资源的Windows函数促使Windows将数据加载内存并传回指向它的指标。然后您就可以对程序做任何操作。您会发现对于储存和存取各种自己的数据，这要比把数据储存在外部文件中，再使用文件输入函数存取它要方便得多。

例如，您有一个文件叫做BINDATA.BIN，它包含程序需要显示的一些数据。您可以选择这个文件的格式。如果在MYPROG项目中有MYPROG.RC资源描述文件，您就可以在Developer Studio中从「Insert」菜单中选择「Resource」并按「 Custom」按钮，来建立自订的资源。键入表示资源的名称：例如，BINTYPE。然后，Developer Studio会生成资源名称（在这种情况下是IDR_BINTYPE1）并显示让您输入二进制数据的窗口。但是您不必输入什么，用鼠标右键单击IDR_BINTYPE1名称，并选择 Properties，然后就可以输入一个文件名称：例如，BINDATA.BIN。

资源描述文件就会包含以下的一行叙述：

IDR_BINTYPE1 BINTYPE BINDATA.BIN

除了我们刚刚生成的BINTYPET资源型态外，这个叙述与ICONDEMO中的ICON叙述一样。有了图标后，您可以对资源名称使用文字的名称，而不是数字的标识符。

当您编译并连结程序，整个BINDATA.BIN文件会被并入MYPROG.EXE文件中。

在程序的初始化（比如，在处理WM_CREATE消息时）期间，您可以获得资源的句柄：

```
hResource = LoadResource(hInstance,
                    FindResource (hInstance, TEXT ("BINTYPE"),
                    MAKEINTRESOURCE (IDR_BINTYPE1))) ;
```

变量hResource定义为HGLOBAL型态，它是指向内存区块的句柄。不管它的名称是什么，LoadResource不会立即将资源加载内存。把LoadResource和FindResource函数如上例般合在一起使用，在实质上就类似于LoadIcon和LoadCursor函数的做法。事实上，LoadIcon和LoadCursor函数就用到了LoadResource和FindResource函数。

当您需要存取文字时，呼叫LockResource：

pData = LockResource (hResource) ;

LockResource将资源加载内存（如果还没有加载的话），然后它会传回一个指向资源的指标。当结束对资源的使用时，您可以从内存中释放它：

FreeResource (hResource) ;

当您的程序终止时，也会释放资源，即使您没有呼叫FreeResource.。

让我们看一个使用三种资源 – 一个图标、一个字符串表和一个自订的资源 – 的范例程序。程序10-3所示的POEPOEM程序在其显示区域显示Edgar Allan Poe的「Annabel Lee」文字。自订的资源是文件POEPOEM.TXT，它包含了一段诗文，此文本文件以反斜线（\）结束。

程序10-3 POEPOEM

POEPOEM.C

```
/*---------------------------------------------------------------------------
POEPOEM.C -- Demonstrates Custom Resource
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HINSTANCE hInst ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  TCHAR szAppName [16], szCaption [64], szErrMsg [64] ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  LoadString ( hInstance, IDS_APPNAME, szAppName,
    sizeof (szAppName) / sizeof (TCHAR)) ;

  LoadString ( hInstance, IDS_CAPTION, szCaption,
    sizeof (szCaption) / sizeof (TCHAR)) ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    LoadStringA (hInstance, IDS_APPNAME, (char *) szAppName,
      sizeof (szAppName)) ;
    LoadStringA (hInstance, IDS_ERRMSG, (char *) szErrMsg,
      sizeof (szErrMsg)) ;
    MessageBoxA (NULL, (char *) szErrMsg,
      (char *) szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, szCaption,
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static char * pText ;
```

```
static HGLOBAL hResource ;
static HWND hScroll ;
static int iPosition, cxChar, cyChar, cyClient, iNumLines, xScroll ;
HDC hdc ;
PAINTSTRUCT ps ;
RECT rect ;
TEXTMETRIC tm ;

switch (message)
{
case WM_CREATE :
  hdc = GetDC (hwnd) ;
  GetTextMetrics (hdc, &tm) ;
  cxChar = tm.tmAveCharWidth ;
  cyChar = tm.tmHeight + tm.tmExternalLeading ;
  ReleaseDC (hwnd, hdc) ;

  xScroll = GetSystemMetrics (SM_CXVSCROLL) ;
  hScroll = CreateWindow (TEXT ("scrollbar"), NULL,
    WS_CHILD | WS_VISIBLE | SBS_VERT,
    0, 0, 0, 0,
    hwnd, (HMENU) 1, hInst, NULL) ;
  hResource = LoadResource (hInst,
    FindResource (hInst, TEXT ("AnnabelLee"),
    TEXT ("TEXT"))) ;
  pText = (char *) LockResource (hResource) ;
  iNumLines = 0 ;

  while (*pText != '\\' && *pText != '\0')
  {
    if (*pText == '\n')
      iNumLines ++ ;
    pText = AnsiNext (pText) ;
  }
  *pText = '\0' ;
  SetScrollRange (hScroll, SB_CTL, 0, iNumLines, FALSE) ;
  SetScrollPos (hScroll, SB_CTL, 0, FALSE) ;
  return 0 ;
case WM_SIZE :
  MoveWindow (hScroll, LOWORD (lParam) - xScroll, 0,
    xScroll, cyClient = HIWORD (lParam), TRUE) ;
  SetFocus (hwnd) ;
  return 0 ;
case WM_SETFOCUS :
  SetFocus (hScroll) ;
  return 0 ;
case WM_VSCROLL :
  switch (wParam)
  {
  case SB_TOP :
    iPosition = 0 ;
    break ;
  case SB_BOTTOM :
    iPosition = iNumLines ;
    break ;
  case SB_LINEUP :
    iPosition -= 1 ;
    break ;
  case SB_LINEDOWN :
    iPosition += 1 ;
    break ;
  case SB_PAGEUP :
    iPosition -= cyClient / cyChar ;
    break ;
  case SB_PAGEDOWN :
    iPosition += cyClient / cyChar ;
    break ;
  case SB_THUMBPOSITION :
    iPosition = LOWORD (lParam) ;
```

```
     break ;
    }
    iPosition = max (0, min (iPosition, iNumLines)) ;

    if (iPosition != GetScrollPos (hScroll, SB_CTL))
    {
      SetScrollPos (hScroll, SB_CTL, iPosition, TRUE) ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;
    case WM_PAINT :
      hdc = BeginPaint (hwnd, &ps) ;
      pText = (char *) LockResource (hResource) ;

      GetClientRect (hwnd, &rect) ;
      rect.left += cxChar ;
      rect.top += cyChar * (1 - iPosition) ;
      DrawTextA (hdc, pText, -1, &rect, DT_EXTERNALLEADING) ;

      EndPaint (hwnd, &ps) ;
      return 0 ;
    case WM_DESTROY :
      FreeResource (hResource) ;
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

POEPOEM.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// TEXT
ANNABELLEE  TEXT DISCARDABLE "poepoem.txt"

////////////////////////////////////////////////////////////////////////////
// Icon
POEPOEM   ICON DISCARDABLE "poepoem.ico"

////////////////////////////////////////////////////////////////////////////
// String Table
STRINGTABLE DISCARDABLE
BEGIN
IDS_APPNAME "PoePoem"
IDS_CAPTION """Annabel Lee"" by Edgar Allan Poe"
IDS_ERRMSG "This program requires Windows NT!"
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by PoePoem.rc
#define IDS_APPNAME   1
#define IDS_CAPTION   2
```

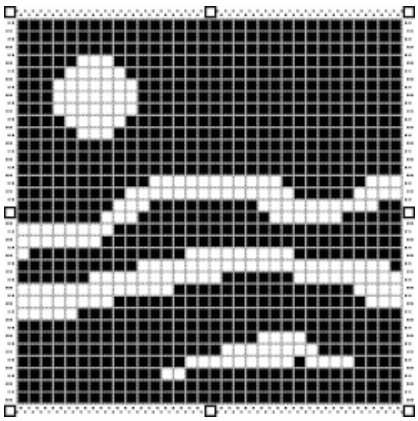
#define IDS_ERRMSG 3

POEPOEM.TXT

```
   It was many and many a year ago,
     In a kingdom by the sea,
   That a maiden there lived whom you may know
   By the name of Annabel Lee;
   And this maiden she lived with no other thought
     Than to love and be loved by me.
   I was a child and she was a child
     In this kingdom by the sea,
   But we loved with a love that was more than love --
```

```
     I and my Annabel Lee --
   With a love that the winged seraphs of Heaven
     Coveted her and me.
   And this was the reason that, long ago,
     In this kingdom by the sea,
   A wind blew out of a cloud, chilling
     My beautiful Annabel Lee;
   So that her highborn kinsmen came
     And bore her away from me,
   To shut her up in a sepulchre
   In this kingdom by the sea.
   The angels, not half so happy in Heaven,
     Went envying her and me --
   Yes! that was the reason (as all men know,
     In this kingdom by the sea)
   That the wind came out of the cloud by night,
     Chilling and killing my Annabel Lee.
   But our love it was stronger by far than the love Of those who were older than we -- Of many far wiser
than we --
   And neither the angels in Heaven above
     Nor the demons down under the sea
   Can ever dissever my soul from the soul
     Of the beautiful Annabel Lee:
   For the moon never beams, without bringing me dreams
     Of the beautiful Annabel Lee;
   And the stars never rise, but I feel the bright eyes
     Of the beautiful Annabel Lee:
   And so, all the night-tide, I lie down by the side
   Of my darling -- my darling -- my life and my bride,
     In her sepulchre there by the sea --
     In her tomb by the sounding sea.
     [May, 1849]
   \
```

POEPOEM.ICO



在POEPOEM.RC资源描述文件中，使用者定义的资源被定义为TEXT型态，取名为AnnabelLee：

ANNABELLEE TEXT POEPOEM.TXT

在WndProc处理WM_CREATE时，使用FindResource和LoadResource取得资源句柄。使用LockResource锁定资源，并且使用一个小程序将文件末尾的反斜线（\）换成0，这有利于后面WM_PAINT消息处理期间使用的DrawText函数。

注意，这里使用的是子窗口的滚动条，而不是窗口滚动条，这是因为子窗口滚动条有一个自动的键盘接口，因此在POEPOEM中没有处理WM_KEYDOWN。

POEPOEM还使用三个字符串，它们的ID在RESOURCE.H表头文件中定义。在程序的开始，

IDS_APPNAME和IDS_CAPTIONPOEPOEM字符串由LoadString加载内存：

```
LoadString (hInstance, IDS_APPNAME, szAppName, sizeof (szAppName) /
          sizeof (TCHAR)) ;
LoadString (hInstance, IDS_CAPTION, szCaption, sizeof (szCaption) /
          sizeof (TCHAR)) ;
```

注意RegisterClass前面的两个呼叫。如果您在Windows 98下执行Unicode版本的POEPOEM，这两个呼叫就都会失败。因此，LoadStringA比LoadStringW要复杂得多（LoadStringA必须将资源字符串由Unicode转化为ANSI，而LoadStringW仅是直接加载它），LoadStringW在Windows 98下不被支持。这意味着在Windows 98下，当RegisterClassW函数失败时，MessageBoxW函数（Windows 98支持）就不能使用LoadStringW加载程序的字符串。由于这个原因，程序使用LoadStringA加载IDS_APPNAME和IDS_ERRMSG字符串，并使用MessageBoxA显示自订的消息框：

```
if (!RegisterClass (&wndclass))
{
  LoadStringA (hInstance, IDS_APPNAME, (char *) szAppName,
    sizeof (szAppName)) ;
  LoadStringA (hInstance, IDS_ERRMSG, (char *) szErrMsg,
    sizeof (szErrMsg)) ;
  MessageBoxA (NULL, (char *) szErrMsg,
    (char *) szAppName, MB_ICONERROR) ;
  return 0 ;
}
```

注意，TCHAR字符串变量是指向char的指针。

既然我们已经定义了用于POEPOEM的所有字符串资源，那么翻译者将程序转换成外语版本就很容易了。当然，它们将不得不翻译「Annabel Lee」这个名字 – 我想，这会是一项困难得多的工作。

菜单

您还记得Monty Python有关奶酪店的幽默短剧吗？那故事内容是这样的：一个客人走进奶酪店想买某种奶酪。当然，店里没有这种奶酪。因此他又问有没有另一种奶酪，然后再问另一种，再问另一种，不断的问店家有没有另一种奶酪（最后总共问了40种的奶酪），回答仍然是没有，没有，没有，没有，没有。

这个不幸的事件可以通过菜单的使用来避免。一个菜单是一列可用的选项，它告诉饥饿的用餐者，厨房可以提供哪些服务，并且 – 对于Windows程序来说 – 还告诉使用者一个应用程序能够执行哪些操作。

菜单可能是Windows程序提供的一致使用者接口中最重要的部分，而在您的程序中增加菜单，是Windows程序设计中相对简单的部分。您在Developer Studio中定义菜单。每个可选的菜单项被赋予唯一的ID。您在窗口类别结构中指定菜单名称。当使用者选择一个菜单项时，Windows给您的程序发送包含该ID的WM_COMMAND消息。

讨论完菜单后，我还将讨论键盘快捷键，它们是一些键的组合，主要用于启动菜单功能。

菜单概念

窗口的菜单列紧接在标题栏的下方显示，这个菜单列有时被称为「主菜单」或「顶层菜单」。列在顶层菜单的项目通常是下拉式菜单，也叫做「弹出式菜单」或「子菜单」。您也可以定义多重嵌套的弹出式菜单，也就是说，在弹出式菜单上的项目可以存取另一个弹出式菜单。有时弹出式菜

单上的项目呼叫对话框以获得更多的信息（对话框在下一章介绍）。在标题栏的最左端，很多父窗口都显示程序的小图标，这个图标可以启动系统菜单。它实际上是另一个弹出式菜单。

弹出式菜单的各项可以是「被选中的」，这意味着Windows在菜单文字的左端显示一个小的选中标记，选中标记让使用者知道从菜单中选中了哪些选项。这些选项之间可以是互斥的，也可以不互斥。顶层菜单项不能被选中。

顶层菜单或弹出式菜单项可以被「启用」、「禁用」或「无效化」。「启动」和「不启动」有时候被当作「启用」和「禁用」的同义词。被启用或禁用的菜单项在使用者看来是一样的，但是无效化的菜单项是使用灰色文字来显示的。

从使用者的角度来看，启用、禁用和无效化的菜单项都是可以「选择的」（被选择的菜单项目会被加高亮度显示），也就是说，使用者可以使用鼠标选择被禁用的菜单项，将反相显示光标列移动到禁用的菜单项上，或者使用菜单项的关键词母来选择该菜单项。然而，从程序写作者的角度来看，启用、禁用和无效化菜单项的功能是不同的。Windows只为启用的菜单项向程序发送WM_COMMAND消息。要让选项变得无效，可以把那些菜单项禁用和无效化。如果您想让使用者知道选择是无效的，那么您可以让一个菜单项无效化。

菜单结构

当您建立或改变程序中的菜单时，把顶层菜单和每一个弹出式菜单想象成各自独立的菜单是有用的。顶层菜单有一个菜单句柄，在顶层菜单中的每一个弹出式菜单也有它自己的菜单句柄。系统菜单（也是一个弹出式菜单）也有菜单句柄。

菜单中的每一项都有三个特性。第一个特性是菜单中显示什么，它可以是字符串或位图。第二个特性是WM_COMMAND消息中Windows发送给程序的菜单ID，或者是在使用者选择菜单项时Windows显示的弹出式菜单的句柄。第三个特性是菜单项的属性，包括是否被禁用、无效化或被选中。

定义菜单

要使用Developer Studio来给程序资源描述文件添加菜单，可以从Insert菜单中选择Resource并选择Menu（或者您可能已经知道了）。然后，您可以用交谈式的方式定义菜单。菜单中每一项都有一个相关的 Menu Item Properties对话框，指出该项目的字符串。如果选中了Pop-up复选框，该项目就会呼叫一个弹出式菜单，并且没有ID与此项目相联系。如果没有选中Pop-up复选框，该项目被选中时就会产生带有特定ID的WM_COMMAND消息。这两类菜单项分别出现在资源描述文件的POPUP和MENUITEM叙述中。

当您为菜单中的项目键入文字时，可以键入一个「&」符号，指出后面一个字符在Windows显示菜单时要加底线。这种底线字符是在您使用Alt键选择菜单项时Windows要寻找的比对字符。如果在文字中不包括「&」符号，就不显示任何底线，Windows会将菜单项文字的第一个字母用于Alt键查找。

如果在Menu Items Properties对话框中选中Grayed选项，则菜单项是不能启动的，它的文字是灰色的，该项不产生WM_COMMAND消息。如果选中 Inactive选项，则菜单项也是不能启动的，也不产生WM_COMMAND消息，但是它的文字显示正常。 Checked选项在菜单项边上放置一个选中标记。Separator选项在弹出式菜单上产生一个分栏的横线。

在弹出式菜单的项目上，可以在字符串中使用制表符\t。紧接着\t的文字被放置在距离弹出式菜单的第一列右边新的一列上。在本章后面，会看到在使用键盘快捷键时它起的作用。字符串中的\a使跟着它的文字向右对齐。

您指定的ID值是Windows发送给窗口消息处理程序中菜单消息中的数值。在菜单中ID值应该是唯一的。按照惯例，我使用以IDM（「ID for a Menu」）开头的标识符。

在程序中引用菜单

大多数Windows应用程序在资源描述文件中只有一个菜单。您可以给菜单起一个与程序名称相同的文字的名称。程序写作者经常将程序名用于菜单名称，以便相同的字符串可以用于窗口类别、程序的图标名称和菜单名称。然后，程序在窗口的定义中为菜单引用该名称：

wndclass.lpszMenuName = szAppName ;

虽然存取菜单资源的最常用方法是在窗口类别中指定菜单，您也可以使用其它方法。Windows应用程序可以使用LoadMenu函数将菜单资源加载内存中，如同LoadIcon和LoadCursor函数一样。LoadMenu传回一个菜单句柄。如果您在资源描述文件中为菜单使用了名称，叙述如下：

hMenu = LoadMenu (hInstance, TEXT ("MyMenu")) ;

如果使用了数值，那么LoadMenu呼叫采用如下的形式：

hMenu = LoadMenu (hInstance, MAKEINTRESOURCE (ID_MENU)) ;

然后，您可以将这个菜单句柄作为CreateWindow的第九个参数：

```
hwnd = CreateWindow ( TEXT ("MyClass"), TEXT ("Window Caption"),
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, hMenu, hInstance, NULL) ;
```

在这种情况下，CreateWindow呼叫中指定的菜单可以覆盖窗口类别中指定的任何菜单。如果CreateWindow的第九个参数是NULL，那么您可以把窗口类别中的菜单看作是这种窗口类别的窗口内定使用的菜单。这样，您可以为依据同一窗口类别建立的几个窗口使用不同的菜单。

您也可以在窗口类别中指定NULL菜单，并且在CreateWindow呼叫中也指定NULL菜单，然后在窗口被建立后再给窗口指定一个菜单：

SetMenu (hwnd, hMenu) ;

这种形式使您可以动态地修改窗口的菜单。在本章后面的NOPOPUPS程序中我们将会看到这方面的例子。

当窗口被清除时，与窗口相关的所有菜单都将被清除。与窗口不相关的菜单在程序结束前通过呼叫DestroyMenu主动清除。

菜单和消息

当使用者选择一个菜单项时，Windows通常向窗口消息处理程序发送几个不同的消息。在大多数情况下，您的程序可以忽略大部分消息，只需把它们传递给DefWindowProc即可。WM_INITMENU就是这一类的消息，它具有下列参数：

wParam： 主菜单句柄

lParam： 0

wParam值是您的主菜单句柄，即使使用者选择的是系统菜单中的项目。Windows程序通常忽略WM_INITMENU消息。尽管在选中该项之前的消息已经给程序提供了修改菜单的机会，但是我们觉得此刻改变顶层菜单是会扰乱使用者的。

程序也会接收到WM_MENUSELECT消息。随着使用者在菜单项中移动光标或者鼠标，程序会收到许多WM_MENUSELECT消息。这对实作那些包含对菜单项的文字描述的状态列是很有帮助的。

WM_MENUSELECT的参数如下所示：

LOWORD (wParam)：被选中项目：菜单ID或者弹出式菜单句柄

HIWORD (wParam)：选择旗标

lParam: 包含被选中项目的菜单句柄

WM_MENUSELECT是一个菜单追踪消息，wParam的值告诉您目前选择的是菜单中的哪一项（加高亮度显示的那个），wParam的高字组中的「选择旗标」可以是下列这些旗标的组合：MF_GRAYED、MF_DISABLED、MF_CHECKED、MF_BITMAP、MF_POPUP、MF_HELP、MF_SYSMENU和MF_MOUSESELECT。如果您需要根据对菜单项的选择来改变窗口显示区域的内容，那么您可以使用WM_MENUSELECT消息。许多程序把该消息发送给DefWindowProc。

当Windows准备显示一个弹出式菜单时，它给窗口消息处理程序发送一个WM_INITMENUPOPUP消息，参数如下：

wParam： 弹出式菜单句柄

LOWORD (lParam)：弹出式菜单索引

HIWORD (lParam)： 系统菜单为1，其它为0

如果您需要在显示弹出式菜单之前启用或者禁用菜单项，那么这个消息就很重要。例如，假定程序使用弹出式菜单上的 **Paste** 命令从**剪贴簿**复制文字，当您收到弹出式菜单中的WM_INITMENUPOPUP消息时，应确定剪贴簿内是否有文字存在。如果没有，那么应该使 **Paste** 菜单项无效化。我们将在本章后面修改的POPPAD程序中看到这样的例子。

最重要的菜单消息是WM_COMMAND，它表示使用者已经从菜单中选中了一个被启用的菜单项。第八章中的WM_COMMAND消息也可以由子窗口控件产生。如果您碰巧为菜单和子窗口控件使用同一ID码，那么您可以通过lParam的值来区别它们，菜单项的lParam其值为0，请参见表10-1。

表10-1

	菜单	**控件**
LOWORD (wParam):	菜单ID	控件ID
HIWORD (wParam):	0	通知码
lParam:	0	子窗口句柄

WM_SYSCOMMAND消息类似于WM_COMMAND消息，只是WM_SYSCOMMAND表示使用者从系统菜单中选择一个启用的菜单项：

wParam: 菜单ID

lParam: 0

然而，如果WM_SYSCOMMAND消息是由按鼠标按键产生的，LOWORD（lParam）和HIWORD（lParam）将包含鼠标光标位置的x和y屏幕坐标。

对于WM_SYSCOMMAND，菜单ID指示系统菜单中的哪一项被选中。对于预先定义的系统菜单项，较低的那四个位应该和0xFFF0进行AND运算来屏蔽掉，结果值应该为下列之一：SC_SIZE、SC_MOVE、SC_MINIMIZE、SC_MAXIMIZE、SC_NEXTWINDOW、SC_PREVWINDOW、SC_CLOSE、SC_VSCROLL、SC_HSCROLL、SC_ARRANGE、SC_RESTORE和SC_TASKLIST。此外，wParam可以是SC_MOUSEMENU或SC_KEYMENU。

如果您在系统菜单中添加菜单项，那么wParam的低字组将是您定义的菜单ID。为了避免与预

先定义的菜单ID相冲突，应用程序应该使用小于0xF000的值，这对于将一般的WM_SYSCOMMAND消息发送给DefWindowProc是很重要的。如果您不这样做，那么您实际上就是禁用了正常的系统菜单命令。

我们将讨论的最后一个消息是WM_MENUCHAR。实际上，它根本不是菜单消息。在下列两种情况之一发生时，Windows会把这个消息发送到窗口消息处理程序：如果使用者按下Alt和一个与菜单项不匹配的字符时，或者在显示弹出式菜单而使用者按下一个与弹出式菜单里的项目不匹配的字符键时。随WM_MENUCHAR消息一起发送的参数如下所示：

LOWORD (wParam)：字符代码（ASCII或Unicode）

HIWORD (wParam)：选择码

lParam：菜单句柄

选择码是：

0 不显示弹出式菜单

MF_POPUP 显示弹出式菜单
MF_SYSMENU 显示系统弹出式菜单

Windows程序通常把该消息传递给DefWindowProc，它一般给Windows传回0，这会使Windows发出哔声。在第十四章GRAFMENU程序中会看到WM_MENUCHAR消息的使用。

范例程序

让我们来看一个简单的例子。程序10-4所示的MENUDEMO程序，在主菜单中有五个选择项－File、Edit、Background、Timer和Help，每一项都与一个弹出式菜单相连。MENUDEMO只完成了最简单、最通用的菜单处理操作，包括拦截WM_COMMAND消息和检查wParam的低字组。

程序10-4 MENUDEMO

MENUDEMO.C

```
/*------------------------------------------------------------------------
MENUDEMO.C -- Menu Demonstration
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define ID_TIMER 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName[] = TEXT ("MenuDemo") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
```

```
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Menu Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int idColor [5] = { WHITE_BRUSH, LTGRAY_BRUSH, GRAY_BRUSH,
    DKGRAY_BRUSH, BLACK_BRUSH } ;
  static int iSelection = IDM_BKGND_WHITE ;
  HMENU hMenu ;

  switch (message)
  {
  case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_NEW:
    case IDM_FILE_OPEN:
    case IDM_FILE_SAVE:
    case IDM_FILE_SAVE_AS:
      MessageBeep (0) ;
      return 0 ;
    case IDM_APP_EXIT:
      SendMessage (hwnd, WM_CLOSE, 0, 0) ;
      return 0 ;
    case IDM_EDIT_UNDO:
    case IDM_EDIT_CUT:
    case IDM_EDIT_COPY:
    case IDM_EDIT_PASTE:
    case IDM_EDIT_CLEAR:
      MessageBeep (0) ;
      return 0 ;
    case IDM_BKGND_WHITE: // Note: Logic below
    case IDM_BKGND_LTGRAY: // assumes that IDM_WHITE
    case IDM_BKGND_GRAY: // through IDM_BLACK are
    case IDM_BKGND_DKGRAY: // consecutive numbers in
    case IDM_BKGND_BLACK: // the order shown here.

      CheckMenuItem (hMenu, iSelection, MF_UNCHECKED) ;
      iSelection = LOWORD (wParam) ;
      CheckMenuItem (hMenu, iSelection, MF_CHECKED) ;

      SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
        GetStockObject
        (idColor [LOWORD (wParam) - IDM_BKGND_WHITE])) ;
```

```
     InvalidateRect (hwnd, NULL, TRUE) ;
     return 0 ;
   case IDM_TIMER_START:
     if (SetTimer (hwnd, ID_TIMER, 1000, NULL))
     {
       EnableMenuItem (hMenu, IDM_TIMER_START, MF_GRAYED) ;
       EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_ENABLED) ;
     }
     return 0 ;
   case IDM_TIMER_STOP:
     KillTimer (hwnd, ID_TIMER) ;
     EnableMenuItem (hMenu, IDM_TIMER_START, MF_ENABLED) ;
     EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_GRAYED) ;
     return 0 ;
   case IDM_APP_HELP:
     MessageBox (hwnd, TEXT ("Help not yet
       implemented!"),
       szAppName, MB_ICONEXCLAMATION | MB_OK) ;
     return 0 ;
   case IDM_APP_ABOUT:
     MessageBox (hwnd,TEXT ("Menu Demonstration
       Program\n")
       TEXT ("(c) Charles Petzold, 1998"),
       szAppName, MB_ICONINFORMATION | MB_OK) ;
     return 0 ;
   }
   break ;
   case WM_TIMER:
     MessageBeep (0) ;
     return 0 ;
   case WM_DESTROY:
     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

MENUDEMO.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
MENUDEMO MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&New",  IDM_FILE_NEW
MENUITEM "&Open",  IDM_FILE_OPEN
MENUITEM "&Save",  IDM_FILE_SAVE
MENUITEM "Save &As...", IDM_FILE_SAVE_AS
MENUITEM SEPARATOR
MENUITEM "E&xit",  IDM_APP_EXIT
END
POPUP "&Edit"
BEGIN
MENUITEM "&Undo",  IDM_EDIT_UNDO
MENUITEM SEPARATOR
MENUITEM "C&ut",   IDM_EDIT_CUT
MENUITEM "&Copy",  IDM_EDIT_COPY
MENUITEM "&Paste",  IDM_EDIT_PASTE
MENUITEM "De&lete",  IDM_EDIT_CLEAR
END
POPUP "&Background"
BEGIN
MENUITEM "&White", IDM_BKGND_WHITE, CHECKED
MENUITEM "&Light Gray", IDM_BKGND_LTGRAY
MENUITEM "&Gray", IDM_BKGND_GRAY
```

```
MENUITEM "&Dark Gray", IDM_BKGND_DKGRAY
MENUITEM "&Black", IDM_BKGND_BLACK
END
POPUP "&Timer"
BEGIN
MENUITEM "&Start", IDM_TIMER_START
MENUITEM "S&top", IDM_TIMER_STOP, GRAYED
END
POPUP "&Help"
BEGIN
MENUITEM "&Help...",  IDM_APP_HELP
MENUITEM "&About MenuDemo...", IDM_APP_ABOUT
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by MenuDemo.rc
#define IDM_FILE_NEW  40001
#define IDM_FILE_OPEN  40002
#define IDM_FILE_SAVE  40003
#define IDM_FILE_SAVE_AS  40004
#define IDM_APP_EXIT  40005
#define IDM_EDIT_UNDO  40006
#define IDM_EDIT_CUT  40007
#define IDM_EDIT_COPY  40008
#define IDM_EDIT_PASTE  40009
#define IDM_EDIT_CLEAR  40010
#define IDM_BKGND_WHITE  40011
#define IDM_BKGND_LTGRAY  40012
#define IDM_BKGND_GRAY  40013
#define IDM_BKGND_DKGRAY  40014
#define IDM_BKGND_BLACK  40015
#define IDM_TIMER_START  40016
#define IDM_TIMER_STOP  40017
#define IDM_APP_HELP  40018
#define IDM_APP_ABOUT  40019
```

MENUDEMO.RC资源描述文件给了您定义菜单的提示。菜单的名称为「MenuDemo」。大多数项目有底线字母，这就是说您必须在字母前键入『&』。MENUITEM SEPARATOR叙述是在「Menu Item Properties」对话框中选中「Separator」框产生的。注意菜单中有一个项目具有「 Checked」选项，另一个具有「Grayed」选项。还有，「 Background」弹出式菜单中的五个项目应该按顺序输入，确保标识符是以数值的顺序，本程序需要这样。所有菜单项的标识符定义在RESOURCE.H中。

当收到弹出式菜单「File」和「Edit」各项有关的WM_COMMAND消息时，MENUDEMO程序只使系统发出哔声。「 Background」弹出式菜单列出MENUDEMO用来给背景着色的五种现有画刷。在MENUDEMO.RC资源描述文件中，「 White」菜单项（菜单ID为IDM_BKGND_WHITE）被标以「 CHECKED」，它在菜单项旁边设定选中标记。在MENUDEMO.C中，iSelection的值被初始化为IDM_BKGND_WHITE。

「Background」弹出式菜单上的五种画刷相互排斥。当MENUDEMO.C收到一个WM_COMMAND消息，而该消息中的wParam是「 Background」弹出式菜单上的五项之一时，它必须从先前选中的背景颜色中除掉选中标记，并把标记加到新的背景颜色上。为此，首先要得到菜单句柄：

hMenu = GetMenu (hwnd) ;

CheckMenuItem函数用来取消目前被选中的项目：

CheckMenuItem (hMenu, iSelection, MF_UNCHECKED) ;

iSelection的值被设定为wParam的值，新的背景颜色被选中：

iSelection = wParam ;

CheckMenuItem (hMenu, iSelection, MF_CHECKED) ;

窗口类别中的背景颜色于是被替换为新的背景颜色，窗口显示区域变为无效状态，Windows使用新的背景颜色清除窗口。

Timer弹出式菜单列出了两个选项－「Start」和「Stop」。开始时，「Stop」选项变为灰色的（就像在资源描述文件中的菜单定义一样）。当您选择「Start」选项时，MENUDEMO试图启动一个定时器，如果成功，则无效化「Start」选项，并启用「Stop」选项：

EnableMenuItem (hMenu, IDM_TIMER_START, MF_GRAYED) ;

EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_ENABLED) ;

当收到一条WM_COMMAND消息，并且wParam等于IDM_TIMER_STOP时，MENUDEMO程序会停止计数，启用「Start」项，然后无效化「Stop」选项：

EnableMenuItem (hMenu, IDM_TIMER_START, MF_ENABLED) ;

EnableMenuItem (hMenu, IDM_TIMER_STOP, MF_GRAYED) ;

请注意，在定时器执行时，MENUDEMO程序不可能收到wParam等于IDM_TIMER_START的WM_COMMAND消息。同样地，在定时器关闭时，MENUDEMO程序也不可能收到wParam等于IDM_TIMER_STOP的WM_COMMAND消息。

当MENUDEMO收到一个WM_COMMAND消息，而该消息的参数wParam等于IDM_APP_ABOUT或IDM_APP_HELP时，MENUDEMO程序显示一个消息框（在下一章中，我们将把消息框变为对话框）。

当MENUDEMO程序收到一个WM_COMMAND消息，其参数wParam等于IDM_APP_EXIT时，它给自己发送一个WM_CLOSE消息。这个消息与DefWindowProc收到WM_SYSCOMMAND消息且wParam等于SC_CLOSE时发送给窗口消息处理程序的消息相同。我们将在本章后面介绍POPPAD2时再仔细研究这个问题。

菜单设计规范

在MENUDEMO中的「File」和「Edit」弹出式菜单的格式与其它Windows程序中的格式非常类似。Windows的目的之一是为使用者提供一种易懂的接口，而不要求使用者为每个程序重新学习基本操作方式。如果「File」和「Edit」菜单在每个Windows程序中看起来都一样，并且都使用同样的字母和Alt键来进行选择，那么当然有助于减轻使用者的学习负担。

除了「File」和「Edit」弹出式菜单外，大多数Windows程序的菜单都是不同的。当设计一个菜单时，您应该看一看现有的Windows程序以尽量保持一致。当然，如果您认为别的程序是不对的，而您知道正确的方法，那么没有人能够阻止您。同时记住，修改一个菜单，通常只需要修改资源描述文件而不必修改您的程序代码。即使以后要改变菜单项的位置，也不会有多大的问题。

虽然您的程序菜单在顶层可以有MENUITEM叙述，但这是不合规范的，因为这样会很容易导致错误的选择。如果您要这样做，那么请在字符串后面加一个惊叹号，表示菜单项不会启动弹出式菜单。

较难的一种菜单定义方法

在程序的资源描述文件中定义菜单，通常是在您的窗口中添加菜单的最简单方法，但不是唯一的方法。如果您没有使用资源描述文件，那么可以使用CreateMenu和AppendMenu两个函数在程序中建立菜单。在您定义完菜单后，您可以将菜单句柄发送给CreateWindow，或者使用SetMenu来设定窗口的菜单。

以下是具体的做法。CreateMenu简单地把一个句柄传回给新菜单：

hMenu = CreateMenu () ;

菜单一开始为空。AppendMenu将菜单项插入菜单中。您必须为顶层菜单项和每一个弹出式菜单提供不同的菜单句柄。弹出式菜单是单独构成的，然后将弹出式菜单句柄插入顶层菜单。程序10-5中所示的程序代码就是用这种方法建立菜单的，实际上，这个菜单与MENUDEMO程序中的菜单相同。为了简化说明，代码使用ASCII字符串。

程序10-5　不使用资源描述文件建立与MENUDEMO程序相同菜单的C程序代码

```
hMenu = CreateMenu () ;
hMenuPopup = CreateMenu () ;
AppendMenu (hMenuPopup, MF_STRING, IDM_FILE_NEW, "&New");
AppendMenu (hMenuPopup, MF_STRING, IDM_FILE_OPEN, "&Open...");
AppendMenu (hMenuPopup, MF_STRING, IDM_FILE_SAVE, "&Save");
AppendMenu (hMenuPopup, MF_STRING, IDM_FILE_SAVE_AS, "Save &As...");
AppendMenu (hMenuPopup, MF_SEPARATOR, 0, NULL) ;
AppendMenu (hMenuPopup, MF_STRING, IDM_APP_EXIT, "E&xit") ;
AppendMenu (hMenu, MF_POPUP, hMenuPopup, "&File") ;

hMenuPopup = CreateMenu () ;
AppendMenu (hMenuPopup, MF_STRING, IDM_EDIT_UNDO,"&Undo") ;
AppendMenu (hMenuPopup, MF_SEPARATOR, 0, NULL) ;
AppendMenu (hMenuPopup, MF_STRING,IDM_EDIT_CUT, "Cu&t") ;
AppendMenu (hMenuPopup, MF_STRING,IDM_EDIT_COPY,"&Copy") ;
AppendMenu (hMenuPopup, MF_STRING,IDM_EDIT_PASTE,"&Paste") ;
AppendMenu (hMenuPopup, MF_STRING,IDM_EDIT_CLEAR,"De&lete") ;
AppendMenu (hMenu, MF_POPUP, hMenuPopup, "&Edit") ;

hMenuPopup = CreateMenu () ;
AppendMenu (hMenuPopup, MF_STRING| MF_CHECKED, IDM_BKGND_WHITE, "&White");
AppendMenu (hMenuPopup, MF_STRING, IDM_BKGND_LTGRAY, "&Light Gray");
AppendMenu (hMenuPopup, MF_STRING, IDM_BKGND_GRAY, "&Gray") ;
AppendMenu (hMenuPopup, MF_STRING, IDM_BKGND_DKGRAY, "&Dark Gray");
AppendMenu (hMenuPopup, MF_STRING, IDM_BKGND_BLACK, "&Black") ;

AppendMenu (hMenu, MF_POPUP, hMenuPopup, "&Background") ;
hMenuPopup = CreateMenu () ;
AppendMenu (hMenuPopup, MF_STRING, IDM_TIMER_START, "&Start") ;
AppendMenu (hMenuPopup, MF_STRING | MF_GRAYED, IDM_TIMER_STOP, "S&top") ;

AppendMenu (hMenu, MF_POPUP, hMenuPopup, "&Timer") ;

hMenuPopup = CreateMenu () ;

AppendMenu (hMenuPopup, MF_STRING, IDM_HELP_HELP, "&Help") ;
AppendMenu (hMenuPopup, MF_STRING, IDM_APP_ABOUT, "&About MenuDemo...") ;

AppendMenu (hMenu, MF_POPUP, hMenuPopup, "&Help") ;
```

我认为您会同意底下这个观点：使用资源描述文件菜单模板来制作菜单，会更容易而且更清楚。我并不鼓励您使用这里的方法定义菜单，而只是提供了一种实作菜单的方法。当然，您可以使用包含所有菜单项字符串、ID和旗标等的结构数组来压缩程序代码大小。不过，如果您这么做了，那么您还可以利用Windows定义菜单的第三种方法。LoadMenuIndirect函数接受一个指向MENUITEMTEMPLATE型态的结构指针，并传回菜单的句柄，该函数在载入资源描述文件中的常规菜单模板后，在Windows中构造菜单，读者不妨自己尝试一下。

浮动弹出式菜单

您还可以在没有顶层菜单列的情况下使用菜单，也就是说，您可以使弹出式菜单出现在屏幕顶层的任何位置。一种方法是使用鼠标右键来启动弹出式菜单。程序10-6所示的POPMENU说明了这种方法。

程序10-6 POPMENU

POPMENU.C

```
/*---------------------------------------------------------------------
POPMENU.C -- Popup Menu Demonstration
(c) Charles Petzold, 1998
---------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HINSTANCE hInst ;
TCHAR szAppName[] = TEXT ("PopMenu") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hInst = hInstance ;
  hwnd = CreateWindow ( szAppName, TEXT ("Popup Menu Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HMENU hMenu ;
  static int idColor [5] = {WHITE_BRUSH, LTGRAY_BRUSH, GRAY_BRUSH,
    DKGRAY_BRUSH, BLACK_BRUSH } ;
```

```
     static int iSelection = IDM_BKGND_WHITE ;
     POINT point ;

     switch (message)
     {
     case WM_CREATE:
       hMenu = LoadMenu (hInst, szAppName) ;
       hMenu = GetSubMenu (hMenu, 0) ;
       return 0 ;
     case WM_RBUTTONUP:
       point.x = LOWORD (lParam) ;
       point.y = HIWORD (lParam) ;
       ClientToScreen (hwnd, &point) ;

       TrackPopupMenu (hMenu, TPM_RIGHTBUTTON, point.x, point.y, 0, hwnd, NULL) ;
       return 0 ;
     case WM_COMMAND:
       switch (LOWORD (wParam))
       {
       case IDM_FILE_NEW:
       case IDM_FILE_OPEN:
       case IDM_FILE_SAVE:
       case IDM_FILE_SAVE_AS:
       case IDM_EDIT_UNDO:
       case IDM_EDIT_CUT:
       case IDM_EDIT_COPY:
       case IDM_EDIT_PASTE:
       case IDM_EDIT_CLEAR:
         MessageBeep (0) ;
         return 0 ;
       case IDM_BKGND_WHITE: // Note: Logic below
       case IDM_BKGND_LTGRAY: // assumes that IDM_WHITE
       case IDM_BKGND_GRAY: // through IDM_BLACK are
       case IDM_BKGND_DKGRAY: // consecutive numbers in
       case IDM_BKGND_BLACK: // the order shown here.

         CheckMenuItem (hMenu, iSelection, MF_UNCHECKED) ;
         iSelection = LOWORD (wParam) ;
         CheckMenuItem (hMenu, iSelection, MF_CHECKED) ;

         SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG)
           GetStockObject
           (idColor [LOWORD (wParam) - IDM_BKGND_WHITE])) ;

         InvalidateRect (hwnd, NULL, TRUE) ;
         return 0 ;
       case IDM_APP_ABOUT:
         MessageBox (hwnd, TEXT ("Popup Menu Demonstration Program\n")
           TEXT ("(c) Charles Petzold, 1998"),
           szAppName, MB_ICONINFORMATION | MB_OK) ;
         return 0 ;
       case IDM_APP_EXIT:
         SendMessage (hwnd, WM_CLOSE, 0, 0) ;
         return 0 ;
       case IDM_APP_HELP:
         MessageBox (hwnd, TEXT ("Help not yet implemented!"),
           szAppName, MB_ICONEXCLAMATION | MB_OK) ;
         return 0 ;
       }
       break ;

       case WM_DESTROY:
         PostQuitMessage (0) ;
         return 0 ;
     }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

POPMENU.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
POPMENU MENU DISCARDABLE
BEGIN
POPUP "MyMenu"
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&New", IDM_FILE_NEW
MENUITEM "&Open", IDM_FILE_OPEN
MENUITEM "&Save", IDM_FILE_SAVE
MENUITEM "Save &As", IDM_FILE_SAVE_AS
MENUITEM SEPARATOR
MENUITEM "E&xit", IDM_APP_EXIT
END
POPUP "&Edit"
BEGIN
MENUITEM "&Undo", IDM_EDIT_UNDO
MENUITEM SEPARATOR
MENUITEM "Cu&t", IDM_EDIT_CUT
MENUITEM "&Copy", IDM_EDIT_COPY
MENUITEM "&Paste", IDM_EDIT_PASTE
MENUITEM "De&lete", IDM_EDIT_CLEAR
END
POPUP "&Background"
BEGIN
MENUITEM "&White", IDM_BKGND_WHITE, CHECKED
MENUITEM "&Light Gray", IDM_BKGND_LTGRAY
MENUITEM "&Gray", IDM_BKGND_GRAY
MENUITEM "&Dark Gray", IDM_BKGND_DKGRAY
MENUITEM "&Black", IDM_BKGND_BLACK
END
POPUP "&Help"
BEGIN
MENUITEM "&Help...", IDM_APP_HELP
MENUITEM "&About PopMenu...", IDM_APP_ABOUT
END
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by PopMenu.rc
#define IDM_FILE_NEW   40001
#define IDM_FILE_OPEN   40002
#define IDM_FILE_SAVE   40003
#define IDM_FILE_SAVE_AS  40004
#define IDM_APP_EXIT   40005
#define IDM_EDIT_UNDO   40006
#define IDM_EDIT_CUT  40007
#define IDM_EDIT_COPY   40008
#define IDM_EDIT_PASTE  40009
#define IDM_EDIT_CLEAR  40010
#define IDM_BKGND_WHITE  40011
#define IDM_BKGND_LTGRAY  40012
#define IDM_BKGND_GRAY  40013
#define IDM_BKGND_DKGRAY  40014
#define IDM_BKGND_BLACK  40015
#define IDM_APP_HELP   40016
#define IDM_APP_ABOUT   40017
```

资源描述文件POPMENU.RC定义的菜单与MENUDEMO.RC中的菜单非常相似。不同的是，在顶层菜单中只包含一项 – 一个弹出式菜单「MyMenu」，它呼叫「File」、「Edit」、「Background」

和「Help」选项。这四个选项垂直一行地出现在弹出式菜单上，而不是水平一列地出现在主菜单上。

在WndProc中的WM_CREATE处理期间，POPMENU取得此弹出式菜单的句柄，就是带有文字「MyMenu」的那个弹出式菜单：

hMenu = LoadMenu (hInst, szAppName) ;

hMenu = GetSubMenu (hMenu, 0) ;

在WM_RBUTTONUP消息处理期间，POPMENU提供了鼠标指针的位置，将此位置转换为屏幕坐标，再将坐标值传递给TrackPopupMenu：

```
point.x = LOWORD (lParam) ;
point.y = HIWORD (lParam) ;
ClientToScreen (hwnd, &point) ;
TrackPopupMenu (hMenu, TPM_RIGHTBUTTON, point.x, point.y,
                0, hwnd, NULL) ;
```

然后，Windows显示出具有「File」、「Edit」、「Background」和「Help」项的弹出式菜单。选择其中任何一项都可以使嵌套的弹出式菜单显示在右边，菜单函数与一般的菜单一样。

如果要使用与该程序的主菜单相同的菜单并带有TrackPopupMenu，您会遇到一些问题，因为函数需要弹出式菜单句柄。在「Microsoft Knowledge Base」文章ID Q99806有提供一些信息。

使用系统菜单

使用WS_SYSMENU样式建立的父窗口，在其标题栏的左侧有一个系统菜单按钮。如果您愿意，可以修改这个菜单。在Windows程序设计的早期，程序写作者一般把「About」菜单项放入系统菜单。虽然这种方法不常见，但是修改系统菜单往往是一种在短程序中添加菜单的快速偷懒方法。这里唯一的限制是：在系统菜单中增加的命令其ID值必须小于0xF000；否则它们将会与Windows系统菜单命令所使用的ID值相冲突。还要记住，当您为这些新菜单项在窗口消息处理程序中处理WM_SYSCOMMAND 消息时，您必须把其它的WM_SYSCOMMAND 消息发送给DefWindowProc。如果您不这样做，那么实际上是禁用了系统菜单上的所有正常选项。

程序10-7中所示的POORMENU（「设计不当的个人菜单」）在系统菜单中加入了一个分隔条和三个命令，最后一个命令将删除这些附加的菜单项。

程序10-7　POORMENU

POORMENU.C

```
/*-----------------------------------------------------------------------
POORMENU.C -- The Poor Person's Menu
(c) Charles Petzold, 1998
-----------------------------------------------------------------------*/
#include <windows.h>
#define IDM_SYS_ABOUT 1
#define IDM_SYS_HELP 2
#define IDM_SYS_REMOVE 3

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
static TCHAR szAppName[] = TEXT ("PoorMenu") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HMENU hMenu ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("The Poor-Person's Menu"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  hMenu = GetSystemMenu (hwnd, FALSE) ;
  AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
  AppendMenu (hMenu, MF_STRING, IDM_SYS_ABOUT, TEXT ("About...")) ;
  AppendMenu (hMenu, MF_STRING, IDM_SYS_HELP, TEXT ("Help...")) ;
  AppendMenu (hMenu, MF_STRING, IDM_SYS_REMOVE, TEXT ("Remove Additions")) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  switch (message)
  {
  case WM_SYSCOMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_SYS_ABOUT:
      MessageBox ( hwnd, TEXT ("A Poor-Person's Menu Program\n")
        TEXT ("(c) Charles Petzold, 1998"),
        szAppName, MB_OK | MB_ICONINFORMATION) ;
      return 0 ;
    case IDM_SYS_HELP:
      MessageBox ( hwnd, TEXT ("Help not yet implemented!"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
      return 0 ;
    case IDM_SYS_REMOVE:
      GetSystemMenu (hwnd, TRUE) ;
      return 0 ;
    }
    break ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

三个菜单ID在POORMENU.C的开始部分定义：

```
#define IDM_ABOUT   1
#define IDM_HELP   2
```

```
#define IDM_REMOVE   3
```

在程序窗口建立之后，POORMENU得到一个系统菜单的句柄：

hMenu = GetSystemMenu (hwnd, FALSE) ;

第一次呼叫GetSystemMenu时，您应该为修改菜单作准备，将第二个参数设定为FALSE。

使用四个AppendMenu呼叫来实作对菜单的修改：

```
AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
AppendMenu (hMenu, MF_STRING, IDM_SYS_ABOUT, TEXT ("About...")) ;
AppendMenu (hMenu, MF_STRING, IDM_SYS_HELP, TEXT ("Help...")) ;
AppendMenu (hMenu, MF_STRING, IDM_SYS_REMOVE, TEXT ("Remove Additions"));
```

第一个AppendMenu呼叫是添加分隔条。选择「Remove Additions」菜单项将使POORMENU删除这些附加的菜单项，这只要把第二个参数设定为TRUE，再次呼叫GetSystemMenu即可：

GetSystemMenu (hwnd, TRUE) ;

标准系统菜单有下列选项：Restore、Move、Size、Minimize、Maximize和Close。它们产生wParam分别等于SC_RESTORE、SC_MOVE、SC_SIZE、SC_MINIMUM、SC_MAXIMUM和SC_CLOSE的WM_SYSCOMMAND消息。尽管Windows程序一般不这样做，但是您可以自己处理这些消息，而不把它们留给DefWindowProc。您也可以使用下面所述的方法来禁止或者除掉系统菜单的标准选项。Windows文件中还介绍了一些系统菜单的标准附加项目，这些附加项目使用标识符SC_NEXTWINDOW、SC_PREVWINDOW、SC_VSCROLL、SC_HSCROLL和SC_ARRANGE。您也许会发现，在一些应用程序中将这些命令加入系统菜单是合适的。

改变菜单

我们已经看到了如何使用AppendMenu函数为程序定义菜单以及将菜单项加入到系统菜单中。在Windows 3.0之前，您不得不被迫使用ChangeMenu函数来完成这种工作。ChangeMenu函数有很多功能，至少在当时，整个Windows中它是最复杂的函数之一。现在，许多函数都比ChangeMenu函数还要复杂，并且ChangeMenu的功能被分解为五个新的函数：

AppendMenu在菜单尾部添加一个新的菜单项目

DeleteMenu删除菜单中一个现有的菜单项并清除该项目
InsertMenu在菜单中插入一个新项目
ModifyMenu修改一个现有的菜单项目
RemoveMenu从菜单中移走某一项目

如果菜单项是一个弹出式菜单，那么DeleteMenu和RemoveMenu之间的区别就很重要。DeleteMenu清除弹出式菜单，但RemoveMenu不清除它。

其它菜单命令

下面是在使用菜单时一些有用的函数。

当您改变顶层菜单项时，直到Windows重画菜单列时才显示所做的改变。您可以通过下列呼叫来强迫执行菜单更新：

DrawMenuBar (hwnd) ;

注意，DrawMenuBar的参数是窗口句柄而不是菜单句柄。

您可以使用下列命令来获得弹出式菜单的句柄：

hMenuPopup = GetSubMenu (hMenu, iPosition) ;

其中iPosition是hMenu指示的顶层菜单中弹出式菜单项的索引（开始为0）。然后您可以在其它函数中使用弹出式菜单句柄（例如在AppendMenu函数中）。

您可以使用下列命令获得顶层菜单或者弹出式菜单中目前的项数：

iCount = GetMenuItemCount (hMenu) ;

您可以取得弹出式菜单项的菜单ID：

id = GetMenuItemID (hMenuPopup, iPosition) ;

其中iPosition是菜单项在弹出式菜单中的位置（以0开始）。

在MENUDEMO中您已经看到如何选中、或者取消选中弹出式菜单中的某一项：

CheckMenuItem (hMenu, id, iCheck) ;

在MENUDEMO中，hMenu是顶层菜单的句柄，id是菜单ID，而iCheck的值是MF_CHECKED或MF_UNCHECKED。如果hMenu是弹出式菜单句柄，那么参数id是位置索引而不是菜单ID。如果使用索引会更方便的话，那么您可以在第三个参数中包含MF_BYPOSITION，例如：

CheckMenuItem (hMenu, iPosition, MF_CHECKED | MF_BYPOSITION) ;

除了第三个参数是MF_ENABLED、MF_DISABLED或MF_GRAYED外，EnableMenuItem函数与CheckMenuItem函数所完成的工作类似。如果您在具有弹出式菜单的顶层菜单项上使用EnableMenuItem，那么必须在第三个参数中使用MF_BYPOSITION标识符，因为菜单项没有菜单ID。我们将在本章后面所示的POPPAD2程序中看到EnableMenuItem的一个例子。HiliteMenuItem也类似于CheckMenuItem和EnableMenuItem，但是它使用的是MF_HILITE和MF_UNHILITE。当您在菜单项之间移动时，Windows使用反白显示方式加亮显示菜单项。您通常不需要使用HiliteMenuItem。

您还需要对您的菜单做些什么呢？还记得我们在菜单中使用了哪些字符串吗？您可以透过下面的呼叫来回顾一下：

iCharCount = GetMenuString (hMenu, id, pString, iMaxCount, iFlag) ;

iFlag可以是MF_BYCOMMAND（其中id是菜单ID），也可以是MF_BYPOSITION（其中的id是位置索引）。函数将字符串的iMaxCount个字节复制到pString中，并传回复制的字节数。

或许您也想知道菜单项目前的属性是什么：

iFlags = GetMenuState (hMenu, id, iFlag) ;

同样地，iFlag可以是MF_BYCOMMAND或MF_BYPOSITION。传回值iFlags是目前所有属性的组合，您可以通过对MF_DISABLED、MF_GRAYED、MF_CHECKED、MF_MENUBREAK、MF_MENUBARBREAK和MF_SEPARATOR标识符的检测来决定目前的属性。

也许现在您对菜单有了一些了解。这时您可能想知道，如果您不再需要菜单时又应该如何处理。您可以使用下面的命令来清除菜单：

DestroyMenu (hMenu) ;

从而使菜单句柄无效。

建立菜单的非正统方法

现在让我们稍微偏离我们所讨论的主题。如果在您的程序中没有下拉式菜单，而是建立了多个没有弹出式菜单的顶层菜单，并呼叫SetMenu在顶层菜单之间切换，那会是什么样的情形呢？就像Lotus 1-2-3中老式的文字模式菜单那样。程序10-8中的NOPOPUPS程序展示了处理这种情况。在

这个程序中，「File」和「Edit」项与MENUDEMO程序中的类似，但是却以另一种顶层菜单显示出来。

程序10-8 NOPOPUPS

NOPOPUPS.C

```
/*---------------------------------------------------------------------------
NOPOPUPS.C -- Demonstrates No-Popup Nested Menu
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("NoPopUps") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName,
    TEXT ("No-Popup Nested Menu Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HMENU hMenuMain, hMenuEdit, hMenuFile ;
  HINSTANCE hInstance ;
  switch (message)
  {
  case WM_CREATE:
    hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;

    hMenuMain = LoadMenu (hInstance, TEXT ("MenuMain")) ;
    hMenuFile = LoadMenu (hInstance, TEXT ("MenuFile")) ;
```

```
    hMenuEdit = LoadMenu (hInstance, TEXT ("MenuEdit")) ;

    SetMenu (hwnd, hMenuMain) ;
    return 0 ;
  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_MAIN:
      SetMenu (hwnd, hMenuMain) ;
      return 0 ;
    case IDM_FILE:
      SetMenu (hwnd, hMenuFile) ;
      return 0 ;
    case IDM_EDIT:
      SetMenu (hwnd, hMenuEdit) ;
      return 0 ;
    case IDM_FILE_NEW:
    case IDM_FILE_OPEN:
    case IDM_FILE_SAVE:
    case IDM_FILE_SAVE_AS:
    case IDM_EDIT_UNDO:
    case IDM_EDIT_CUT:
    case IDM_EDIT_COPY:
    case IDM_EDIT_PASTE:
    case IDM_EDIT_CLEAR:
      MessageBeep (0) ;
      return 0 ;
    }
    break ;
    case WM_DESTROY:
      SetMenu (hwnd, hMenuMain) ;
      DestroyMenu (hMenuFile) ;
      DestroyMenu (hMenuEdit) ;

      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

NOPOPUPS.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
MENUMAIN MENU DISCARDABLE
BEGIN
MENUITEM "MAIN:", 0, INACTIVE
MENUITEM "&File...", IDM_FILE
MENUITEM "&Edit...", IDM_EDIT
END
MENUFILE MENU DISCARDABLE
BEGIN
MENUITEM "FILE:", 0, INACTIVE
MENUITEM "&New", IDM_FILE_NEW
MENUITEM "&Open...", IDM_FILE_OPEN
MENUITEM "&Save", IDM_FILE_SAVE
MENUITEM "Save &As", IDM_FILE_SAVE_AS
MENUITEM "(&Main)", IDM_MAIN
END
MENUEDIT MENU DISCARDABLE
BEGIN
MENUITEM "EDIT:", 0, INACTIVE
MENUITEM "&Undo", IDM_EDIT_UNDO
MENUITEM "Cu&t", IDM_EDIT_CUT
MENUITEM "&Copy", IDM_EDIT_COPY
```

```
MENUITEM "&Paste", IDM_EDIT_PASTE
MENUITEM "De&lete", IDM_EDIT_CLEAR
MENUITEM "(&Main)", IDM_MAIN
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by NoPopups.rc
#define IDM_FILE   40001
#define IDM_EDIT   40002
#define IDM_FILE_NEW  40003
#define IDM_FILE_OPEN  40004
#define IDM_FILE_SAVE  40005
#define IDM_FILE_SAVE_AS  40006
#define IDM_MAIN   40007
#define IDM_EDIT_UNDO  40008
#define IDM_EDIT_CUT  40009
#define IDM_EDIT_COPY  40010
#define IDM_EDIT_PASTE  40011
#define IDM_EDIT_CLEAR  40012
```

在Microsoft Developer Studio中，您建立了三个菜单，而不是一个。从「Insert」中选择「Resource」三次，每个菜单有一个不同的名称。当窗口消息处理程序处理WM_CREATE消息时，Windows将每个菜单资源加载内存：

```
hMenuMain = LoadMenu (hInstance, TEXT ("MenuMain")) ;
hMenuFile = LoadMenu (hInstance, TEXT ("MenuFile")) ;
hMenuEdit = LoadMenu (hInstance, TEXT ("MenuEdit")) ;
```

开始时，程序只显示主菜单：

SetMenu (hwnd, hMenuMain) ;

主菜单使用字符串「MAIN:」、「File...」和「Edit...」列出这三个选项。然而，「MAIN:」是禁用的，因此它不能使WM_COMMAND消息被发送到窗口消息处理程序。「File」和「Edit」菜单项以「FILE:」和「EDIT:」开始，表示它们是子菜单。每个菜单的最后一项都是字符串「(Main)」，表示传回到主菜单。在这三个菜单之间进行切换是很简单的：

```
case WM_COMMAND :
  switch (wParam)
  {
  case IDM_MAIN :
    SetMenu (hwnd, hMenuMain) ;
    return 0 ;
  case IDM_FILE :
    SetMenu (hwnd, hMenuFile) ;
    return 0 ;
  case IDM_EDIT :
    SetMenu (hwnd, hMenuEdit) ;
    return 0 ;

    //其它行程序
  }
  break ;
```

在WM_DESTROY消息处理期间，NOPOPUPS将程序的菜单设定为主菜单，并呼叫DestroyMenu来清除「File」和「Edit」菜单。当窗口被清除时，主菜单将被自动清除。

键盘快捷键

快捷键是产生WM_COMMAND消息（有些情况下是WM_SYSCOMMAND）的键组合。许多时候，程序使用快捷键来重复常用菜单项的动作（然而，快捷键还可以用于执行非菜单功能）。例

如，许多Windows程序都有一个包含「Delete」或「Clear」选项的「Edit」菜单，这些程序习惯上都将Del键指定为该选项的快捷键。使用者可以通过「 Alt键」从菜单中选择「Delete」选项，或者只需按下快捷键 Del。当窗口消息处理程序收到一个WM_COMMAND消息时，它不必确定使用的是菜单还是快捷键。

为什么要使用快捷键

您也许会问：为什么我应该使用快捷键?为什么不能直接拦截WM_KEYDOWN或WM_CHAR消息而自己实作同样的菜单功能呢？好处又在哪里呢？对于一个单窗口应用程序，您当然可以拦截键盘消息，但是使用快捷键可以得到一些好处：您不需要把菜单和快捷键的处理方式重写一遍。

对于有多个窗口和多个窗口消息处理程序的应用程序来说，快捷键是非常重要的。正如我们所看到的，Windows将键盘消息发送给目前活动窗口的窗口消息处理程序。然而对于快捷键，Windows把WM_COMMAND消息发送给窗口消息处理程序，该窗口消息处理程序的句柄在Windows函数TranslateAccelerator中给出。通常这是主窗口，也是拥有菜单的窗口，这意味着无须每个窗口消息处理程序都把快捷键的操作处理程序重写一遍。

如果您在主窗口的显示区域中，使用了非系统模态对话框（在下一章中会讨论）或者子窗口，那么这种好处就变得非常重要。如果定义一个特定的快捷键以便在不同的窗口之间移动，那么，只需要一个窗口消息处理程序有这个处理程序。子窗口就不会收到快捷键引发的WM_COMMAND消息。

安排快捷键的几条规则

理论上，您可以使用任何虚拟键或者字符键连同Shift键、Ctrl键或Alt键来定义快捷键。然而，您应该尽力使应用程序之间协调一致，并且尽量避免干扰Windows的键盘使用。在快捷键中，应该避免使用Tab、Enter、Esc和Spacebar键，因为这些键常常用于完成系统功能。

快捷键最经常的用途是操作程序的「Edit」菜单中的各项。为这些菜单项推荐的快捷键在Windows 3.0和Windows 3.1之间已有不同，因此通常都要支持如下所列的新旧两套快捷键：

表10-2

功能	旧快捷键	新快捷键
Undo	Alt+Backspace	Ctrl+Z
Cut	Shift+Del	Ctrl+X
Copy	Ctrl+Ins	Ctrl+C
Paste	Shift+Ins	Ctrl+V
Delete或Clear	Del	Del

另一种常用的虚拟键是启动辅助信息的功能键F1。应该避免使用F4、F5和F6键，因为这些键常用在多重文件接口（MDI）程序中来完成特殊的功能（将在第十九章中讨论）。

快捷键表

您可以在Developer Studio中定义快捷键表。为了让程序中加载加速键表更为容易，给它和程序名相同的名称（与菜单和图标名也相同）。

每个快捷键都有在Accel Properties对话框中定义的ID和按键组合。如果您已经定义了菜单，则菜单ID会出现在下拉式清单方块中，因此不需要键入它们。

快捷键可以是虚拟键或ASCII字符与Shift、Ctrl或Alt键的组合。可以通过在字母前键入『^』来指定带有Ctrl键的ASCII字符。也可以从下拉式清单方块中选取虚拟键。

当您为菜单项定义快捷键时，应该将键的组合包含到菜单项的文字中。制表符（\t）将文字与快捷键分割开，将快捷键列在第二列。为了在菜单中为快捷键做上标记，可以在文字「Ctrl」、「Shift」或「Alt」之后跟上一个「+」号和一个键名（例如，「Shift+F6」或「Ctrl+F6」）。

快捷键表的加载

在您的程序中，您使用LoadAccelerators函数把快捷键表加载内存，并获得该表的句柄。LoadAccelerators叙述非常类似于LoadIcon、LoadCursor和LoadMenu叙述。

首先，把快捷键表的句柄定义为型态HANDLE：

HANDLE hAccel ;

然后加载加速键表：

hAccel = LoadAccelerators (hInstance, TEXT ("MyAccelerators")) ;

正如图标、光标和菜单一样，您可以使用一个数值代替快捷键表的名称，然后在LoadAccelerators叙述中和MAKEINTRESOURCE宏一起使用该数值，或者把它放在双引号内，前面冠以字符「#」。

键盘代码转换

现在我们将讨论底下这三行程序代码，在本书中，截至目前为止建立的所有Windows程序中都使用过它们。这些程序代码是标准的消息循环：

```
while (GetMessage (&msg, NULL, 0, 0))
{
  TranslateMessage (&msg) ;
  DispatchMessage (&msg) ;
}
```

下面把上头那段程序代码加以修改，以便使用加速键：

```
while (GetMessage (&msg, NULL, 0, 0))
{
  if (!TranslateAccelerator (hwnd, hAccel, &msg))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
}
```

TranslateAccelerator函数确认存放在msg消息结构中的消息是否为键盘消息。如果是，该函数将找寻句柄为hAccel的快捷键表。如果找到了一个符合的，则呼叫句柄为hwnd的窗口消息处理程序。如果快捷键ID与系统菜单的菜单项一致，则消息就是WM_SYSCOMMAND；否则，消息为WM_COMMAND。

当TranslateAccelerator传回时，如果消息已经被转换（并且已经被发送给窗口消息处理程序），那么传回值为非零；否则，传回值为0。如果TranslateAccelerator传回一个非零值，则不呼叫TranslateMessage和DispatchMessage，而是经过循环回到GetMessage呼叫中。

TranslateMessage中的参数hwnd看起来有点累赘，因为消息循环中的其它三个函数都没有要求这个参数。此外，消息结构本身（结构变量msg）有一个叫做hwnd的成员，它是窗口句柄。

该函数有些不同的原因在于：msg结构的字段由GetMessage呼叫填入。当GetMessage的第二个参数为NULL时，函数会找寻应用程序所有窗口的消息。当GetMessage传回时，msg结构的hwnd是将要获得消息之窗口的窗口句柄。然而，当TranslateAccelerator把键盘消息转换为WM_COMMAND或WM_SYSCOMMAND消息时，它使用函数的第一个参数指定的窗口句柄hwnd来代替窗口代号msg.hwnd。Windows就是这样把所有快捷键消息发送给同一窗口消息处理

程序的，即使另一个应用窗口目前拥有输入焦点。当系统模态对话框或者消息框拥有输入焦点时，TranslateAccelerator不会转换键盘消息，因为这些窗口的消息是不经过程序的消息循环的。

在某些情况下，当您程序的另一个窗口（比如一个非系统模态对话框）拥有输入焦点时，您也许不想转换快捷键。您将在下一章中看到如何处理这种情况。

接收快捷键消息

当快捷键与系统菜单中的菜单项相对应时，TranslateAccelerator给窗口消息处理程序发送一个WM_SYSCOMMAND消息，否则，TranslateAccelerator给窗口消息处理程序发送一个WM_COMMAND消息。下表所示为几种可能接收到的WM_COMMAND消息，这些消息用于快捷键、菜单命令以及子窗口控件：

表10-3

	快捷键	菜单	控件
LOWORD (wParam)	快捷键ID	菜单ID	控件ID
HIWORD (wParam)	1	0	通知码
lParam	0	0	子窗口句柄

如果快捷键与一个菜单项对应，那么窗口消息处理程序还会收到WM_INITMENU、WM_INITMENUPOPUP和WM_MENUSELECT消息，就好像选中了菜单选项一样。在处理WM_INITMENUPOPUP时，程序往往启用和禁用弹出式菜单中的菜单项，因此，在使用快捷键时，您仍然能够实作这类功能。如果快捷键与一个禁用或者无效化的菜单项相对应，那么，TranslateAccelerator函数就不会向窗口消息处理程序发送WM_COMMAND或WM_SYSCOMMAND消息。

如果活动窗口已经被最小化，那么TranslateAccelerator将为与启用的系统菜单项相对应的快捷键向窗口消息处理程序发送WM_SYSCOMMAND消息，而不是WM_COMMAND消息。TranslateAccelerator也会为没有任何菜单项与之对应的快捷键，来向窗口消息处理程序发送WM_COMMAND消息。

菜单与快捷键应用程序POPPAD

在第九章，我们建立了一个叫做POPPAD1的程序，它使用了子窗口编辑控件来实作基本的笔记本功能。在这一章中，我们将加入「File」和「Edit」菜单，并称此程序为POPPAD2。「Edit」菜单的菜单项的功能全部可用；我们将在第十一章中完成「File」功能，在第十三章中完成「Print」功能。POPPAD2如程序10-9所示。

程序10-9 POPPAD2

POPPAD2.C

```
/*---------------------------------------------------------------------------
POPPAD2.C -- Popup Editor Version 2 (includes menu)
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define ID_EDIT 1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
TCHAR szAppName[] = TEXT ("PopPad2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
```

```
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, szAppName,
    WS_OVERLAPPEDWINDOW,
    GetSystemMetrics (SM_CXSCREEN) / 4,
    GetSystemMetrics (SM_CYSCREEN) / 4,
    GetSystemMetrics (SM_CXSCREEN) / 2,
    GetSystemMetrics (SM_CYSCREEN) / 2,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

AskConfirmation (HWND hwnd)
{
  return MessageBox ( hwnd, TEXT ("Really want to close PopPad2?"),
    szAppName, MB_YESNO | MB_ICONQUESTION) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HWND hwndEdit ;
  int iSelect, iEnable ;

  switch (message)
  {
  case WM_CREATE:
    hwndEdit = CreateWindow (TEXT ("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, (HMENU) ID_EDIT,
      ((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_INITMENUPOPUP:
  if (lParam == 1)
  {
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_UNDO,
      SendMessage (hwndEdit, EM_CANUNDO,
      0, 0) ? MF_ENABLED : MF_GRAYED) ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_PASTE,
      IsClipboardFormatAvailable(CF_TEXT) ? MF_ENABLED : MF_GRAYED) ;

    iSelect = SendMessage (hwndEdit, EM_GETSEL, 0, 0) ;

    if (HIWORD (iSelect) == LOWORD (iSelect))
      iEnable = MF_GRAYED ;
    else
      iEnable = MF_ENABLED ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CLEAR, iEnable) ;
    return 0 ;
  }
  break ;
case WM_COMMAND:
  if (lParam)
  {
    if (LOWORD (lParam) == ID_EDIT &&
      (HIWORD (wParam) == EN_ERRSPACE ||
      HIWORD (wParam) == EN_MAXTEXT))
      MessageBox (hwnd, TEXT ("Edit control out of space."),
      szAppName, MB_OK | MB_ICONSTOP) ;
    return 0 ;
  }
  else switch (LOWORD (wParam))
  {
case IDM_FILE_NEW:
case IDM_FILE_OPEN:
case IDM_FILE_SAVE:
case IDM_FILE_SAVE_AS:
case IDM_FILE_PRINT:
  MessageBeep (0) ;
  return 0 ;
case IDM_APP_EXIT:
  SendMessage (hwnd, WM_CLOSE, 0, 0) ;
  return 0 ;
case IDM_EDIT_UNDO:
  SendMessage (hwndEdit, WM_UNDO, 0, 0) ;
  return 0 ;
case IDM_EDIT_CUT:
  SendMessage (hwndEdit, WM_CUT, 0, 0) ;
  return 0 ;
case IDM_EDIT_COPY:
  SendMessage (hwndEdit, WM_COPY, 0, 0) ;
  return 0 ;
case IDM_EDIT_PASTE:
  SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
  return 0 ;
case IDM_EDIT_CLEAR:
  SendMessage (hwndEdit, WM_CLEAR, 0, 0) ;
  return 0 ;
case IDM_EDIT_SELECT_ALL:
  SendMessage (hwndEdit, EM_SETSEL, 0, -1) ;
  return 0 ;
case IDM_HELP_HELP:
  MessageBox (hwnd, TEXT ("Help not yet implemented!"),
```

```
     szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    return 0 ;
  case IDM_APP_ABOUT:
    MessageBox (hwnd, TEXT ("POPPAD2 (c) Charles Petzold, 1998"),
      szAppName, MB_OK | MB_ICONINFORMATION) ;
    return 0 ;
    }
    break ;
  case WM_CLOSE:
    if (IDYES == AskConfirmation (hwnd))
      DestroyWindow (hwnd) ;
    return 0 ;
  case WM_QUERYENDSESSION:
    if (IDYES == AskConfirmation (hwnd))
      return 1 ;
    else
      return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

POPPAD2.RC （摘录）

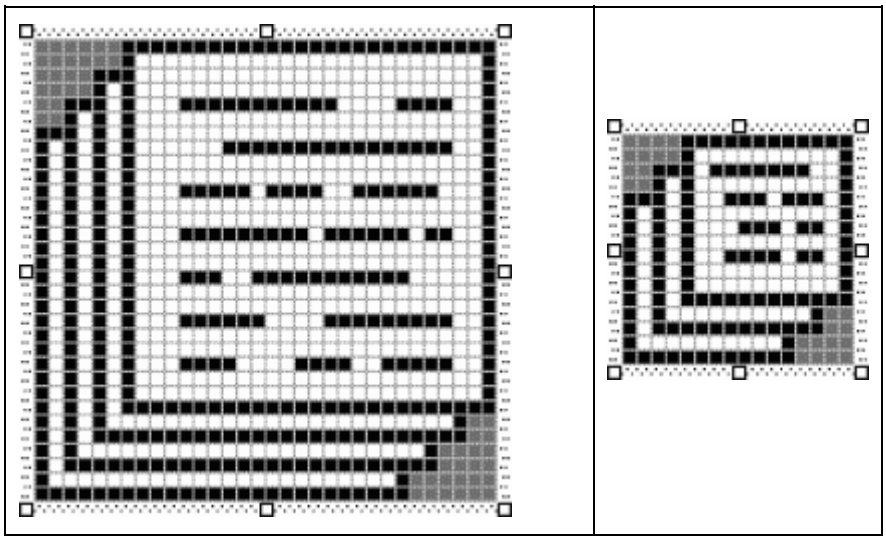
```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Icon
POPPAD2  ICON DISCARDABLE "poppad2.ico"
//////////////////////////////////////////////////////////////////////////////
// Menu
POPPAD2 MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&New",   IDM_FILE_NEW
MENUITEM "&Open...",  IDM_FILE_OPEN
MENUITEM "&Save",   IDM_FILE_SAVE
MENUITEM "Save &As...",  IDM_FILE_SAVE_AS
MENUITEM SEPARATOR
MENUITEM "&Print",   IDM_FILE_PRINT
MENUITEM SEPARATOR
MENUITEM "E&xit",  IDM_APP_EXIT
END
POPUP "&Edit"
BEGIN
MENUITEM "&Undo\tCtrl+Z",  IDM_EDIT_UNDO
MENUITEM SEPARATOR
MENUITEM "Cu&t\tCtrl+X",  IDM_EDIT_CUT
MENUITEM "&Copy\tCtrl+C",  IDM_EDIT_COPY
MENUITEM "&Paste\tCtrl+V",  IDM_EDIT_PASTE
MENUITEM "De&lete\tDel",  IDM_EDIT_CLEAR
MENUITEM SEPARATOR
MENUITEM "&Select All",  IDM_EDIT_SELECT_ALL
END
POPUP "&Help"
BEGIN
MENUITEM "&Help...",   IDM_HELP_HELP
MENUITEM "&About PopPad2...", IDM_APP_ABOUT
END
END

//////////////////////////////////////////////////////////////////////////////
// Accelerator
POPPAD2 ACCELERATORS DISCARDABLE
```

```
BEGIN
VK_BACK,  IDM_EDIT_UNDO, VIRTKEY, ALT, NOINVERT
VK_DELETE,  IDM_EDIT_CLEAR, VIRTKEY, NOINVERT
VK_DELETE,  IDM_EDIT_CUT, VIRTKEY, SHIFT, NOINVERT
VK_F1,  IDM_HELP_HELP, VIRTKEY, NOINVERT
VK_INSERT,  IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
VK_INSERT,  IDM_EDIT_PASTE, VIRTKEY, SHIFT, NOINVERT
"^C", IDM_EDIT_COPY, ASCII, NOINVERT
"^V", IDM_EDIT_PASTE, ASCII, NOINVERT
"^X", IDM_EDIT_CUT, ASCII, NOINVERT
"^Z", IDM_EDIT_UNDO, ASCII, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by POPPAD2.RC
#define IDM_FILE_NEW   40001
#define IDM_FILE_OPEN  40002
#define IDM_FILE_SAVE  40003
#define IDM_FILE_SAVE_AS  40004
#define IDM_FILE_PRINT  40005
#define IDM_APP_EXIT   40006
#define IDM_EDIT_UNDO  40007
#define IDM_EDIT_CUT   40008
#define IDM_EDIT_COPY  40009
#define IDM_EDIT_PASTE  40010
#define IDM_EDIT_CLEAR  40011
#define IDM_EDIT_SELECT_ALL  40012
#define IDM_HELP_HELP  40013
#define IDM_APP_ABOUT  40014
```

POPPAD2.ICO



POPPAD2.RC资源描述文件包含菜单和快捷键。您将注意到，所有快捷键都表示在制表符 (\t) 后的「Edit」弹出式菜单的字符串中。

启用菜单项

窗口消息处理程序的工作包括启用和无效化「Edit」菜单中的选项，这项工作在处理WM_INITMENUPOPUP时完成。首先，程序检查是否要显示「Edit」弹出式菜单。因为菜单里「Edit」的位置索引（「File」从0开始）是1，因此如果即将显示「Edit」弹出式菜单，那么lParam应该等于

1。

为了确定是否启用「Undo」选项，POPPAD2给编辑控件发送一条EM_CANUNDO消息。如果编辑控件能够执行「Undo」动作，那么SendMessage呼叫传回非零值。在这种情况下，选项被启用；否则，选项无效化：

```
EnableMenuItem (wParam, IDM_UNDO,
             SendMessage (hwndEdit, EM_CANUNDO, 0, 0) ?
             MF_ENABLED : MF_GRAYED) ;
```

只有当剪贴簿中包含文字时，「Paste」选项才能够被启用。我们可以使用CF_TEXT标识符通过IsClipboardFormatAvailable呼叫来确定这一点：

```
EnableMenuItem (wParam, IDM_PASTE,
             IsClipboardFormatAvailable (CF_TEXT) ? MF_ENABLED : MF_GRAYED) ;
```

只有选择了编辑控件中的文字，「Cut」、「Copy」和「Delete」选项才能够被启用。给编辑控件发送一条EM_GETSEL消息，并传回包含此信息的整数：

iSelect = SendMessage (hwndEdit, EM_GETSEL, 0, 0) ;

iSelect的低位字是第一个被选中字符的位置，iSelect的高字组是下一个被选中字符的位置。如果这两个字相等，则表示没有选中文字：

```
if (HIWORD (iSelect) == LOWORD (iSelect))
  iEnable = MF_GRAYED ;
else
  iEnable = MF_ENABLED ;
    然后可以将iEnable的值用于「Cut」、「Copy」和「Delete」选项：
EnableMenuItem (wParam, IDM_CUT, iEnable) ;
EnableMenuItem (wParam, IDM_COPY, iEnable) ;
EnableMenuItem (wParam, IDM_DEL, iEnable) ;
```

处理菜单项

当然，如果POPPAD2程序不使用子窗口编辑控件，那么我们将面临一些问题，这涉及如何完成「Edit」菜单中的「Undo」、「Cut」、「Copy」、「Paste」、「Clear」和「Select All」选项。正是编辑控件使得这种处理变得容易，因为对于每一个选项我们只需向编辑控件发送一个消息即可：

```
case IDM_UNDO :
  SendMessage (hwndEdit, WM_UNDO, 0, 0) ;
  return 0 ;
case IDM_CUT :
  SendMessage (hwndEdit, WM_CUT, 0, 0) ;
  return 0 ;
case IDM_COPY :
  SendMessage (hwndEdit, WM_COPY, 0, 0) ;
  return 0 ;
case IDM_PASTE :
  SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
  return 0 ;
case IDM_DEL :
  SendMessage (hwndEdit, WM_DEL, 0, 0) ;
  return 0 ;
case IDM_SELALL :
  SendMessage (hwndEdit, EM_SETSEL, 0, -1) ;
  return 0 ;
```

注意，我们可以更进一步简化这些处理 – 只要使IDM_UNDO、IDM_CUT等等的值等于相对应的窗口消息WM_UNDO、WM_CUT的值。

「File」弹出式菜单上的「About」选项启动一个简单的消息框：

```
case IDM_ABOUT :
  MessageBox (hwnd, TEXT ("POPPAD2 (c) Charles Petzold, 1998"),
    szAppName, MB_OK | MB_ICONINFORMATION) ;
```

```
  return 0 ;
```

在下一章中，我们将把它变成一个对话框。当您从菜单中选择「Help」选项或者按下F1快捷键时，同样可以启动一个消息框。

「Exit」选项向窗口消息处理程序发送一个WM_CLOSE消息：

```
case IDM_EXIT :
  SendMessage (hwnd, WM_CLOSE, 0, 0) ;
  return 0 ;
```

这正是DefWindowProc收到一个wParam等于SC_CLOSE的WM_SYSCOMMAND消息时所完成的工作。

在前面的那些程序中，我们没有在窗口消息处理程序中处理WM_CLOSE消息，而只是简单地把它送给DefWindowProc。DefWindowProc对WM_CLOSE的处理非常简单：呼叫DestroyWindow函数。可以不把WM_CLOSE消息送给DefWindowProc，而让POPPAD2来处理它。这个事实到目前为止并不重要，但是在第十一章中当POPPAD可以真正编辑文字时，它就变得非常重要了。

```
case WM_CLOSE :
  if (IDYES == AskConfirmation (hwnd))
    DestroyWindow (hwnd) ;
  return 0 ;
```

AskConfirmation是POPPAD2中的一个函数，它显示一个请求确认关闭程序的消息框：

```
AskConfirmation (HWND hwnd)
{
  return MessageBox (hwnd, TEXT ("Really want to close Poppad2?"),
    szAppName, MB_YESNO | MB_ICONQUESTION) ;
}
```

如果选择了Yes按钮的话，消息框（以及AskConfirmation函数）将传回IDYES。只有这样，程序才会呼叫DestroyWindow，否则，程序不会结束。

如果要在程序结束之前确认使用者真的要结束程序，那么您还必须处理WM_QUERYENDSESSION消息。当使用者要关闭Windows时，Windows开始向每个窗口消息处理程序发送一个WM_QUERYENDSESSION消息。如果有任何一个窗口消息处理程序处理这个消息后传回0，那么Windows将不会结束。我们如下处理了WM_QUERYENDSESSION：

```
case WM_QUERYENDSESSION :
  if (IDYES == AskConfirmation (hwnd))
    return 1 ;
  else
    return 0 ;
```

如果要在程序结束之前要求使用者的确认，必须处理WM_CLOSE和WM_QUERYENDSESSION这两个消息，这就是为什么我们使POPPAD2中的「Exit」菜单选项只向窗口消息处理程序发送一个WM_CLOSE消息的原因。这样做，我们避免了在别处进行请求确认的动作。

如果要处理WM_QUERYENDSESSION消息，那么您也许还会对WM_ENDSESSION消息感兴趣。Windows把这个消息发送给先前收到WM_QUERYENDSESSION消息的每个窗口消息处理程序。如果由于另一个程序从WM_QUERYENDSESSION传回了0而不能结束Windows的执行，那么WM_ENDSESSION的wParam参数为0。WM_ENDSESSION消息实际上回答了这个问题：我告诉过Windows可以把我结束掉，但是我真的被结束掉了吗？

尽管在POPPAD2的「File」菜单中我加上了常见的「New」、「Open」、「Save」和「Save As」选项，但是它们现在并不起作用。要处理这些命令，我们需要使用对话框。现在是讨论对话框的时机，也是您准备学习它们的时候了。

第十一章 对话框

如果有很多输入超出了菜单可以处理的程度，那么我们可以使用对话框来取得输入信息。程序写作者可以通过在某选项后面加上省略号（…）来表示该菜单项将启动一个对话框。

对话框的一般形式是包含多种子窗口控件的弹出式窗口，这些控件的大小和位置在程序资源描述文件的「对话框模板」中指定。虽然程序写作者能够「手工」定义对话框模板，但是现在通常是在Visual C++ Developer Studio中以交谈式操作的方式设计的，然后由Developer Studio建立对话框模板。

当程序呼叫依据模板建立的对话框时，Microsoft Windows 98负责建立弹出式对话框窗口和子窗口控件，并提供处理对话框消息（包括所有键盘和鼠标输入）的窗口消息处理程序。有时候称呼完成这些功能的Windows内部程序代码为「对话框管理器」。

Windows的内部对话框窗口消息处理程序所处理的许多消息也传递给您自己程序中的函数，这个函数即是所谓的「对话框程序」或者「对话程序」。对话程序与普通的窗口消息处理程序类似，但是也存在着一些重要区别。一般来说，除了在建立对话框时初始化子窗口控件，处理来自子窗口控件的消息以及结束对话框之外，程序写作者不需要再给对话框程序增加其它功能。对话程序通常不处理WM_PAINT消息，也不直接处理键盘和鼠标输入。

对话框这个主题的含义太广了，因为它还包含子窗口控件的使用。不过，我们已经在第九章研究了子窗口控件。当您在对话框中使用子窗口控件时，第九章所提到的许多工作都可以由Windows的对话框管理器来完成。尤其是，在程序COLORS1中遇到在滚动条之间切换输入焦点的问题也不会在对话框中出现。Windows会处理对话框中的控件之间切换输入焦点所必需完成的全部工作。

不过，在程序中添加对话框要比添加图标或者菜单更麻烦一些。我们将从一个简单的对话框开始，让您对各部分之间的相互联系有所了解。

模态对话框

对话框分为两类:「模态的」和「非模态的」，其中模态对话框最为普遍。当您的程序显示一个模态对话框时，使用者不能在对话框与同一个程序中的另一个窗口之间进行切换，使用者必须主动结束该对话框，这藉由通过按一下「OK」或者「Cancel」键来完成。不过，在显示模态对话框时，使用者通常可以从目前的程序切换到另一个程序。而有些对话框（称为「系统模态」）甚至连这样的切换程序操作也不允许。在Windows中，显示了系统模态对话框之后，要完成其它任何工作，都必须先结束该对话框。

建立「About」对话框

Windows程序即使不需要接收使用者输入，也通常具有由菜单上的「About」选项启动的对话框，该对话框用来显示程序的名字、图标、版权旗标和标记为「OK」的按键，也许还会有其它信息（例如技术支持的电话号码）。我们将要看到的第一个程序除了显示一个「About」对话框外，别无它用。这个ABOUT1程序如程序11-1所示:

程序11-1 ABOUT1

ABOUT1.C

```
/*---------------------------------------------------------------------------
ABOUT1.C -- About Box Demo Program No. 1
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("About1") ;
  MSG msg ;
  HWND hwnd ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("About Box Demo Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HINSTANCE hInstance ;
  switch (message)
  {
  case WM_CREATE :
    hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
    return 0 ;
  case WM_COMMAND :
    switch (LOWORD (wParam))
    {
    case IDM_APP_ABOUT :
      DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
      break ;
```

```
    }
    return 0 ;
    case WM_DESTROY :
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG :
    return TRUE ;
  case WM_COMMAND :
    switch (LOWORD (wParam))
    {
    case IDOK :
    case IDCANCEL :
      EndDialog (hDlg, 0) ;
      return TRUE ;
    }
    break ;
  }
  return FALSE ;
}
```

ABOUT1.RC （摘录）

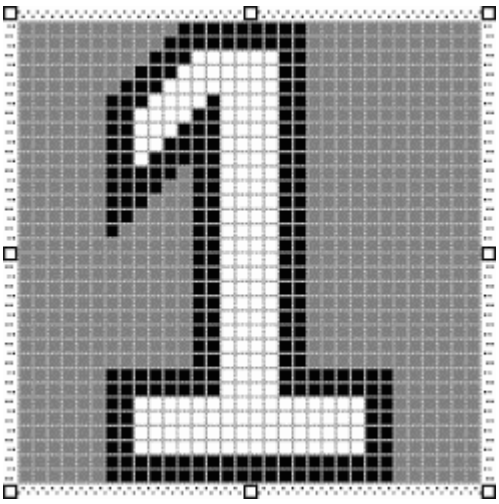
```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON  "OK",IDOK,66,80,50,14
ICON      "ABOUT1",IDC_STATIC,7,7,21,20
CTEXT      "About1",IDC_STATIC,40,12,100,8
CTEXT    "About Box Demo Program",IDC_STATIC,7,40,166,8
CTEXT    "(c) Charles Petzold,
1998",IDC_STATIC,7,52,166,8
END

////////////////////////////////////////////////////////////////////////////
// Menu
ABOUT1 MENU DISCARDABLE
BEGIN
POPUP "&Help"
BEGIN
MENUITEM "&About About1...",    IDM_APP_ABOUT
END
END

////////////////////////////////////////////////////////////////////////////
// Icon
ABOUT1  ICON DISCARDABLE "About1.ico"
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by About1.rc
#define IDM_APP_ABOUT  40001
```

```
#define IDC_STATIC    -1
```

ABOUT1.ICO



藉由后面章节中介绍的方法，您还可以在程序中建立图标和菜单。图标和菜单的ID名均为「About1」。菜单有一个选项，它产生一条ID名为IDM_APP_ABOUT的WM_COMMAND消息。这使得程序显示的图11-1所示的对话框。

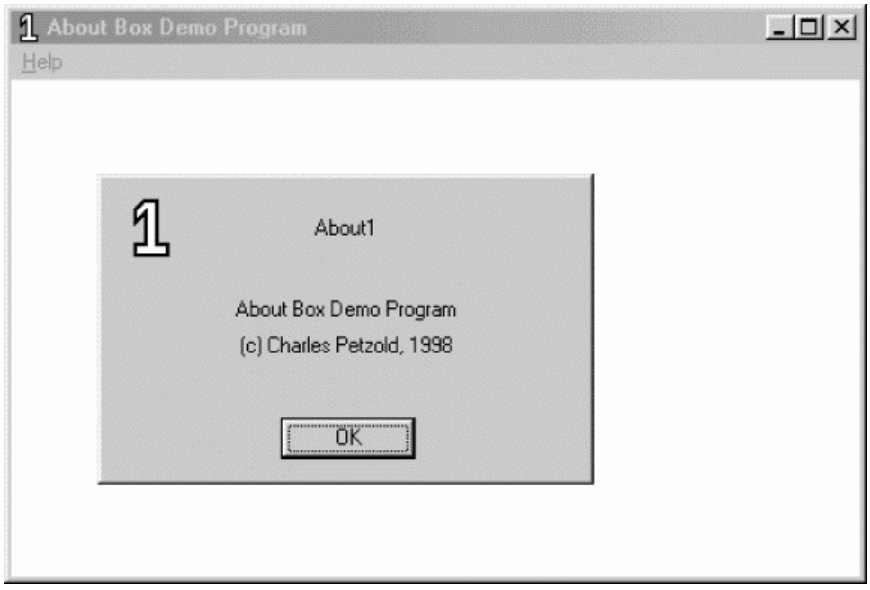


图11-1 程序ABOUT1的对话框

对话框及其模板

要把一个对话框添加到Visual C++ Developer Studio会有的应用程序上，可以先从Insert菜单中选择 Resource，然后选择Dialog Box。现在一个对话框出现在您的眼前，该对话框带有标题栏、标题（Dialog）以及 OK和Cancel按钮。Controls工具列允许您在对话框中插入不同的控件。

Developer Studio将对话框的ID设为标准的IDD_DIALOG1。您可以在此名称上（或者在对话框本身）单击右键，然后从菜单中选择 Properties。在本程序中，将ID改为「AboutBox」（带有引号）。为了与我建立的对话框保持一致，请将 X Pos和Y Pos字段改为32。这表示对话框相对于程序窗口显示区域左上角的显示位置待会会有关于对话框坐标的详细讨论）。

现在，继续在Properties对话框中选择Styles页面标签。因为此对话框没有标题栏，所以不要选取 Title Bar复选框。然后请单击Properties对话框的 关闭按钮。

现在可以设计对话框了。因为不需要Cancel按钮，所以先单击该按钮，然后按下键盘上的Delete键。接着单击OK按钮，将其移动到对话框的底部。在Developer Studio窗口下面的工具列上有一个小位图，它可使控件在窗口内水平居中对齐，请按下此钮。

如果您要让程序的图标出现在对话框中，可以这样做：先在浮动的Controls工具列中按下「 Pictures」按钮。将鼠标移动到对话框的表面，按下左键，然后拉出一个矩形。这就是图标将出现的位置。然后在次矩形上按下鼠标右键，从菜单中选择 Properties。保持ID为IDC_STATIC。此标识符在RESOURCE.H中定义为-1，用于程序中不使用的所有ID。将 Type改为Icon。您可以在Image字段输入程序图标的名称，或者，如果您已经建立了一个图标，那么您也可以从下拉式清单方块中选择一个名称（About1）。

对于对话框中的三个静态字符串，可以从Controls工具列中选择 Static Text，然后确定文字在对话框中的位置。右键单击控件，然后从菜单中选择 Properties。在Properties框的 Caption字段中输入要显示的文字。选择Styles页面标签，从 Align Text字段选择Center。

在添加这些字符串的时候，若希望对话框可以更大一些，请先选中对话框，然后拖曳边框。您也可以选择并缩放控件。通常用键盘上的光标移动键完成此操作会更容易些。箭头键本身移动控件，按下Shift键后按箭头键，可以改变控件的大小。所选控件的坐标和大小显示在Developer Studio窗口的右下角。

如果您建立了一个应用程序，那么以后在查看资源描述文件ABOUT1.RC时，您将发现Developer Studio建立的模板。我所设计的对话框模板如下：

```
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON "OK",IDOK,66,80,50,14
ICON      "ABOUT1",IDC_STATIC,7,7,21,20
CTEXT      "About1",IDC_STATIC,40,12,100,8
CTEXT "About Box Demo Program",IDC_STATIC,7,40,166,8
CTEXT "(c) Charles Petzold, 1998",IDC_STATIC,7,52,166,8
END
```

第一行给出了对话框的名称（这里为ABOUTBOX）。如同其它资源，您也可以使用数字作为对话框的名称。名称后面是关键词DIALOG和DISCARDABLE以及四个数字。前两个数字是对话框左上角的x、y坐标，该坐标在程序呼叫对话框时，是相对于父窗口显示区域的。后两个数字是对话框的宽度和高度。

这些坐标和大小的单位都不是像素。它们实际上依据一种特殊的坐标系统，该系统只用于对话框模板。数字依据对话框使用字体的大小而定（这里是8点的MS Sans Serif字体）：x坐标和宽度的单位是字符平均宽度的1/4；y坐标和高度的单位是字符高度的1/8。因此，对这个对话框来说，对话框左上角距离主窗口显示区域的左边是5个字符，距离顶边是2-1/2个字符。对话框本身宽40个字符，高10个字符。

这样的坐标系使得程序写作者可以使用坐标和大小来大致勾勒对话框的尺寸和外观，而不管视讯显示器的分辨率是多少。由于系统字体字符的高度大致为其宽度的两倍，所以，x轴和y轴的量度差不多相等。

模板中的STYLE叙述类似于CreateWindow呼叫中的style字段。对于模态对话框，通常使用WS_POPUP和DS_MODALFRAME，我们将在稍后介绍其它的选项。

在BEGIN和END叙述（或者是左右大括号，手工设计对话框模板时，您可能会使用）之间，定义出现在对话框中的子窗口控件。这个对话框使用了三种型态的子窗口控件，它们分别是DEFPUSHBUTTON（内定按键）、ICON（图标）和CTEXT（文字居中）。这些叙述的格式为：

control-type "text" id, xPos, yPos, xWidth, yHeight, iStyle

其中，后面的iStyle项是可选的，它使用Windows表头文件中定义的标识符来指定其它窗口样式。

DEFPUSHBUTTON、ICON和CTEXT等标识符只可以在对话框中使用，它们是某种特定窗口类别和窗口样式的缩写。例如，CTEXT指示这个子窗口控件类别是「静态的」，其样式为：

WS_CHILD | SS_CENTER | WS_VISIBLE | WS_GROUP

虽然前面没有出现过WS_GROUP标识符，但是在第九章的COLORS1程序中已经出现过WS_CHILD、SS_CENTER和WS_VISIBLE窗口样式，我们在建立静态子窗口文字控件时已经用到了它们。

对于图标，文字字段是程序的图标资源名称，它也在ABOUT1资源描述文件中定义。对于按键，文字字段是出现在按键里的文字，这个文字相同于在程序中建立子窗口控件时呼叫CreateWindow所指定的第二个参数。

id字段是子窗口在向其父窗口发送消息（通常为WM_COMMMAND消息）时用来标示它自身的值。这些子窗口控件的父窗口就是对话框本身，它将这些消息发送给Windows的一个窗口消息处理程序。不过，这个窗口消息处理程序也将这些消息发送给您在程序中给出的对话框程序。ID值相同于我们在第九章建立子窗口时，在CreateWindow函数中使用的子窗口ID。由于文字和图标控件不向父窗口回送消息，所以这些值被设定为IDC_STATIC，它在RESOURCE.H中定义为-1。按键的ID值为IDOK，它在WINUSER.H中定义为1。

接下来的四个数字设定子窗口的位置（相对于对话框显示区域的左上角）和大小，它们是以系统字体平均宽度的1/4和平均高度的1/8为单位来表示的。对于ICON叙述，宽度和高度将被忽略。

对话框模板中的DEFPUSHBUTTON叙述，除了包含DEFPUSHBUTTON关键词所隐含的窗口样式，还包含窗口样式WS_GROUP。稍后讨论该程序的第二个版本ABOUT2时，还会详细说明WS_GROUP（以及相关的WS_TABSTOP样式）。

对话框程序

您程序内的对话框程序处理传送给对话框的消息。尽管看起来很像是窗口消息处理程序，但是它并不是真实的窗口消息处理程序。对话框的窗口消息处理程序在Windows内部定义，这个窗口过程调用您编写的对话框程序，把它所接收到的许多消息作为参数。下面是ABOUT1的对话框程序：

```
BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG :
    return TRUE ;

  case WM_COMMAND :
    switch (LOWORD (wParam))
    {
    case IDOK :
    case IDCANCEL :
      EndDialog (hDlg, 0) ;
      return TRUE ;
    }
    break ;
  }
  return FALSE ;
}
```

该函数的参数与常规窗口消息处理程序的参数相同，与窗口消息处理程序类似，对话框程序都

必须定义为一个CALLBACK（callback）函数。尽管我使用了hDlg作为对话框窗口的句柄，但是您也可以按照您自己的意思使用hwnd。首先，让我们来看一下这个函数与窗口消息处理程序的区别：

窗口消息处理程序传回一个LRESULT。对话框传回一个BOOL，它在Windows表头文件中定义为int型态。

如果窗口消息处理程序不处理某个特定的消息，那么它将呼叫DefWindowProc。如果对话框程序处理一个消息，那么它传回TRUE（非0），如果不处理，则传回FALSE（0）。

对话框程序不需要处理WM_PAINT或WM_DESTROY消息。对话框程序不接收WM_CREAT消息，而是在特殊的WM_INITDIALOG消息处理期间，对话框程序执行初始化操作。

WM_INITDIALOG消息是对话框接收到的第一个消息，这个消息只发送给对话框程序。如果对话框程序传回TRUE，那么Windows将输入焦点设定给对话框中第一个具有WS_TABSTOP样式（我们将在ABOUT2的讨论中加以解释）的子窗口控件。在这个对话框中，第一个具有WS_TABSTOP样式的子窗口控件是按键。另外，对话框程序也可以在处理WM_INITDIALOG时使用SetFocus来将输入焦点设定为对话框中的某个子窗口控件，然后传回FALSE。

此外，对话框程序只处理WM_COMMAND消息。这是当按键被鼠标点中，或者在按钮具有输入焦点的情况下按下空格键时，按键控件发送给其父窗口的消息。这个控件的ID（我们在对话框模板中将其设定为IDOK）在wParam的低字组中。对于这个消息，对话框过程调用EndDialog，它告诉Windows清除对话框。对于所有其它消息，对话框程序传回FALSE，并告诉Windows内部的对话框窗口消息处理程序：我们的对话框程序不处理这些消息。

模态对话框的消息不通过您程序的消息队列，所以不必担心对话框中键盘快捷键的影响。

激活对话框

在WndProc中处理WM_CREATE消息时，ABOUT1取得程序的执行实体句柄并将它放在静态变量中：

```
hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
```

ABOUT1检查WM_COMMAND消息，以确保消息wParam的低位字等于IDM_APP_ABOUT。当它获得这样一个消息时，程序呼叫DialogBox：

```
DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
```

该函数需要执行实体句柄（在处理WM_CREATE时储存的）、对话框名称（在资源描述文件中定义的）、对话框的父窗口（也是程序的主窗口）和对话框程序的地址。如果您使用一个数字而不是对话框模板名称，那么可以用MAKEINTRESOURCE宏将它转换为一个字符串。

从菜单中选择「About About1」，将显示图11-2所示的对话框。您可以使用鼠标单击「OK」按钮、按空格键或者按Enter键来结束这个对话框。对任何包含内定按钮的对话框，在按下Enter键或空格键之后，Windows发送一个WM_COMMAND消息给对话框，并令wParam的低字组等于内定按键的ID，此时的ID为IDOK。按下Escape键也可以关闭对话框，这时Windows将发送一个WM_COMMAND消息，并令ID等于IDCANCEL。

直到对话框结束之后，用来显示对话框的DialogBox才将控制权传回给WndProc。DialogBox的传回值是对话框程序内部呼叫的EndDialog函数的第二个参数（这个值未在ABOUT1中使用，但会在ABOUT2中使用）。然后，WndProc可以将控制权传回给Windows。

即使在显示对话框时，WndProc也可以继续接收消息。实际上，您可以从对话框程序内部给WndProc发送消息。ABOUT1的主窗口是弹出式对话框窗口的父窗口，所以AboutDlgProc中的

SendMessage呼叫可以使用如下叙述来开始：

SendMessage (GetParent (hDlg), ...) ;

不同的主题

虽然Visual C++ Developer Studio中的对话框编辑器和其它资源编辑器，使我们几乎不用考虑资源描述的写作问题，但是学习一些资源描述的语法还是有用的。尤其对于对话框模板来说，知道了语法，您就可以近一步了解对话框的范围和限制。甚至当它不能满足您的需要时，您还可以自己建立一个对话框模板（就像本章后面的HEXCALC程序）。资源编译器和资源描述语法的文件位于/Platform SDK/Windows Programming Guidelines/Platform SDK Tools/Compiling/Using the Resource Compiler。

在Developer Studio的「Properties」对话框中指定了对话框的窗口样式，它翻译成对话框模板中的STYLE叙述。对于ABOUT1，我们使用模态对话框最常用的样式；

STYLE WS_POPUP | DS_MODALFRAME

然而，您也可以尝试其它样式。有些对话框有标题栏，标题栏用于指出对话框的用途，并允许使用者通过鼠标在显示屏上移动对话框。此样式为WS_CAPTION。如果您使用WS_CAPTION，那么DIALOG叙述中所指定的x坐标和y坐标是对话框显示区域的坐标，并相对于父窗口显示区域的左上角。标题栏将在y坐标之上显示。

如果使用了标题栏，那么您可以用CAPTION叙述将文字放入标题中。在对话框模板中，CAPTION叙述在STYLE叙述的后面：

CAPTION "Dialog Box Caption"

另外，在对话框程序处理WM_INITDIALOG消息处理期间，您还可以呼叫：

SetWindowText (hDlg, TEXT ("Dialog Box Caption")) ;

如果您使用WS_CAPTION样式，也可以添加一个WS_SYSMENU样式的系统菜单按钮。此样式允许使用者从系统菜单中选择 Move或Close。

从Properties对话框的Border清单方块中选择 Resizing（相同于样式WS_THICKFRAME），允许使用者缩放对话框，仅管此操作并不常用。如果您不介意更特殊一点的话，还可以着为此对话框样式添加最大化方块。

您甚至可以给对话框添加一个菜单。这时对话框模板将包括下面的叙述：

MENU menu-name

其参数不是菜单的名称，就是资源描述中的菜单号。模态对话框很少使用菜单。如果使用了菜单，那么您必须确保菜单和对话框控件中的所有ID都是唯一的；或者不是唯一的，却表达了相同的命令。

FONT叙述使您可以设定非系统字体，以供对话框文字使用。这在过去的对话框中不常用，但现在却非常普遍。事实上，在内定情况下，Developer Studio为您建立的每一个对话框都选用8点的MS Sans Serif字体。一个Windows程序能把自己外观打点得非常与众不同，这只需为程序的对话框及其它文字输出单独准备一种字体即可。

尽管对话框窗口消息处理程序通常位于Windows内部，但是您也可以使用自己编写的窗口消息处理程序来处理对话框消息。要这样做，您必须在对话框模板中指定一个窗口类别名：

CLASS "class-name"

这种用法很少见，但是在本章后面所示的HEXCALC程序中我们将用到它。

当您使用对话框模板的名称来呼叫DialogBox时，Windows通过呼叫普通的CreateWindow函数来完成建立弹出式窗口所需要完成的一切操作。Windows从对话框模板中取得窗口的坐标、大小、窗口样式、标题和菜单，从DialogBox的参数中获得执行实体句柄和父窗口句柄。它所需要的唯一其它信息是一个窗口类别（假设对话框模板不指定窗口类别的话）。Windows为对话框注册一个专用的窗口类别，这个窗口类别的窗口消息处理程序可以存取对话框程序地址（该地址是您在DialogBox呼叫中指定的），所以它可以使程序获得该弹出式窗口所接收的消息。当然，您可以通过自己建立弹出式窗口来建立和维护自己的对话框。不过，使用DialogBox则更简单。

也许您希望受益于Windows对话框管理器，但不希望（或者能够）在资源描述中定义对话框模板，也可能您希望程序在执行时可以动态地建立对话框。这时可以完成这种功能的函数是DialogBoxIndirect，此函数用数据结构来定义模板。

在ABOUT1.RC的对话框模板中，我们使用缩写CTEXT、ICON和DEFPUSHBUTTON来定义对话框所需要的三种型态的子窗口控件。您还可以使用其它型态，每种型态都隐含一个特定的预先定义窗口类别和一种窗口样式。下表显示了与一些控件型态相同的窗口类别和窗口样式：

表 11-1

控件型态	窗口类别	窗口样式
PUSHBUTTON	按钮	BS_PUSHBUTTON | WS_TABSTOP
DEFPUSHBUTTON	按钮	BS_DEFPUSHBUTTON | WS_TABSTOP
CHECKBOX	按钮	BS_CHECKBOX | WS_TABSTOP
RADIOBUTTON	按钮	BS_RADIOBUTTON | WS_TABSTOP
GROUPBOX	按钮	BS_GROUPBOX | WS_TABSTOP
LTEXT	静态文字	SS_LEFT | WS_GROUP
CTEXT	静态文字	SS_CENTER | WS_GROUP
RTEXT	静态文字	SS_RIGHT | WS_GROUP
ICON	静态图标	SS_ICON
EDITTEXT	编辑	ES_LEFT | WS_BORDER | WS_TABSTOP
SCROLLBAR	滚动条	SBS_HORZ
LISTBOX	清单方块	LBS_NOTIFY | WS_BORDER | WS_VSCROLL
COMBOBOX	下拉式清单方块	CBS_SIMPLE | WS_TABSTOP

资源编译器是唯一能够识别这些缩写的程序。除了表中所示的窗口样式外，每个控件还具有下面的样式：

WS_CHILD | WS_VISIBLE

对于这些控件型态，除了EDITTEXT、SCROLLBAR、LISTBOX和COMBOBOX之外，控件叙述的格式为：

control-type "text", id, xPos, yPos, xWidth, yHeight, iStyle

对于EDITTEXT、SCROLLBAR、LISTBOX和COMBOBOX，其格式为：

control-type id, xPos, yPos, xWidth, yHeight, iStyle

其中没有文字字段。在这两种叙述中，iStyle参数都是选择性的。

在第九章，我讨论了确定预先定义子窗口的宽度和高度的规则。您可能需要回到第九章去参考这些规则，这时请记住：对话框模板中指定大小的单位为平均字符宽度的1/4，及平均字符高度的1/8。

控件叙述的**style**字段是可选的。它允许您包含其它窗口样式标识符。例如，如果您想建立在正方形框左边包含文字的复选框，那么可以使用：

CHECKBOX "text", id, xPos, yPos, xWidth, yHeight, BS_LEFTTEXT

注意，控件型态EDITTEXT会自动添加一个边框。如果您想建立一个没有边框的子窗口编辑控件，您可以使用：

EDITTEXT id, xPos, yPos, xWidth, yHeight, NOT WS_BORDER

资源编译器也承认与下面叙述类似的专用控件叙述：

CONTROL "text", id, "class", iStyle, xPos, yPos, xWidth, yHeight

此叙述允许您通过指定窗口类别和完整的窗口样式，来建立任意型态的子窗口控件。例如，要取代：

PUSHBUTTON "OK", IDOK, 10, 20, 32, 14

您可以使用：

CONTROL "OK", IDOK, "button", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON | WS_TABSTOP, 10, 20, 32, 14

当编译资源描述文件时，这两条叙述在.RES和.EXE文件中的编码是相同的。在Developer Studio中，您可以使用**Controls**工具列中的**Custom Control**选项来建立此叙述。在ABOUT3程序中，我向您展示了如何用此选项建立一个控件，且在您的程序中已定义了该控件的窗口类别。

当您在对话框模板中使用CONTROL叙述时，不必包含WS_CHILD和WS_VISIBLE样式。在建立子窗口时，Windows已经包含了这些窗口样式。CONTROL叙述的格式也说明Windows对话框管理器在建立对话框时就完成了此项操作。首先，就像我前面所讨论的，它建立一个弹出式窗口，其父窗口句柄在DialogBox函数中提供。然后，对话框管理器为对话框模板中的每个控件建立一个子窗口。所有这些控件的父窗口均是这个弹出式对话框。上面给出的CONTROL叙述被转换成一个CreateWindow呼叫，形式如下所示：

```
hCtrl =CreateWindow (TEXT ("button"), TEXT ("OK"),
                  WS_CHILD | WS_VISIBLE | WS_TABSTOP |
                  BS_PUSHBUTTON,
                  10 * cxChar / 4, 20 * cyChar / 8,
                  32 * cxChar / 4, 14 * cyChar / 8,
                  hDlg, IDOK, hInstance, NULL) ;
```

其中，cxChar和cyChar是系统字体字符的宽度和高度，以像素为单位。hDlg参数是从建立该对话框窗口的CreateWindow呼叫传回的值；hInstance参数是从DialogBox呼叫获得的。

更复杂的对话框

ABOUT1中的简单对话框展示了设计和执行一个对话框的要点，现在让我们来看一个稍微复杂的例子。程序11-2给出的ABOUT2程序展示了如何在对话框程序中管理控件（这里用单选按钮）以

及如何在对话框的显示区域中绘图。

程序11-2 ABOUT2

ABOUT2.C

```
/*------------------------------------------------------------------------
ABOUT2.C -- About Box Demo Program No. 2
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;

int iCurrentColor = IDC_BLACK,
iCurrentFigure = IDC_RECT ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("About2") ;
  MSG msg ;
  HWND hwnd ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("About Box Demo Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void PaintWindow (HWND hwnd, int iColor, int iFigure)
{
  static COLORREF crColor[8] = { RGB ( 0, 0, 0), RGB ( 0, 0, 255),
    RGB ( 0, 255, 0), RGB ( 0, 255, 255),
    RGB (255, 0, 0), RGB (255, 0, 255),
    RGB (255, 255, 0), RGB (255, 255, 255)} ;
```

```
  HBRUSH hBrush ;
  HDC hdc ;
  RECT rect ;

  hdc = GetDC (hwnd) ;
  GetClientRect (hwnd, &rect) ;
  hBrush = CreateSolidBrush (crColor[iColor - IDC_BLACK]) ;
  hBrush = (HBRUSH) SelectObject (hdc, hBrush) ;

  if (iFigure == IDC_RECT)
    Rectangle (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
  else
    Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
  DeleteObject (SelectObject (hdc, hBrush)) ;
  ReleaseDC (hwnd, hdc) ;
}

void PaintTheBlock (HWND hCtrl, int iColor, int iFigure)
{
  InvalidateRect (hCtrl, NULL, TRUE) ;
  UpdateWindow (hCtrl) ;
  PaintWindow (hCtrl, iColor, iFigure) ;
}
LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HINSTANCE hInstance ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
    return 0 ;
  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_APP_ABOUT:
      if (DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc))
        InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
    }
    break ;
    case WM_PAINT:
      BeginPaint (hwnd, &ps) ;
      EndPaint (hwnd, &ps) ;

      PaintWindow (hwnd, iCurrentColor, iCurrentFigure) ;
      return 0 ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hCtrlBlock ;
  static int iColor, iFigure ;

  switch (message)
  {
  case WM_INITDIALOG:
    iColor = iCurrentColor ;
    iFigure = iCurrentFigure ;

    CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE, iColor) ;
    CheckRadioButton (hDlg, IDC_RECT, IDC_ELLIPSE, iFigure) ;
```

```
    hCtrlBlock = GetDlgItem (hDlg, IDC_PAINT) ;

    SetFocus (GetDlgItem (hDlg, iColor)) ;
    return FALSE ;
  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDOK:
      iCurrentColor = iColor ;
      iCurrentFigure = iFigure ;
      EndDialog (hDlg, TRUE) ;
      return TRUE ;
    case IDCANCEL:
      EndDialog (hDlg, FALSE) ;
      return TRUE ;
    case IDC_BLACK:
    case IDC_RED:
    case IDC_GREEN:
    case IDC_YELLOW:
    case IDC_BLUE:
    case IDC_MAGENTA:
    case IDC_CYAN:
    case IDC_WHITE:
      iColor = LOWORD (wParam) ;
      CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE, LOWORD (wParam)) ;
      PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
      return TRUE ;
    case IDC_RECT:
    case IDC_ELLIPSE:
      iFigure = LOWORD (wParam) ;
      CheckRadioButton (hDlg, IDC_RECT, IDC_ELLIPSE, LOWORD (wParam)) ;
      PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
      return TRUE ;
    }
    break ;
    case WM_PAINT:
      PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
      break ;
  }
  return FALSE ;
}
```

ABOUT2.RC （摘录）

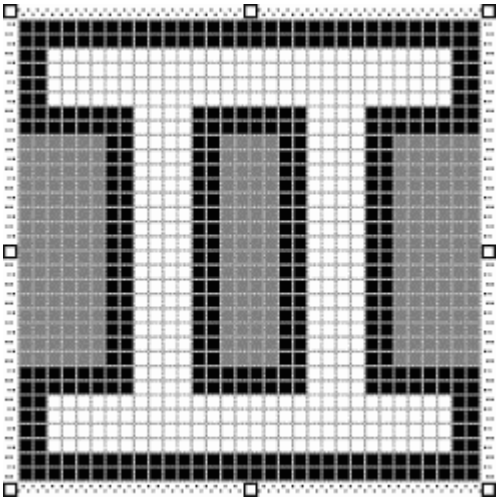
```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 200, 234
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
FONT 8, "MS Sans Serif"
BEGIN
ICON    "ABOUT2",IDC_STATIC,7,7,20,20
CTEXT "About2",IDC_STATIC,57,12,86,8
CTEXT "About Box Demo Program",IDC_STATIC,7,40,186,8
LTEXT "",IDC_PAINT,114,67,74,72
GROUPBOX   "&Color",IDC_STATIC,7,60,84,143
RADIOBUTTON  "&Black",IDC_BLACK,16,76,64,8,WS_GROUP | WS_TABSTOP
RADIOBUTTON "B&lue",IDC_BLUE,16,92,64,8
RADIOBUTTON "&Green",IDC_GREEN,16,108,64,8
RADIOBUTTON "Cya&n",IDC_CYAN,16,124,64,8
RADIOBUTTON "&Red",IDC_RED,16,140,64,8
RADIOBUTTON "&Magenta",IDC_MAGENTA,16,156,64,8
RADIOBUTTON "&Yellow",IDC_YELLOW,16,172,64,8
RADIOBUTTON "&White",IDC_WHITE,16,188,64,8
GROUPBOX  "&Figure",IDC_STATIC,109,156,84,46,WS_GROUP
RADIOBUTTON  "Rec&tangle",IDC_RECT,116,172,65,8,WS_GROUP | WS_TABSTOP
```

```
RADIOBUTTON  "&Ellipse",IDC_ELLIPSE,116,188,64,8
DEFPUSHBUTTON  "OK",IDOK,35,212,50,14,WS_GROUP
PUSHBUTTON   "Cancel",IDCANCEL,113,212,50,14,WS_GROUP
END

/////////////////////////////////////////////////////////////////////////////
// Icon
ABOUT2 ICON DISCARDABLE "About2.ico"

/////////////////////////////////////////////////////////////////////////////
// Menu
ABOUT2 MENU DISCARDABLE
BEGIN
POPUP "&Help"
BEGIN
MENUITEM "&About",  IDM_APP_ABOUT
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by About2.rc
#define IDC_BLACK   1000
#define IDC_BLUE   1001
#define IDC_GREEN   1002
#define IDC_CYAN   1003
#define IDC_RED    1004
#define IDC_MAGENTA   1005
#define IDC_YELLOW   1006
#define IDC_WHITE   1007
#define IDC_RECT   1008
#define IDC_ELLIPSE   1009
#define IDC_PAINT   1010
#define IDM_APP_ABOUT  40001
#define IDC_STATIC   -1
```

ABOUT2.ICO



ABOUT2中的About框有两组单选按钮。一组用来选择颜色，另一组用来选择是矩形还是椭圆形。所选的矩形或者椭圆显示在对话框内，其内部以目前选择的颜色着色。使用者按下「OK」按钮后，对话框会终止，程序的窗口消息处理程序在它自己的显示区域内绘出所选图形。如果您按下「Cancel」，则主窗口的显示区域会保持原样。对话框如图11-2所示。尽管ABOUT2使用预先定义的标识符IDOK和IDCANCEL作为两个按键，但是每个单选按钮均有自己的标识符，它们以前缀IDC开头（用于控件的ID）。这些标识符在RESOURCE.H中定义。

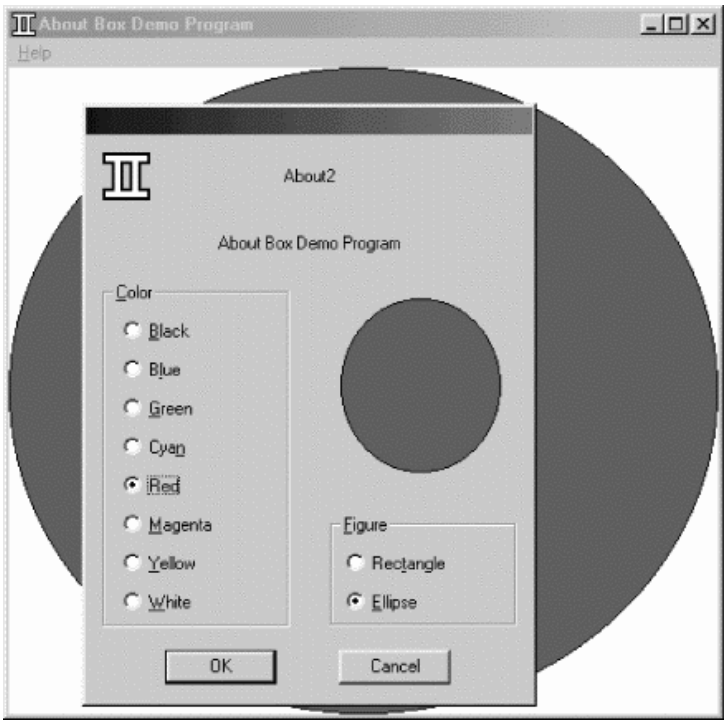


图11-2 ABOUT2程序的对话框

当您在ABOUT2对话框中建立单选按钮时，请按显示顺序建立。这能保证Developer Studio依照顺序定义标识符的值，程序将使用这些值。另外，每个单选按钮都不要选中「Auto」选项。「Auto Radio Button」需要的程序代码较少，但基本上处理起来更深奥些。然后请依照ABOUT2.RC中的定义来设定它们的标识符。

选中「Properties」对话框中下列对象的「Group」选项：「OK」和「Cancel」按钮、「Figure」分组方块、每个分组方块中的第一个单选按钮（「Black」和「Rectangle」）。选中这两个单选按钮的「Tab Stop」复选框。

当您有全部控件在对话框中的近似位置和大小时，就可以从「Layout」菜单选择「Tab Order」选项。按ABOUT2.RC资源描述中显示的顺序单击每一个控件。

使用对话框控件

在第九章中，您会发现大多数子窗口控件发送WM_COMMAND消息给其父窗口（唯一例外的是滚动条控件）。您还看到，经由发送消息给子窗口控件，父窗口可以改变子窗口控件的状态（例如，选择或不选择单选按钮、复选框）。您也可以用类似方法在对话框程序中改变控件。例如，如果您设计了一系列单选按钮，就可以发送消息给它们，以选择或者不选择这些按钮。不过，Windows也提供了几种使用对话框控件的简单办法。我们来看一看对话框程序与子窗口控件相互通信的方式。

ABOUT2的对话框模板显示在程序11-2的ABOUT2.RC资源描述文件中。GROUPBOX控件只是一个带标题（标题为「Color」或者「Figure」）的分组方块，每组单选按钮都由这样的分组方块包围。前一组的八个单选按钮是互斥的，第二组的两个单选按钮也是如此。

当用鼠标单击其中一个单选按钮时（或者当单选按钮拥有输入焦点时按空格键），子窗口向其父窗口发送一个WM_COMMAND消息，消息的wParam的低字组被设为控件的ID，wParam的高字组是一个通知码，lParam值是控件的窗口句柄。对于单选按钮，这个通知码是BN_CLICKED或者0。然后Windows中的对话框窗口消息处理程序将这个WM_COMMAND消息发送给ABOUT2.C

内的对话框程序。当对话框程序收到一个单选按钮的WM_COMMAND消息时，它为此按钮设定选中标记，并为组中其它按钮清除选中标记。

您可能还记得在第九章中已经提过，选中和不选中按钮均需要向子窗口控件发送BM_CHECK消息。要设定一个按钮选中标记，您可以使用：

SendMessage (hwndCtrl, BM_SETCHECK, 1, 0) ;

要消除选中标记，您可以使用：

SendMessage (hwndCtrl, BM_SETCHECK, 0, 0) ;

其中hwndCtrl参数是子窗口按钮控件的窗口句柄。

但是在对话框程序中使用这种方法是时有点问题的，因为您不知道所有单选按钮的窗口句柄，只是从您获得的消息中知道其中一个句柄。幸运的是，Windows为您提供了一个函数，可以用对话框句柄和控件ID来取得一个对话框控件的窗口句柄：

hwndCtrl = GetDlgItem (hDlg, id) ;

（您也可以使用如下函数，从窗口句柄中取得控件的ID值：

id = GetWindowLong (hwndCtrl, GWL_ID) ;

但是在大多数情况下这是不必要的。）

您会注意到，在程序11-2所示的表头文件ABOUT2.H中，八种颜色的ID值是从IDC_BLACK到IDC_WHITE连续变化的，这种安排在处理来自单选按钮的WM_COMMAND消息时将会很有用。在第一次尝试选中或者不选中单选按钮时，您可能会在对话框程序中编写如下的程序：

static int iColor ;

其它行程序

```
case WM_COMMAND:
  switch (LOWORD (wParam))
  {
    //其它行程序
  case IDC_BLACK:
  case IDC_RED:
  case IDC_GREEN:
  case IDC_YELLOW:
  case IDC_BLUE:
  case IDC_MAGENTA:
  case IDC_CYAN:
  case IDC_WHITE:
    iColor = LOWORD (wParam) ;
    for (i = IDC_BLACK, i <= IDC_WHITE, i++)
      SendMessage (GetDlgItem (hDlg, i),
      BM_SETCHECK, i == LOWORD (wParam), 0) ;
    return TRUE ;
    //其它行程序
```

这种方法能让人满意地执行。您将新的颜色值储存在iColor中，并且还建立了一个循环，轮流使用所有八种颜色的ID值。您取得每个单选按钮控件的窗口句柄，并用SendMessage给每个句柄发送一条BM_SETCHECK消息。只有对于向对话框窗口消息处理程序发送WM_COMMAND消息的按钮，这个消息的wParam值才被设定为1。

第一种简化的方法是使用专门的对话框程序SendDlgItemMessage：

SendDlgItemMessage (hDlg, id, iMsg, wParam, lParam) ;

它相同于：

SendMessage (GetDlgItem (hDlg, id), id, wParam, lParam) ;

现在，循环将变成这样：

for (i = IDC_BLACK, i <= IDC_WHITE, i++)

SendDlgItemMessage (hDlg, i, BM_SETCHECK, i == LWORD (wParam), 0) ;

稍微有些改进。但是真正的重大突破要等到使用了CheckRadioButton函数时才会出现：

CheckRadioButton (hDlg, idFirst, idLast, idCheck) ;

这个函数将ID在idFirst到idLast之间的所有单选按钮的选中标记都清除掉，除了ID为idCheck的单选按钮，因为它是被选中的。这里，所有ID必须是连续的。从此我们可以完全摆脱循环，并使用：

CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE, LOWORD (wParam)) ;

这正是ABOUT2对话框程序所采用的方法。

在使用复选框时，也提供了类似的简化函数。如果您建立了一个「CHECKBOX」对话框窗口控件，那么可以使用如下的函数来设定和清除选中标记：

CheckDlgButton (hDlg, idCheckbox, iCheck) ;

如果iCheck设定为1，那么按钮被选中；如果设定为0，那么按钮不被选中。您可以使用如下的方法来取得对话框中某个复选框的状态：

iCheck = IsDlgButtonChecked (hDlg, idCheckbox) ;

在对话框程序中，您既可以将选中标记的目前状态储存在一个静态变量中，又可以在收到一个WM_COMMAND消息后，使用如下方法触发按钮：

CheckDlgButton (hDlg, idCheckbox,

!IsDlgButtonChecked (hDlg, idCheckbox)) ;

如果您定义了BS_AUTOCHECKBOX控件，那么完全没有必要处理WM_COMMAND消息。在终止对话框之前，您只要使用IsDlgButtonChecked就可以取得按钮目前的状态。不过，如果您使用BS_AUTORADIOBUTTON样式，那么IsDlgButtonChecked就不能令人满意了，因为需要为每个单选按钮都呼叫它，直到函数传回TRUE。实际上，您还要拦截WM_COMMAND消息来追踪按下的按钮。

「OK」和「Cancel」按钮

ABOUT2有两个按键，分别标记为「OK」和「Cancel」。在ABOUT2.RC的对话框模板中，「OK」按钮的ID值为IDOK（在WINUSER.H中被定义为1），「Cancel」按钮的ID值为IDCANCEL（定义为2），「OK」按钮是内定的：

```
DEFPUSHBUTTON     "OK",IDOK,35,212,50,14
PUSHBUTTON        "Cancel",IDCANCEL,113,212,50,14
```

在对话框中，通常都这样安排「OK」和「Cancel」按钮：将「OK」按钮作为内定按钮有助于用键盘接口终止对话。一般情况下，您通过单击两个鼠标按键之一，或者当所期望的按钮具有输入焦点时按下Spacebar来终止对话框。不过，如果使用者按下Enter，对话框窗口消息处理程序也将产生一个WM_COMMAND消息，而不管哪个控件具有输入焦点。wParam的低字组被设定为对话框中内定按键的ID值，除非另一个按键拥有输入焦点。在后一种情况下，wParam的低字组被设定为具有输入焦点之按键的ID值。如果对话框中没有内定按键，那么Windows向对话框程序发送一

个WM_COMMAND消息，消息中wParam的低字组被设定为IDOK。如果使用者按下Esc键或者Ctrl-Break键，那么Windows令wParam等于IDCANCEL，并给对话框程序发送一个WM_COMMAND消息。所以，您不用在对话框程序中加入单独的处理键盘操作，因为通常终止对话框的按键会由Windows将这两个按键动作转换为WM_COMMAND消息。

AboutDlgProc函数通过呼叫EndDialog来处理这两种WM_COMMAND消息：

```
switch (LWORD (wParam))
{
case IDOK:
  iCurrentColor = iColor ;
  iCurrentFigure = iFigure ;
  EndDialog (hDlg, TRUE) ;
  return TRUE ;
case IDCANCEL :
  EndDialog (hDlg, FALSE) ;
  return TRUE ;
```

ABOUT2的窗口消息处理程序在程序的显示区域中绘制矩形或椭圆时，使用了整体变量iCurrentColor和iCurrentFigure。AboutDlgProc在对话框中画图时使用了静态区域变量iColor和iFigure。

注意EndDialog的第二个参数的值不同，这个值是在WndProc中作为原DialogBox函数的传回值传回的：

```
case IDM_ABOUT:
  if (DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc))
    InvalidateRect (hwnd, NULL, TRUE) ;
  return 0 ;
```

如果DialogBox传回TRUE（非0），则意味着按下了「OK」按钮，然后需要使用新的颜色来更新WndProc显示区域。当AboutDlgProc收到一个WM_COMMAND消息并且消息的wParam的低字组等于IDOK时，AboutDlgProc将图形和颜色储存在整体变量iCurrentColor和iCurrentFigure中。如果DialogBox传回FALSE，则主窗口继续使用iCurrentColor和iCurrentFigure的原始设定。

TRUE和FALSE通常用于EndDialog呼叫中，以告知主窗口消息处理程序使用者是用「OK」还是用「Cancel」来终止对话框的。不过，EndDialog的参数实际上是一个int值，而DialogBox也传回一个int值。所以，用这种方法能比仅用TRUE或者FALSE传回更多的信息。

避免使用整体变量

在ABOUT2中使用整体变量可能会、也可能不会影响您。一些程序写作者（包括我自己）较喜欢少用整体变量。ABOUT2中的整体变量iCurrentColor和iCurrentFigure看来使用得完全合法，因为它们必须同时在窗口消息处理程序和对话框程序中使用。不过，在一个有一大堆对话框的程序中，每个对话框都可能改变一堆变量的值，使整体变量的数量容易用得过多。

您可能更喜欢将程序中的对话框与数据结构相联系，该数据结构含有对话框可以改变的所有变量。您将在typedef叙述中定义这些结构。例如，在ABOUT2中，可以定义与「About」方块相联系的结构：

```
typedef struct
{
  int iColor, iFigure ;
}
ABOUTBOX_DATA ;
```

在WndProc中，您可以依据此结构来定义并初始化一个静态变量：

static ABOUTBOX_DATA ad = { IDC_BLACK, IDC_RECT } ;

在WndProc中也是这样，用ad.iColor和ad.iFigure替换了所有的iCurrentColor和iCurrentFigure。呼叫对话框时，使用DialogBoxParam而不用DialogBox。此函数的第五个参数可以是任意的32位值。一般来说，此值设定为指向一个结构的指针，在这里是WndProc中的ABOUTBOX_DATA结构。

```
case IDM_ABOUT:
  if (DialogBoxParam (hInstance, TEXT ("AboutBox"),
    hwnd, AboutDlgProc, &ad))
    InvalidateRect (hwnd, NULL, TRUE) ;
  return 0 ;
```

这是关键：DialogBoxParam的最后一个参数是作为WM_INITDIALOG消息中的lParam传递给对话框程序的。

对话框程序有两个ABOUTBOX_DATA结构型态的静态变量（一个结构和一个指向结构的指针）：

static ABOUTBOX_DATA ad, * pad ;

在AboutDlgProc中，此定义代替了iColor和iFigure的定义。在WM_INITDIALOG消息的开始部分，对话框程序根据lParam设定了这两个变量的值：

pad = (ABOUTBOX_DATA *) lParam ;

ad = * pad ;

第一道叙述中，pad设定为lParam的指标。亦即，pad实际是指向在WndProc定义的ABOUTBOX_DATA结构。第二个参数完成了从WndProc中的结构，到DlgProc中的区域结构的字段对字段内容复制。

现在，除了使用者按下「OK」按钮时所用的程序代码以之外，所有的AboutDlgProc都用ad.iColor和ad.iFigure替换了iFigure和iColor。这时，将区域结构的内容复制回WndProc中的结构：

```
case IDOK:
  * pad = ad ;
  EndDialog (hDlg, TRUE) ;
  return TRUE ;
```

Tab停留和分组

在第九章，我们利用窗口子类别化为COLORS1增加功能，使我们能够按下Tab键从一个滚动条转移到另一个滚动条。在对话框中，窗口子类别化是不必要的，因为Windows完成了将输入焦点从一个控件移动到另一个控件的所有工作。尽管如此，您必须在对话框模板中使用WS_TABSTOP和WS_GROUP窗口样式达到此目的。对于所有想要使用Tab键存取的控件，都要在其窗口样式中指定WS_TABSTOP。

如果参阅表11-1，您就会注意到许多控件将WS_TABSTOP定义为内定样式，其它一些则没有将它作为内定样式。一般而言，不包含WS_TABSTOP样式的控件（特别是静态控件）不应该取得输入焦点，因为即使有了输入焦点，它们也不能完成操作。除非在处理WM_INITDIALOG消息时您将输入焦点设定给一个特定的控件，并从消息中传回FALSE。否则Windows将输入焦点设定为对话框内第一个具有WS_TABSTOP样式的控件。

Windows给对话框增加的第二个键盘接口包括光标移动键，这种接口对于单选按钮有特殊的重要性。如果您使用Tab键移动到某一组内目前选中的单选按钮，那么，就需要使用光标移动键，将输入焦点从该单选按钮移动到组内其它单选按钮上。使用WS_GROUP窗口样式即可获得这个功能。

对于对话框模板中的特定控件序列，Windows将使用光标移动键把输入焦点从第一个具有WS_GROUP样式的控制权切换到下一个具有WS_GROUP样式的控件中。如果有必要，Windows将从对话框的最后一个控件循环到第一个控件，以便找到分组的结尾。

在内定设定下，控件LTEXT、CTEXT、RTEXT和ICON包含有WS_GROUP样式，这种样式方便地标记了分组的结尾。您必须经常将WS_GROUP样式加到其它型态的控件中。

让我们来看一看ABOUT2.RC中的对话框模板。四个具有WS_TABSTOP样式的控件是每个组的第一个单选按钮（明显地包含）和两个按键（内定设定）。在第一次启动对话框时，您可以使用Tab键在这四个控件之间移动。

在每组单选按钮中，您可以使用光标移动键切换输入焦点并改变选中标记。例如， Color下拉式清单方块的第一个单选按钮（Black）和 Figure下拉式清单方块都具有WS_GROUP样式。这意味着您可以用光标移动键将焦点从「Black」单选按钮移动到 Figure分组方块中。类似的情形，Figure分组方块的第一个单选按钮（ Rectangle）和DEFPUSHBUTTON都具有WS_GROUP样式，所以您可以使用光标移动键在组内两个单选按钮 – Rectangle和Ellipse之间移动。两个按键都有WS_GROUP样式，以阻止光标移动键在按键具有输入焦点时起作用。

使用ABOUT2时，Windows的对话框管理器在两组单选按钮中完成一些相当复杂的处理。正如所预期的那样，处于单选按钮组内时，光标移动键切换输入焦点，并给对话框程序发送WM_COMMAND消息。但是，当您改变了组内选中的单选按钮时，Windows也给新选中的单选按钮设定了WS_TABSTOP样式。当您下一次使用Tab切换到这一组后，Windows将会把输入焦点设定为选中的单选按钮。

文字字段中的「&」将导致紧跟其后的字母以底线显示，这就增加了另一种键盘接口，您可以通过按底线字母来将输入焦点移动到任意单选按钮上。透过按下C（代表 Color下拉式清单方块）或者F（代表Figure下拉式清单方块），您可以将输入焦点移动到相对应组内目前选中的单选按钮上。

尽管程序写作者通常让对话框管理器来完成这些工作，但是Windows提供了两个函数，以便程序写作者找寻下一个或者前一个Tab键停留项或者组项。这些函数为：

```
hwndCtrl = GetNextDlgTabItem (hDlg, hwndCtrl, bPrevious) ;
```

和

```
hwndCtrl = GetNextDlgGroupItem (hDlg, hwndCtrl, bPrevious) ;
```

如果bPrevious为TRUE，那么函数传回前一个Tab键停留项或组项；如果为FALSE，则传回下一个Tab键停留项或者组项。

在对话框上画图

ABOUT2还完成了一些相对说来很特别的事情，亦即在对话框上画图。让我们来看一看它是怎样做的。在ABOUT2.RC的对话框模板内，使用位置和大小为我们想要画图的区域定义了一块空白文字控件：

```
LTEXT    ""    IDC_PAINT, 114, 67, 72, 72
```

这个区域为18个字符宽和9个字符高。由于这个控件没有文字，所以窗口消息处理程序为「静态」类别所做的工作，只是在必须重绘这个子窗口控件时清除其背景。

在目前颜色或图形选择发生改变，或者对话框自身获得一个WM_PAINT消息时，对话框过程调用PaintTheBlock，这个函数在ABOUT2.C中：

```
PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
```

在AboutDlgProc中，窗口句柄hCtrlBlock已经在处理WM_INITDIALOG消息时被设定：

hCtrlBlock = GetDlgItem (hDlg, IDD_PAINT) ;

下面是PaintTheBlock函数：

```
void PaintTheBlock (HWND hCtrl, int iColor, int iFigure)
{
  InvalidateRect (hCtrl, NULL, TRUE) ;
  UpdateWindow (hCtrl) ;
  PaintWindow (hCtrl, iColor, iFigure) ;
}
```

这个函数使得子窗口控件无效，并为控件窗口消息处理程序产生一个WM_PAINT消息，然后呼叫ABOUT2中的另一个函数PaintWindow 。

PaintWindow函数取得一个设备内容句柄，并将其放到hCtrl中，画出所选图形，根据所选颜色用一个着色画刷填入图形。子窗口控件的大小从GetClientRect获得。尽管对话框模板以字符为单位定义了控件的大小，但GetClientRect取得以像素为单位的尺寸。您也可以使用函数MapDialogRect将对话框中的字符坐标转换为显示区域中的像素坐标。

我们并非真的绘制了对话框的显示区域，实际绘制的是子窗口控件的显示区域。每当对话框得到一个WM_PAINT消息时，就令子窗口控件的显示区域失效，并更新它，使它确信现在其显示区域又有效了，然后在其上画图。

将其它函数用于对话框

大多数可以用在子窗口的函数也可以用于对话框中的控件。例如，如果您想捣乱的话，那么可以使用MoveWindow在对话框内移动控件，强迫使用者用鼠标来追踪它们。

有时，您需要根据其它控件的设定，动态地启用或者禁用某些控件，这需要呼叫：

EnableWindow (hwndCtrl, bEnable) ;

当bEnable为TRUE（非0）时，它启用控件；当bEnable为FALSE（0）时，它禁用控件。在控件被禁用时，它不再接收键盘或者鼠标输入。您不能禁用一个拥有输入焦点的控件。

定义自己的控件

尽管Windows承揽了许多维护对话框和子窗口控件的工作，它同时也为您提供了各种加入程序代码的方法。前面我们已经看到了在对话框上绘图的方法。您也可以使用第九章中讨论的窗口子类别化来改变子窗口控件的操作。

您还可以定义自己的子窗口控件，并将它们用到对话框中。例如，假定您特别不喜欢普通的矩形按键，而倾向于建立椭圆形按键，那么您可以通过注册一个窗口类别，并使用自己编写的窗口消息处理程序处理来自您所建立窗口的消息，从而建立椭圆形按键。在Developer Studio中，您可以在与自订控件相联系的「Properties」对话框中指定这个窗口类别，这将转换成对话框模板中的CONTROL叙述。程序11-3所示的ABOUT3程序正是这样做的。

程序11-3 ABOUT3

ABOUT3.C

```
/*---------------------------------------------------------------------------
ABOUT3.C -- About Box Demo Program No. 3
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;
```

```
LRESULT CALLBACK EllipPushWndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("About3") ;
  MSG msg ;
  HWND hwnd ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = EllipPushWndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = NULL ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) (COLOR_BTNFACE + 1) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = TEXT ("EllipPush") ;

  RegisterClass (&wndclass) ;
  hwnd = CreateWindow ( szAppName, TEXT ("About Box Demo Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HINSTANCE hInstance ;
  switch (message)
  {
  case WM_CREATE :
    hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
    return 0 ;
  case WM_COMMAND :
    switch (LOWORD (wParam))
    {
    case IDM_APP_ABOUT :
```

```
     DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
     return 0 ;
    }
    break ;
    case WM_DESTROY :
     PostQuitMessage (0) ;
     return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG :
    return TRUE ;
  case WM_COMMAND :
    switch (LOWORD (wParam))
    {
    case IDOK :
     EndDialog (hDlg, 0) ;
     return TRUE ;
    }
    break ;
  }
  return FALSE ;
}

LRESULT CALLBACK EllipPushWndProc (HWND hwnd, UINT message,WPARAM wParam, LPARAM lParam)
{
  TCHAR szText[40] ;
  HBRUSH hBrush ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_PAINT :
    GetClientRect (hwnd, &rect) ;
    GetWindowText (hwnd, szText, sizeof (szText)) ;

    hdc = BeginPaint (hwnd, &ps) ;

    hBrush = CreateSolidBrush (GetSysColor (COLOR_WINDOW)) ;
    hBrush = (HBRUSH) SelectObject (hdc, hBrush) ;
    SetBkColor (hdc, GetSysColor (COLOR_WINDOW)) ;
    SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT)) ;

    Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;
    DrawText (hdc, szText, -1, &rect,
     DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

    DeleteObject (SelectObject (hdc, hBrush)) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_KEYUP :
    if (wParam != VK_SPACE)
     break ;// fall through
  case WM_LBUTTONUP :
    SendMessage (GetParent (hwnd), WM_COMMAND,
     GetWindowLong (hwnd, GWL_ID), (LPARAM) hwnd) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

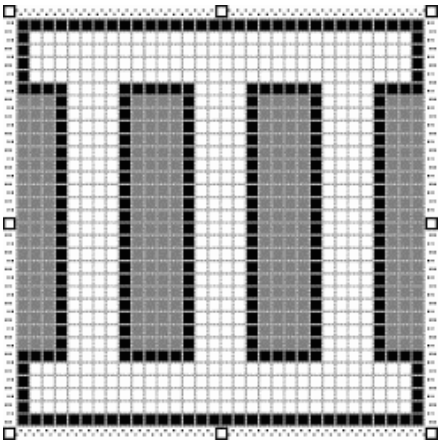
ABOUT3.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
CONTROL  "OK",IDOK,"EllipPush",WS_GROUP | WS_TABSTOP,73,79,32,14
ICON "ABOUT3",IDC_STATIC,7,7,20,20
CTEXT "About3",IDC_STATIC,40,12,100,8
CTEXT "About Box Demo Program",IDC_STATIC,7,40,166,8
CTEXT  "(c) Charles Petzold, 1998",IDC_STATIC,7,52,166,8
END

////////////////////////////////////////////////////////////////////////////
// Menu
ABOUT3 MENU DISCARDABLE
BEGIN
POPUP "&Help"
BEGIN
MENUITEM "&About About3...",   IDM_APP_ABOUT
END
END

////////////////////////////////////////////////////////////////////////////
// Icon
ABOUT3 ICON DISCARDABLE "icon1.ico"
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by About3.rc
#define IDM_APP_ABOUT  40001
#define IDC_STATIC  -1

#define IDM_APP_ABOUT 40001
#define IDC_STATIC -1
```

ABOUT3.ICO



我们所注册的窗口类别叫做「EllipPush」（椭圆形按键）。在Developer Studio的对话框编辑器中，删除「Cancel」和「OK」按钮。要添加依据此窗口类别的控件，请从「 Controls」工具列选择「Custom Control」。在此控件的「Properties」对话框的「 Class」字段输入「EllipPush」。在对话框模板中我们没有使用DEFPUSHBUTTON叙述，而是用CONTROL叙述来指定此窗口类别：

CONTROL "OK" IDOK, "EllipPush", TABGRP, 64, 60, 32, 14

当在对话框中建立子窗口控件时，对话框管理器把这个窗口类别用于CreateWindow呼叫中。

ABOUT3.C程序在WinMain中注册了EllipPush窗口类别：

```
wndclass.style = CS_HREDRAW | CS_VREDRAW ;
wndclass.lpfnWndProc = EllipPushWndProc ;
wndclass.cbClsExtra = 0 ;
wndclass.cbWndExtra = 0 ;
wndclass.hInstance = hInstance ;
wndclass.hIcon = NULL ;
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
wndclass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1) ;
wndclass.lpszMenuName = NULL ;
wndclass.lpszClassName = TEXT ("EllipPush") ;
RegisterClass (&wndclass) ;
```

该窗口类别指定窗口消息处理程序为EllipPushWndProc，在ABOUT3.C中正是这样。

EllipPushWndProc窗口消息处理程序只处理三种消息：WM_PAINT、WM_KEYUP和WM_LBUTTONUP。在处理WM_PAINT消息时，它从GetClientRect中取得窗口的大小，从GetWindowText中取得显示在按键上的文字，用Windows函数Ellipse和DrawText来输出椭圆和文字。

WM_KEYUP和WM_LBUTTONUP消息的处理非常简单：

```
case WM_KEYUP :
  if (wParam != VK_SPACE)
    break ; // fall through
case WM_LBUTTONUP :
  SendMessage (GetParent (hwnd), WM_COMMAND,
    GetWindowLong (hwnd, GWL_ID), (LPARAM) hwnd) ;
  return 0 ;
```

窗口消息处理程序使用GetParent来取得其父窗口（即对话框）的句柄，并发送一个WM_COMMAND消息，消息的wParam等于控件的ID，这个ID是用GetWindowLong取得的。然后，对话框窗口消息处理程序将这个消息传给ABOUT3内的对话框程序，结果得到一个使用者自订的按键，如图11-3所示。您可以用同样的方法来建立其它自订对话框控件。

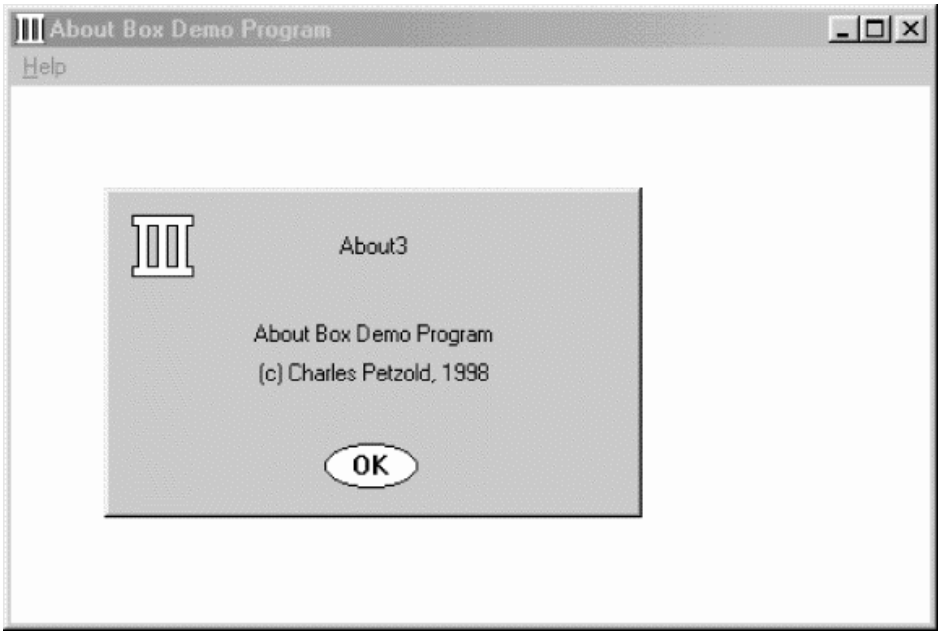


图11-3 ABOUT3建立的自订按键

这就是全部要做的吗？其实不然。通常，对于维护子窗口控件所需要的处理而言，EllipPushWndProc只是一个空架子。例如，按钮不会像普通的按键那样闪烁。要翻转按键内的颜色，窗口消息处理程序必须处理WM_KEYDOWN（来自空格键）和WM_LBUTTONDOWN消息。窗口消息处理程序还必须在收到WM_LBUTTONDOWN消息时拦截鼠标，并且，如果当按钮还处于按下状态，而鼠标移到了子窗口的显示区域之外，那么得要释放鼠标拦截（并将按钮的内部颜色回复为正常状态）。只有在鼠标被拦截时松开该按钮，子窗口才会给其父窗口送回一个WM_COMMAND消息。

EllipPushWndProc也不处理WM_ENABLE消息。如上所述，对话框程序可以使用EnableWindow函数来禁用某窗口。于是，子窗口将显示灰色文字，而不再是黑色文字，以表示它已经被禁用，并且不能再接收任何消息了。

如果子窗口控件的窗口消息处理程序需要为所建立的每个窗口存放各自不同的数据，那么它可以通过使用窗口类别结构中的cbWndExtra值来做到。这样就在内部窗口结构中保留了空间，并可以用SetWindowLong和GetWindowLong来存取该数据。

非模态对话框

在本章的开始，我曾经说过对话框分为「模态的」和「非模态的」两种。现在我们已经研究过这两种对话框中最常见的一种 – 模态对话框。模态对话框（不包括系统模态对话框）。允许使用者在对话框与其它程序之间进行切换。但是，使用者不能切换到同一程序的另一个窗口，直到模态对话框被清除为止。非模态对话框允许使用者在对话框与其它程序之间进行切换，又可以在对话框与建立对话框的窗口之间进行切换。因此，非模态对话框与使用者程序常见的普通弹出式窗口可能更为相似。

当使用者觉得让对话框保留片刻会更加方便时，使用非模态对话框是合适的。例如，文书处理程序经常使用非模态对话框来进行「Find」和「Change」操作。如果「Find」对话框是模态的，那么使用者必须从菜单中选择「Find」，然后输入要寻找的字符串，结束对话框，传回到文件中，接着再重复整个程序来寻找同一字符串的另一次出现。允许使用者在文件与对话框之间进行切换则会方便得多。

您已经看到，模态对话框是用DialogBox来建立的。只有在清除对话框之后，函数才会传回值。在对话框程序内使用EndDialog呼叫来终止对话框，DialogBox传回的是该呼叫的第二个参数的值。非模态对话框是使用CreateDialog来建立的，该函数所使用的参数与DialogBox相同。

```
hDlgModeless = CreateDialog (       hInstance, szTemplate,
                                    hwndParent, DialogProc) ;
```

区别是CreateDialog函数立即传回对话框的窗口句柄，并通常将这个窗口句柄存放到整体变量中。

尽管将DialogBox这一名字用于模态对话框而CreateDialog用于非模态对话框是随意的，但是您可以通过非模态对话框与普通窗口类似这一点来记住这两个函数的区别。CreateDialog可以令人想起CreateWindow函数来，而后者建立的是普通窗口。

模态对话框与非模态对话框的区别

使用非模态对话框与使用模态对话框相似，但是也有一些重要的区别：

首先，非模态对话框通常包含一个标题栏和一个系统菜单按钮。当您在Developer Studio中建立对话框时，这些是内定选项。用于非模态对话框的对话框模板中的STYLE叙述形如：

STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE

标题栏和系统菜单允许使用者，使用鼠标或者键盘将非模态对话框移动到另一个显示区域。对于模态对话框，您通常无须提供标题栏和系统菜单，因为使用者不能在其下面的窗口中做任何其它的事情。

第二项重要的区别是：注意，在我们的范例STYLE叙述中包含有WS_VISIBLE样式。在 Developer Studio中，从「Dialog Properties」对话框的「More Styles」页面卷标中选择此选项。如果省略了WS_VISIBLE，那么您必须在CreateDialog呼叫之后呼叫ShowWindow：

hDlgModeless = CreateDialog (. . .) ;

ShowWindow (hDlgModeless, SW_SHOW) ;

如果您既没有包含WS_VISIBLE样式，又没有呼叫ShowWindow，那么非模态对话框将不会被显示。如果忽略这个事实，那么习惯于模态对话框的程序写作者在第一次试图建立非模态对话框时，经常会出现问题。

第三项区别：与模态对话框和消息框的消息不同，非模态对话框的消息要经过程序式的消息队列。要将这些消息传送给对话框窗口消息处理程序，则必须改变消息队列。方法如下：当您使用CreateDialog建立非模态对话框时，应该将从呼叫中传回的对话框句柄储存在一个整体变量（如hDlgModeless）中，并将消息循环改变为：

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

如果消息是发送给非模态对话框的，那么IsDialogMessage将它发送给对话框中窗口消息处理程序，并传回TRUE（非0）；否则，它将传回FALSE（0）。只有hDlgModeless为0或者消息不是该对话框的消息时，才必须呼叫TranslateMessage和DispatchMessage函数。如果您将键盘快捷键用于您的程序窗口，那么消息循环将如下所示：

```
while (GetMessage (&msg, NULL, 0, 0))
{
  if (hDlgModeless == 0 || !IsDialogMessage (hDlgModeless, &msg))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
}
```

由于整体变量被初始化为0，所以hDlgModeless将为0，直到建立对话框为止，从而保证不会使用无效的窗口句柄来呼叫IsDialogMessage。在清除非模态对话框时，您也必须注意这一点，正如最后一点所说明的。

hDlgModeless变量也可以由程序的其它部分使用，以便对非模态对话框是否存在加以验证。例如，程序中的其它窗口可以在hDlgModeless不等于0时给对话框发送消息。

最后一项重要的区别：使用DestroyWindow而不是EndDialog来结束非模态对话框。当您呼

叫DestroyWindow后，将hDlgModeless整体变量设定为0。

使用者习惯于从系统菜单中选择「Close」来结束非模态对话框。尽管启用了「Close」选项，Windows内的对话框窗口消息处理程序并不处理WM_CLOSE消息。您必须自己在对话框程序中处理它：

```
case WM_CLOSE :
  DestroyWindow (hDlg) ;
  hDlgModeless = NULL ;
  break ;
```

注意这两个窗口句柄之间的区别：DestroyWindow的hDlg参数是传递给对话框程序的参数；hDlgModeless是从CreateDialog传回的整体变量，程序在消息循环内检验它。

您也可以允许使用者使用按键来关闭非模态对话框，处理方式与处理WM_CLOSE消息一样。对话框必须传回给建立它的窗口之任何数据都可以储存在整体变量中。如果不喜欢使用整体变量，那么您也可以用CreateDialogParam来建立非模态对话框，并按前面介绍的方法让它储存一个结构指针。

新的COLORS程序

第九章中所描述的COLORS1程序建立了九个子窗口，以便显示三个滚动条和六个文字项。那时候，这个程序还是我们所写过的程序中相当复杂的一个。如果将COLORS1转换为使用非模态对话框则会使程序 – 特别是WndProc函数 – 变得令人难以置信的简单，修正后的COLORS2程序如程序11-4所示。

程序11-4 COLORS2

COLORS2.C

```
/*------------------------------------------------------------------------
COLORS2.C -- Version using Modeless Dialog Box
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK ColorScrDlg (HWND, UINT, WPARAM, LPARAM) ;
HWND hDlgModeless ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Colors2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = CreateSolidBrush (0L) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
```

```
  hwnd = CreateWindow (szAppName, TEXT ("Color Scroll"),
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hDlgModeless = CreateDialog (hInstance, TEXT ("ColorScrDlg"),
    hwnd, ColorScrDlg) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (hDlgModeless == 0 || !IsDialogMessage (hDlgModeless, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  switch (message)
  {
  case WM_DESTROY :
    DeleteObject ((HGDIOBJ) SetClassLong (hwnd, GCL_HBRBACKGROUND,
      (LONG) GetStockObject (WHITE_BRUSH))) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK ColorScrDlg (HWND hDlg, UINT message,WPARAM wParam, LPARAM lParam)
{
  static int iColor[3] ;
  HWND hwndParent, hCtrl ;
  int iCtrlID, iIndex ;

  switch (message)
  {
  case WM_INITDIALOG :
    for (iCtrlID = 10 ; iCtrlID < 13 ; iCtrlID++)
    {
      hCtrl = GetDlgItem (hDlg, iCtrlID) ;
      SetScrollRange (hCtrl, SB_CTL, 0, 255, FALSE) ;
      SetScrollPos (hCtrl, SB_CTL, 0, FALSE) ;
    }
    return TRUE ;
  case WM_VSCROLL :
    hCtrl = (HWND) lParam ;
    iCtrlID = GetWindowLong (hCtrl, GWL_ID) ;
    iIndex = iCtrlID - 10 ;
    hwndParent = GetParent (hDlg) ;

    switch (LOWORD (wParam))
    {
    case SB_PAGEDOWN :
      iColor[iIndex] += 15 ; // fall through
    case SB_LINEDOWN :
      iColor[iIndex] = min (255, iColor[iIndex] + 1) ;
      break ;
    case SB_PAGEUP :
      iColor[iIndex] -= 15 ; // fall through
    case SB_LINEUP :
      iColor[iIndex] = max (0, iColor[iIndex] - 1) ;
      break ;
```

```
    case SB_TOP :
      iColor[iIndex] = 0 ;
      break ;
    case SB_BOTTOM :
      iColor[iIndex] = 255 ;
      break ;
    case SB_THUMBPOSITION :
    case SB_THUMBTRACK :
      iColor[iIndex] = HIWORD (wParam) ;
      break ;
    default :
      return FALSE ;
    }
    SetScrollPos (hCtrl, SB_CTL, iColor[iIndex], TRUE) ;
    SetDlgItemInt (hDlg, iCtrlID + 3, iColor[iIndex], FALSE) ;

    DeleteObject ((HGDIOBJ) SetClassLong (hwndParent, GCL_HBRBACKGROUND,
      (LONG) CreateSolidBrush (
      RGB (iColor[0], iColor[1], iColor[2])))) ;

    InvalidateRect (hwndParent, NULL, TRUE) ;
    return TRUE ;
  }
  return FALSE ;
}
```

COLORS2.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Dialog
COLORSCRDLG DIALOG DISCARDABLE 16, 16, 120, 141
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Color Scroll Scrollbars"
FONT 8, "MS Sans Serif"
BEGIN
CTEXT     "&Red",IDC_STATIC,8,8,24,8,NOT WS_GROUP
SCROLLBAR    10,8,20,24,100,SBS_VERT | WS_TABSTOP
CTEXT    "0",13,8,124,24,8,NOT WS_GROUP
CTEXT     "&Green",IDC_STATIC,48,8,24,8,NOT WS_GROUP
SCROLLBAR    11,48,20,24,100,SBS_VERT | WS_TABSTOP
CTEXT    "0",14,48,124,24,8,NOT WS_GROUP
CTEXT     "&Blue",IDC_STATIC,89,8,24,8,NOT WS_GROUP
SCROLLBAR    12,89,20,24,100,SBS_VERT | WS_TABSTOP
CTEXT    "0",15,89,124,24,8,NOT WS_GROUP
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Colors2.rc
#define IDC_STATIC   -1
```

原来的COLORS1程序所显示的滚动条大小是依据窗口大小决定的，而新程序在非模态对话框内以固定的尺寸来显示它们，如图11-4所示。

当您建立对话框模板时，直接将三个滚动条的ID分别设为10、11和12，将显示滚动条目前值的三个静态文字字段的ID分别设为13、14和15。将每个滚动条都设定为Tab Stop样式，而从所有的六个静态文字字段中删除Group样式。

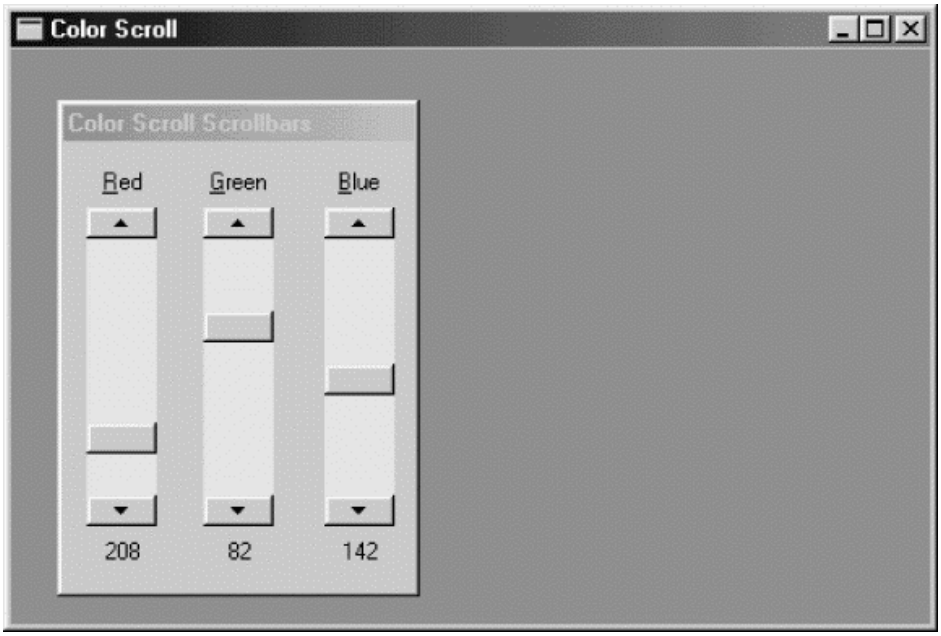


图11-4 COLORS2的屏幕显示

在COLORS2中，非模态对话框是在WinMain函数里建立的，紧跟在程序主窗口的ShowWindow呼叫之后。注意，主窗口的窗口样式包含WS_CLIPCHILDREN，这允许程序无须擦除对话框就能够重画主窗口。

如上所述，从CreateDialog传回的对话框窗口句柄存放在整体变量hDlgModeless中，并在消息循环中被测试。不过，在这个程序中，不需要将句柄存放在整体变量中，也不需要在呼叫IsDialogMessage之前测试这个值。消息循环可以编写如下：

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

由于对话框是在程序进入消息循环前建立，并且直到程序结束时才会被清除，所以hDlgModeless的值将总是有效的。我加入了如下的处理方式，以便您可能会往对话框的窗口消息处理程序中加入一段清除对话框的程序代码：

```
case WM_CLOSE :
  DestroyWindow (hDlg) ;
  hDlgModeless = NULL ;
  break ;
```

在原来的COLORS1程序中，SetWindowText在使用wsprintf将三个数值卷标转换为文字之后才设定它们的值。叙述为：

wsprintf (szBuffer, TEXT ("%i"), color[i]) ;

SetWindowText (hwndValue[i], szBuffer) ;

i的值为目前处理的滚动条的ID，hwndValue是一个数组，它包含颜色数值的三个静态文字子窗口的窗口句柄。

新版本使用SetDlgItemInt为每个子窗口的每个文字字段设定一个号码：

SetDlgItemInt (hDlg, iCtrlID + 3, color [iCtrlID], FALSE) ;

尽管SetDlgItemInt和与其对应的GetDlgItemInt在编辑控件中用得最多，它们也可以用来设定其它控件的文字字段，如静态文字控件等。iCtrlID变量是滚动条的ID，给ID加上3使之变成对应数字卷标的ID。第三个参数是颜色值。通常，第四个参数表示第三个参数的值是解释为有正负号的（第四个参数为TRUE）还是无正负号的（第四个参数为FALSE）。但是，对于这个程序，值的范围是从0到256，所以这个参数没有意义。

在将COLORS1转换为COLORS2的程序中，我们把越来越多的工作交给了Windows。旧版本呼叫了CreateWindow 10次；而新版本只呼叫了CreateWindow和CreateDialog各一次。但是，如果您认为我们已经把呼叫CreateWindow的次数降到最少，那么您就错了，请看下一个程序。

HEXCALC：窗口还是对话框?

HEXCALC程序可能是写程序偷懒的经典之作，如程序11-5所示。这个程序完全不呼叫CreateWindow，也不处理WM_PAINT消息，不取得设备内容，也不处理鼠标消息。但是它只用了不到150行的原始码，就构成了一个具有完整键盘和鼠标接口以及10种运算的十六进制计算器。计算器如图11-5所示。

程序11-5 HEXCALC

HEXCALC.C

```
/*------------------------------------------------------------------------
HEXCALC.C -- Hexadecimal Calculator
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("HexCalc") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = DLGWINDOWEXTRA ; // Note!
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) (COLOR_BTNFACE + 1) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateDialog (hInstance, szAppName, 0, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
```

```
void ShowNumber (HWND hwnd, UINT iNumber)
{
  TCHAR szBuffer[20] ;
  wsprintf (szBuffer, TEXT ("%X"), iNumber) ;
  SetDlgItemText (hwnd, VK_ESCAPE, szBuffer) ;
}

DWORD CalcIt (UINT iFirstNum, int iOperation, UINT iNum)
{
  switch (iOperation)
  {
  case '=': return iNum ;
  case '+': return iFirstNum + iNum ;
  case '-': return iFirstNum - iNum ;
  case '*': return iFirstNum * iNum ;
  case '&': return iFirstNum & iNum ;
  case '|': return iFirstNum | iNum ;
  case '^': return iFirstNum ^ iNum ;
  case '<': return iFirstNum << iNum ;
  case '>': return iFirstNum >> iNum ;
  case '/': return iNum ? iFirstNum / iNum: MAXDWORD ;
  case '%': return iNum ? iFirstNum % iNum: MAXDWORD ;
  default : return 0 ;
  }
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL bNewNumber = TRUE ;
  static int iOperation = '=' ;
  static UINT iNumber, iFirstNum ;
  HWND hButton ;

  switch (message)
  {
  case WM_KEYDOWN: // left arrow --> backspace
    if (wParam != VK_LEFT)
      break ;
    wParam = VK_BACK ;
    // fall through
  case WM_CHAR:
    if ((wParam = (WPARAM) CharUpper ((TCHAR *) wParam)) == VK_RETURN)
      wParam = '=' ;

    if (hButton = GetDlgItem (hwnd, wParam))
    {
      SendMessage (hButton, BM_SETSTATE, 1, 0) ;
      Sleep (100) ;
      SendMessage (hButton, BM_SETSTATE, 0, 0) ;
    }
    else
    {
      MessageBeep (0) ;
      break ;
    }
    // fall through
  case WM_COMMAND:
    SetFocus (hwnd) ;

    if (LOWORD (wParam) == VK_BACK) //backspace
      ShowNumber (hwnd, iNumber /= 16) ;

    else if (LOWORD (wParam) == VK_ESCAPE) // escape
      ShowNumber (hwnd, iNumber = 0) ;

    else if (isxdigit (LOWORD (wParam))) // hex digit
    {
      if (bNewNumber)
```

```
    {
      iFirstNum = iNumber ;
      iNumber = 0 ;
    }
    bNewNumber = FALSE ;
    if (iNumber <= MAXDWORD >> 4)
      ShowNumber (hwnd, iNumber = 16 * iNumber + wParam -
      (isdigit (wParam) ? '0': 'A' - 10)) ;
    else
      MessageBeep (0) ;
  }
  else // operation
  {
    if (!bNewNumber)
      ShowNumber (hwnd, iNumber =
      CalcIt (iFirstNum, iOperation, iNumber)) ;
    bNewNumber = TRUE ;
    iOperation = LOWORD (wParam) ;
  }
  return 0 ;
 case WM_DESTROY:
  PostQuitMessage (0) ;
  return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

HEXCALC.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Icon
HEXCALC    ICON DISCARDABLE   "HexCalc.ico"

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

#include "hexcalc.dlg"
```

HEXCALC.DLG

```
/*------------------------------
HEXCALC.DLG dialog script
--------------------------------*/
HexCalc DIALOG -1, -1, 102, 122
STYLE WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CLASS "HexCalc"
CAPTION "Hex Calculator"
{
 PUSHBUTTON "D",     68, 8, 24, 14, 14
  PUSHBUTTON "A",     65, 8, 40, 14, 14
  PUSHBUTTON "7",     55, 8, 56, 14, 14
  PUSHBUTTON "4",   52, 8, 72, 14, 14
  PUSHBUTTON "1",       49, 8, 88, 14, 14
  PUSHBUTTON "0",     48, 8, 104,14, 14
  PUSHBUTTON "0",    27, 26, 4, 50, 14
  PUSHBUTTON "E",    69, 26, 24, 14, 14
  PUSHBUTTON "B",    66, 26, 40, 14, 14
  PUSHBUTTON "8",       56, 26, 56, 14, 14
  PUSHBUTTON "5",    53, 26, 72, 14, 14
  PUSHBUTTON "2",    50, 26, 88, 14, 14
  PUSHBUTTON "Back",  8,  26, 104,32, 14
  PUSHBUTTON "C",    67, 44, 40, 14, 14
  PUSHBUTTON "F",    70, 44, 24, 14, 14
  PUSHBUTTON "9",     57, 44, 56, 14, 14
  PUSHBUTTON "6",     54, 44, 72, 14, 14
```

```
    PUSHBUTTON "3",     51, 44, 88, 14, 14
    PUSHBUTTON "+",     43, 62, 24, 14, 14
    PUSHBUTTON "-",     45, 62, 40, 14, 14
    PUSHBUTTON "*",     42, 62, 56, 14, 14
    PUSHBUTTON "/",     47, 62, 72, 14, 14
    PUSHBUTTON "%",     37, 62, 88, 14, 14
    PUSHBUTTON "Equals",  61, 62, 104,32, 14
    PUSHBUTTON "&&",38, 80, 24, 14, 14
    PUSHBUTTON "|",   124, 80, 40, 14, 14
    PUSHBUTTON "^",     94, 80, 56, 14, 14
    PUSHBUTTON "<",   60, 80, 72, 14, 14
    PUSHBUTTON ">",   62, 80, 88, 14, 14
}
```

HEXCALC.ICO

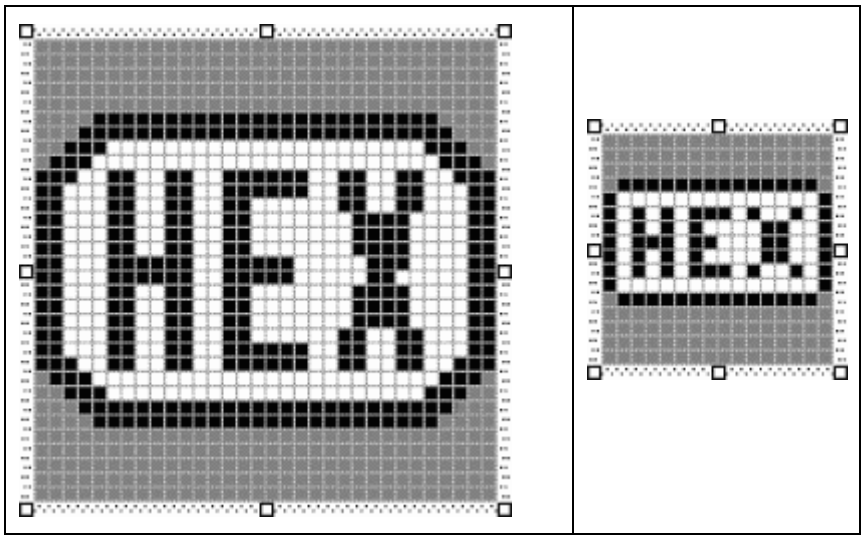


图11-5 HEXCALC的屏幕显示

HEXCALC是一个普通的中序表达式计算器，使用C语言的符号表示方式进行计算。它对无正负号32位整数作加、减、乘、除和取余数运算，位AND, OR, exclusive-OR运算，还有左右位移运算。被0除将导致结果被设定为FFFFFFFF。

在HEXCALC中既可以使用鼠标又可以使用键盘。您从按键点入」或者输入第一个数（最多8位十六进制数字）开始，然后输入运算子，然后是第二个数。接着，您可以透过单击「Equals」按钮或者按下等号键或Enter键便可以显示运算结果。为了更正输入，您可以使用「Back」按钮、Backspace或者左箭头键。单击「display」方块或者按下Esc键即可清除目前的输入。

HEXCALC比较奇怪的一点是，屏幕上显示的窗口似乎是普通的重迭式窗口与非模态对话框的混合体。一方面，HEXCALC的所有消息都在函数的WndProc中处理，这个函数与通常的窗口消息处理程序相似，该函数传回一个长整数，它处理WM_DESTROY消息，呼叫DefWindowProc，就像普通的窗口消息处理程序一样。另一方面，窗口是在WinMain中呼叫CreateDialog并使用HEXCALC.DLG中的对话框模板建立的。那么，HEXCALC到底是一个普通的可重迭窗口，还是一个非模态对话框呢?

简单的回答是，对话框就是窗口。通常，Windows使用它自己内部的窗口消息处理程序处理对话框窗口的消息，然后，Windows将这些消息传送给建立对话框的程序内的对话框程序。在HEXCALC中，我们让Windows使用对话框模板建立一个窗口，但是自己写程序处理这个窗口的消息。

不幸的是，在Developer Studio的Dialog Editor中，对话框模板需要一些我们不能添加的东西。因此，对话框模板包含在HEXCALC.DLG文件中，而且需要手工输入。依照下面的方法，您可以将一个文本文件添加到任何项目中：从「 File」菜单选择「New」，再选择「 Files」页面卷标，然后从文件型态列表中选择「Text File」。像这样的文件 – 包含附加资源定义 – 需要包含在资源描述中。从「 View」菜单选择「Resource Includes」。这显示一个对话框。在「Compile-time Directives」编辑栏输入

```
#include "hexcalc.dlg"
```

这一行将插入到HEXCALC.RC资源描述中，像上面所显示的一样。

仔细看一下HEXCALC.DLG文件中的对话框模板，您将发现HEXCALC如何为对话框使用它自己的窗口消息处理程序。对话框模板的上方如下：

```
HexCalc DIALOG -1, -1, 102, 122
STYLE WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CLASS "HexCalc"
CAPTION "Hex Calculator"
```

注意诸如WS_OVERLAPPED和WS_MINIMIZEBOX等标识符，我们可以将它们用在CreateWindow呼叫中以建立普通的窗口。CLASS叙述是这个对话框与曾经建立过的对话框之间最重要的区别（而且它也是Developer Studio中的Dialog Editor不允许我们指定的）。当对话框模板省略了这个叙述时，Windows为对话框注册一个窗口类别，并使用它自己的窗口消息处理程序处理对话框消息。这里，包含CLASS叙述就告诉Windows将消息发送到其它的地方 – 具体的说，就是发送到在HexCalc窗口类别中指定的窗口消息处理程序。

HexCalc窗口类别是在HEXCALC的WinMain函数中注册的，就像普通窗口的窗口类别一样。但是，请注意有个十分重要的区别：WNDCLASS结构的cbWndExtra字段设定为DLGWINDOWEXTRA。对于您自己注册的对话框程序，这是必需的。

在注册窗口类别之后，WinMain呼叫CreateDialog：

```
hwnd = CreateDialog (hInstance, szAppName, 0, NULL) ;
```

第二个参数（字符串「HexCaEc」）是对话框模板的名字。第三个参数通常是父窗口的窗口句柄，这里设定为0，因为窗口没有父窗口。最后一个参数，通常是对话框程序的地址，这里不需要。因为Windows不会处理这些消息，因而也不会将消息发送给对话框程序。

这个CreateDialog呼叫与对话框模板一起，被Windows有效地转换为一个CreateWindow呼叫。该CreateWindow呼叫的功能与下面的呼叫相同：

```
hwnd = CreateWindow (TEXT ("HexCalc"), TEXT ("Hex Calculator"),
                     WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
                     WS_MINIMIZEBOX,
                     CW_USEDEFAULT, CW_USEDEFAULT,
                     102 * 4 / cxChar, 122 * 8 / cyChar,
                     NULL, NULL, hInstance, NULL) ; ;
```

其中，cxChar和cyChar变量分别是系统字体字符的宽度和高度。

我们通过让Windows来进行CreateWindow呼叫而收获甚丰：Windows不会在建立弹出式窗口1后就停止，它还会为对话框模板中定义的其它29个子窗口按键控件呼叫CreateWindow。所有这些控件都给父窗口的窗口消息处理程序发送WM_COMMAND消息，该程序正是WndProc。对于建立一个包含许多子窗口的窗口来说，这是一个很好的技巧。

下面是使HEXCALC的程序代码量下降到最少的另一种方法：或许您会注意到HEXCALC没有表头文件，表头文件中通常包含对话框模板中，需要为所有子窗口控件定义的标识符。我们之所以可以不要这个文件，是因为每个按键控件的ID设定为出现在控件上的文字的ASCII码。这意味着，WndProc可以完全相同地对待WM_COMMAND消息和WM_CHAR消息。在每种情况下，wParam的低字组都是按钮的ASCII码。

当然，对键盘消息进行一些处理是必要的。WndProc拦截WM_KEYDOWN消息，将左箭头键转换为Backspace键。在处理WM_CHAR消息时，WndProc将字符代码转换为大写，Enter键转换为等号键的ASCII码。

WM_CHAR消息的有效性是通过呼叫GetDlgItem来检验的。如果GetDlgItem函数传回0，那么键盘字符不是对话框模板中定义的ID之一。如果字符是ID之一，则通过给相应的按钮发送一对BM_SETSTATE消息，来使之闪烁：

```
if (hButton = GetDlgItem (hwnd, wParam))
{
  SendMessage (hButton, BM_SETSTATE, 1, 0) ;
  Sleep (100) ;
  SendMessage (hButton, BM_SETSTATE, 0, 0) ;
}
```

这样做，用最小的代价，却为HEXCALC的键盘接口增色不少。Sleep函数将程序暂停100毫秒。这会防止按钮被按得太快而让人注意不到。

当WndProc处理WM_COMMAND消息时，它总是将输入焦点设定给父窗口：

```
case          WM_COMMAND :
              SetFocus (hwnd) ;
```

否则，一旦使用鼠标单击某按钮，输入焦点就会切换到该按钮上。

通用对话框

Windows的一个主要目的是推动标准的使用者接口。对许多常用的菜单项来说，这推行得很快，几乎所有软件厂商都采用Alt-File-Open选择来打开一个文件。然而，实际的文件开启对话框却经常

各不相同。

从Windows 3.1开始，对这个问题有了一个可行的解决方案，这是一种叫做「通用对话框链接库」的增强。这个链接库由几个函数组成，这些函数启动标准对话框来进行打开和储存文件、搜索和替换、选择颜色、选择字体（我将在本章讨论以上的这些内容）以及打印（我将在第十三章讨论）。

为了使用这些函数，您基本上都要初始化某一结构的各个字段，并将该结构的指针传送给通用对话框链接库的某个函数，该函数会建立并显示对话框。当使用者关闭对话框时，被呼叫的函数将控制权传回给程序，您可以从传送给它的结构中获得信息。

在使用通用对话框链接库的任何C原始码文件时，您都需要含入COMMDLG.H表头文件。通用对话框的文件在/Platform SDK/User Interface Services/User Input/Common Dialog Box Library中。

增强POPPAD

当我们往第十章的POPPAD中增加菜单时，还有几个标准菜单项没有实作。现在我们已经准备好在POPPAD中加入打开文件、读入文件以及在磁盘上储存编辑过文件的功能。在处理中，我们还将在POPPAD中加入字体选择和搜索替换功能。

实作POPPAD3程序的文件如程序11-6所示。

程序11-6 POPPAD3

POPPAD.C

```
/*----------------------------------------------------------------------------
POPPAD.C -- Popup Editor
(c) Charles Petzold, 1998
----------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "resource.h"

#define EDITID 1
#define UNTITLED TEXT ("(untitled)")

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutDlgProc (HWND, UINT, WPARAM, LPARAM) ;

// Functions in POPFILE.C

void PopFileInitialize (HWND) ;
BOOL PopFileOpenDlg (HWND, PTSTR, PTSTR) ;
BOOL PopFileSaveDlg (HWND, PTSTR, PTSTR) ;
BOOL PopFileRead (HWND, PTSTR) ;
BOOL PopFileWrite (HWND, PTSTR) ;

// Functions in POPFIND.C

HWND PopFindFindDlg (HWND) ;
HWND PopFindReplaceDlg (HWND) ;
BOOL PopFindFindText (HWND, int *, LPFINDREPLACE) ;
BOOL PopFindReplaceText (HWND, int *, LPFINDREPLACE) ;
BOOL PopFindNextText (HWND, int *) ;
BOOL PopFindValidFind (void) ;

// Functions in POPFONT.C

void PopFontInitialize (HWND) ;
BOOL PopFontChooseFont (HWND) ;
void PopFontSetFont (HWND) ;
void PopFontDeinitialize (void) ;
// Functions in POPPRNT.C
```

```
BOOL PopPrntPrintFile (HINSTANCE, HWND, HWND, PTSTR) ;

// Global variables

static HWND hDlgModeless ;
static TCHAR szAppName[] = TEXT ("PopPad") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  MSG msg ;
  HWND hwnd ;
  HACCEL hAccel ;
  WNDCLASS wndclass ;

  wndclass.style  = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc  = WndProc ;
  wndclass.cbClsExtra  = 0 ;
  wndclass.cbWndExtra  = 0 ;
  wndclass.hInstance  = hInstance ;
  wndclass.hIcon  = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor  = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, NULL,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, szCmdLine) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  hAccel = LoadAccelerators (hInstance, szAppName) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (hDlgModeless == NULL || !IsDialogMessage (hDlgModeless, &msg))
    {
      if (!TranslateAccelerator (hwnd, hAccel, &msg))
      {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
      }
    }
  }
  return msg.wParam ;
}

void DoCaption (HWND hwnd, TCHAR * szTitleName)
{
  TCHAR szCaption[64 + MAX_PATH] ;
  wsprintf (szCaption, TEXT ("%s - %s"), szAppName,
    szTitleName[0] ? szTitleName : UNTITLED) ;
  SetWindowText (hwnd, szCaption) ;
}

void OkMessage (HWND hwnd, TCHAR * szMessage, TCHAR * szTitleName)
{
  TCHAR szBuffer[64 + MAX_PATH] ;
```

```
  wsprintf (szBuffer, szMessage, szTitleName[0] ? szTitleName : UNTITLED) ;
  MessageBox (hwnd, szBuffer, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

short AskAboutSave (HWND hwnd, TCHAR * szTitleName)
{
  TCHAR szBuffer[64 + MAX_PATH] ;
  int iReturn ;

  wsprintf (szBuffer, TEXT ("Save current changes in %s?"),
    szTitleName[0] ? szTitleName : UNTITLED) ;

  iReturn = MessageBox (hwnd, szBuffer, szAppName,
    MB_YESNOCANCEL | MB_ICONQUESTION) ;
  if (iReturn == IDYES)
    if (!SendMessage (hwnd, WM_COMMAND, IDM_FILE_SAVE, 0))
      iReturn = IDCANCEL ;

    return iReturn ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL bNeedSave = FALSE ;
  static HINSTANCE hInst ;
  static HWND hwndEdit ;
  static int iOffset ;
  static TCHAR szFileName[MAX_PATH], szTitleName[MAX_PATH] ;
  static UINT messageFindReplace ;
  int  iSelBeg, iSelEnd, iEnable ;
  LPFINDREPLACE pfr ;

  switch (message)
  {
  case WM_CREATE:
    hInst = ((LPCREATESTRUCT) lParam) -> hInstance ;
    // Create the edit control child window
    hwndEdit = CreateWindow (TEXT ("edit"), NULL,
      WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |
      WS_BORDER | ES_LEFT | ES_MULTILINE |
      ES_NOHIDESEL | ES_AUTOHSCROLL | ES_AUTOVSCROLL,
      0, 0, 0, 0,
      hwnd, (HMENU) EDITID, hInst, NULL) ;

    SendMessage (hwndEdit, EM_LIMITTEXT, 32000, 0L) ;
    // Initialize common dialog box stuff
    PopFileInitialize (hwnd) ;
    PopFontInitialize (hwndEdit) ;

    messageFindReplace = RegisterWindowMessage (FINDMSGSTRING) ;
    DoCaption (hwnd, szTitleName) ;
    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_INITMENUPOPUP:
    switch (lParam)
    {
    case 1: // Edit menu

      // Enable Undo if edit control can do it

      EnableMenuItem ((HMENU) wParam, IDM_EDIT_UNDO,
        SendMessage (hwndEdit, EM_CANUNDO, 0, 0L) ?
        MF_ENABLED : MF_GRAYED) ;
```

```
    // Enable Paste if text is in the clipboard

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_PASTE,
      IsClipboardFormatAvailable (CF_TEXT) ?
      MF_ENABLED : MF_GRAYED) ;

    // Enable Cut, Copy, and Del if text is selected

    SendMessage (hwndEdit, EM_GETSEL, (WPARAM) &iSelBeg,
      (LPARAM) &iSelEnd) ;

    iEnable = iSelBeg != iSelEnd ? MF_ENABLED : MF_GRAYED ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CLEAR, iEnable) ;
    break ;
  case 2: // Search menu

    // Enable Find, Next, and Replace if modeless
    // dialogs are not already active
    iEnable = hDlgModeless == NULL ? MF_ENABLED : MF_GRAYED ;
    EnableMenuItem ((HMENU) wParam, IDM_SEARCH_FIND, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_SEARCH_NEXT, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_SEARCH_REPLACE, iEnable) ;
    break ;
  }
  return 0 ;
  case WM_COMMAND:
    // Messages from edit control
    if (lParam && LOWORD (wParam) == EDITID)
    {
      switch (HIWORD (wParam))
      {
      case EN_UPDATE :
        bNeedSave = TRUE ;
        return 0 ;
      case EN_ERRSPACE :
      case EN_MAXTEXT :
        MessageBox (hwnd, TEXT ("Edit control out of space."),
          szAppName, MB_OK | MB_ICONSTOP) ;
        return 0 ;
      }
      break ;
    }

    switch (LOWORD (wParam))
    {
      // Messages from File menu
    case IDM_FILE_NEW:
      if (bNeedSave && IDCANCEL == AskAboutSave (hwnd, szTitleName))
        return 0 ;
      SetWindowText (hwndEdit, TEXT ("\0")) ;
      szFileName[0] = '\0' ;
      szTitleName[0] = '\0' ;
      DoCaption (hwnd, szTitleName) ;
      bNeedSave = FALSE ;
      return 0 ;
    case IDM_FILE_OPEN:
      if (bNeedSave && IDCANCEL == AskAboutSave (hwnd, szTitleName))
        return 0 ;
      if (PopFileOpenDlg (hwnd, szFileName, szTitleName))
      {
        if (!PopFileRead (hwndEdit, szFileName))
        {
          OkMessage (hwnd, TEXT ("Could not read file %s!"),
            szTitleName) ;
          szFileName[0] = '\0' ;
          szTitleName[0] = '\0' ;
```

```
        }
      }
      DoCaption (hwnd, szTitleName) ;
      bNeedSave = FALSE ;
      return 0 ;
    case IDM_FILE_SAVE:
      if (szFileName[0])
      {
        if (PopFileWrite (hwndEdit, szFileName))
        {
          bNeedSave = FALSE ;
          return 1 ;
        }
        else
        {
          OkMessage (hwnd, TEXT ("Could not write file %s"),
            szTitleName) ;
          return 0 ;
        }
      }
      //fall through
    case IDM_FILE_SAVE_AS:
      if (PopFileSaveDlg (hwnd, szFileName, szTitleName))
      {
        DoCaption (hwnd, szTitleName) ;

        if (PopFileWrite (hwndEdit, szFileName))
        {
          bNeedSave = FALSE ;
          return 1 ;
        }
        else
        {
          OkMessage (hwnd, TEXT ("Could not write file %s"),
            szTitleName) ;
          return 0 ;
        }
      }
      return 0 ;
    case IDM_FILE_PRINT:
      if (!PopPrntPrintFile (hInst, hwnd, hwndEdit, szTitleName))
        OkMessage ( hwnd, TEXT ("Could not print file %s"),
        szTitleName) ;
      return 0 ;
    case IDM_APP_EXIT:
      SendMessage (hwnd, WM_CLOSE, 0, 0) ;
      return 0 ;
      // Messages from Edit menu
    case IDM_EDIT_UNDO:
      SendMessage (hwndEdit, WM_UNDO, 0, 0) ;
      return 0 ;
    case IDM_EDIT_CUT:
      SendMessage (hwndEdit, WM_CUT, 0, 0) ;
      return 0 ;
    case IDM_EDIT_COPY:
      SendMessage (hwndEdit, WM_COPY, 0, 0) ;
      return 0 ;
    case IDM_EDIT_PASTE:
      SendMessage (hwndEdit, WM_PASTE, 0, 0) ;
      return 0 ;
    case IDM_EDIT_CLEAR:
      SendMessage (hwndEdit, WM_CLEAR, 0, 0) ;
      return 0 ;
    case IDM_EDIT_SELECT_ALL:
      SendMessage (hwndEdit, EM_SETSEL, 0, -1) ;
      return 0 ;
      // Messages from Search menu
    case IDM_SEARCH_FIND:
      SendMessage (hwndEdit, EM_GETSEL, 0, (LPARAM) &iOffset) ;
```

```
      hDlgModeless = PopFindFindDlg (hwnd) ;
      return 0 ;
    case IDM_SEARCH_NEXT:
      SendMessage (hwndEdit, EM_GETSEL, 0, (LPARAM) &iOffset) ;

      if (PopFindValidFind ())
        PopFindNextText (hwndEdit, &iOffset) ;
      else
        hDlgModeless = PopFindFindDlg (hwnd) ;

      return 0 ;
    case IDM_SEARCH_REPLACE:
      SendMessage (hwndEdit, EM_GETSEL, 0, (LPARAM) &iOffset) ;
      hDlgModeless = PopFindReplaceDlg (hwnd) ;
      return 0 ;
    case IDM_FORMAT_FONT:
      if (PopFontChooseFont (hwnd))
        PopFontSetFont (hwndEdit) ;

      return 0 ;
      // Messages from Help menu
    case IDM_HELP:
      OkMessage (hwnd, TEXT ("Help not yet implemented!"),
        TEXT ("\0")) ;
      return 0 ;
    case IDM_APP_ABOUT:
      DialogBox (hInst, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
      return 0 ;
 }
 break ;
case WM_CLOSE:
  if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd, szTitleName))
    DestroyWindow (hwnd) ;

  return 0 ;
case WM_QUERYENDSESSION :
  if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd, szTitleName))
    return 1 ;

  return 0 ;
case WM_DESTROY:
  PopFontDeinitialize () ;
  PostQuitMessage (0) ;
  return 0 ;
default:
  // Process "Find-Replace" messages
  if (message == messageFindReplace)
  {
    pfr = (LPFINDREPLACE) lParam ;
    if (pfr->Flags & FR_DIALOGTERM)
      hDlgModeless = NULL ;

    if (pfr->Flags & FR_FINDNEXT)
      if (!PopFindFindText (hwndEdit, &iOffset, pfr))
        OkMessage (hwnd, TEXT ("Text not found!"),
        TEXT ("\0")) ;

    if (pfr->Flags & FR_REPLACE || pfr->Flags & FR_REPLACEALL)
      if (!PopFindReplaceText (hwndEdit, &iOffset, pfr))
          OkMessage (hwnd, TEXT ("Text not found!"),
          TEXT ("\0")) ;

    if (pfr->Flags & FR_REPLACEALL)
       while (PopFindReplaceText (hwndEdit, &iOffset, pfr)) ;

    return 0 ;
  }
  break ;
 }
```

```
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message,WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG:
    return TRUE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDOK:
      EndDialog (hDlg, 0) ;
      return TRUE ;
    }
    break ;
  }
  return FALSE ;
}
```

POPFILE.C

```
/*------------------------------------------------------------------------
POPFILE.C -- Popup Editor File Functions
------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>

static OPENFILENAME ofn ;
void PopFileInitialize (HWND hwnd)
{
  static TCHAR szFilter[] = TEXT ("Text Files (*.TXT)\0*.txt\0") \
    TEXT ("ASCII Files (*.ASC)\0*.asc\0") \
    TEXT ("All Files (*.*)\0*.*\0\0") ;

  ofn.lStructSize = sizeof (OPENFILENAME) ;
  ofn.hwndOwner = hwnd ;
  ofn.hInstance = NULL ;
  ofn.lpstrFilter = szFilter ;
  ofn.lpstrCustomFilter = NULL ;
  ofn.nMaxCustFilter = 0 ;
  ofn.nFilterIndex = 0 ;
  ofn.lpstrFile = NULL ; // Set in Open and Close functions
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = NULL ; // Set in Open and Close functions
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.lpstrInitialDir = NULL ;
  ofn.lpstrTitle = NULL ;
  ofn.Flags = 0 ; // Set in Open and Close functions
  ofn.nFileOffset = 0 ;
  ofn.nFileExtension = 0 ;
  ofn.lpstrDefExt = TEXT ("txt") ;
  ofn.lCustData = 0L ;
  ofn.lpfnHook = NULL ;
  ofn.lpTemplateName = NULL ;
}

BOOL PopFileOpenDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
  ofn.hwndOwner = hwnd ;
  ofn.lpstrFile = pstrFileName ;
  ofn.lpstrFileTitle = pstrTitleName ;
  ofn.Flags = OFN_HIDEREADONLY | OFN_CREATEPROMPT ;

  return GetOpenFileName (&ofn) ;
}
```

```
BOOL PopFileSaveDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
  ofn.hwndOwner = hwnd ;
  ofn.lpstrFile = pstrFileName ;
  ofn.lpstrFileTitle = pstrTitleName ;
  ofn.Flags = OFN_OVERWRITEPROMPT ;

  return GetSaveFileName (&ofn) ;
}

BOOL PopFileRead (HWND hwndEdit, PTSTR pstrFileName)
{
  BYTE bySwap ;
  DWORD dwBytesRead ;
  HANDLE hFile ;
  int i, iFileLength, iUniTest ;
  PBYTE pBuffer, pText, pConv ;

  // Open the file.
  if (INVALID_HANDLE_VALUE ==
    (hFile = CreateFile (pstrFileName, GENERIC_READ, FILE_SHARE_READ,
    NULL, OPEN_EXISTING, 0, NULL)))
    return FALSE ;
  // Get file size in bytes and allocate memory for read.
  // Add an extra two bytes for zero termination.

  iFileLength = GetFileSize (hFile, NULL) ;
  pBuffer = malloc (iFileLength + 2) ;

  // Read file and put terminating zeros at end.
  ReadFile (hFile, pBuffer, iFileLength, &dwBytesRead, NULL) ;
  CloseHandle (hFile) ;
  pBuffer[iFileLength] = '\0' ;
  pBuffer[iFileLength + 1] = '\0' ;

  // Test to see if the text is Unicode
  iUniTest = IS_TEXT_UNICODE_SIGNATURE | IS_TEXT_UNICODE_REVERSE_SIGNATURE ;
  if (IsTextUnicode (pBuffer, iFileLength, &iUniTest))
  {
    pText = pBuffer + 2 ;
    iFileLength -= 2 ;

    if (iUniTest & IS_TEXT_UNICODE_REVERSE_SIGNATURE)
    {
      for (i = 0 ; i < iFileLength / 2 ; i++)
      {
        bySwap = ((BYTE *) pText) [2 * i] ;
        ((BYTE *) pText) [2 * i] = ((BYTE *) pText) [2 * i + 1] ;
        ((BYTE *) pText) [2 * i + 1] = bySwap ;
      }
    }

    // Allocate memory for possibly converted string
    pConv = malloc (iFileLength + 2) ;
    // If the edit control is not Unicode, convert Unicode text to
    // non-Unicode (i.e., in general, wide character).
#ifndef UNICODE
    WideCharToMultiByte (CP_ACP, 0, (PWSTR) pText, -1, pConv,
      iFileLength + 2, NULL, NULL) ;
    // If the edit control is Unicode, just copy the string
#else
    lstrcpy ((PTSTR) pConv, (PTSTR) pText) ;
#endif

  }
  else // the file is not Unicode
  {
    pText = pBuffer ;
    // Allocate memory for possibly converted string.
```

```
    pConv = malloc (2 * iFileLength + 2) ;
    // If the edit control is Unicode, convert ASCII text.
#ifdef UNICODE
    MultiByteToWideChar (CP_ACP, 0, pText, -1, (PTSTR) pConv,
      iFileLength + 1) ;
    // If not, just copy buffer
#else
    lstrcpy ((PTSTR) pConv, (PTSTR) pText) ;
#endif
  }

  SetWindowText (hwndEdit, (PTSTR) pConv) ;
  free (pBuffer) ;
  free (pConv) ;

  return TRUE ;
}

BOOL PopFileWrite (HWND hwndEdit, PTSTR pstrFileName)
{
  DWORD dwBytesWritten ;
  HANDLE hFile ;
  int iLength ;
  PTSTR pstrBuffer ;
  WORD wByteOrderMark = 0xFEFF ;
  // Open the file, creating it if necessary

  if (INVALID_HANDLE_VALUE ==
    (hFile = CreateFile (pstrFileName, GENERIC_WRITE, 0,
    NULL, CREATE_ALWAYS, 0, NULL)))
    return FALSE ;
  // Get the number of characters in the edit control and allocate
  // memory for them.

  iLength = GetWindowTextLength (hwndEdit) ;
  pstrBuffer = (PTSTR) malloc ((iLength + 1) * sizeof (TCHAR)) ;

  if (!pstrBuffer)
  {
    CloseHandle (hFile) ;
    return FALSE ;
  }

  // If the edit control will return Unicode text, write the
  // byte order mark to the file.

#ifdef UNICODE
  WriteFile (hFile, &wByteOrderMark, 2, &dwBytesWritten, NULL) ;
#endif
  // Get the edit buffer and write that out to the file.
  GetWindowText (hwndEdit, pstrBuffer, iLength + 1) ;
  WriteFile (hFile, pstrBuffer, iLength * sizeof (TCHAR),
    &dwBytesWritten, NULL) ;
  if ((iLength * sizeof (TCHAR)) != (int) dwBytesWritten)
  {
    CloseHandle (hFile) ;
    free (pstrBuffer) ;
    return FALSE ;
  }

  CloseHandle (hFile) ;
  free (pstrBuffer) ;

  return TRUE ;
}
```

POPFIND.C

```
/*---------------------------------------------------------------------------
POPFIND.C -- Popup Editor Search and Replace Functions
---------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include <tchar.h> // for _tcsstr (strstr for Unicode & non-Unicode)

#define MAX_STRING_LEN 256

static TCHAR szFindText [MAX_STRING_LEN] ;
static TCHAR szReplText [MAX_STRING_LEN] ;

HWND PopFindFindDlg (HWND hwnd)
{
  static FINDREPLACE fr ; // must be static for modeless dialog!!!

  fr.lStructSize = sizeof (FINDREPLACE) ;
  fr.hwndOwner = hwnd ;
  fr.hInstance = NULL ;
  fr.Flags = FR_HIDEUPDOWN | FR_HIDEMATCHCASE | FR_HIDEWHOLEWORD ;
  fr.lpstrFindWhat = szFindText ;
  fr.lpstrReplaceWith = NULL ;
  fr.wFindWhatLen = MAX_STRING_LEN ;
  fr.wReplaceWithLen = 0 ;
  fr.lCustData = 0 ;
  fr.lpfnHook = NULL ;
  fr.lpTemplateName = NULL ;

  return FindText (&fr) ;
}

HWND PopFindReplaceDlg (HWND hwnd)
{
  static FINDREPLACE fr ; // must be static for modeless dialog!!!

  fr.lStructSize = sizeof (FINDREPLACE) ;
  fr.hwndOwner = hwnd ;
  fr.hInstance = NULL ;
  fr.Flags = FR_HIDEUPDOWN | FR_HIDEMATCHCASE | FR_HIDEWHOLEWORD ;
  fr.lpstrFindWhat = szFindText ;
  fr.lpstrReplaceWith = szReplText ;
  fr.wFindWhatLen = MAX_STRING_LEN ;
  fr.wReplaceWithLen = MAX_STRING_LEN ;
  fr.lCustData = 0 ;
  fr.lpfnHook = NULL ;
  fr.lpTemplateName = NULL ;

  return ReplaceText (&fr) ;
}

BOOL PopFindFindText (HWND hwndEdit, int * piSearchOffset, LPFINDREPLACE pfr)
{
  int iLength, iPos ;
  PTSTR pstrDoc, pstrPos ;

  // Read in the edit document
  iLength = GetWindowTextLength (hwndEdit) ;

  if (NULL == (pstrDoc = (PTSTR) malloc ((iLength + 1) * sizeof (TCHAR))))
    return FALSE ;

  GetWindowText (hwndEdit, pstrDoc, iLength + 1) ;

  // Search the document for the find string
  pstrPos = _tcsstr (pstrDoc + * piSearchOffset, pfr->lpstrFindWhat) ;
  free (pstrDoc) ;

  // Return an error code if the string cannot be found
  if (pstrPos == NULL)
```

```
    return FALSE ;

  // Find the position in the document and the new start offset
  iPos = pstrPos - pstrDoc ;
  * piSearchOffset = iPos + lstrlen (pfr->lpstrFindWhat) ;

  // Select the found text
  SendMessage (hwndEdit, EM_SETSEL, iPos, * piSearchOffset) ;
  SendMessage (hwndEdit, EM_SCROLLCARET, 0, 0) ;

  return TRUE ;
}
BOOL PopFindNextText (HWND hwndEdit, int * piSearchOffset)
{
  FINDREPLACE fr ;
  fr.lpstrFindWhat = szFindText ;
  return PopFindFindText (hwndEdit, piSearchOffset, &fr) ;
}

BOOL PopFindReplaceText (HWND hwndEdit, int * piSearchOffset, LPFIND,REPLACE pfr)
{
  // Find the text
  if (!PopFindFindText (hwndEdit, piSearchOffset, pfr))
    return FALSE ;

  // Replace it
  SendMessage (hwndEdit, EM_REPLACESEL, 0, (LPARAM) pfr->
    lpstrReplaceWith) ;
  return TRUE ;
}

BOOL PopFindValidFind (void)
{
  return * szFindText != '\0' ;
}
```

POPFONT.C

```
/*-----------------------------------------------------
POPFONT.C -- Popup Editor Font Functions
-----------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>

static LOGFONT logfont ;
static HFONT hFont ;

BOOL PopFontChooseFont (HWND hwnd)
{
  CHOOSEFONT cf ;
  cf.lStructSize = sizeof (CHOOSEFONT) ;
  cf.hwndOwner = hwnd ;
  cf.hDC = NULL ;
  cf.lpLogFont = &logfont ;
  cf.iPointSize = 0 ;
  cf.Flags = CF_INITTOLOGFONTSTRUCT | CF_SCREENFONTS | CF_EFFECTS ;
  cf.rgbColors = 0 ;
  cf.lCustData = 0 ;
  cf.lpfnHook = NULL ;
  cf.lpTemplateName = NULL ;
  cf.hInstance = NULL ;
  cf.lpszStyle = NULL ;
  cf.nFontType = 0 ; // Returned from ChooseFont
  cf.nSizeMin = 0 ;
  cf.nSizeMax = 0 ;

  return ChooseFont (&cf) ;
}
```

```
void PopFontInitialize (HWND hwndEdit)
{
  GetObject (GetStockObject (SYSTEM_FONT), sizeof (LOGFONT),
    (PTSTR) &logfont) ;
  hFont = CreateFontIndirect (&logfont) ;
  SendMessage (hwndEdit, WM_SETFONT, (WPARAM) hFont, 0) ;
}

void PopFontSetFont (HWND hwndEdit)
{
  HFONT hFontNew ;
  RECT rect ;

  hFontNew = CreateFontIndirect (&logfont) ;
  SendMessage (hwndEdit, WM_SETFONT, (WPARAM) hFontNew, 0) ;
  DeleteObject (hFont) ;
  hFont = hFontNew ;
  GetClientRect (hwndEdit, &rect) ;
  InvalidateRect (hwndEdit, &rect, TRUE) ;
}

void PopFontDeinitialize (void)
{
  DeleteObject (hFont) ;
}
```

POPPRNT0.C

```
/*-----------------------------------------------------------------------
POPPRNT0.C -- Popup Editor Printing Functions (dummy version)
-----------------------------------------------------------------------*/
#include <windows.h>
BOOL PopPrntPrintFile ( HINSTANCE hInst, HWND hwnd, HWND hwndEdit,
                   PTSTR pstrTitleName)
{
  return FALSE ;
}
```

POPPAD.RC （摘录）

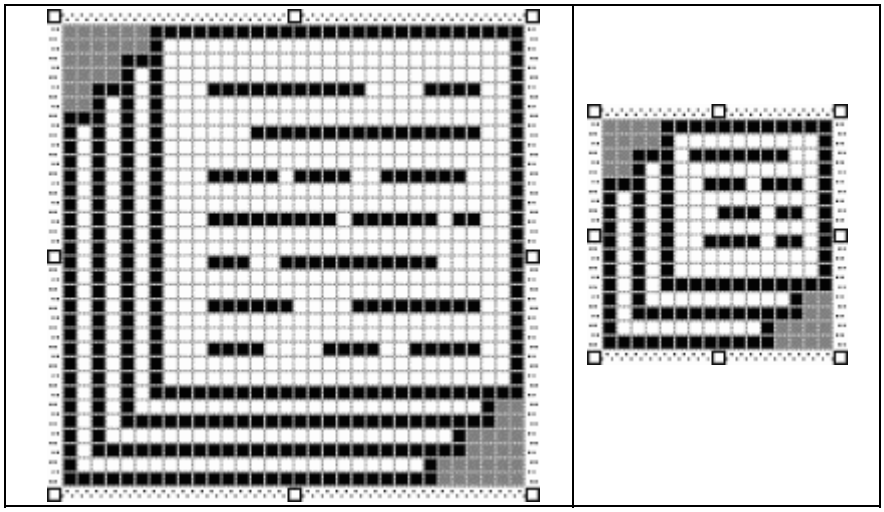
```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
ABOUTBOX DIALOG DISCARDABLE 32, 32, 180, 100
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON "OK",IDOK,66,80,50,14
ICON    "POPPAD",IDC_STATIC,7,7,20,20
CTEXT    "PopPad",IDC_STATIC,40,12,100,8
CTEXT "Popup Editor for Windows",IDC_STATIC,7,40,166,8
CTEXT "(c) Charles Petzold, 1998",IDC_STATIC,7,52,166,8
END
PRINTDLGBOX DIALOG DISCARDABLE 32, 32, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "PopPad"
FONT 8, "MS Sans Serif"
BEGIN
PUSHBUTTON "Cancel",IDCANCEL,67,74,50,14
CTEXT   "Sending",IDC_STATIC,8,8,172,8
CTEXT "",IDC_FILENAME,8,28,172,8
CTEXT "to print spooler.",IDC_STATIC,8,48,172,8
END
```

```
//////////////////////////////////////////////////////////////////////////////
// Menu
POPPAD MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&New\tCtrl+N", IDM_FILE_NEW
MENUITEM "&Open...\tCtrl+O",IDM_FILE_OPEN
MENUITEM "&Save\tCtrl+S", IDM_FILE_SAVE
MENUITEM "Save &As...", IDM_FILE_SAVE_AS
MENUITEM SEPARATOR
MENUITEM "&Print\tCtrl+P", IDM_FILE_PRINT
MENUITEM SEPARATOR
MENUITEM "E&xit", IDM_APP_EXIT
END
POPUP "&Edit"
BEGIN
MENUITEM "&Undo\tCtrl+Z", IDM_EDIT_UNDO
MENUITEM SEPARATOR
MENUITEM "Cu&t\tCtrl+X", IDM_EDIT_CUT
MENUITEM "&Copy\tCtrl+C", IDM_EDIT_COPY
MENUITEM "&Paste\tCtrl+V", IDM_EDIT_PASTE
MENUITEM "De&lete\tDel", IDM_EDIT_CLEAR
MENUITEM SEPARATOR
MENUITEM "&Select All", IDM_EDIT_SELECT_ALL
END
POPUP "&Search"
BEGIN
MENUITEM "&Find...\tCtrl+F",IDM_SEARCH_FIND
MENUITEM "Find &Next\tF3", IDM_SEARCH_NEXT
MENUITEM "&Replace...\tCtrl+R", IDM_SEARCH_REPLACE
END
POPUP "F&ormat"
BEGIN
MENUITEM "&Font...",
END
POPUP "&Help"
BEGIN
MENUITEM "&Help", IDM_HELP
MENUITEM "&About PopPad...", IDM_APP_ABOUT
END
END
//////////////////////////////////////////////////////////////////////////////
// Accelerator
POPPAD ACCELERATORS DISCARDABLE
BEGIN
VK_BACK, IDM_EDIT_UNDO, VIRTKEY, ALT, NOINVERT
VK_DELETE, IDM_EDIT_CLEAR, VIRTKEY, NOINVERT
VK_DELETE, IDM_EDIT_CUT, VIRTKEY, SHIFT, NOINVERT
VK_F1, IDM_HELP, VIRTKEY, NOINVERT
VK_F3, IDM_SEARCH_NEXT, VIRTKEY, NOINVERT
VK_INSERT, IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
VK_INSERT,  IDM_EDIT_PASTE, VIRTKEY, SHIFT, NOINVERT
"^C", IDM_EDIT_COPY, ASCII, NOINVERT
"^F", IDM_SEARCH_FIND, ASCII, NOINVERT
"^N", IDM_FILE_NEW, ASCII, NOINVERT
"^O", IDM_FILE_OPEN, ASCII, NOINVERT
"^P", IDM_FILE_PRINT, ASCII, NOINVERT
"^R", IDM_SEARCH_REPLACE, ASCII, NOINVERT
"^S", IDM_FILE_SAVE, ASCII, NOINVERT
"^V", IDM_EDIT_PASTE, ASCII, NOINVERT
"^X", IDM_EDIT_CUT, ASCII, NOINVERT
"^Z", IDM_EDIT_UNDO, ASCII, NOINVERT
END

//////////////////////////////////////////////////////////////////////////////
// Icon
POPPAD   ICON
DISCARDABLE  "poppad.ico"
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by poppad.rc
#define IDC_FILENAME   1000
#define IDM_FILE_NEW   40001
#define IDM_FILE_OPEN   40002
#define IDM_FILE_SAVE   40003
#define IDM_FILE_SAVE_AS  40004
#define IDM_FILE_PRINT  40005
#define IDM_APP_EXIT   40006
#define IDM_EDIT_UNDO   40007
#define IDM_EDIT_CUT   40008
#define IDM_EDIT_COPY   40009
#define IDM_EDIT_PASTE  40010
#define IDM_EDIT_CLEAR  40011
#define IDM_EDIT_SELECT_ALL 40012
#define IDM_SEARCH_FIND  40013
#define IDM_SEARCH_NEXT  40014
#define IDM_SEARCH_REPLACE 40015
#define IDM_FORMAT_FONT  40016
#define IDM_HELP    40017
#define IDM_APP_ABOUT   40018
```

POPPAD.ICO



为了避免在第十三章中重复原始码，我在POPPAD.RC的菜单中加入了打印项目和一些其它的支持。

POPPAD.C包含了程序中所有的基本原始码。POPFILE.C具有启动File Open和File Save对话框的程序代码，它还包含文件I/O例程。POPFIND.C中包含了搜寻和替换文字功能。POPFONT.C包含了字体选择功能。POPPRNT0.C不完成什么工作：在第十三章中将使用POPPRNT.C替换POPPRNT0.C以建立最终的POPPAD程序。

让我们先来看一看POPPAD.C。POPPAD.C含有两个文件名字符串：第一个，储存在WndProc，名称为szFileName，含有详细的驱动器名称、路径名称和文件名称；第二个，储存为szTitleName，是程序本身的文件名称。它用在POPPAD3的DoCaption函数中，以便将文件名称显示在窗口的标题栏上；也用在OKMessage函数和AskAboutSave函数中，以便向使用者显示消息框。

POPFILE.C包含了几个显示「File Open」和「File Save」对话框以及实际执行文件I/O的函数。

对话框是使用函数GetOpenFileName和GetSaveFileName来显示的。这两个函数都使用一个型态为OPENFILENAME的结构，这个结构在COMMDLG.H中定义。在POPFILE.C中，使用了一个该结构型态的整体变量，取名为ofn。ofn的大多数字段在PopFileInitialize函数中被初始化，POPPAD.C在WndProc中处理WM_CREATE消息时呼叫该函数。

将ofn作为静态整体结构变量会比较方便，因为GetOpenFileName和GetSaveFileName给该结构传回的一些信息，并将在以后呼叫这些函数时用到。

尽管通用对话框具有许多选项－包括设定自己的对话框模板，以及为对话框程序增加「挂勾（hook）」－POPFILE.C中使用的「File Open」和「File Save」对话框是最基本的。OPENFILENAME结构中被设定的字段只有lStructSize（结构的长度）、hwndOwner（对话框拥有者）、lpstrFilter（下面将简要讨论）、lpstrFile和nMaxFile（指向接收完整文件名称的缓冲区指标和该缓冲区的大小）、lpstrFileTitle和nMaxFileTitle（文件名称缓冲区及其大小）、Flags（设定对话框的选项）和lpstrDefExt（如果使用者在对话框中输入文件名时不指定文件扩展名，那么它就是内定的文件扩展名）。

当使用者在「File」菜单中选择「Open」时，POPPAD3呼叫POPFILE的PopFileOpenDlg函数，将窗口句柄、一个指向文件名称缓冲区的指标和一个指向文件标题缓冲区的指标传给它。PopFileOpenDlg恰当地设定OPENFILENAME结构的hwndOwner、lpstrFile和lpstrFileTitle字段，将Flags设定为OFN_ CREATEPROMPT，然后呼叫GetOpenFileName，显示如图11-6所示的普通对话框。

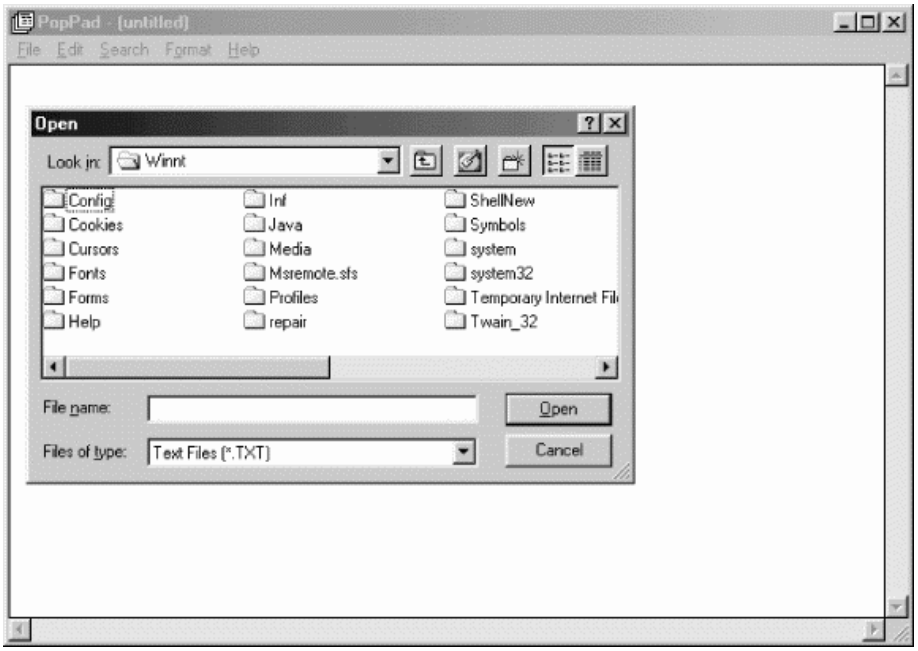


图11-6 「File Open」对话框

当使用者结束这个对话框时，GetOpenFileName函数传回。OFN_CREATEPROMPT旗标指示GetOpenFileName显示一个消息框，询问使用者如果所选文件不存在，是否要建立该文件。

左下角的下拉式清单方块列出了将要显示在文件列表中的文件型态，此清单方块被称为「筛选清单」。使用者可以通过从下拉式清单方块列表中选择另一种文件型态，来改变筛选条件。在POPFILE.C的PopFileInitialize函数中，我在变量szFilter（一个字符串数组）中为三种型态的文件定义了一个筛检清单：带有.TXT扩展名的文本文件、带有.ASC扩展名的ASCII文件和所有文件。OPENFILENAME结构的lpstrFilter字段储存指向此数组第一个字符串的指针。

如果使用者在对话框处于活动状态时改变了筛选条件，那么OPENFILENAME的nFilterIndex字段反映出使用者的选择。由于该结构是静态变量，下次启动对话框时，筛选条件将被设定为选中的文件型态。

POPFILE.C 中的 PopFileSaveDlg 函数与此类似，它将 Flags 参数设定为 OFN_OVERWRITEPROMPT，并呼叫 GetSaveFileName 启动「File Save」对话框。OFN_OVERWRITEPROMPT旗标导致显示一个消息框，如果被选文件已经存在，那么将询问使用者是否覆盖该文件。

Unicode文件I/O

对于本书中的大多数程序，您都不必注意Unicode和非Unicode版的区别。例如，在POPPAD3的Unicode中，编辑控件将保留Unicode文字和使用Unicode字符串的所有通用对话框。例如，当程序需要搜索和替换时，所有的操作都会处理Unicode字符串，而不需要转换。

不过，POPPAD3得处理文件I/O，也就是说，程序不能闭门造车。如果Unicode版的POPPAD3获得了编辑缓冲区的内容并将其写入磁盘，文件将是使用Unicode存放的。如果非Unicode版的POPPAD3读取了该文件，并将其写入编辑缓冲区，其结果将是一堆垃圾。Unicode版读取由非Unicode版储存的文件时也会这样。

解决的办法在于辨别和转换。首先，在POPFILE.C的PopFileWrite函数中，您将看到Unicode版的程序将在文件的开始位置写入0xFEFF。这定义为字节顺序标记，以表示文本文件含有Unicode文字。

其次，在PopFileRead函数中，程序用IsTextUnicode函数来决定文件是否含有字节顺序标记。此函数甚至检测字节顺序标记是否反向了，亦即Unicode文本文件在Macintosh或者其它使用与Intel处理器相反的字节顺序的机器上建立的。这时，字节的顺序都经过翻转。如果文件是Unicode版，但是被非Unicode版的POPPAD3读取，这时，文字将被WideCharToMultiChar转换。WideCharToMultiChar实际上是一个宽字符ANSI函数（除非您执行远东版的Windows）。只有这时文字才能放入编辑缓冲区。

同样地，如果文件是非Unicode文本文件，而执行的是Unicode版的程序，那么文字必须用MultiCharToWideChar转换。

改变字体

我们将在第十七章`详细讨论字体，但那些都不能代替通用对话框函数来选择字体。

在WM_CREATE消息处理期间，POPFONT.C中的POPPAD呼叫PopFontInitialize。这个函数取得一个依据系统字体建立的LOGFONT结构，由此建立一种字体，并向编辑控件发送一个WM_SETFONT消息来设定一种新的字体（内定编辑控件字体是系统字体，而PopFontInitialize为编辑控件建立一种新的字体，因为最终该字体将被删除，而删除现有系统字体是不明智的）。

当 POPPAD 收到来自程序的字体选项的 WM_COMMAND 消息时，它呼叫 PopFontChooseFont。这个函数初始化一个CHOOSEFONT结构，然后呼叫ChooseFont显示字体选择对话框。如果使用者按下「OK」按钮，那么ChooseFont将传回TRUE。随后，POPPAD呼叫PopFontSetFont来设定编辑控件中的新字体，旧字体将被删除。

最后，在WM_DESTROY消息处理期间，POPPAD呼叫PopFontDeinitialize来删除最近一次由PopFontSetFont建立的字体。

搜寻与替换

通用对话框链接库也提供两个用于文字搜寻和替换函数的对话框，这两个函数（FindText和

ReplaceText）使用一个型态为FINDREPLACE的结构。图10-11中所示的POPFIND.C文件有两个例程（PopFindFindDlg和PopFindReplaceDlg）呼叫这些函数，还有两个函数在编辑控件中搜寻和替换文字。

使用搜寻和替换函数有一些考虑。首先，它们启动的对话框是非模态对话框，这意味着必须改写消息循环，以便在对话框活动时呼叫IsDialogMessage。第二，传送给FindText和ReplaceText的FINDREPLACE结构必须是一个静态变量，因为对话框是模态的，函数在对话框显示之后传回，而不是在对话框结束之后传回；而对话框程序必须仍然能够存取该结构。

第三，在显示FindText和ReplaceText对话框时，它们通过一条特殊消息与拥有者窗口联络，消息编号可以通过以FINDMSGSTRING为参数呼叫RegisterWindowMessage函数来获得。这是在WndProc中处理WM_CREATE消息时完成的，消息号存放在静态变量中。

在处理内定消息时，WndProc将消息变量与RegisterWindowMessage传回的值相比较。lParam消息参数是一个指向FINDREPLACE结构的指针，Flags字段指示使用者使用对话框是为了搜寻文字还是替换文字，以及是否要终止对话框。POPPAD3是呼叫POPFIND.C中的PopFindFindText和PopFindReplaceText函数来执行搜寻和替换功能的。

只呼叫一个函数的Windows程序

到现在为止，我们已经说明了两个程序，让您浏览选择颜色，这两个程序分别是第九章中的COLORS1和本章中的COLORS2。现在是讲解COLORS3的时候了，这个程序只有一个Windows函数呼叫。COLORS3的原始码如程序11-7所示。

COLORS3所呼叫的唯一Windows函数是ChooseColor，这也是通用对话框链接库中的函数，它显示如图11-7所示的对话框。颜色选择类似于COLORS1和COLORS2，但是它与使用者交谈互动能力更强。

程序11-7　COLORS3

COLORS3.C

```
/*---------------------------------------------------------------------------
COLORS3.C -- Version using Common Dialog Box
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static CHOOSECOLOR cc ;
  static COLORREF crCustColors[16] ;

  cc.lStructSize = sizeof (CHOOSECOLOR) ;
  cc.hwndOwner = NULL ;
  cc.hInstance = NULL ;
  cc.rgbResult = RGB (0x80, 0x80, 0x80) ;
  cc.lpCustColors = crCustColors ;
  cc.Flags = CC_RGBINIT | CC_FULLOPEN ;
  cc.lCustData = 0 ;
  cc.lpfnHook = NULL ;
  cc.lpTemplateName = NULL ;

  return ChooseColor (&cc) ;
}
```

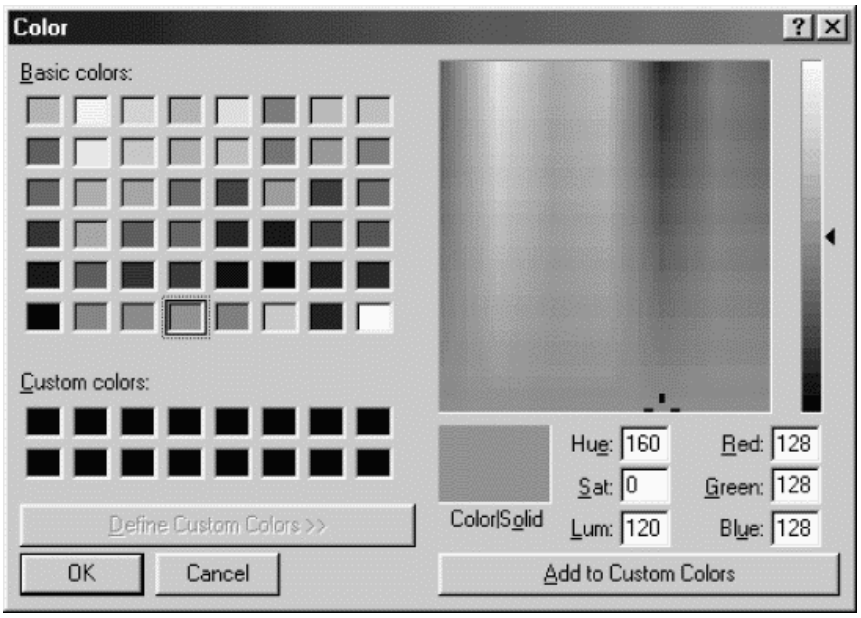


图11-7 COLORS3的屏幕显示

ChooseColor函数使用一个CHOOSECOLOR型态的结构和含有16个DWORD的数组来存放常用颜色，使用者将从对话框中选择这些颜色之一。rgbResult字段可以初始化为一个颜色值，如果Flags字段的CC_RGBINIT旗标被设立，则显示该颜色。通常在使用这个函数时，rgbResult将被设定为使用者选择的颜色。

请注意，Color对话框的hwndOwner字段被设定为NULL。在ChooseColor函数呼叫DialogBox以显示对话框时，DialogBox的第三个参数也被设定为NULL。这是完全合法的，其含义是对话框不为另一个窗口所拥有。对话框的标题将显示在工作列中，而对话框就像一个普通的窗口那样执行。

您也可以在自己程序的对话框中使用这种技巧。使Windows程序只建立对话框，其它事情都在对话框程序中完成，这是可能的。

第十二章 剪贴簿

Microsoft Windows剪贴簿允许把数据从一个程序传送到另一个程序中。它的原理相对而言比较简单，把数据存放到剪贴簿上的程序或从剪贴簿上取出数据的程序都无须太多的负担。Windows 98和Microsoft Windows NT都提供了剪贴簿浏览程序，该程序可以显示剪贴簿的目前内容。

许多处理文件或者其它数据的程序都包含一个「Edit」菜单，其中包括「Cut」、「Copy」和「Paste」选项。当使用者选择「Cut」或者「Copy」时，程序将数据传送给剪贴簿。这个数据使用某种格式，如文字、位图（一种按位排列的矩形数组，其中的位与平面显示的像素相对应）或者metafile（用二进制元数值内容表示的绘图命令集）等。当使用者从菜单中选择「Paste」时，程序检查剪贴簿中包含的数据，看看使用的是否是程序可以接受的一种格式。如果是，那么数据将从剪贴簿传送到程序中。

如果使用者不发出明确的指令，程序就不能把数据送入或移出剪贴簿。例如，在某个程序中执行剪下或复制（或者按Ctrl-X及Ctrl-C）操作的使用者，应该能够假定数据将储存在剪贴簿上，直到下次剪下或复制操作为止。

回忆一下第十和第十一章所示的POPPAD程序的修订版中，我们加上了「Edit」菜单，但是在那边这菜单的作用只是发送消息给编辑控件而已。多数情况下，处理剪贴簿并不方便，您必须自己呼叫剪贴簿传输函数。

本章集中讨论将文字传入和移出剪贴簿。在后面的章节里，我将向您展示如何用剪贴簿处理位图（第十四、十五和十六章）和metafile（第十八章）。

剪贴簿的简单使用

我们由分析把数据传送到剪贴簿（剪下或复制）和存取剪贴簿数据（粘贴）的程序代码开始。

标准剪贴簿数据格式

Windows支持不同的预先定义剪贴簿格式，这些格式在WINUSER.H定义成以CF为前缀的标识符。

首先介绍三种能够储存在剪贴簿上的文字数据型态，以及一个与剪贴簿格式相关的数据型态：

CF_TEXT以NULL结尾的ANSI字符集字符串。它在每行末尾包含一个carriage return和linefeed字符，这是最简单的剪贴簿数据格式。传送到剪贴簿的数据存放在整体内存块中，并且是利用内存块句柄进行传送的（我将简短地讨论此项概念）。这个内存块专供剪贴簿使用，建立它的程序不应该继续使用它。

CF_OEMTEXT含有文字数据（与CF_TEXT类似）的内存块。但是它使用的是OEM字符集。通常Windows程序不必关心这一点；它只有与在窗口中执行MS-DOS程序一起使用剪贴簿时才会使用。

CF_UNICODETEXT含有Unicode文字的内存块。与CF_TEXT类似，它在每一行的末尾包含一个carriage return和linefeed字符，以及一个NULL字符（两个0字节）以表示数据结束。CF_UNICODETEXT只支援Windows NT。

CF_LOCALE一个国家地区标识符的句柄。表示剪贴簿文字使用的国别地区设定。

下面是两种附加的剪贴簿格式，它们在概念上与CF_TEXT格式相似（也就是说，它们都是文字数据），但是它们不需要以NULL结尾，因为格式已经定义了数据的结尾。现在已经很少使用这些格式了：

CF_SYLK包含Microsoft 「符号连结」数据格式的整体内存块。这种格式用在Microsoft的Multiplan、Chart和Excel程序之间交换数据，它是一种ASCII码格式，每一行都用carriage return和linefeed结尾。

CF_DIF包含数据交换格式(DIF)之数据的整体内存块。这种格式是由Software Arts公司提出的，用于把数据送到VisiCalc电子表格程序中。这也是一种ASCII码格式，每一行都使用carriage return和linefeed结尾。

下面三种剪贴簿格式与位图有关。所谓位图就是数据位的矩形数组，其中的数据位与输出设备的像素相对应。 第十四和第十五章将详细讨论位图以及这些位图剪贴簿的格式：

CF_BITMAP与设备相关的位图格式。位图是通过位图句柄传送给剪贴簿的。同样，在把这个位图传送给剪贴簿之后，程序不应该再继续使用这个位图。

CF_DIB定义一个设备无关位图（在第十五章中描述）的内存块。这种内存块是以位图信息结构开始的，后面跟着可用的颜色表和位图数据位。

CF_PALETTE调色盘句柄。它通常与CF_DIB配合使用，以定义与设备相关的位图所使用的颜色调色盘。

在剪贴簿中，还有可能以工业标准的TIFF格式储存的位图数据：

CF_TIFF含有标号图像文件格式(TIFF)数据的整体内存块。这种格式由Microsoft、Aldus公司和Hewlett-Packard公司以及一些硬件厂商推荐使用。这一格式可从Hewlett-Packard的网站上获得。

下面是两个metafile格式，我将在第十八章详细讨论。一个metafile就是一个以二进制格式储存的画图命令集：

CF_METAFILEPICT以旧的metafile格式存放的「图片」。

CF_ENHMETAFILE增强型metafile（32位Windows支持的）句柄。

最后介绍几个混合型的剪贴簿格式：

CF_PENDATA与Windows的笔式输入扩充功能联合使用。

CF_WAVE声音（波形）文件。

CF_RIFF使用资源交换文件格式（Resource Interchange File Format）的多媒体数据。

CF_HDROP与拖放服务相关的文件列表。

内存配置

程序向剪贴簿传输一些数据的时候，必须配置一个内存块，并且将这块内存交给剪贴簿处理。在本书早期的程序中需要配置内存时，我们只需使用标准C执行时期链接库所支持的malloc函数。但是，由于在Windows中执行的应用程序之间必须要共享剪贴簿所储存的内存块，这时malloc函数就有些不适任这项任务了。

实际上，我们必须把早期Windows所开发的内存配置函数再拿出来使用，那时的操作系统在16位的实际模式内存结构中执行。现在的Windows仍然支持这些函数，您还可以使用它们，但不

是必须使用这些函数就是了。

要用Windows API来配置一个内存块，可以呼叫：

hGlobal = GlobalAlloc (uiFlags, dwSize) ;

此函数有两个参数：一系列可能的旗标和内存块的字节大小。函数传回一个HGLOBAL型态的句柄，称为「整体内存块句柄」或「整体句柄」。传回值为NULL表示不能配置足够的内存。

虽然GlobalAlloc的两个参数略有不同，但它们都是32位的无正负号整数。如果将第一个参数设定为0，那么您就可以更有效地使用旗标GMEM_FIXED。在这种情况下，GlobalAlloc传回的整体句柄实际是指向所配置内存块的指针。

如果不喜欢将内存块中的每一位都初始化为0，那么您也能够使用旗标GMEM,_ZEROINIT。在Windows表头文件中，简洁的GPTR旗标定义为GMEM_FIXED和GMEM_ZEROINIT旗标的组合：

#define GPTR (GMEM_FIXED | GMEM_ZEROINIT)

下面是一个重新配置函数：

hGlobal = GlobalReAlloc (hGlobal, dwSize, uiFlags) ;

如果内存块扩大了，您可以用GMEM_ZEROINIT旗标将新的字节设为0。

下面是获得内存块大小的函数：

dwSize = GlobalSize (hGlobal) ;

释放内存块的函数：

GlobalFree (hGlobal) ;

在早期16位的Windows中，因为Windows不能在物理内存中移动内存块，所以禁止使用GMEM_FIXED旗标。在32位的Windows中，GMEM_FIXED旗标很常见。这是因为它将传回一个虚拟地址，并且操作系统也能够通过改变内存页映像表在物理内存中移动内存块。因此为16位的Windows写程序时，GlobalAlloc推荐使用GMEM_MOVEABLE旗标。在Windows的表头文件中还定义了一个简写标识符，用此标识符可以在可移动的内存之外填0：

#define GHND (GMEM_MOVEABLE | GMEM_ZEROINIT)

GMEM_MOVEABLE旗标允许Windows在虚拟内存中移动一个内存块。这不是说将在物理内存中移动内存块，只是应用程序用于读写这块内存的地址可以被变动。

尽管GMEM_MOVEABLE是16位Windows的通则，但是它的作用现在已经少得多了。如果您的应用程序频繁地配置、重新配置以及释放不同大小的内存块，应用程序的虚拟地址空间将会变得支离破碎。可以想象得到，最后虚拟内存地址空间就会被用完。如果这是个可能会发生的问题，那么您将希望内存是可移动的。下面就介绍如何让内存块成为可搬移位置的。

首先定义一个指标（例如，一个int型态的）和一个GLOBALHANDLE型态的变量：

```
int * p ;
GLOBALHANDLE hGlobal ;
```

然后配置内存。例如：

hGlobal = GlobalAlloc (GHND, 1024) ;

与处理其它Windows句柄一样，您不必担心数字的实际意义，只要照著作就好了。需要存取内存块时，可以呼叫：

p = (int *) GlobalLock (hGlobal) ;

此函数将句柄转换为指标。在内存块被锁定期间，Windows将固定虚拟内存中的地址，不再移动那块内存。存取结束后呼叫：

GlobalUnlock (hGlobal) ;

这将使Windows可以在虚拟内存中移动内存块。要真正确保此程序正常运作（体验早期Windows程序写作者的痛苦经历），您应该在单一个消息处理期间锁定和解锁内存块。

在释放内存时，呼叫GlobalFree应使用句柄而不是指标。如果您现在不能存取句柄，可以使用下面的函数：

hGlobal = GlobalHandle (p) ;

在解锁之前，您能够多次锁定一个内存块。Windows保留一个锁定次数，而且在内存块可被自由移动之前，每次锁定都需要相对应的解锁。当Windows在虚拟内存中移动一个内存块时，不需要将字节从一个位置复制到另一个，只需巧妙地处理内存页映像表。通常，让32位Windows为您的程序配置可移动的内存块，其唯一确实的理由只是避免虚拟内存的空间碎裂出现。使用剪贴簿时，也应该使用可移动内存。

为剪贴簿配置内存时，您应该以GMEM_MOVEABLE和GMEM_SHARE旗标呼叫GlobalAlloc函数。GMEM_SHARE旗标使得其它应用程序也可以使用那块内存。

将文字传送到剪贴簿

让我们想象把一个ANSI字符串传送到剪贴簿上，并且我们已经有了指向这个字符串的指针(pString)。现在希望传送这个字符串的iLength字符，这些字符可能以NULL结尾，也可能不以NULL结尾。

首先，通过使用GlobalAlloc来配置一个足以储存字符串的内存块，其中还包括一个终止字符NULL：

hGlobal = GlobalAlloc (GHND | GMEM_SHARE, iLength + 1) ;

如果未能配置到内存块，hGlobal的值将为NULL 。如果配置成功，则锁定这块内存，并得到指向它的一个指标：

pGlobal = GlobalLock (hGlobal) ;

将字符串复制到内存块中：

```
for (i = 0 ; i < wLength ; i++)
    *pGlobal++ = *pString++ ;
```

由于GlobalAlloc的GHND旗标已使整个内存块在配置期间被清除为零，所以不需要增加结尾的NULL 。以下叙述为内存块解锁：

GlobalUnlock (hGlobal) ;

现在就有了表示以NULL结尾的文字所在内存块的内存句柄。为了把它送到剪贴簿中，打开剪贴簿并把它清空：

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
```

利用CF_TEXT标识符把内存句柄交给剪贴簿，关闭剪贴簿：

SetClipboardData (CF_TEXT, hGlobal) ;

CloseClipboard () ;

工作告一段落。

下面是关于此过程的一些规则：

在处理同一个消息的过程中呼叫OpenClipboard和CloseClipboard。不需要时，不要打开剪贴簿。

不要把锁定的内存句柄交给剪贴簿。

当呼叫SetClipboardData后，请不要再继续使用该内存块。它不再属于使用者程序，必须把句柄看成是无效的。如果需要继续存取数据，可以制作数据的副本，或从剪贴簿中读取它（如下节所述）。您也可以在SetClipboardData呼叫和CloseClipboard呼叫之间继续使用内存块，但是不要使用传递给SetClipboardData函数的整体句柄。事实上，此函数也传回一个整体句柄，必需锁定这些代码以存取内存。在呼叫CloseClipboard之前，应先为此句柄解锁。

从剪贴簿上取得文字

从剪贴簿上取得文字只比把文字传送到剪贴簿上稍微复杂一些。您必须首先确定剪贴簿是否含有CF_TEXT格式的数据，最简单的方法是呼叫

bAvailable = IsClipboardFormatAvailable (CF_TEXT) ;

如果剪贴簿上含有CF_TEXT数据，这个函数将传回TRUE（非零）。我们在第十章的POPPAD2程序中已使用了这个函数，用它来确定「Edit」菜单中「Paste」项是被启用还是被停用的。IsClipboardFormatAvailable是少数几个不需先打开剪贴簿就可以使用的剪贴簿函数之一。但是，如果您之后想再打开剪贴簿以取得这个文字，就应该再做一次检查（使用同样的函数或其它方法），以便确定CF_TEXT数据是否仍然留在剪贴簿中。

为了传送出文字，首先打开剪贴簿：

OpenClipboard (hwnd) ;

会得到代表文字的内存块代号：

hGlobal = GetClipboardData (CF_TEXT) ;

如果剪贴簿不包含CF_TEXT格式的数据，此句柄就为NULL。这是确定剪贴簿是否含有文字的另一种方法。如果GetClipboardData传回NULL，则关闭剪贴簿，不做其它任何工作。

从GetClipboardData得到的句柄并不属于使用者程序－它属于剪贴簿。仅在GetClipboardData和CloseClipboard呼叫之间这个句柄才有效。您不能释放这个句柄或更改它所引用的数据。如果需要继续存取这些数据，必须制作这个内存块的副本。

这里有一种将数据复制到使用者程序中的方法。首先，配置一块与剪贴簿数据块大小相同的内存块，并配置一个指向该块的指标：

pText = (char *) malloc (GlobalSize (hGlobal)) ;

再次呼叫hGlobal ，而hGlobal是从GetClipboardData呼叫传回的整体句柄。现在锁定句柄，获得一个指向剪贴簿块的指标：

pGlobal = GlobalLock (hGlobal) ;

现在就可以复制数据了：

strcpy (pText, pGlobal) ;

或者，您可以使用一些简单的C程序代码：

while (*pText++ = *pGlobal++) ;

在关闭剪贴簿之前先解锁内存块：

GlobalUnlock (hGlobal) ;

CloseClipboard () ;

现在您有了一个叫做pText的指针，以后程序的使用者就可以用它来复制文字了。

打开和关闭剪贴簿

在任何时候，只有一个程序可以打开剪贴簿。呼叫OpenClipboard的作用是当一个程序使用剪贴簿时，防止剪贴簿的内容发生变化。OpenClipboard传回BOOL值，它说明是否已经成功地打开了剪贴簿。如果另一个应用程序没有关闭剪贴簿，那么它就不能被打开。如果每个程序在响应使用者的命令时都尽快地、遵守规范地打开然后关闭剪贴簿，那么您将永远不会遇到不能打开剪贴簿的问题。

但是，在不遵守规范程序和优先权式多任务环境中，总会发生一些问题。即使在您的程序将某些东西放入剪贴簿和使用者启动一个「Paste」选项期间，您的程序并没有失去输入焦点，但是您也不能假定您放入的东西仍然在那里，一个背景程序有可能已经在这段期间存取过剪贴簿了。

而且，请留意一个与消息框有关的更微妙问题：如果不能配置足够的内存来将内容复制到剪贴簿，那么您可能希望显示一个消息框。但是，如果这个消息框不是系统模态的，那么使用者可以在显示消息框期间切换到另一个应用程序中。您应该使用系统模态的消息框，或者在您显示消息框之前关闭剪贴簿。

如果您在显示一个对话框时将剪贴簿保持为打开状态，那么您还可能遇到其它问题，对话框中的编辑字段会使用剪贴簿进行文字的剪贴。

剪贴簿和Unicode

迄今为止，我只讨论了用剪贴簿处理ANSI文字（每个字符对应一个字节）。我们用CF_TEXT标识符时就是这种格式。您可能对CF_OEMTEXT和CF_UNICODETEXT还不熟悉吧。

我有一些好消息：在处理您所想要的文字格式时，您只需呼叫SetClipboardData和GetClipboardData，Windows将处理剪贴簿中所有的文字转换。例如，在Windows NT中，如果一个程序用SetClipboardData来处理CF_TEXT剪贴簿数据型态，程序也能用CF_OEMTEXT呼叫GetClipboardData。同样地，剪贴簿也能将CF_OEMTEXT数据转换为CF_TEXT。

在Windows NT中，转换发生在CF_UNICODETEXT、CF_TEXT和CF_OEMTEXT之间。程序应该使用对程序本身而言最方便的一种文字格式来呼叫SetClipboardData 。同样地，程序应该用程序需要的文字格式来呼叫GetClipboardData。我们已经知道，本书附上的程序在编写时可以带有或不带UNICODE标识符。如果您的程序也依此编写，那么在定义了UNICODE标识符之后，程序将执行带有CF_UNICODETEXT参数的SetClipboardData以及GetClipboardData呼叫，而不是CF_TEXT。

CLIPTEXT程序，如程序12-1所示，展示了一种可行的方法。

程序12-1 CLIPTEXT

CLIPTEXT.C

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

```
CLIPTEXT.C -- The Clipboard and Text
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
#ifdef UNICODE
#define CF_TCHAR CF_UNICODETEXT
TCHAR szDefaultText[] = TEXT ("Default Text - Unicode Version") ;
TCHAR szCaption[] = TEXT ("Clipboard Text Transfers - Unicode Version") ;
#else
#define CF_TCHAR CF_TEXT
TCHAR szDefaultText[] = TEXT ("Default Text - ANSI Version") ;
TCHAR szCaption[] = TEXT ("Clipboard Text Transfers - ANSI Version") ;

#endif
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("ClipText") ;
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, szCaption,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  hAccel = LoadAccelerators (hInstance, szAppName) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static PTSTR pText ;
  BOOL bEnable ;
```

```
HGLOBAL hGlobal ;
HDC hdc ;
PTSTR pGlobal ;
PAINTSTRUCT ps ;
RECT rect ;

switch (message)
{
case WM_CREATE:
  SendMessage (hwnd, WM_COMMAND, IDM_EDIT_RESET, 0) ;
  return 0 ;
case WM_INITMENUPOPUP:
  EnableMenuItem ((HMENU) wParam,IDM_EDIT_PASTE,
    IsClipboardFormatAvailable (CF_TCHAR) ? MF_ENABLED : MF_GRAYED) ;

  bEnable = pText ? MF_ENABLED : MF_GRAYED ;

  EnableMenuItem ((HMENU) wParam,IDM_EDIT_CUT, bEnable) ;
  EnableMenuItem ((HMENU) wParam,IDM_EDIT_COPY, bEnable) ;
  EnableMenuItem ((HMENU) wParam,IDM_EDIT_CLEAR,bEnable) ;
  break ;
case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_EDIT_PASTE:
    OpenClipboard (hwnd) ;

    if (hGlobal = GetClipboardData (CF_TCHAR))
    {
      pGlobal = GlobalLock (hGlobal) ;
      if (pText)
      {
        free (pText) ;
        pText = NULL ;
      }
      pText = malloc (GlobalSize (hGlobal)) ;
      lstrcpy (pText, pGlobal) ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    CloseClipboard () ;
    return 0 ;
  case IDM_EDIT_CUT:
  case IDM_EDIT_COPY:
    if (!pText)
      return 0 ;

    hGlobal = GlobalAlloc (GHND | GMEM_SHARE,
      (lstrlen (pText) + 1) * sizeof (TCHAR)) ;
    pGlobal = GlobalLock (hGlobal) ;
    lstrcpy (pGlobal, pText) ;
    GlobalUnlock (hGlobal) ;

    OpenClipboard (hwnd) ;
    EmptyClipboard () ;
    SetClipboardData (CF_TCHAR, hGlobal) ;
    CloseClipboard () ;

    if ( LOWORD (wParam) == IDM_EDIT_COPY)
      return 0 ;
    // fall through for IDM_EDIT_CUT
  case IDM_EDIT_CLEAR:
    if (pText)
    {
      free (pText) ;
      pText = NULL ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case IDM_EDIT_RESET:
```

```
     if (pText)
     {
       free (pText) ;
       pText = NULL ;
     }
     pText = malloc ((lstrlen (szDefaultText) + 1) * sizeof (TCHAR)) ;
     lstrcpy (pText, szDefaultText) ;
     InvalidateRect (hwnd, NULL, TRUE) ;
     return 0 ;
   }
   break ;
   case WM_PAINT:
     hdc = BeginPaint (hwnd, &ps) ;

     GetClientRect (hwnd, &rect) ;

     if (pText != NULL)
       DrawText (hdc, pText, -1, &rect, DT_EXPANDTABS | DT_WORDBREAK) ;

     EndPaint (hwnd, &ps) ;
     return 0 ;
   case WM_DESTROY:
     if ( pText)
       free (pText) ;

     PostQuitMessage (0) ;
     return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CLIPTEXT.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
CLIPTEXT MENU DISCARDABLE
BEGIN
POPUP "&Edit"
BEGIN
MENUITEM "Cu&t\tCtrl+X",   IDM_EDIT_CUT
MENUITEM "&Copy\tCtrl+C",   IDM_EDIT_COPY
MENUITEM "&Paste\tCtrl+V",  IDM_EDIT_PASTE
MENUITEM "De&lete\tDel",   IDM_EDIT_CLEAR
MENUITEM SEPARATOR
MENUITEM "&Reset",   IDM_EDIT_RESET
END
END

//////////////////////////////////////////////////////////////////////////////
// Accelerator
CLIPTEXT ACCELERATORS DISCARDABLE
BEGIN
"C",  IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
"V",  IDM_EDIT_PASTE, VIRTKEY, CONTROL, NOINVERT
VK_DELETE,   IDM_EDIT_CLEAR, VIRTKEY, NOINVERT
"X",  IDM_EDIT_CUT, VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H（摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ClipText.rc
#define IDM_EDIT_CUT   40001
```

```
#define IDM_EDIT_COPY   40002
#define IDM_EDIT_PASTE  40003
#define IDM_EDIT_CLEAR  40004
#define IDM_EDIT_RESET  40005
```

这是在Windows NT下执行Unicode版和ANSI版程序的概念，而且可以看到，剪贴簿是如何在两种字符集之间转换的。注意CLIPTEXT.C顶部的#ifdef叙述。如果定义了UNICODE标识符，那么CF_TCHAR(我命名的一种常用的剪贴簿格式)就等于CF_UNICODETEXT;否则,它就等于CF_TEXT。程序后面呼叫的IsClipboardFormatAvailable、GetClipboardData和SetClipboardData函数都使用CF_TCHAR来指定数据型态。

在程序的开始部分（以及您从「Edit」菜单中选择「Reset」选项时），静态变量pText包含一个指针，在Unicode版的程序中，指针指向Unicode字符串「Default Text -Unicode version」;在非Unicode版的程序中，指针指向「Default Text - ANSI version」。您可以用「Cut」或「Copy」命令将字符串传递给剪贴簿，用「Cut」或「Delete」命令从程序中删除字符串。「Paste」命令将剪贴簿中的文字内容复制到pText。在WM_PAINT消息处理期间，pText将字符串显示在程序的显示区域。

如果您先在Unicode版的CLIPTEXT中选择了「Copy」命令，然后在非Unicode版中选择「Paste」命令，那么您就能看到文字已经从Unicode转换成了ANSI。类似地，如果您执行相反的操作，那么文字就会从ANSI转换成Unicode。

复杂的剪贴簿用法

我们已经看到，在将数据准备好之后，从剪贴簿传输数据时需要四个呼叫:

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (iFormat, hGlobal) ;
CloseClipboard () ;
```

存取这些数据需要三个呼叫

```
OpenClipboard (hwnd) ;
hGlobal = GetClipboardData (iFormat) ;
//其它行程序
CloseClipboard () ;
```

在GetClipboardData和CloseClipboard呼叫之间，可以复制剪贴簿数据或以其它方式来使用它。很多应用程序都需要采用这种方法，但也可以用更复杂的方式来使用剪贴簿。

利用多个数据项

当打开剪贴簿并把数据传送给它时，必须先呼叫EmptyClipboard，通知Windows释放或删除剪贴簿上的内容。不能在现有的剪贴簿内容中附加其它东西。所以，从这种意义上说，剪贴簿每次只能保留一个数据项。

但是，可以在EmptyClipboard和CloseClipboard呼叫之间多次呼叫SetClipboardData，每次都使用不同的剪贴簿格式。例如，如果想在剪贴簿中储存一个很短的文字字符串，可以把这个文字写入metafile，也可以把这个文字写入位图。把位图选进内存设备内容中，并把这个字符串写进位图中。利用这种方法可以使字符串不仅能为从剪贴簿上读取文字的程序所使用，也可以为从剪贴簿上读取位图和metafile的程序所使用。当然，这些程序并不能知道metafile或位图实际上包含了一个字符串。

如果想把一些句柄写到剪贴簿上，对每个句柄均可以呼叫SetClipboardData：

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (CF_TEXT, hGlobalText) ;
SetClipboardData (CF_BITMAP, hBitmap) ;
SetClipboardData (CF_METAFILEPICT, hGlobalMFP) ;
CloseClipboard () ;
```

当这三种格式的数据同时位于剪贴簿上时，用CF_TEXT、CF_BITMAP或CF_METAFILEPICT参数呼叫IsClipboardFormatAvailable将传回TRUE。通过下列呼叫程序可以存取这些代码：

hGlobalText = GetClipboardData (CF_TEXT) ;

或

hBitmap = GetClipboardData (CF_BITMAP) ;

或

hGlobalMFP = GetClipboardData (CF_METAFILEPICT) ;

下一次程序呼叫EmptyClipboard时，Windows将释放或删除剪贴簿上保留的所有三个句柄。

在将不同的文字格式、不同的位图格式或者不同的metafile格式添加到剪贴簿时，不要使用这种技术。只使用一种文字格式、一种位图格式以及一种metafile格式。就像我所说的那样，Windows将在CF_TEXT、CF_OEMTEXT和CF_UNICODETEXT之间转换，也可以在CF_BITMAP和CF_DIB之间，以及在CF_METAFILEPICT和CF_ENHMETAFILE之间进行转换。

透过首先打开剪贴簿，然后呼叫EnumClipboardFormats，程序可以确定剪贴簿储存的所有格式。开始时设定变量iFormat为0：

iFormat = 0 ;

OpenClipboard (hwnd) ;

现在从0值开始逐次进行连续的EnumClipboardFormats呼叫。函数将为目前在剪贴簿中的每种格式传回一个正的iFormat值。当函数传回0时，表示完成：

```
while (iFormat = EnumClipboardFormats (iFormat))
{
  //各个iFormat值的处理方式
}
CloseClipboard () ;
```

您可以通过下面的呼叫来取得目前在剪贴簿中之不同格式的个数：

iCount = CountClipboardFormats () ;

延迟提出

当把数据放入剪贴簿中时，一般来说要制作一份数据的副本，并将包含这份副本的内存块句柄传给剪贴簿。对非常大的数据项来说，这种方法会浪费内存空间。如果使用者不想把数据粘贴到另一个程序里，那么，在被其它内容取代之前，它将一直占据着内存空间。

通过使用一种叫做「延迟提出」的技术可以避免这个问题。实际上，直到另一个程序需要数据，程序才提供这份数据。为此，不将数据句柄传给Windows，而是在SetClipboardData呼叫中使用NULL：

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (iFormat, NULL) ;
CloseClipboard () ;
```

可以有多个使用不同iFormat值的SetClipboardData呼叫，对其中某些呼叫可使用NULL值。而对其他一些则使用实际的句柄值。

前面的过程比较简单，以下的过程就要稍微复杂一些了。当另一个程序呼叫GetClipboardData时，Windows将检查那种格式的句柄是否为NULL。如果是，Windows将给「剪贴簿所有者」（您的程序）发送一个消息，要求取得数据的实际句柄，这时您的程序必须提供这个句柄。

更具体地说，「剪贴簿所有者」是将数据放入剪贴簿的最后一个窗口。当一个程序呼叫OpenClipboard时，Windows储存呼叫这个函数时所用的窗口句柄，这个句柄标示打开剪贴簿的窗口。一旦收到一个EmptyClipboard呼叫，Windows就使这个窗口作为新的剪贴簿所有者。

使用延迟提出技术的程序在它的窗口消息处理程序中必须处理三个消息：WM_RENDERFORMAT、WM_RENDERALLFORMATS和WM_DESTROYCLIPBOARD。当另一个程序呼叫GetClipboardData时，Windows给窗口消息处理程序发送一个WM_RENDERFORMAT消息，wParam的值是所要求的格式。在处理WM_RENDERFORMAT消息时，不要打开或清空剪贴簿。为wParam所指定的格式建立一个整体内存块，把数据传给它，并用正确的格式和相应句柄呼叫SetClipboardData。很明显地，为了在处理WM_RENDERFORMAT时正确地构造出此数据，需要在程序中保留这些信息。当另一个程序呼叫EmptyClipboard时，Windows给您的程序发送一个WM_DESTROYCLIPBOARD消息，告诉您不再需要构造剪贴簿数据的信息。您的程序不再是剪贴簿的所有者。

如果程序在它自己仍然是剪贴簿所有者的时候就要终止执行，并且剪贴簿上仍然包含着该程序用SetClipboardData设定的NULL数据句柄，它将收到WM_RENDERALLFORMATS消息。这时，应该打开剪贴簿，清空它，把数据加载内存块中，并为每种格式呼叫SetClipboardData，然后关闭剪贴簿。WM_RENDERALLFORMATS消息是窗口消息处理程序最后收到的消息之一。它后面跟有WM_DESTROYCLIPBOARD消息（由于已经提出了所有数据），然后是正常的WM_DESTROY消息。

如果您的程序只能向剪贴簿传输一种格式的数据（例如文字），那么您可以把WM_RENDERALLFORMATS和WM_RENDERFORMAT处理结合在一起。这些程序代码应该类似下面这样：

```
case WM_RENDERALLFORMATS :
  OpenClipboard (hwnd) ;
  EmptyClipboard () ;
  // fall through
case WM_RENDERFORMAT :
  // 将文字放入整体内存块
  SetClipboardData (CF_TEXT, hGlobal) ;
  if (message == WM_RENDERALLFORMATS)
    CloseClipboard () ;
  return 0 ;
```

如果您的程序使用好几种剪贴簿格式，那么您可能想为wParam所要求的格式处理WM_RENDERFORMAT。除非程序在存放构造数据所需的信息时遇到困难，否则不需要处理WM_DESTROYCLIPBOARD消息。

自订数据格式

到目前为止，我们仅处理了Windows定义的标准剪贴簿资料格式。但是，您可能想用剪贴簿来储存「自订数据格式」。许多文书处理程序使用这种技术来储存包含着字体和格式化信息的文字。

初看之下，这个概念似乎是没有意义的。如果剪贴簿的作用是在应用程序之间传送数据，那么，为什么剪贴簿中要含有只有一个应用程序才能理解的数据呢？答案很简单：剪贴簿允许在同一个程序的内部（或者可能在一个程序中的不同执行实体之间）传送数据。很明显地，这些执行实体能理

解它们自己的自订数据格式。

有几种使用自订数据格式的方法。最简单的方法用到一种表面上是标准剪贴簿格式（文字、位图或metafile）的数据，可是该数据实际上只对您的程序有意义。这种情况下，在SetClipboardData和GetClipboardData呼叫中可使用下列wFormat值：CF_DSPTEXT、CF_DSPBITMAP、CF_DSPMETAFILEPICT或CF_DSPENHMETAFILE（字母DSP代表「显示器」）。这些格式允许Windows按文字、位图或metafile来浏览或显示资料。但是，另一个使用常规的CF_TEXT、CF_BITMAP、CF_DIB、CF_METAFILEPICT或CF_ENHMETAFILE格式呼叫GetClipboardData的程序将不能取得这个数据。

如果用其中一种格式把数据放入剪贴簿中，则必须使用同样的格式读出数据。但是，如何知道数据是来自程序的另一个执行实体，还是来自使用其中某种数据格式的另一个程序呢？这里有一种方法，可以透过下列呼叫首先获得剪贴簿所有者：

```
hwndClipOwner = GetClipboardOwner () ;
```

然后可以得到此窗口句柄的窗口类别名称：

```
TCHAR szClassName [32] ;
//其它行程序
GetClassName (hwndClipOwner, szClassName, 32) ;
```

如果类别名称与程序名称相同，那么数据是由程序的另一个执行实体传送到剪贴簿中的。

使用自订数据格式的第二种方法涉及到CF_OWNERDISPLAY旗标。SetClipboardData的整体内存句柄是NULL：

```
SetClipboardData (CF_OWNERDISPLAY, NULL) ;
```

这是某些文书处理程序在Windows的剪贴簿浏览器的显示区域中显示格式化文字时所采用的方法。很明显地，剪贴簿浏览器不知道如何显示这种格式化文字。当一个文书处理程序指定CF_OWNERDISPLAY格式时，它也就承担起在剪贴簿浏览器的显示区域中绘图的责任。

由于整体内存句柄为NULL，所以用CF_OWNERDISPLAY格式（剪贴簿所有者）呼叫SetClipboardData的程序必须处理由Windows发往剪贴簿所有者的延迟提出消息、以及5条附加消息。这5个消息是由剪贴簿浏览器发送到剪贴簿所有者的：

WM_ASKCBFORMATNAME剪贴簿浏览器把这个消息发送到剪贴簿所有者，以得到数据格式名称。lParam参数是指向缓冲区的指标，wParam是这个缓冲区能容纳的最大字符数目。剪贴簿所有者必须把剪贴簿数据格式的名字复制到这个缓冲区中。

WM_SIZECLIPBOARD这个消息通知剪贴簿所有者，剪贴簿浏览器的显示区域大小已发生了变化。wParam参数是剪贴簿浏览器的句柄，lParam是指向包含新尺寸的RECT结构的指针。如果RECT结构中都是0，则剪贴簿浏览器退出或最小化。尽管Windows的剪贴簿浏览器只允许它自己的一个执行实体执行，但其它剪贴簿浏览器也能把这个消息发送给剪贴簿所有者。应付多个剪贴簿浏览器并非不可能（假定wParam标识特定的浏览器），但剪贴簿所有者处理起来也不容易。

WM_PAINTCLIPBOARD这个消息通知剪贴簿所有者修改剪贴簿浏览器的显示区域。同时，wParam是剪贴簿浏览器窗口的句柄，lParam是指向PAINTSTRUCT结构的整体指针。剪贴簿所有者可以从此结构的hdc栏中得到剪贴簿浏览器设备内容的句柄。

WM_HSCROLLCLIPBOARD和WM_VSCROLLCLIPBOARD这两个消息通知剪贴簿所有者，使用者已经卷动了剪贴簿浏览器的卷动列。wParam参数是剪贴簿浏览器窗口的句柄，

lParam的低字组是卷动请求，并且，如果低字组是SB_THUMBPOSITION，那么lParam的高字组就是滑块位置。

处理这些消息比较麻烦，看来并不值得这样做。但是，这种处理对使用者来说是有益的。当从文书处理程序把文字复制到剪贴簿时，使用者在剪贴簿浏览器的显示区域中看见文字还保持着格式时心里会舒坦些。

使用私有剪贴簿数据格式的第三种方法是注册自己的剪贴簿格式名。您向Windows提供格式名，Windows给程序提供一个序号，它可以用作SetClipboardData和GetClipboardData的格式参数。一般来说，采用这种方法的程序也要以一种标准格式把数据复制到剪贴簿。这种方法允许剪贴簿浏览器在它的显示区域中显示数据（没有与CF_OWNERDISPLAY相关的冲突），并且允许其它程序从剪贴簿上复制数据。

例如，假定我们已经编写了一个以位图格式、metafile格式和自己的已注册的剪贴簿格式把数据复制到剪贴簿中的向量绘图程序。剪贴簿浏览器将显示metafile或者位图，其它从剪贴簿上读取位图和metafile的程序将获得这几种格式。但是，当我们的向量绘图程序需要从剪贴簿上读数据时，它会按照自己已注册的格式复制数据，这是因为这种格式可能包含着比位图文件或者metafile更多的信息。

程序透过下面的呼叫来注册一个新的剪贴簿格式：

iFormat = RegisterClipboardFormat (szFormatName) ;

iFormat 的值介于 0xC000 和 0xFFFF 之间。剪贴簿浏览器（或一个通过呼叫EnumClipboardFormats取得目前所有剪贴簿数据格式的程序）可以取得这种数据格式的ASCII名称，这是通过下面呼叫实作的：

GetClipboardFormatName (iFormat, psBuffer, iMaxCount) ;

Windows将多达iMaxCount个字符复制到psBuffer中。

使用这种方法把数据复制到剪贴簿中的程序写作者，可能需要公开数据格式名称和实际的数据格式。如果这个程序流行起来，那么其它程序就会以这种格式从剪贴簿中复制数据。

实作剪贴簿浏览器

监视剪贴簿内容变化的程序称为「剪贴簿浏览器」。您可以在Windows中得到一个剪贴簿浏览器，但是您也可以编写自己的剪贴簿浏览器程序。剪贴簿浏览器通过传递到浏览器窗口消息处理程序的消息来监视剪贴簿内容的变化。

剪贴簿浏览器链

任意数量的剪贴簿浏览器应用程序都可以同时在Windows下执行，它们都可以监视剪贴簿内容的变化。但是，从Windows的角度来看，只存在一个剪贴簿浏览器，我们称之为「目前剪贴簿浏览器」。Windows只保留一个识别目前剪贴簿浏览器的窗口句柄，并且当剪贴簿的内容发生变化时只把消息发送到那个窗口中。

剪贴簿浏览器应用程序有必要加入「剪贴簿浏览器链」，以便执行的所有剪贴簿浏览器都可以收到Windows发送给目前剪贴簿浏览器的消息。当一个程序将自己注册为一个剪贴簿浏览器时，它就成为目前的剪贴簿浏览器。Windows把先前的目前浏览器窗口句柄交给这个程序，并且此程序将储存这个句柄。当此程序收到一个剪贴簿浏览器消息时，它把这个消息发送给剪贴簿链中下一个程序的窗口消息处理程序。

剪贴簿浏览器的函数和消息

程序透过呼叫SetClipboardViewer函数可以成为剪贴簿浏览器链的一部分。如果程序的主要作用是作为剪贴簿浏览器，那么这个程序在WM_CREATE消息处理期间可以呼叫这个函数，该函数传回前一个目前剪贴簿浏览器的窗口句柄。程序应该把这个句柄储存在静态变量中：

```
static HWND hwndNextViewer ;
//其它行程序
case WM_CREATE :
  //其它行程序
  hwndNextViewer = SetClipboardViewer (hwnd) ;
```

如果在Windows的一次执行期间，您的程序成为剪贴簿浏览器的第一个程序，那么hwndNextViewer将为NULL。

不管剪贴簿中的内容怎样变化，Windows都将把WM_DRAWCLIPBOARD消息发送给目前的剪贴簿浏览器（最近注册为剪贴簿浏览器的窗口）。剪贴簿浏览器链中的每个程序都应该用SendMessage把这个消息发送到下一个剪贴簿浏览器。浏览器链中的最后一个程序（第一个将自己注册为剪贴簿浏览器的窗口）所储存的hwndNextViewer为NULL。如果hwndNextViewer为NULL，那么程序只简单地将控件权还给系统而已，而不向其它程序发送任何消息（不要把WM_DRAWCLIPBOARD消息和WM_PAINTCLIPBOARD消息混淆了。WM_PAINTCLIPBOARD是由剪贴簿浏览器发送给使用CF_OWNERDISPLAY剪贴簿数据格式的程序，而WM_DRAWCLIPBOARD消息是由Windows发往目前剪贴簿浏览器的）。

处理WM_DRAWCLIPBOARD消息的最简单方法是将消息发送给下一个剪贴簿浏览器（除非hwndNextViewer为NULL），并使窗口的显示区域无效：

```
case WM_DRAWCLIPBOARD :
  if ( hwndNextViewer)
    SendMessage (hwndNextViewer, message, wParam, lParam) ;
  InvalidateRect (hwnd, NULL, TRUE) ;
  return 0 ;
```

在处理WM_PAINT消息处理期间，通过使用常规的OpenClipboard、GetClipboardData和CloseClipboard呼叫可以读取剪贴簿的内容。

当某个程序想从剪贴簿浏览器链中删除它自己时，它必须呼叫ChangeClipboardChain。这个函数接收脱离浏览器链的程序之窗口句柄，和下一个剪贴簿浏览器的窗口句柄：

ChangeClipboardChain (hwnd, hwndNextViewer) ;

当程序呼叫ChangeClipboardChain时，Windows发送WM_CHANGECBCHAIN消息给目前的剪贴簿浏览器。wParam参数是从链中移除它自己的那个浏览器窗口句柄（ChangeClipboardChain的第一个参数），lParam是从链中移除自己后的下一个剪贴簿浏览器的窗口句柄（ChangeClipboardChain的第二个参数）。

当程序接收到WM_CHANGECBCHAIN消息时，必须检查wParam是否等于已经储存的hwndNextViewer的值。如果是这样，程序必须设定hwndNextViewer为lParam。这项工作保证将来的WM_DRAWCLIPBOARD消息不会发送给从剪贴簿浏览器链中删除了自己的窗口。如果wParam不等于hwndNextViewer ，并且hwndNextViewer不为NULL，则把消息送到下一个剪贴簿浏览器。

```
case WM_CHANGECBCHAIN :
  if ((HWND) wParam == hwndNextViewer)
    hwndNextViewer = (HWND) lParam ;
  else if (hwndNextViewer)
    SendMessage (hwndNextViewer, message, wParam, lParam) ;
  return 0 ;
```

不一定要使用else if叙述，它只用于保证hwndNextViewer为非NULL的值。hwndNextViewer的值为NULL时，执行这段程序代码的程序就是链中最后一个浏览器，而这是不可能的。

当程序快结束时，如果它仍然在剪贴簿浏览器链中，则必须从链中删除它。您可以在处理WM_DESTROY消息时呼叫ChangeClipboardChain来完成这项工作。

```
case WM_DESTROY :
  ChangeClipboardChain (hwnd, hwndNextViewer) ;
  PostQuitMessage (0) ;
  return 0 ;
```

Windows还有一个允许程序获得第一个剪贴簿浏览器窗口句柄的函数：

hwndViewer = GetClipboardViewer () ;

一般来说不需要这个函数。如果没有目前的剪贴簿浏览器，则传回值为NULL。

下面是一个说明剪贴簿浏览器链如何工作的例子。当Windows刚启动时，目前剪贴簿浏览器是NULL：

剪贴簿浏览器：NULL

一个具有hwnd1窗口句柄的程序呼叫SetClipboardViewer。这个函数传回的NULL成为这个程序中的hwndNextViewer值：

目前剪贴簿浏览器：hwnd1

hwnd1的下一个浏览器：NULL

第二个具有hwnd2窗口句柄的程序呼叫SetClipboardViewer ，并传回hwnd1：

目前的剪贴簿浏览器：hwnd2

hwnd2的下一个浏览器：hwnd1

hwnd1的下一个浏览器：NULL

每三个程序(hwnd3)和第四个程序(hwnd4) 也呼叫SetClipboardViewer ，并且传回hwnd2和hwnd3：

目前的剪贴簿浏览器：hwnd4

hwnd4的下一个浏览器：hwnd3

hwnd3的下一个浏览器：hwnd2

hwnd2的下一个浏览器：hwnd1

hwnd1的下一个浏览器：NULL

当剪贴簿的内容发生变化时，Windows发送一个WM_DRAWCLIPBOARD消息给hwnd4，hwnd4发送消息给hwnd3，hwnd3发送消息给hwnd2，hwnd2发送消息给hwnd1，hwnd1传回。

现在hwnd2决定通过下列呼叫从链中删除自己：

ChangeClipboardChain (hwnd2, hwnd1) ;

Windows将wParam等于hwnd2、lParam等于hwnd1的WM_CHANGECBCHAIN消息发送给hwnd4。由于hwnd4的下一个测览器是hwnd3，所以hwnd4把这个消息传给hwnd3。现在hwnd3注意到wParam等于它的下一个测览器(hwnd2)，所以将下一个浏览器设定为lParam(hwnd1)并且传回。这样工作就完成了。现在剪贴簿浏览器链如下：

目前剪贴簿浏览器：hwnd4

hwnd4的下一个浏览器：hwnd3

hwnd3的下一个浏览器：hwnd1

hwnd1的下一个浏览器：NULL

一个简单的剪贴簿浏览器

剪贴簿浏览器不一定要像Windows所提供的那样完善，例如，剪贴簿浏览器可以只显示一种剪贴簿数据格式。程序12-2中所示的CLIPVIEW程序是一种只能显示CF_TEXT格式的剪贴簿浏览器。

程序12-2　CLIPVIEW

CLIPVIEW.C

```
/*---------------------------------------------------------------------------
CLIPVIEW.C --Simple Clipboard Viewer
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("ClipView") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName,
    TEXT ("Simple Clipboard Viewer (Text Only)"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
```

```
{
  static HWND hwndNextViewer ;
  HGLOBAL hGlobal ;
  HDC hdc ;
  PTSTR pGlobal ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    hwndNextViewer = SetClipboardViewer (hwnd) ;
    return 0 ;
  case WM_CHANGECBCHAIN:
    if ((HWND) wParam == hwndNextViewer)
      hwndNextViewer = (HWND) lParam ;

    else if(hwndNextViewer)
      SendMessage (hwndNextViewer, message, wParam, lParam) ;

    return 0 ;
  case WM_DRAWCLIPBOARD:
    if (hwndNextViewer)
      SendMessage (hwndNextViewer, message, wParam, lParam) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    GetClientRect (hwnd, &rect) ;
    OpenClipboard (hwnd) ;

#ifdef UNICODE
    hGlobal = GetClipboardData (CF_UNICODETEXT) ;
#else
    hGlobal = GetClipboardData (CF_TEXT) ;
#endif
    if (hGlobal != NULL)
    {
      pGlobal = (PTSTR) GlobalLock (hGlobal) ;
      DrawText (hdc, pGlobal, -1, &rect, DT_EXPANDTABS) ;
      GlobalUnlock (hGlobal) ;
    }

    CloseClipboard () ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    ChangeClipboardChain (hwnd, hwndNextViewer) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CLIPVIEW 依上面所讨论的方法来处理 WM_CREATE、WM_CHANGECBCHAIN、WM_DRAWCLIPBOARD和WM_DESTROY消息。WM_PAINT消息处理打开剪贴簿，并用CF_TEXT格式呼叫GetClipboardData。如果函数传回一个整体内存句柄，那么CLIPVIEW将锁定它，并用DrawText在显示区域显示文字。

处理标准格式（如Windows提供的那个剪贴簿一样）以外的数据格式的剪贴簿浏览器还需要完成一些其它工作，比如显示剪贴簿中目前所有数据格式的名称。使用者可以通过呼叫EnumClipboardFormats并使用GetClipboardFormatName得到非标准数据格式名称来完成这项工作。使用CF_OWNERDISPLAY数据格式的剪贴簿浏览器必须把下面四个消息送往剪贴簿数据

的拥有者以显示该资料：

WM_PAINTCLIPBOARD	WM_VSCROLLCLIPBOARD
WM_SIZECLIPBOARD	WM_HSCROLLCLIPBOARD

如果您想编写这样的剪贴簿浏览器，那么必须使用GetClipboardOwner获得剪贴簿所有者的窗口句柄，并当您需要修改剪贴簿的显示区域时，将这些消息发送给该窗口。

图像篇

第十三章 使用打印机

为了处理文字和图形而使用视讯显示器时，设备无关的概念看来非常完美，但对于打印机，设备无关的概念又怎样呢?

总的说来，效果也很好。在Windows程序中，用于视讯显示器的GDI函数一样可以在印表纸上打印文字和图形，在以前讨论的与设备无关的许多问题（多数都与平面显示的尺寸、分辨率以及颜色数有关）都可以用相同的方法解决。当然，一台打印机不像使用阴极射线管的显示器那么简单，它们使用的是印表纸。它们之间有一些比较大的差异。例如，我们从来不必考虑视讯显示器没有与显示卡连结好，或者显示器出现「屏幕空间不够」的错误，但打印机off line和缺纸却是经常会遇到的问题。

我们也不必担心显示卡不能执行某些图形操作，更不用担心显示卡能否处理图形，因为，如果它不能处理图形，就根本不能使用Windows。但有些打印机不能打印图形（尽管它们能在Windows环境中使用）。绘图机尽管可以打印向量图形，却存在位图块的传输问题。

以下是其它一些需要考虑的问题：

打印机比视讯显示器慢。尽管我们没有机会将程序性能调整到最佳状态，却不必担心视讯显示器更新所需的时间。然而，没有人想在做其它工作前一直等待打印机完成打印任务。

程序可以用新的输出覆盖原有的显示输出，以重新使用视讯显示器表面。这对打印机是不可能的，打印机只能用完一整页纸，然后在新一页的纸上打印新的内容。

在视讯显示器上，不同的应用程序都被窗口化。而对于打印机，不同应用程序的输出必须分成不同的文件或打印作业。

为了在GDI的其余部分中加入打印机支持功能，Windows提供几个只用于打印机的函数。这些限用在打印机上的函数（StartDoc、EndDoc、StartPage和EndPage）负责将打印机的输出组织打印到纸页上。而一个程序呼叫普通的GDI函数在一张纸上显示文字和图形，和在屏幕上显示的方式一样。

在第十五、十七和十八章有打印位图、格式化的文字以及metafile的其它信息。

打印入门

当您在Windows下使用打印机时，实际上启动了一个包含GDI32动态链接库模块、打印驱动程序动态连结模块（带.DRV扩展名）、Windows后台打印程序，以及有用到的其它相关模块。在写打印机打印程序之前，让我们先看一看这个程序是如何进行的。

打印和背景处理

当应用程序要使用打印机时，它首先使用CreateDC或PrintDlg来取得指向打印机设备内容的

句柄，于是使得打印机设备驱动程序动态链接库模块被加载到内存（如果还没有加载内存的话）并自己进行初始化。然后，程序呼叫StartDoc函数，通知说一个新文件开始了。StartDoc函数是由GDI模块来处理的，GDI模块呼叫打印机设备驱动程序中的Control函数告诉设备驱动程序准备进行打印。

打印一个文件的程序以StartDoc呼叫开始，以EndDoc呼叫结束。这两个呼叫对于在文件页面上书写文字或者绘制图形的GDI命令来说，其作用就像分隔页面的书挡一样。每页本身是这样来划清界限的：呼叫StartPage来开始一页，呼叫EndPage来结束该页。

例如，如果应用程序想在一页纸上画出一个椭圆，它首先呼叫StartDoc开始打印任务，然后再呼叫StartPage通知这是新的一页，接着呼叫Ellipse，正如同在屏幕上画一个椭圆一样。GDI模块通常将程序对打印机设备内容做出的GDI呼叫储存在磁盘上的metafile中，该文件名以字符串~EMF（代表「增强型metafile」）开始，且以.TMP为扩展名。然而，我在这里应该指出，打印机驱动程序可能会跳过这一步骤。

当绘制第一页的GDI呼叫结束时，应用程序呼叫EndPage。现在，真正的工作开始了。打印机驱动程序必须把存放在metafile中的各种绘图命令翻译成打印机输出数据。绘制一页图形所需的打印机输出数据量可能非常大，特别是当打印机没有高级页面制作语言时，更是如此。例如，一台每英寸600点且使用8.5×11英寸印表纸的激光打印机，如果要定义一个图形页，可能需要4百万以上字节的数据。

为此，打印机驱动程序经常使用一种称作「打印分带」的技术将一页分成若干称为「输出带」的矩形。GDI模块从打印机驱动程序取得每个输出带的大小，然后设定一个与目前要处理的输出带相等的剪裁区，并为metafile中的每个绘图函数呼叫打印机设备驱动程序的Output函数，这个程序叫做「将metafile输出到设备驱动程序」。对设备驱动程序所定义的页面上的每个输出带，GDI模块必须将整个metafile「输出到」设备驱动程序。这个程序完成以后，该metafile就可以删除了。

对每个输出带，设备驱动程序将这些绘图函数转换为在打印机上打印这些图形所需要的输出数据。这种输出数据的格式是依照打印机的特性而异的。对点阵打印机，它将是包括图形序列在内的一系列控制命令序列的集合（打印机驱动程序也能呼叫在GDI模块中的各种「helper」辅助例程，用来协助这种输出的构造）。对于带有高阶页面制作语言（如PostScript）的激光打印机，打印机将用这种语言进行输出。

打印驱动程序将打印输出的每个输出带传送到GDI模块。随后，GDI模块将该打印输出存入另一个临时文件中，该临时文件名以字符串~SPL开始，带有.TMP扩展名。当处理好整页之后，GDI模块对后台打印程序进行一个程序间呼叫，通知它一个新的打印页已经准备好了。然后，应用程序就转向处理下一页。当应用程序处理完所有要打印的输出页后，它就呼叫EndDoc发出一个信号，表示打印作业已经完成。图13-1显示了应用程序、GDI模块和打印驱动程序的交互作用程序。

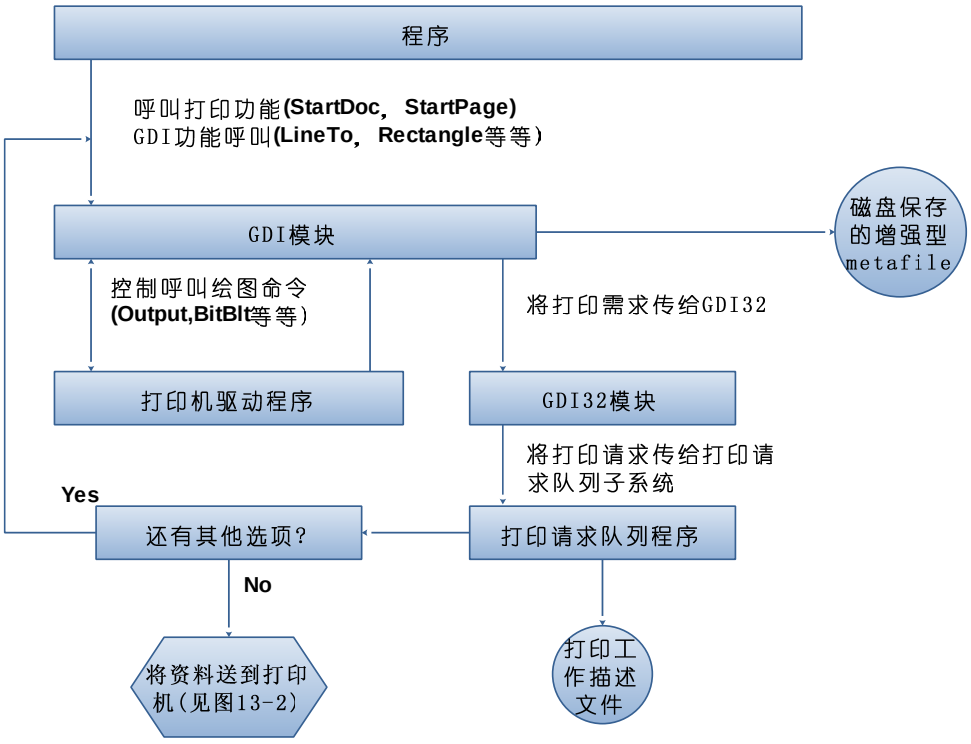


图13-1 应用程序、GDI模块、打印驱动程序和打印队列程序的交互作用过程

Windows后台打印程序实际上是几个组件的一种组合（见表13-1）。

表13-1

打印队列程序组件	说明
打印请求队列程序	将数据流传递给打印功能提供者
本地打印功能提供者	为本地打印机建立背景文件
网络打印功能提供者	为网络打印机建立背景文件
打印处理程序	将打印队列中与设备无关的数据转换为针对目的打印机的格式
打印端口监视程序	控件连结打印机的端口
打印语言监视程序	控件可以双向通讯的打印机，设定设备设定并检测打印机状态

打印队列程序可以减轻应用程序的打印负担。Windows在启动时就加载打印队列程序，因此，当应用程序开始打印时，它已经是活动的了。当程序行印一个文件时，GDI模块会建立包含打印输出数据的文件。后台打印程序的任务是将这些文件发往打印机。GDI模块发出一个消息来通知它一个新的打印作业开始，然后它开始读文件并将文件直接传送到打印机。为了传送这些文件，打印队列程序依照打印机所连结的并列端口或串行埠使用各种不同的通信函数。在打印队列程序向打印机发送文件的操作完成后，它就将包含输出数据的临时文件删除。这个交互作用过程如图13-2所示。

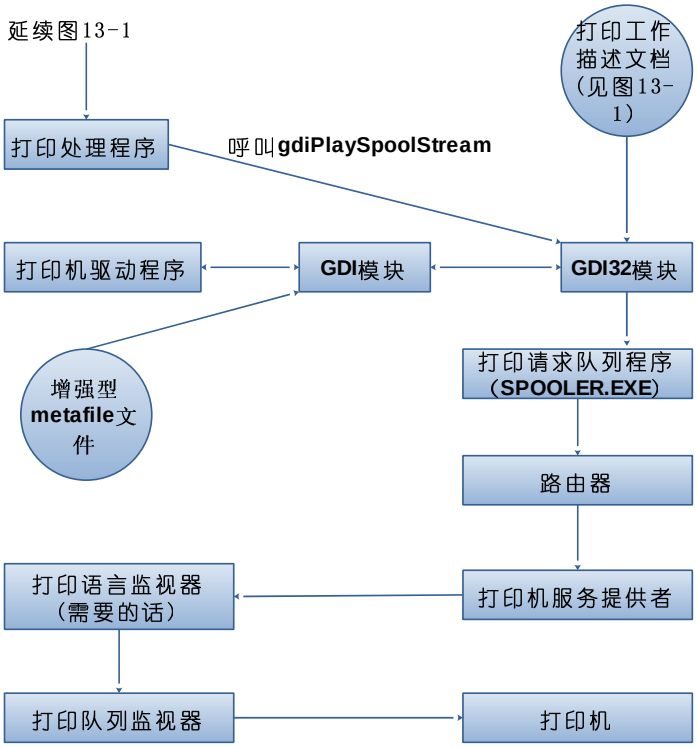


图13-2 后台打印程序的操作程序

这个程序的大部分对应用程序来说是透明的。从应用程序的角度来看，「打印」只发生在GDI模块将所有打印输出数据储存到磁盘文件中的时候，在这之后（如果打印是由第二个线程来操作的，甚至可以在这之前）应用程序可以自由地进行其它操作。真正的文件打印操作成了后台打印程序的任务，而不是应用程序的任务。通过打印机文件夹，使用者可以暂停打印作业、改变作业的优先级或取消打印作业。这种管理方式使应用程序能更快地将打印数据以实时方式打印，况且这样必须等到打印完一页后才能处理下一页。

我们已经描述了一般的打印原理，但还有一些例外情况。其中之一是Windows程序要使用打印机时，并非一定需要后台打印程序。使用者可以在打印机属性表格的详细数据属性页中关闭打印机的背景操作。

为什么使用者希望不使用背景操作呢？因为使用者可能使用了比Windows打印队列程序更快的硬件或软件后台打印程序，也可能是打印机在一个自身带有打印队列器的网络上使用。一般的规则是，使用一个打印队列程序比使用两个打印队列程序更快。去掉Windows后台打印程序可以加快打印速度，因为打印输出数据不必储存在硬盘上，而可以直接输出到打印机，并被外部的硬件打印队列器或软件的后台打印程序所接收。

如果没有启用Windows打印队列程序，GDI模块就不把来自设备驱动程序的打印输出数据存入文件中，而是将这些输出数据直接输出到打印输出埠。与打印队列程序进行的打印不同，GDI进行的打印一定会让应用程序暂停执行一段时间（特别是进行打印中的程序）直到打印完成。

还有另一个例外。通常，GDI模块将定义一页所需的所有函数存入一个增强型metafile中，然后替驱动程序定义的每个打印输出带输出一遍该metafile到打印驱动程序中。然而，如果打印驱动程序不需要打印分带的话，就不会建立这个metafile；GDI只需简单地将绘图函数直接送往驱动程序。进一步的变化是，应用程序也可能得承担起对打印输出数据进行打印分带的责任，这就使得应用程序中的打印程序代码更加复杂了，但却免去了GDI模块建立metafile的麻烦。这样，GDI只需简单地为每个输出带将函数传到打印驱动程序。

或许您现在已经发现了从一个Windows应用程序进行打印操作要比使用视讯显示器的负担更大，这样可能出现一些问题 – 特别是，如果GDI模块在建立metafile或打印输出文件时耗尽了磁盘空间。您可以更关切这些问题，并尝试着处理这些问题并告知使用者，或者您当然也可以置之不理。

对于一个应用程序，打印文件的第一步就是如何取得打印机设备的内容。

打印机设备内容

正如在视讯显示器上绘图前需要得到设备内容句柄一样，在打印之前，使用者必须取得一个打印机设备内容句柄。一旦有了这个句柄（并为建立一个新文件呼叫了StartDoc以及呼叫StartPage开始一页），就可以用与使用视讯显示设备内容句柄相同的方法来使用打印机设备内容句柄，该句柄即为各种GDI呼叫的第一个参数。

大多数应用程序经由呼叫PrintDlg函数打开一个标准的打印对话框（本章后面会展示该函数的用法）。这个函数还为使用者提供了一个在打印之前改变打印机或者指定其它特性的机会。然后，它将打印机设备内容句柄交给应用程序。该函数能够省下应用程序的一些工作。然而，某些应用程序（例如Notepad）仅需要取得打印机设备内容，而不需要那个对话框。要做到这一点，需要呼叫CreateDC函数。

在第五章中，您已知道如何通过如下的呼叫来为整个视讯显示器取得指向设备内容的句柄：

hdc = CreateDC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;

您也可以使用该函数来取得打印机设备内容句柄。然而，对打印机设备内容，CreateDC的一般语法为：

hdc = CreateDC (NULL, szDeviceName, NULL, pInitializationData) ;

pInitializationData参数一般被设为NULL。szDeviceName参数指向一个字符串，以告诉Windows打印机设备的名称。在设定设备名称之前，您必须知道有哪些打印机可用。

一个系统可能有不只一台连结着的打印机，甚至可以有其它程序，如传真软件，将自己伪装成打印机。不论连结的打印机有多少台，都只能有一台被认为是「目前的打印机」或者「内定打印机」，这是使用者最近一次选择的打印机。许多小型的Windows程序只使用内定打印机来进行打印。

取得内定打印机设备内容的方式不断在改变。目前，标准的方法是使用EnumPrinters函数来获得。该函数填入一个包含每个连结着的打印机信息的数组结构。根据所需的细节层次，您还可以选择几种结构之一作为该函数的参数。这些结构的名称为PRINTER_INFO_x，x是一个数字。

不幸的是，所使用的函数还取决于您的程序是在Windows 98上执行还是在Windows NT上执行。程序13-1展示了GetPrinterDC函数在两种操作系统上工作的用法。

程序13-1 GETPRNDC

GETPRNDC.C

```
/*------------------------------------------------------------------------
GETPRNDC.C -- GetPrinterDC function
------------------------------------------------------------------------*/
#include <windows.h>
HDC GetPrinterDC (void)
{
  DWORD dwNeeded, dwReturned ;
  HDC hdc ;
  PRINTER_INFO_4 * pinfo4 ;
  PRINTER_INFO_5 * pinfo5 ;

  if (GetVersion () & 0x80000000) // Windows 98
  {
```

```
    EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, NULL,
      0, &dwNeeded, &dwReturned) ;
    pinfo5 = (PRINTER_INFO_5*)malloc (dwNeeded) ;
    EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, (PBYTE) pinfo5,
      dwNeeded, &dwNeeded, &dwReturned) ;
    hdc = CreateDC (NULL, pinfo5->pPrinterName, NULL, NULL) ;
    free (pinfo5) ;
  }
  else
    //Windows NT
  {
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, NULL,
      0, &dwNeeded, &dwReturned) ;
    pinfo4 = (PRINTER_INFO_4*)malloc (dwNeeded) ;
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, (PBYTE) pinfo4,
      dwNeeded, &dwNeeded, &dwReturned) ;
    hdc = CreateDC (NULL, pinfo4->pPrinterName, NULL, NULL) ;
    free (pinfo4) ;
  }
  return hdc ;
}
```

这些函数使用GetVersion函数来确定程序是执行在Windows 98上还是Windows NT上。不管是什么操作系统，函数呼叫EnumPrinters两次：一次取得它所需结构的大小，一次填入结构。在Windows 98上，函数使用PRINTER_INFO_5结构；在Windows NT上，函数使用PRINTER_INFO_4结构。这些结构在EnumPrinters文件（/Platform SDK/Graphics and Multimedia Services/GDI/Printing and Print Spooler/Printing and Print Spooler Reference/Printing and Print Spooler Functions/EnumPrinters，范例小节的前面）中有说明，它们是「容易而快速」的。

修改后的DEVCAPS程序

第五章的DEVCAPS1程序只显示了从GetDeviceCaps函数获得的关于视讯显示的基本信息。程序13-2所示的新版本显示了关于视讯显示和连结到系统之所有打印机的更多信息。

程序13-2 DEVCAPS2

DEVCAPS2.C

```
/*--------------------------------------------------------------------------
DEVCAPS2.C -- Displays Device Capability Information (Version 2)
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
void DoBasicInfo (HDC, HDC, int, int) ;
void DoOtherInfo (HDC, HDC, int, int) ;
void DoBitCodedCaps (HDC, HDC, int, int, int) ;

typedef struct
{
  int iMask ;
  TCHAR * szDesc ;
}
BITS ;
#define IDM_DEVMODE 1000
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("DevCaps2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, NULL,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static TCHAR szDevice[32], szWindowText[64] ;
  static int cxChar, cyChar,nCurrentDevice = IDM_SCREEN,
    nCurrentInfo = IDM_BASIC ;
  static DWORD dwNeeded, dwReturned ;
  static PRINTER_INFO_4 * pinfo4 ;
  static PRINTER_INFO_5 * pinfo5 ;
  DWORD i ;
  HDC hdc, hdcInfo ;
  HMENU hMenu ;
  HANDLE hPrint ;
  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) ;
    // fall through
  case WM_SETTINGCHANGE:
    hMenu = GetSubMenu (GetMenu (hwnd), 0) ;

    while (GetMenuItemCount (hMenu) > 1)
      DeleteMenu (hMenu, 1, MF_BYPOSITION) ;

    // Get a list of all local and remote printers
    //
    // First, find out how large an array we need; this
    // call will fail, leaving the required size in dwNeeded
```

```
  //
  // Next, allocate space for the info array and fill it
  //
  // Put the printer names on the menu

  if (GetVersion () & 0x80000000) // Windows 98
  {
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, NULL,
      0, &dwNeeded, &dwReturned) ;

    pinfo5 = malloc (dwNeeded) ;

    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, (PBYTE) pinfo5,
      dwNeeded, &dwNeeded, &dwReturned) ;

    for (i = 0 ; i < dwReturned ; i++)
    {
      AppendMenu (hMenu, (i+1) % 16 ? 0 : MF_MENUBARBREAK, i + 1,
        pinfo5[i].pPrinterName) ;
    }
    free (pinfo5) ;
  }
  else
    // Windows NT
  {
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, NULL,
      0, &dwNeeded, &dwReturned) ;
    pinfo4 = malloc (dwNeeded) ;
    EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, (PBYTE) pinfo4,
      dwNeeded, &dwNeeded, &dwReturned) ;
    for (i = 0 ; i < dwReturned ; i++)
    {
      AppendMenu (hMenu, (i+1) % 16 ? 0 : MF_MENUBARBREAK, i + 1,
        pinfo4[i].pPrinterName) ;
    }
    free (pinfo4) ;
  }

  AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
  AppendMenu (hMenu, 0, IDM_DEVMODE, TEXT ("Properties")) ;

  wParam = IDM_SCREEN ;
  // fall through
case WM_COMMAND :
  hMenu = GetMenu (hwnd) ;

  if ( LOWORD (wParam) == IDM_SCREEN || // IDM_SCREEN & Printers
    LOWORD (wParam) < IDM_DEVMODE)
  {
    CheckMenuItem (hMenu, nCurrentDevice, MF_UNCHECKED) ;
    nCurrentDevice = LOWORD (wParam) ;
    CheckMenuItem (hMenu, nCurrentDevice, MF_CHECKED) ;
  }
  else if (LOWORD (wParam) == IDM_DEVMODE) // Properties selection
  {
    GetMenuString (hMenu, nCurrentDevice, szDevice,
      sizeof (szDevice) / sizeof (TCHAR), MF_BYCOMMAND);

    if (OpenPrinter (szDevice, &hPrint, NULL))
    {
      PrinterProperties (hwnd, hPrint) ;
      ClosePrinter (hPrint) ;
    }
  }
  else
    // info menu items
  {
    CheckMenuItem (hMenu, nCurrentInfo, MF_UNCHECKED) ;
    nCurrentInfo = LOWORD (wParam) ;
```

```
      CheckMenuItem (hMenu, nCurrentInfo, MF_CHECKED) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  case WM_INITMENUPOPUP :
    if (lParam == 0)
      EnableMenuItem (GetMenu (hwnd), IDM_DEVMODE,
      nCurrentDevice == IDM_SCREEMF_GRAYED : MF_ENABLED) ;
    return 0 ;
  case WM_PAINT :
    lstrcpy (szWindowText, TEXT ("Device Capabilities: ")) ;

    if (nCurrentDevice == IDM_SCREEN)
    {
      lstrcpy (szDevice, TEXT ("DISPLAY")) ;
      hdcInfo = CreateIC (szDevice, NULL, NULL, NULL) ;
    }
    else
    {
      hMenu = GetMenu (hwnd) ;
      GetMenuString (hMenu, nCurrentDevice, szDevice,
        sizeof (szDevice), MF_BYCOMMAND) ;
      hdcInfo = CreateIC (NULL, szDevice, NULL, NULL) ;
    }

    lstrcat (szWindowText, szDevice) ;
    SetWindowText (hwnd, szWindowText) ;

    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

    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) ;
}

void DoBasicInfo (HDC hdc, HDC hdcInfo, int cxChar, int cyChar)
{
  static struct
  {
    int nIndex ;
    TCHAR * szDesc ;
  }
  info[] =
```

```
  {
    HORZSIZE, TEXT ("HORZSIZE Width in millimeters:"),
      VERTSIZE, TEXT ("VERTSIZE Height in millimeters:"),
      HORZRES, TEXT ("HORZRES Width in pixels:"),
      VERTRES, TEXT ("VERTRES Height in raster lines:"),
      BITSPIXEL, TEXT ("BITSPIXEL Color bits per pixel:"),
      PLANES, TEXT ("PLANES Number of color planes:"),
      NUMBRUSHES, TEXT ("NUMBRUSHES Number of device brushes:"),
      NUMPENS, TEXT ("NUMPENS Number of device pens:"),
      NUMMARKERS, TEXT ("NUMMARKERS Number of device markers:"),
      NUMFONTS, TEXT ("NUMFONTS Number of device fonts:"),
      NUMCOLORS, TEXT ("NUMCOLORS Number of device colors:"),
      PDEVICESIZE, TEXT("PDEVICESIZESize of device structure:"),
      ASPECTX, TEXT("ASPECTX Relative width of pixel:"),
      ASPECTY, TEXT("ASPECTY Relative height of pixel:"),
      ASPECTXY, TEXT("ASPECTXY Relative diagonal of pixel:"),
      LOGPIXELSX, TEXT("LOGPIXELSX Horizontal dots per inch:"),
      LOGPIXELSY, TEXT("LOGPIXELSY Vertical dots per inch:"),
      SIZEPALETTE, TEXT("SIZEPALETTE Number of palette entries:"),
      NUMRESERVED, TEXT("NUMRESERVED Reserved palette entries:"),
      COLORRES, TEXT("COLORRES Actual color resolution:"),
      PHYSICALWIDTH, TEXT("PHYSICALWIDTH Printer page pixel width:"),
      PHYSICALHEIGHT,TEXT("PHYSICALHEIGHT Printer page pixel height:"),
      PHYSICALOFFSETX,TEXT("PHYSICALOFFSETX Printer page x offset:"),
      PHYSICALOFFSETY,TEXT("PHYSICALOFFSETY Printer page y offset:")
  } ;
  int i ;
  TCHAR szBuffer[80] ;

  for (i = 0 ; i < sizeof (info) / sizeof (info[0]) ; i++)
    TextOut (hdc, cxChar, (i + 1) * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("%-45s%8d"), info[i].szDesc,
    GetDeviceCaps (hdcInfo, info[i].nIndex))) ;
}

void DoOtherInfo (HDC hdc, HDC hdcInfo, int cxChar, int cyChar)
{
  static BITS clip[] =
  {
    CP_RECTANGLE, TEXT ("CP_RECTANGLE Can Clip To Rectangle:")
  } ;

  static BITS raster[] =
  {
    RC_BITBLT, TEXT ("RC_BITBLT Capable of simple BitBlt:"),
      RC_BANDING, TEXT ("RC_BANDING Requires banding support:"),
      RC_SCALING, TEXT ("RC_SCALING Requires scaling support:"),
      RC_BITMAP64, TEXT ("RC_BITMAP64 Supports bitmaps >64K:"),
      RC_GDI20_OUTPUT, TEXT ("RC_GDI20_OUTPUT Has 2.0 output calls:"),
      RC_DI_BITMAP, TEXT ("RC_DI_BITMAP Supports DIB to memory:"),
      RC_PALETTE, TEXT ("RC_PALETTE Supports a palette:"),
      RC_DIBTODEV, TEXT ("RC_DIBTODEV Supports bitmap conversion:"),
      RC_BIGFONT, TEXT ("RC_BIGFONT Supports fonts >64K:"),
      RC_STRETCHBLT,TEXT ("RC_STRETCHBLT Supports StretchBlt:"),
      RC_FLOODFILL, TEXT ("RC_FLOODFILL Supports FloodFill:"),
      RC_STRETCHDIB,TEXT ("RC_STRETCHDIB Supports StretchDIBits:")
  } ;

  static TCHAR * szTech[]= { TEXT ("DT_PLOTTER (Vector plotter)"),
    TEXT ("DT_RASDISPLAY (Raster display)"),
    TEXT ("DT_RASPRINTER (Raster printer)"),
    TEXT ("DT_RASCAMERA (Raster camera)"),
    TEXT ("DT_CHARSTREAM (Character stream)"),
    TEXT ("DT_METAFILE (Metafile)"),
    TEXT ("DT_DISPFILE (Display file)") } ;
  int i ;
  TCHAR szBuffer[80] ;

  TextOut (hdc, cxChar, cyChar, szBuffer,
```

```
    wsprintf (szBuffer, TEXT ("%-24s%04XH"), TEXT ("DRIVERVERSION:"),
    GetDeviceCaps (hdcInfo, DRIVERVERSION))) ;
  TextOut (hdc, cxChar, 2 * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("%-24s%-40s"), TEXT ("TECHNOLOGY:"),
    szTech[GetDeviceCaps (hdcInfo, TECHNOLOGY)])) ;
  TextOut (hdc, cxChar, 4 * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("CLIPCAPS (Clipping capabilities)"))) ;
  for (i = 0 ; i < sizeof (clip) / sizeof (clip[0]) ; i++)
    TextOut (hdc, 9 * cxChar, (i + 6) * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("%-45s %3s"), clip[i].szDesc,
    GetDeviceCaps (hdcInfo, CLIPCAPS) & clip[i].iMask ?
    TEXT ("Yes") : TEXT ("No"))) ;
  TextOut (hdc, cxChar, 8 * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("RASTERCAPS (Raster capabilities)"))) ;
  for (i = 0 ; i < sizeof (raster) / sizeof (raster[0]) ; i++)
    TextOut (hdc, 9 * cxChar, (i + 10) * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("%-45s %3s"), raster[i].szDesc,
    GetDeviceCaps (hdcInfo, RASTERCAPS) & raster[i].iMask ?
    TEXT ("Yes") : TEXT ("No"))) ;
}

void DoBitCodedCaps ( HDC hdc, HDC hdcInfo, int cxChar, int cyChar,int iType)
{
  static BITS curves[] =
  {
    CC_CIRCLES, TEXT ("CC_CIRCLES Can do circles:"),
      CC_PIE, TEXT ("CC_PIE Can do pie wedges:"),
      CC_CHORD, TEXT ("CC_CHORD Can do chord arcs:"),
      CC_ELLIPSES, TEXT ("CC_ELLIPSES Can do ellipses:"),
      CC_WIDE, TEXT ("CC_WIDE Can do wide borders:"),
      CC_STYLED, TEXT ("CC_STYLED Can do styled borders:"),
      CC_WIDESTYLED, TEXT ("CC_WIDESTYLED Can do wide and styled borders:"),
      CC_INTERIORS, TEXT ("CC_INTERIORS Can do interiors:")
  } ;

  static BITS lines[] =
  {
    LC_POLYLINE, TEXT ("LC_POLYLINE Can do polyline:"),
      LC_MARKER, TEXT ("LC_MARKER Can do markers:"),
      LC_POLYMARKER, TEXT ("LC_POLYMARKER Can do polymarkers"),
      LC_WIDE, TEXT ("LC_WIDE Can do wide lines:"),
      LC_STYLED, TEXT ("LC_STYLED Can do styled lines:"),
      LC_WIDESTYLED, TEXT ("LC_WIDESTYLED Can do wide and styled lines:"),
      LC_INTERIORS, TEXT ("LC_INTERIORS Can do interiors:")
  } ;

  static BITS poly[] =
  {
    PC_POLYGON,
      TEXT ("PC_POLYGON Can do alternate fill polygon:"),
      PC_RECTANGLE, TEXT ("PC_RECTANGLE Can do rectangle:"),
      PC_WINDPOLYGON,
      TEXT ("PC_WINDPOLYGON Can do winding number fill polygon:"),
      PC_SCANLINE, TEXT ("PC_SCANLINE Can do scanlines:"),
      PC_WIDE, TEXT ("PC_WIDE Can do wide borders:"),
      PC_STYLED, TEXT ("PC_STYLED Can do styled borders:"),
      PC_WIDESTYLED,
      TEXT ("PC_WIDESTYLED Can do wide and styled borders:"),
      PC_INTERIORS, TEXT ("PC_INTERIORS Can do interiors:")
  } ;

  static BITS text[] =
  {
    TC_OP_CHARACTER, TEXT ("TC_OP_CHARACTER Can do character output precision:"),
      TC_OP_STROKE, TEXT ("TC_OP_STROKE Can do stroke output precision:"),
      TC_CP_STROKE, TEXT ("TC_CP_STROKE Can do stroke clip precision:"),
      TC_CR_90, TEXT ("TC_CP_90 Can do 90 degree character rotation:"),
      TC_CR_ANY, TEXT ("TC_CR_ANY Can do any character rotation:"),
      TC_SF_X_YINDEP, TEXT ("TC_SF_X_YINDEP Can do scaling independent of X and Y:"),
```

```
    TC_SA_DOUBLE, EXT ("TC_SA_DOUBLE Can do doubled character for scaling:"),
    TC_SA_INTEGER, TEXT ("TC_SA_INTEGER Can do integer multiples for scaling:"),
    TC_SA_CONTIN, TEXT ("TC_SA_CONTIN Can do any multiples for exact scaling:"),
    TC_EA_DOUBLE, TEXT ("TC_EA_DOUBLE Can do double weight characters:"),
    TC_IA_ABLE, TEXT ("TC_IA_ABLE Can do italicizing:"),
    TC_UA_ABLE, TEXT ("TC_UA_ABLE Can do underlining:"),
    TC_SO_ABLE, TEXT ("TC_SO_ABLE Can do strikeouts:"),
    TC_RA_ABLE, TEXT ("TC_RA_ABLE Can do raster fonts:"),
    TC_VA_ABLE, TEXT ("TC_VA_ABLE Can do vector fonts:")
  } ;

  static struct
  {
    int iIndex ;
    TCHAR * szTitle ;
    BITS (*pbits)[] ;
    int iSize ;
  }
  bitinfo[] =
  {
    CURVECAPS, TEXT ("CURVCAPS (Curve Capabilities)"),
      (BITS (*)[]) curves, sizeof (curves) / sizeof (curves[0]),
      LINECAPS, TEXT ("LINECAPS (Line Capabilities)"),
      (BITS (*)[]) lines, sizeof (lines) / sizeof (lines[0]),
      POLYGONALCAPS, TEXT ("POLYGONALCAPS (Polygonal Capabilities)"),
      (BITS (*)[]) poly, sizeof (poly) / sizeof (poly[0]),
      TEXTCAPS, TEXT ("TEXTCAPS (Text Capabilities)"),
      (BITS (*)[]) text, sizeof (text) / sizeof (text[0])
  } ;

  static TCHAR szBuffer[80] ;
  BITS (*pbits)[] = bitinfo[iType].pbits ;
  int i, iDevCaps = GetDeviceCaps (hdcInfo, bitinfo[iType].iIndex) ;

  TextOut (hdc, cxChar, cyChar, bitinfo[iType].szTitle,
    lstrlen (bitinfo[iType].szTitle)) ;
  for (i = 0 ; i < bitinfo[iType].iSize ; i++)
    TextOut (hdc, cxChar, (i + 3) * cyChar, szBuffer,
    wsprintf (szBuffer, TEXT ("%-55s %3s"), (*pbits)[i].szDesc,
    iDevCaps & (*pbits)[i].iMask ? TEXT ("Yes") : TEXT ("No")));
}
```

DEVCAPS2.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
DEVCAPS2 MENU DISCARDABLE
BEGIN
POPUP "&Device"
BEGIN
MENUITEM "&Screen",IDM_SCREEN, CHECKED
END
POPUP "&Capabilities"
BEGIN
MENUITEM "&Basic Information",IDM_BASIC
MENUITEM "&Other Information",IDM_OTHER
MENUITEM "&Curve Capabilities",IDM_CURVE
MENUITEM "&Line Capabilities",IDM_LINE
MENUITEM "&Polygonal Capabilities",IDM_POLY
MENUITEM "&Text Capabilities",IDM_TEXT
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DevCaps2.rc
#define IDM_SCREEN  40001
#define IDM_BASIC   40002
#define IDM_OTHER   40003
#define IDM_CURVE   40004
#define IDM_LINE   40005
#define IDM_POLY   40006
#define IDM_TEXT   40007
```

因为DEVCAPS2只取得打印机的信息内容，使用者仍然可以从DEVCAPS2的菜单中选择所需打印机。如果使用者想比较不同打印机的功能，可以先用打印机文件夹增加各种打印驱动程序。

PrinterProperties呼叫

DEVCAPS2的「Device」菜单中上还有一个称为「Properties」的选项。要使用这个选项，首先得从 **Device**菜单中选择一个打印机，然后再选择**Properties**，这时弹出一个对话框。对话框从何而来呢？它由打印机驱动程序呼叫，而且至少还让使用者选择纸的尺寸。大多数打印机驱动也可以让使用者在「直印（portrait)」或「横印（landscape)」模式中进行选择。在直印模式（一般为内定模式）下，纸的短边是顶部。在横印模式下，纸的长边是顶部。如果改变该模式，则所作的改变将在DEVCAPS2程序从GetDeviceCaps函数取得的信息中反应出来：水平尺寸和分辨率将与垂直尺寸和分辨率交换。彩色绘图机的「Properties」对话框内容十分广泛，它们要求使用者输入安装在绘图机上之画笔的颜色和使用之绘图纸（或透明胶片）的型号。

所有打印机驱动程序都包含一个称为ExtDeviceMode的输出函数，它呼叫对话框并储存使用者输入的信息。有些打印机驱动程序也将这些信息储存在系统登录的自己拥有的部分中，有些则不然。那些储存信息的打印机驱动程序在下次执行Windows时将存取该信息。

允许使用者选择打印机的Windows程序通常只呼叫PrintDlg（本章后面我会展示用法)。这个有用的函数在准备打印时负责和使用者之间所有的通讯工作，并负责处理使用者要求的所有改变。当使用者单击「Properties」按钮时，PrintDlg还会启动属性表格对话框。

程序还可以通过直接呼叫打印机驱动程序的ExtDeviceMode或ExtDeveModePropSheet函数，来显示打印机的属性对话框，然而，我不鼓励您这样做。像DEVCAPS2那样，透过呼叫PrinterProperties来启动对话框会好得多。

PrinterProperties要求打印机对象的句柄，您可以通过OpenPrinter函数来得到。当使用者取消属性表格对话框时，PrinterProperties传回，然后使用者通过呼叫ClosePrinter，释放打印机句柄。DEVCAPS2就是这样做到这一点的。

程序首先取得刚刚在Device菜单中选择的打印机名称，并将其存入一个名为szDevice的字符数组中。

```
GetMenuString ( hMenu, nCurrentDevice, szDevice,
            sizeof (szDevice) / sizeof (TCHAR), MF_BYCOMMAND) ;
```

然后，使用OpenPrinter获得该设备的句柄。如果呼叫成功，那么程序接着呼叫PrinterProperties启动对话框，然后呼叫ClosePrinter释放设备句柄:

```
if (OpenPrinter (szDevice, &hPrint, NULL))
{
  PrinterProperties (hwnd, hPrint) ;
  ClosePrinter (hPrint) ;
}
```

检查BitBlt支持

您可以用GetDeviceCaps函数来取得页中可打印区的尺寸和分辨率（通常，该区域不会与整

张纸的大小相同)。如果使用者想自己进行缩放操作，也可以获得相对的像素宽度和高度。

打印机能力的大多数信息是用于GDI而不是应用程序的。通常，在打印机不能做某件事时，GDI会仿真出那项功能。然而，这是应用程序应该事先检查的。

以RASTERCAPS（「位映像支持」）参数呼叫GetDeviceCaps，它传回的RC_BITBLT位包含了另一个重要的打印机特性，该位标示设备是否能进行位块传送。大多数点阵打印机、激光打印机和喷墨打印机都能进行位块传送，而大多数绘图机却不能。不能处理位块传送的设备不支持下列GDI函数：CreateCompatibleDC、CreateCompatibleBitmap、PatBlt、BitBlt、StretchBlt、GrayString、DrawIcon、SetPixel、GetPixel、FloodFill、ExtFloodFill、FillRgn、FrameRgn、InvertRgn、PaintRgn、FillRect、FrameRect和InvertRect。这是在视讯显示器上使用GDI函数与在打印机上使用它们的唯一重要区别。

最简单的打印程序

现在可以开始打印了，我们尽可能简单地开始。事实上，我们的第一个程序只是让打印机走纸而已。程序13-3的FORMFEED程序，展示了打印所需的最小需求。

程序13-3 FORMFEED

FORMFEED.C

```
/*------------------------------------------------------------------------
FORMFEED.C -- Advances printer to next page
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
HDC GetPrinterDC (void) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int iCmdShow)
{
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("FormFeed") } ;
  HDC hdcPrint = GetPrinterDC () ;

  if (hdcPrint != NULL)
  {
    if (StartDoc (hdcPrint, &di) > 0)
      if (StartPage (hdcPrint) > 0 && EndPage (hdcPrint) > 0)
        EndDoc (hdcPrint) ;
      DeleteDC (hdcPrint) ;
  }
  return 0 ;
}
```

这个程序也需要前面程序13-1中的GETPRNDC.C文件。

除了取得打印机设备内容（然后再删除它）外，程序只呼叫了我们在本章前面讨论过的四个打印函数。FORMFEED首先呼叫StartDoc开始一个新的文件，它测试从StartDoc传回的值，只有传回值是正数时，才继续下去：

if (StartDoc (hdcPrint, &di) > 0)

StartDoc的第二个参数是指向DOCINFO结构的指针。该结构在第一个字段包含了结构的大小，在第二个字段包含了字符串「FormFeed」。当文件正在被打印或者在等待打印时，这个字符串将出现在打印机任务队列中的「Document Name」列中。通常，该字符串包含进行打印的应用程序名称和被打印的文件名称。

如果StartDoc成功（由一个正的传回值表示），那么FORMFEED呼叫StartPage，紧接着立即呼叫EndPage。这一程序将打印机推进到新的一页，再次对传回值进行测试：

if (StartPage (hdcPrint) > 0 && EndPage (hdcPrint) > 0)

最后，如果不出错，文件就结束：

EndDoc (hdcPrint) ;

要注意的是，只有当没出错时，才呼叫EndDoc函数。如果其它打印函数中的某一个传回错误代码，那么GDI实际上已经中断了文件的打印。如果打印机目前未打印，这种错误代码通常会使打印机重新设定。测试打印函数的传回值是检测错误的最简单方法。如果您想向使用者报告错误，就必须呼叫GetLastError来确定错误。

如果您写过MS-DOS下的简单利用打印机走纸的程序，就应该知道，对于大多数打印机，ASCII码12启动走纸。为什么不简单地使用C的链接库函数open，然后用write输出ASCII码12呢？当然，您完全可以这么做，但是必须确定打印机连结的是串行端口还是并列埠。然后您还要确定另外的程序（例如，打印队列程序）是不是正在使用打印机。您并不希望在文件打印到一半时被别的程序把正在打印的那张纸送出打印机，对不对？最后，您还必须确定ASCII码12是不是所连结打印机的走纸字符，因为并非所有打印机的走纸字符都是12。事实上，在PostScript中的走纸命令便不是12，而是单字showpage。

简单地说，不要试图直接绕过Windows；而应该坚持在打印中使用Windows函数。

打印图形和文字

在一个Windows程序中，打印所需的额外负担通常比FORMFEED程序高得多，而且还要用GDI函数来实际打印一些东西。我们来写个打印一页文字和图形的程序，采用FORMFEED程序中的方法，并加入一些新的东西。该程序将有三个版本PRINT1、PRINT2和PRINT3。为避免程序代码重复，每个程序都用前面所示的GETPRNDC.C文件和PRINT.C文件中的函数，如程序13-4所示。

程序13-4 PRINT

PRINT.C

```
/*---------------------------------------------------------------------------
PRINT.C -- Common routines for Print1, Print2, and Print3
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL PrintMyPage (HWND) ;

extern HINSTANCE hInst ;
extern TCHAR szAppName[] ;
extern TCHAR szCaption[] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
```

```
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hInst = hInstance ;
  hwnd = CreateWindow (szAppName, szCaption,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void PageGDICalls (HDC hdcPrn, int cxPage, int cyPage)
{
  static TCHAR szTextStr[] = TEXT ("Hello, Printer!") ;
  Rectangle (hdcPrn, 0, 0, cxPage, cyPage) ;
  MoveToEx (hdcPrn, 0, 0, NULL) ;
  LineTo (hdcPrn, cxPage, cyPage) ;
  MoveToEx (hdcPrn, cxPage, 0, NULL) ;
  LineTo (hdcPrn, 0, cyPage) ;

  SaveDC (hdcPrn) ;

  SetMapMode (hdcPrn, MM_ISOTROPIC) ;
  SetWindowExtEx (hdcPrn, 1000, 1000, NULL) ;
  SetViewportExtEx (hdcPrn, cxPage / 2, -cyPage / 2, NULL) ;
  SetViewportOrgEx (hdcPrn, cxPage / 2, cyPage / 2, NULL) ;

  Ellipse (hdcPrn, -500, 500, 500, -500) ;
  SetTextAlign (hdcPrn, TA_BASELINE | TA_CENTER) ;
  TextOut (hdcPrn, 0, 0, szTextStr, lstrlen (szTextStr)) ;

  RestoreDC (hdcPrn, -1) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int cxClient, cyClient ;
  HDC hdc ;
  HMENU hMenu ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    hMenu = GetSystemMenu (hwnd, FALSE) ;
    AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
    AppendMenu (hMenu, 0, 1, TEXT ("&Print")) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_SYSCOMMAND:
```

```
    if (wParam == 1)
    {
      if (!PrintMyPage (hwnd))
        MessageBox (hwnd, TEXT ("Could not print page!"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
      return 0 ;
    }
    break ;

  case WM_PAINT :
    hdc = BeginPaint (hwnd, &ps) ;

    PageGDICalls (hdc, cxClient, cyClient) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY :
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

PRINT.C包括函数WinMain、WndProc以及一个称为PageGDICalls的函数。PageGDICalls函数接收打印机设备内容句柄和两个包含打印页面宽度及高度的变量。这个函数还负责画一个包围整个页面的矩形，有两条对角线，页中间有一个椭圆（其直径是打印机高度和宽度中较小的那个的一半），文字「Hello, Printer!」位于椭圆的中间。

处理WM_CREATE消息时，WndProc将一个「Print」选项加到系统菜单上。选择该选项将呼叫PrintMyPage，此函数的功能在程序的三个版本中将不断增强。当打印成功时，PrintMyPage传回TRUE值，如果遇到错误时则传回FALSE。如果PrintMyPage传回FALSE，WndProc就会显示一个消息框以告知使用者发生了错误。

打印的基本程序

打印程序的第一个版本是PRINT1，见程序13-5。经编译后即可执行此程序，然后从系统菜单中选择「Print」。接着，GDI将必要的打印机输出储存在一个临时文件中，然后打印队列程序将它发送给打印机。

程序13-5 PRINT1

PRINT1.C

```
/*------------------------------------------------------------------------
PRINT1.C -- Bare Bones Printing
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
HDC GetPrinterDC (void) ; // in GETPRNDC.C
void PageGDICalls (HDC, int, int) ; // in PRINT.C

HINSTANCE hInst ;
TCHAR szAppName[] = TEXT ("Print1") ;
TCHAR szCaption[] = TEXT ("Print Program 1") ;

BOOL PrintMyPage (HWND hwnd)
{
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print1: Printing") } ;
  BOOL bSuccess = TRUE ;
  HDC hdcPrn ;
  int xPage, yPage ;

  if (NULL == (hdcPrn = GetPrinterDC ()))
    return FALSE ;
```

```
  xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
  yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

  if (StartDoc (hdcPrn, &di) > 0)
  {
    if (StartPage (hdcPrn) > 0)
    {
      PageGDICalls (hdcPrn, xPage, yPage) ;

      if (EndPage (hdcPrn) > 0)
        EndDoc (hdcPrn) ;
      else
        bSuccess = FALSE ;
    }
  }
  else
    bSuccess = FALSE ;

  DeleteDC (hdcPrn) ;
  return bSuccess ;
}
```

我们来看看PRINT1.C中的程序代码。如果PrintMyPage不能取得打印机的设备内容句柄，它就传回FALSE，并且WndProc显示消息框指出错误。如果函数成功取得了设备内容句柄，它就通过呼叫GetDeviceCaps来确定页面的水平和垂直大小（以像素为单位）。

xPage = GetDeviceCaps (hdcPrn, HORZRES) ;

yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

这不是纸的全部大小，只是纸的可打印区域。呼叫后，除了PRINT1在StartPage和EndPage呼叫之间呼叫PageGDICalls，PRINT1的PrintMyPage函数中的程序代码在结构上与FORMFEED中的程序代码相同。仅当呼叫StartDoc、StartPage和EndPage都成功时，PRINT1才呼叫EndDoc打印函数。

使用放弃程序来取消打印

对于大型文件，程序应该提供使用者在应用程序行印期间取消打印任务的便利性。也许使用者只要打印文件中的一页，而不是打印全部的537页。应该要能在印完全部的537页之前纠正这个错误。

在一个程序内取消一个打印任务需要一种被称为「放弃程序」的技术。放弃程序在程序中只是个较小的输出函数，使用者可以使用SetAbortProc函数将该函数的地址传给Windows。然后GDI在打印时，重复呼叫该程序，不断地问：「我是否应该继续打印？」

我们看看将放弃程序加到打印处理程序中去需要些什么，然后检查一些旁枝末节。放弃程序一般命名为AbortProc，其形式为：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
  //其它行程序
}
```

打印前，您必须通过呼叫SetAbortProc来登记放弃程序：

SetAbortProc (hdcPrn, AbortProc) ;

在呼叫StartDoc前呼叫上面的函数，打印完成后不必清除放弃程序。

在处理EndPage呼叫时（亦即，在将metafile放入设备驱动程序并建立临时打印文件时），GDI常常呼叫放弃程序。参数hdcPrn是打印机设备内容句柄。如果一切正常，iCode参数是0，如果GDI模块在生成临时文件时耗尽了磁盘空间，iCode就是SP_OUTOFDISK。

如果打印作业继续，那么AbortProc必须传回TRUE（非零）；如果打印作业异常结束，就传回

FALSE（零）。放弃程序可以被简化为如下所示的形式：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
  MSG msg ;
  while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return TRUE ;
}
```

这个函数看起来有点特殊，其实它看起来像是消息循环。使用者会注意到，这个「消息循环」呼叫 PeekMessage 而不是 GetMessage。我在第五章的 RANDRECT 程序中讨论过 PeekMessage。应该还记得，PeekMessage将会控制权返回给程序，而不管程序的消息队列中是否有消息存在。

只要PeekMessage传回TRUE，那么AbortProc函数中的消息循环就重复呼叫PeekMessage。TRUE 值表示 PeekMessage 已经找到一个消息，该消息可以通过 TranslateMessage 和 DispatchMessage 发送到程序的窗口消息处理程序。若程序的消息队列中没有消息，则PeekMessage的传回值为FALSE，因此AbortProc将控制权返回给Windows。

Windows如何使用AbortProc

当程序进行打印时，大部分工作发生在要呼叫EndPage时。呼叫EndPage前，程序每呼叫一次GDI绘图函数，GDI模块只是简单地将另一个记录加到磁盘上的metafile中。当GDI得到EndPage后，对打印页中由设备驱动程序定义的每个输出带，GDI都将该metafile送入设备驱动程序中。然后，GDI将打印机驱动程序建立的打印输出储存到一个文件中。如果没有启用后台打印，那么GDI模块必须自动将该打印输出写入打印机。

在EndPage呼叫期间，GDI模块呼叫您设定的放弃程序。通常iCode参数为0，但如果由于存在未打印的其它临时文件，而造成GDI执行时磁盘空间不够，iCode参数就为SP_OUTOFDISK（通常您不会检查这个值，但是如果愿意，您可以进行检查）。放弃程序随后进入PeekMessage循环从自己的消息队列中找寻消息。

如果在程序的消息队列中没有消息，PeekMessage会传回FALSE，然后放弃程序跳出它的消息循环并给GDI模块传回一个TRUE值，指示打印应该继续进行。然后GDI模块继续处理EndPage呼叫。

如果有错误发生，那么GDI将中止打印程序，这样，放弃程序的主要目的是允许使用者取消打印。为此，我们还需要一个显示「Cancel」按钮的对话框，让我们采用两个独立的步骤。首先，我们在建立PRINT2程序时增加一个放弃程序，然后在PRINT3中增加一个带有「Cancel」按钮的对话框，使放弃程序可用。

实作放弃程序

现在快速复习一下放弃程序的机制。可以定义一个如下所示的放弃程序：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
  MSG msg ;
  while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return TRUE ;
}
```

当您想打印什么时，使用下面的呼叫将指向放弃程序的指针传给Windows：

SetAbortProc (hdcPrn, AbortProc) ;

在呼叫StartDoc之前进行这个呼叫就行了。

不过，事情没有这么简单。我们忽视了AbortProc程序中PeekMessage循环这个问题，它是个很大的问题。只有在程序处于打印程序时，AbortProc程序才会被呼叫。如果在AbortProc中找到一个消息并把它传送给窗口消息处理程序，就会发生一些非常令人讨厌的事情：使用者可以从菜单中再次选择「Print」，但程序已经处于打印例程之中。程序在打印前一个文件的同时，使用者也可以把一个新文件加载到程序里。使用者甚至可以退出程序！如果这种情况发生了，所有使用者程序的窗口都将被清除。当打印例程执行结束时，除了退到不再有效的窗口例程之外，您无处可去。

这种东西会把人搞得晕头转向，而我们的程序对此并未做任何准备。正是由于这个原因，当设定放弃程序时，首先应禁止程序的窗口接受输入，使它不能接受键盘和鼠标输入。可以用以下的函数完成这项工作：

EnableWindow (hwnd, FALSE) ;

它可以禁止键盘和鼠标的输入进入消息队列。因此在打印程序中，使用者不能对程序做任何工作。当打印完成时，应重新允许窗口接受输入：

EnableWindow (hwnd, TRUE) ;

您可能要问，既然没有键盘或鼠标消息进入消息队列，为什么我们还要进行AbortProc中的TranslateMessage和DispatchMessage呼叫呢？实际上并不一定非得需要TranslateMessage，但是，我们必须使用DispatchMessage，处理WM_PAINT消息进入消息队列中的情况。如果WM_PAINT消息没有得到窗口消息处理程序中的BeginPaint和EndPaint的适当处理，由于PeekMessage不再传回FALSE，该消息就会滞留在队列中并且妨碍工作。

当打印期间阻止窗口处理输入消息时，您的程序不会进行显示输出。但使用者可以切换到其它程序，并在那里进行其它工作，而后台打印程序则能继续将输出文件送到打印机。

程序13-6所示的PRINT2程序在PRINT1中增加了一个放弃程序和必要的支持 – 呼叫AbortProc函数并呼叫EnableWindow两次（第一次阻止窗口接受输入消息，第二次启用窗口）。

程序13-6 PRINT2

PRINT2.C

```
/*------------------------------------------------------------------------
PRINT2.C -- Printing with Abort Procedure
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
HDC GetPrinterDC (void) ; // in GETPRNDC.C
void PageGDICalls (HDC, int, int) ; // in PRINT.C

HINSTANCE hInst ;
TCHAR szAppName[] = TEXT ("Print2") ;
TCHAR szCaption[] = TEXT ("Print Program 2 (Abort Procedure)") ;

BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
  MSG msg ;
  while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return TRUE ;
```

```
}

BOOL PrintMyPage (HWND hwnd)
{
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print2: Printing") } ;
  BOOL bSuccess = TRUE ;
  HDC hdcPrn ;
  short xPage, yPage ;

  if (NULL == (hdcPrn = GetPrinterDC ()))
    return FALSE ;
  xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
  yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

  EnableWindow (hwnd, FALSE) ;
  SetAbortProc (hdcPrn, AbortProc) ;
  if (StartDoc (hdcPrn, &di) > 0)
  {
    if (StartPage (hdcPrn) > 0)
    {
      PageGDICalls (hdcPrn, xPage, yPage) ;
      if (EndPage (hdcPrn) > 0)
        EndDoc (hdcPrn) ;
      else
        bSuccess = FALSE ;
    }
  }
  else
    bSuccess = FALSE ;
  EnableWindow (hwnd, TRUE) ;
  DeleteDC (hdcPrn) ;
  return bSuccess ;
}
```

增加打印对话框

PRINT2还不能令人十分满意。首先，这个程序没有直接指示出何时开始打印和何时结束打印。只有将鼠标指向程序并且发现它没有反应时，才能断定它仍然在处理PrintMyPage例程。PRINT2在进行背景处理时也没有给使用者提供取消打印作业的机会。

您可能注意到，大多数Windows程序都为使用者提供了一个取消目前正在进行打印操作的机会。一个小的对话框出现在屏幕上，它包括一些文字和「Cancel」按键。在GDI将打印输出储存到磁盘文件或（如果停用打印队列程序）打印机正在打印的整个期间，程序都显示这个对话框。它是一个非系统模态对话框，您必须提供对话程序。

通常称这个对话框为「放弃对话框」，称这种对话程序为「放弃对话程序」。为了更清楚地把它和「放弃程序」区别开来，我们称这种对话程序为「打印对话程序」。放弃程序（名为AbortProc）和打印对话程序（将命名为PrintDlgProc）是两个不同的输出函数。如果想以一种专业的Windows式打印方式进行打印工作，就必须拥有这两个函数。

这两个函数的交互作用方式如下：AbortProc中的PeekMessage循环得被修改，以便将非系统模态对话框的消息发送给对话框窗口消息处理程序。PrintDlgProc必须处理WM_COMMAND消息，以检查「Cancel」按钮的状态。如果「Cancel」钮被按下，就将一个叫做bUserAbort的整体变量设为TRUE。AbortProc传回的值正好和bUserAbort相反。您可能还记得，如果AbortProc传回TRUE会继续打印，传回FALSE则放弃打印。在PRINT2中，我们总是传回TRUE。现在，使用者在打印对话框中按下「Cancel」按钮时将传回FALSE。程序13-7所示的PRINT3程序实作了这个处理方式。

程序13-7 PRINT3

PRINT3.C

```
/*---------------------------------------------------------------
PRINT3.C -- Printing with Dialog Box
(c) Charles Petzold, 1998
---------------------------------------------------------------*/
#include <windows.h>
HDC GetPrinterDC (void) ; // in GETPRNDC.C
voidPageGDICalls (HDC, int, int) ; // in PRINT.C

HINSTANCE hInst ;
TCHAR szAppName[] = TEXT ("Print3") ;
TCHAR szCaption[] = TEXT ("Print Program 3 (Dialog Box)") ;

BOOL bUserAbort ;
HWND hDlgPrint ;

BOOL CALLBACK PrintDlgProc (HWND hDlg, UINT message,
                        WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG:
    SetWindowText (hDlg, szAppName) ;
    EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE, MF_GRAYED) ;
    return TRUE ;

  case WM_COMMAND:
    bUserAbort = TRUE ;
    EnableWindow (GetParent (hDlg), TRUE) ;
    DestroyWindow (hDlg) ;
    hDlgPrint = NULL ;
    return TRUE ;
  }
  return FALSE ;
}

BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
  MSG msg ;
  while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return !bUserAbort ;
}

BOOL PrintMyPage (HWND hwnd)
{
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print3: Printing") } ;
  BOOL bSuccess = TRUE ;
  HDC hdcPrn ;
  int xPage, yPage ;

  if (NULL == (hdcPrn = GetPrinterDC ()))
    return FALSE ;
  xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
  yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

  EnableWindow (hwnd, FALSE) ;
  bUserAbort = FALSE ;
  hDlgPrint = CreateDialog (hInst, TEXT ("PrintDlgBox"),
    hwnd, PrintDlgProc) ;
  SetAbortProc (hdcPrn, AbortProc) ;
  if (StartDoc (hdcPrn, &di) > 0)
```

```
  {
    if (StartPage (hdcPrn) > 0)
    {
      PageGDICalls (hdcPrn, xPage, yPage) ;
      if (EndPage (hdcPrn) > 0)
        EndDoc (hdcPrn) ;
      else
        bSuccess = FALSE ;
    }
  }
  else
    bSuccess = FALSE ;
  if (!bUserAbort)
  {
    EnableWindow (hwnd, TRUE) ;
    DestroyWindow (hDlgPrint) ;
  }

  DeleteDC (hdcPrn) ;
  return bSuccess && !bUserAbort ;
}
```

PRINT.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
/////////////////////////////////////////////////////////////////////////////
// Dialog
PRINTDLGBOX DIALOG DISCARDABLE 20, 20, 186, 63
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
PUSHBUTTON    "Cancel",IDCANCEL,67,42,50,14
CTEXT     "Cancel Printing",IDC_STATIC,7,21,172,8
END
```

如果您使用PRINT3，那么最好临时暂停使用后台打印；否则，只有在打印队列程序从PRINT3中接收数据时才可见到的「Cancel」按钮可能会很快消失，让您根本没有机会去按它。如果您按「Cancel」按钮时打印并不立即终止（特别是在一个慢速打印机上），不要惊讶。打印机有一个内部缓冲区，在打印机停止之前其中的数据必须全部送出，按「Cancel」只是告诉GDI不要向打印机的缓冲区发送更多的数据而已。

PRINT3增加了两个整体变量：一个是叫做bUserAbort的布尔变量，另一个是叫做hDlgPrint的对话框窗口句柄。PrintMyPage函数将bUserAbort初始化为FALSE。与PRINT2一样，程序的主窗口是不接收输入消息的。指向AbortProc的指标用于SetAbortProc呼叫中，而指向PrintDlgProc的指标用于CreateDialog呼叫中。CreateDialog传回的窗口句柄储存在hDlgPrint中。

现在，AbortProc中的消息循环如下：

```
while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
{
  if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
}
return !bUserAbort ;
```

只有在bUserAbort为FALSE，也就是使用者还没有终止打印工作时，这段程序代码才会呼叫

PeekMessage。IsDialogMessage函数用来将消息发送给非系统模态对话框。和普通的非系统模态对话框一样，对话框窗口的句柄在这个呼叫之前受到检查。AbortProc的传回值正好与bUserAbort相反。开始时，bUserAbort为FALSE，因此AbortProc传回TRUE，表示继续进行打印；但是bUserAbort可能在打印对话程序中被设定为TRUE。

PrintDlgProc函数是相当简单的。处理WM_INITDIALOG时，该函数将窗口标题设定为程序名称，并且停用系统菜单上的「Close」选项。如果使用者按下了「Cancel」钮，PrintDlgProc将收到WM_COMMAND消息：

```
case WM_COMMAND :
  bUserAbort = TRUE ;
  EnableWindow (GetParent (hDlg), TRUE) ;
  DestroyWindow (hDlg) ;
  hDlgPrint = NULL ;
  return TRUE ;
```

将bUserAbort设定为TRUE，则说明使用者已经决定取消打印操作，主窗口被启动，而对话框被清除（按顺序完成这两项活动是很重要的，否则，在Windows中执行其它程序之一将变成活动程序，而您的程序将消失到背景中）。与通常的情况一样，将hDlgPrint设定为NULL，防止在消息循环中呼叫IsDialogMessage。

只有在AbortProc用PeekMessage找到消息，并用IsDialogMessage将它们传送给对话框窗口消息处理程序时，这个对话框才接收消息。只有在GDI模块处理EndPage函数时，才呼叫AbortProc。如果GDI发现AbortProc的传回值是FALSE，它将控制权从EndPage传回到PrintMyPage。它不传回错误码。至此，PrintMyPage认为打印页已经发完了，并呼叫EndDoc函数。但是，由于GDI模块还没有完成对EndPage呼叫的处理，所以不会打印出什么东西来。

有些清除工作尚待完成。如果使用者没在对话框中取消打印作业，那么对话框仍然会显示着。PrintMyPage重新启用它的主窗口并清除对话框：

```
if (!bUserAbort)
{
  EnableWindow (hwnd, TRUE) ;
  DestroyWindow (hDlgPrint) ;
}
```

两个变量会通知您发生了什么事：bUserAbort可以告诉您使用者是否终止了打印作业，bSuccess会告诉您是否出了故障，您可以用这些变量来完成想做的工作。PrintMyPage只简单地对它们进行逻辑上的AND运算，然后把值传回给WndProc：

return bSuccess && !bUserAbort ;

为POPPAD增加打印功能

现在准备在POPPAD程序中增加打印功能，并且宣布POPPAD已告完毕。这需要第十一章中的各个POPPAD文件，此外，还需要程序13-8中的POPPRNT.C文件。

程序13-8 POPPRNT

POPPRNT.C

```
/*-----------------------------------------------------------------------
POPPRNT.C -- Popup Editor Printing Functions
-----------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "resource.h"

BOOL bUserAbort ;
```

```
HWND hDlgPrint ;

BOOL CALLBACK PrintDlgProc ( HWND hDlg, UINT msg, WPARAM wParam,LPARAM lParam)
{
  switch (msg)
  {
  case WM_INITDIALOG :
    EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE, MF_GRAYED) ;
    return TRUE ;

  case WM_COMMAND :
    bUserAbort = TRUE ;
    EnableWindow (GetParent (hDlg), TRUE) ;
    DestroyWindow (hDlg) ;
    hDlgPrint = NULL ;
    return TRUE ;
  }
  return FALSE ;
}

BOOL CALLBACK AbortProc (HDC hPrinterDC, int iCode)
{
  MSG msg ;
  while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
  {
    if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return !bUserAbort ;
}

BOOL PopPrntPrintFile (HINSTANCE hInst, HWND hwnd, HWND hwndEdit,
                       PTSTR szTitleName)
{
  static DOCINFO di = { sizeof (DOCINFO) } ;
  static PRINTDLG pd ;
  BOOL bSuccess ;
  int yChar, iCharsPerLine, iLinesPerPage, iTotalLines,
    iTotalPages, iPage, iLine, iLineNum ;
  PTSTR pstrBuffer ;
  TCHAR szJobName [64 + MAX_PATH] ;
  TEXTMETRIC tm ;
  WORD iColCopy, iNoiColCopy ;

  // Invoke Print common dialog box
  pd.lStructSize = sizeof (PRINTDLG) ;
  pd.hwndOwner = hwnd ;
  pd.hDevMode = NULL ;
  pd.hDevNames = NULL ;
  pd.hDC = NULL ;
  pd.Flags = PD_ALLPAGES | PD_COLLATE |
    PD_RETURNDC | PD_NOSELECTION ;
  pd.nFromPage = 0 ;
  pd.nToPage = 0 ;
  pd.nMinPage = 0 ;
  pd.nMaxPage = 0 ;
  pd.nCopies = 1 ;
  pd.hInstance = NULL ;
  pd.lCustData = 0L ;
  pd.lpfnPrintHook = NULL ;
  pd.lpfnSetupHook = NULL ;
  pd.lpPrintTemplateName = NULL ;
  pd.lpSetupTemplateName = NULL ;
  pd.hPrintTemplate = NULL ;
  pd.hSetupTemplate = NULL ;
```

```
if (!PrintDlg (&pd))
  return TRUE ;

if (0 == (iTotalLines = SendMessage (hwndEdit, EM_GETLINECOUNT, 0, 0)))
  return TRUE ;

// Calculate necessary metrics for file
GetTextMetrics (pd.hDC, &tm) ;
yChar = tm.tmHeight + tm.tmExternalLeading ;

iCharsPerLine = GetDeviceCaps (pd.hDC, HORZRES) / tm.tmAveCharWidth ;
iLinesPerPage = GetDeviceCaps (pd.hDC, VERTRES) / yChar ;
iTotalPages = (iTotalLines + iLinesPerPage - 1) / iLinesPerPage ;

// Allocate a buffer for each line of text
pstrBuffer = malloc (sizeof (TCHAR) * (iCharsPerLine + 1)) ;

// Display the printing dialog box
EnableWindow (hwnd, FALSE) ;
bSuccess = TRUE ;
bUserAbort = FALSE ;

hDlgPrint = CreateDialog (hInst, TEXT ("PrintDlgBox"),
  hwnd, PrintDlgProc) ;

SetDlgItemText (hDlgPrint, IDC_FILENAME, szTitleName) ;
SetAbortProc (pd.hDC, AbortProc) ;

// Start the document
GetWindowText (hwnd, szJobName, sizeof (szJobName)) ;
di.lpszDocName = szJobName ;
if (StartDoc (pd.hDC, &di) > 0)
{
  // Collation requires this loop and iNoiColCopy
  for (iColCopy = 0 ;
  iColCopy < ((WORD) pd.Flags & PD_COLLATE ? pd.nCopies : 1) ;
  iColCopy++)
  {
    for (iPage = 0 ; iPage < iTotalPages ; iPage++)
    {
      for (iNoiColCopy = 0 ;
      iNoiColCopy < (pd.Flags & PD_COLLATE ? 1 : pd.nCopies);
      iNoiColCopy++)
      {
        // Start the page
        if (StartPage (pd.hDC) < 0)
        {
          bSuccess = FALSE ;
          break ;
        }
        // For each page, print the lines
        for (iLine = 0 ; iLine < iLinesPerPage ; iLine++)
        {
          iLineNum = iLinesPerPage * iPage + iLine ;
          if (iLineNum > iTotalLines)
            break ;
          *(int *) pstrBuffer = iCharsPerLine ;
          TextOut (pd.hDC, 0, yChar * iLine, pstrBuffer,
            (int) SendMessage (hwndEdit, EM_GETLINE,
            (WPARAM) iLineNum, (LPARAM) pstrBuffer));
        }
        if (EndPage (pd.hDC) < 0)
        {
          bSuccess = FALSE ;
          break ;
        }

        if (bUserAbort)
          break ;
```

```
      }

      if (!bSuccess || bUserAbort)
        break ;
    }

    if (!bSuccess || bUserAbort)
      break ;
  }
}
else
  bSuccess = FALSE ;
if (bSuccess)
  EndDoc (pd.hDC) ;

if (!bUserAbort)
{
  EnableWindow (hwnd, TRUE) ;
  DestroyWindow (hDlgPrint) ;
}

free (pstrBuffer) ;
DeleteDC (pd.hDC) ;
return bSuccess && !bUserAbort ;
}
```

与POPPAD尽量利用Windows高阶功能来简化程序的方针一致，POPPRNT.C文件展示了使用PrintDlg函数的方法。这个函数包含在通用对话框链接库（common dialog box library）中，使用一个PRINTDLG型态的结构。

通常，程序的「File」菜单中有个「Print」选项。当使用者选中「Print」选项时，程序可以初始化PRINTDLG结构的字段，并呼叫PrintDlg。

PrintDlg显示一个对话框，它允许使用者选择打印页的范围。因此，这个对话框特别适用于像POPPAD这样能打印多页文件的程序。这种对话框同时也给出了一个确定副本份数的编辑区和名为「Collate（逐份打印）」的复选框。「逐份打印」影响着多个副本页的顺序。例如，如果文件是3页，使用者要求打印三份副本，则这个程序能以两种顺序之一打印它们。选择逐份打印后的副本的页码顺序为1、2、3、1、2、3、1、2、3，未选择逐份打印的副本的页码顺序是1、1、1、2、2、2、3、3、3。程序在这里应负起的责任就是以正确的顺序打印副本。

这个对话框也允许使用者选择非内定打印机，它包括一个标记为「Properties」的按钮，可以启动设备模式对话框。这样，至少允许使用者选择直印或横印。

从PrintDlg函数传回后，PRINTDLG结构的字段指明打印页的范围和是否对多个副本进行逐份打印。这个结构同时也给出了准备使用的打印机设备内容句柄。

在POPPRNT.C中，PopPrntPrintFile函数（当使用者在「File」菜单里选中「Print」选项时，它由POPPAD呼叫）呼叫PrintDlg，然后开始打印文件。PopPrntPrintFile完成某些计算，以确定一行能容纳多少字符和一页能容纳多少行。这个程序涉及到呼叫GetDeviceCaps来确定页的分辨率，呼叫GetTextMetrics来确定字符的大小。

这个程序通过发送一条EM_GETLINECOUNT消息给编辑控件来取得文件中的总行数（在变量iTotalLines中）。储存各行内容的缓冲区配置在局部内存中。对每一行，缓冲区的第一个字被设定为该行中字符的数目。把EM_GETLINE消息发送给编辑控件可以把一行复制到缓冲区中，然后用TextOut把这一行送到打印机设备内容中（POPPRNT.C还没有聪明到对超出打印宽度的文字换到下一行去处理。在第十七章我们会讨论这种文字绕行的技术）。

为了确定副本份数，应注意打印文字的处理方式包括两个for循环。第一个for循环使用了一个叫作iColCopy的变量，当使用者指定将副本逐份打印时，它将会起作用。第二个for循环使用了一

个叫作iNonColCopy的变量，当不对副本进行逐份打印时，它将起作用。

如果StartPage或EndPage传回一个错误，或者如果bUserAbort为TRUE，那么这个程序退出增加页号的那个for循环。如果放弃程序的传回值是FALSE，则EndPage不传回错误。正是由于这个原因，在下一页开始之前，要直接测试bUserAbort。如果没有报告错误，则进行EndDoc呼叫：

```
if (!bError)
  EndDoc (hdcPrn) ;
```

您可能想通过打印多页文件来测试POPPAD。您可以从打印任务窗口中监视打印进展情况。在GDI处理完第一个EndPage呼叫之后，首先打印的文件将显示在打印任务窗口中。此时，后台打印程序开始把文件发送到打印机。然后，如果在POPPAD中取消打印作业，那么后台打印程序将终止打印，这也就是放弃程序传回FALSE的结果。当文件出现在打印任务窗口中，您也可以透过从「Document」菜单中选择「Cancel Printing」来取消打印作业，在这种情况下， POPPAD中的EndPage呼叫会传回一个错误。

Windows的程序设计的新手经常会抱住AbortDoc函数不放，但实际上这个函数几乎不在打印中使用。像在POPPAD中看到的那样，使用者几乎随时可以取消打印作业，或者通过POPPAD的打印对话框及通过打印任务窗口。这两种方法都不需要程序使用AbortDoc函数。 POPPAD中允许AbortDoc的唯一时刻是在对StartDoc的呼叫和对EndPage的第一个呼叫之间，但是程序很快就会执行过去，以至不再需要AbortDoc。

图13-3显示出正确打印多页文件之打印函数的呼叫顺序。检查bUserAbort的值是否为TRUE的最佳位置是在每个EndPage函数之后。只有当对先前的打印函数的呼叫没有产生错误时，才使用EndDoc函数。实际上，如果任何一个打印函数的呼叫出现错误，那么表演就结束了，同时您也可以回家了。

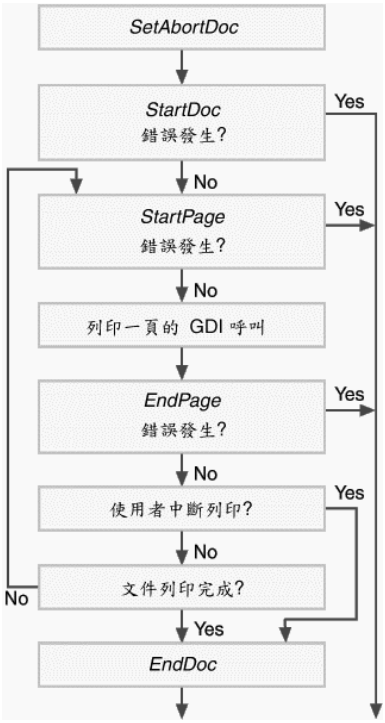


图13-3 打印一个文件时的函数呼叫顺序

第十四章 位图和Bitblt

位图是一个二维的位数组，它与图像的像素一一对应。当现实世界的图像被扫描成位图以后，图像被分割成网格，并以像素作为取样单位。在位图中的每个像素值指明了一个单位网格内图像的平均颜色。单色位图每个像素只需要一位，灰色或彩色位图中每个像素需要多个位。

位图代表了Windows程序内储存图像信息的两种方法之一。储存图像信息的另一种形式是metafile，我将在第十八章讨论。Metafile储存的就是对图像如何生成的描述，而不是将图像以数字化的图标代表。

以后我将更详细地讨论，Microsoft Windows 3.0定义了一种称为设备无关位图（DIB：device-independent bitmap）。我将在下一章讨论DIB。本章主要讨论GDI位图对象，这是一种在Windows中比DIB更早支持的位图形数据。如同本章大量的范例程序所说明的，这种比DIB位图更早被Windows支持的图形格式仍然有其利用价值。

位图入门

位图和metafile在计算机图形处理世界中都占有一席之地。位图经常用来表示来自真实世界的复杂图像，例如数字化的照片或者视讯图像。Metafile更适合于描述由人或者机器产生的图像，比如建筑蓝图。位图和metafile都能存于内存或作为文件存于磁盘上，并且都能通过剪贴簿在Windows应用程序之间传输。

位图和metafile的区别在于位映像图像和向量图像之间的差别。位映像图像用离散的像素来处理输出设备；而向量图像用笛卡尔坐标系统来处理输出设备，其线条和填充对象能被个别拖移。现在大多数的图像输出设备是位映像设备，这包括视讯显示、点阵打印机、激光打印机和喷墨打印机。而笔式绘图机则是向量输出设备。

位图有两个主要的缺点。第一个问题是容易受设备依赖性的影响。最明显的就是对颜色的依赖性，在单色设备上显示彩色位图的效果总是不能令人满意的。另一个问题是位图经常暗示了特定的显示分辨率和图像纵横比。尽管位图能被拉伸和缩小，但是这样的处理通常包括复制或删除像素的某些行和列，这样会破坏图像的大小。而metafile在放大缩小后仍然能保持图形样貌不受破坏。

位图的第二个缺点是需要很大的储存空间。例如，描述完整的640×480像素，16色的视频图形数组（VGA：Video Graphics Array）屏幕的一幅位图需要大于150 KB的空间；一幅1024×768，并且每个像素为24位颜色的图像则需要大于2 MB的空间。Metafile需要通常比位图来得少的空间。位图的储存空间由图像的大小及其包含的颜色决定，而metafile的储存空间则由图像的复杂程度和它所包含的GDI指令数决定。

然而，位图优于metafile之处在于速度。将位图复制给视讯显示器通常比复制基本图形文件的速度要快。最近几年，压缩技术允许压缩位图的文件大小，以使它能有效地通过电话线传输并广泛地用于Internet的网页上。

位图的来源

位图可以手工建立，例如，使用Windows 98附带的「小画家」程序。一些人宁愿使用位映像绘图软件也不使用向量绘图软件。他们假定：图形最后一定会复杂到不能用线条跟填充区域来表达。

位图图像也能由计算机程序计算生成。尽管大多数计算生成的图像能按向量图形metafile储

存，但是高清晰度的画面或碎形图样通常还是需要位图。

现在，位图通常用于描述真实世界的图像，并且有许多硬设备能让您把现实世界的图像输入到计算机。这类硬件通常使用 **电荷耦合设备**（CCD：charge-coupled device），这种设备接触到光就释放电荷。有时这些CCD单元能排列成一组，一个像素对应一个CCD；为节约开支，只用一行CCD扫描图像。

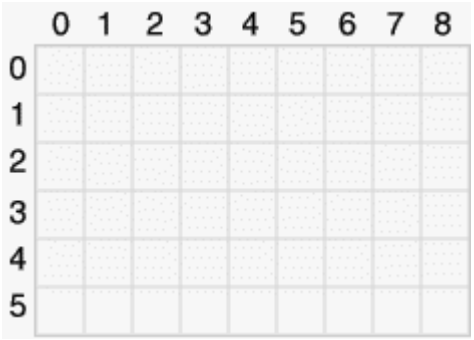
在这些计算机CCD设备中，**扫描仪**是最古老的。它用一行CCD沿着纸上图像（例如照片）的表面扫描。CCD根据光的强度产生电荷。模拟数字转换器（ADC：Analog-to-digital converters）把电荷转换为数字讯号，然后排列成位图。

便携式摄像机也利用CCD单元组来捕捉影像。通常，这些影像是记录到录像带上。不过，这些视讯输出也能直接进入 **影像捕捉器**（frame grabber），该设备能把模拟视讯信号转换为一组像素值。这些影像捕捉器与任何兼容的视讯信号来源都能同时使用，例如VCR、光盘、DVD播放机或有线电视译码器。

最近，数字照相机的价位对于家庭使用者来说开始变得负担得起了。它看起来很像普通照相机。但是数字照相机不使用底片，而用一组CCD来拦截图像，并且在ADC内部把数字图像直接储存在照相机内的内存中。通常，数字照相机与计算机的接口要通过串行端口。

位图尺寸

位图呈矩形，并有空间尺寸，图像的高度和宽度都以像素为单位。例如，此网格可描述一个很小的位图：宽度为9像素，高度为6像素，或者更简单地计为9×6：



习惯上，位图的速记尺寸是先给出宽度。位图总数为9×6或者54像素。我将经常使用符号cx和cy来表示位图的宽度和高度。c表示计数，因此cx和cy是沿着x轴（水平）和y轴（垂直）的像素数。

我们能根据x和y坐标来描述位图上具体的像素。一般（并不都是这样），在网格内计算像素时，位图开始于图像的左上角。这样，在此位图右下角的像素坐标就是(8, 5)。因为从0开始计数，所以此值比图像的宽度和高度小1。

位图的空间尺寸通常也指定了分辨率，但这是一个有争议的词。我们说我们的视讯显示有640×480的分辨率，但是激光打印机的分辨率只有每英寸300点。我喜欢用后一种情况中分辨率的意思作为每单位像素的数量。位图在这种意义上的分辨率指的是位图在特定测量单位中的像素数。不管怎样，当我使用分辨率这个词语时，其定义的内容应该是明确的。

位图是矩形的，但是计算机内存空间是线性的。通常（但并不都是这样）位图按列储存在内存中，且从顶列像素开始到底列结束。（DIB是此规则的一个主要例外）。每一列，像素都从最左边的

像素开始依次向右储存。这就好像储存几列文字中的各个字符。

颜色和位图

除空间尺寸以外，位图还有颜色尺寸。这里指的是每个像素所需要的位数，有时也称为位图的**颜色深度**（color depth）、**位数**（bit-count）或 **位/像素**（bpp：bits per pixel）数。位图中的每个像素都有相同数量的颜色位。

每像素1位的位图称为**二阶**（bilevel）、 **二色**（bicolor）或者**单色** （monochrome）位图。每像素可以是0或1，0表示黑色，1可以表示白色，但并不总是这样。对于其它颜色，一个像素就需要有多个位。可能的颜色值等于2位数值。用2位可以得到4种颜色，用4位可以得16种颜色，8位可得到256种颜色，16位可得到65,536种颜色，而24位可得到16,777,216种颜色。

如何将颜色位的组合与人们所熟悉的颜色相对应是目前处理位图时经常碰到（而且常常是灾难）的问题。

实际的设备

位图可按其颜色位数来分类；在Windows的发展过程中，不同的位图颜色格式取决于常用视讯显示卡的功能。实际上，我们可把视讯显示内存看作是一幅巨大的位图 – 我们从显示器上就可以看见。

Windows 1.0多数采用的显示卡是IBM的彩色图像适配器（CGA：Color Graphics Adapter）和单色图形卡（HGC：Hercules Graphics Card）。HGC是单色设备，而CGA也只能在Windows以单色图形模式使用。单色位图现在还很常用（例如，鼠标的光标一般为单色），而且单色位图除显示图像以外还有其它用途。

随着增强型图形显示卡（EGA：Enhanced Graphics Adapter）的出现，Windows使用者开始接触16色的图形。每个像素需要4个颜色位。（实际上，EGA比这里所讲的更复杂，它还包括一个64种颜色的调色盘，应用程序可以从中选择任意的16种颜色，但Windows只按较简单的方法使用EGA）。在EGA中使用的16种颜色是黑、白、两种灰色、高低亮度的红色、绿和蓝（三原色）、青色（蓝和绿组合的颜色）。现在认为这16种颜色是Windows的最低颜色标准。同样，其它16色位图也可以在Windows中显示。大多数的图标都是16色的位图。通常，简单的卡通图像也可以用这16种颜色制作。

在16色位图中的颜色编码有时称为IRGB（高亮红绿蓝：Intensity-Red-Green-Blue），并且实际上是源自IBM CGA文字模式下最初使用的十六种颜色。每个像素所用的4个IRGB颜色位都映像为表14-1所示的Windows十六进制RGB颜色。

表14-1

IRGB	RGB颜色	颜色名称
0000	00-00-00	黑
0001	00-00-80	暗蓝
0010	00-80-00	暗绿
0011	00-80-80	暗青
0100	80-00-00	暗红
0101	80-00-80	暗洋红
0110	80-80-00	暗黄
0111	C0-C0-C0	亮灰
1000	80-80-80	暗灰

1001	00-00-FF	蓝
1010	00-FF-00	绿
1011	00-FF-FF	青
1100	FF-00-00	红
1101	FF-00-FF	洋红
1110	FF-FF-00	黄
1111	FF-FF-FF	白

EGA的内存组成了四个「颜色面」，也就是说，定义每个像素颜色的四位在内存中是不连续的。然而，这样组织显示内存便于使所有的亮度位都排列在一起、所有的红色位都排在一起，等等。这样听起来就好像一种设备依赖特性，即Windows程序写作者不需要了解所有细节，但这时应或多或少地知道一些。不过，这些颜色面会出现在一些API呼叫中，例如GetDeviceCaps和CreateBitmap。

Windows 98和Microsoft Windows NT需要VGA或分辨率更高的图形卡。这是目前公认的显示卡的最低标准。

1987年，IBM最早发表视讯图像数组（Video Graphics Array：VGA）以及PS/2系列的个人计算机。它提供了许多不同的显示模式，但最好的图像模式（Windows也使用其中之一）是水平显示640个像素，垂直显示480个像素，带有16种颜色。要显示256种颜色，最初的VGA必须切换到320×240的图形模式，这种像素数不适合Windows的正常工作。

一般人们已经忘记了最初VGA卡的颜色限制，因为其它硬件制造商很快就开发了「Super-VGA」（SVGA）显示卡，它包括更多的视讯内存，可显示256种颜色并有多于640×480的模式。这是现在的标准，而且也是一件好事，因为对于现实世界中的图像来说，16种颜色过于简单，有些不适合。

显示256种颜色的显示卡模式采用每像素8位。不过，这些8位值都不必与实际的颜色相符。事实上，显示卡提供了「调色盘对照表（palette lookup table）」，该表允许软件指定这8位的颜色值，以便与实际颜色相符合。在Windows中，应用程序不能直接存取调色盘对照表。实际上，Windows储存了256种颜色中的20种，而应用程序可以通过「Windows调色盘管理器」来自订其余的236种颜色。关于这些内容，我将在第十六章详细介绍。调色盘管理器允许应用程序在256色显示器上显示实际位图。Windows所储存的20种颜色如表14-2所示。

表14-2

IRGB	RGB颜色	颜色名称
00000000	00-00-00	黑
00000001	80-00-00	暗红
00000010	00-80-00	暗绿
00000011	80-80-00	暗黄
00000100	00-00-80	暗蓝
00000101	80-00-80	暗洋红
00000110	00-80-80	暗青
00000111	C0-C0-C0	亮灰
00001000	C0-DC-C0	美元绿
00001001	A6-CA-F0	天蓝
11110110	FF-FB-F0	乳白

11110111	A0-A0-A4	中性灰
11111000	80-80-80	暗灰
11111001	FF-00-00	红
11111010	00-FF-00	绿
11111011	FF-FF-00	黄
11111100	00-00-FF	蓝
11111101	FF-00-FF	洋红
11111110	00-FF-FF	青
11111111	FF-FF-FF	白

最近几年，True-Color显示卡很普遍，它们在每像素使用16位或24位。有时每像素虽然用了16位，其中有1位不用，而其它15位主要近似于红、绿和蓝。这样红、绿和蓝每种都有32色阶，组合起来就可以达到32,768种颜色。更普遍的是，6位用于绿色（人类对此颜色最敏感），这样就可得到65,536种颜色。对于非技术性的PC使用者来说，他们并不喜欢看到诸如32,768或65,536之类的数字，因此通常将这种视讯显示卡称为Hi-Color显示卡，它能提供数以千计的颜色。

到了每个像素24位时，我们总共有了16,777,216种颜色（或者True Color、数百万的颜色），每个像素使用3字节。这与今后的标准很相似，因为它大致代表了人类感官的极限而且也很方便。

在呼叫GetDeviceCaps时（参见第五章的DEVCAPS程序），您能利用BITSPIXEL和PLANES常数来获得显示卡的颜色单位，这些值显示如表14-3所示

表14-3

BITSPIXEL	PLANES	颜色数
1	1	2
1	4	16
8	1	256
15或16	1	32,768或65 536
24或32	1	16 777 216

最近，您应该不会再碰到单色显示器了，但即便碰到了，您的应用程序也应该不会发生问题。

GDI支援的位图

Windows图形设备接口（GDI：Graphics Device Interface）从1.0版开始支持位图。不过，一直到Windows 3.0以前，Windows下唯一支持GDI对象的只有位图，以位图句柄来使用。这些GDI位图对象是单色的，或者与实际的图像输出设备（例如视讯显示器）有相同的颜色单位。例如，与16色VGA兼容的位图有四个颜色面。问题是这些颜色位图不能储存，也不能用于颜色单位不同的图像输出设备（如每像素占8位就可以产生256种颜色的设备）上。

从Windows 3.0开始，定义了一种新的位图格式，我们称之为设备无关位图（device-independent bitmap），或者DIB。DIB包括了自己的调色盘，其中显示了与RGB颜色相对应的像素位。DIB能显示在任何位映像输出设备上。这里唯一的问题是DIB的颜色通常一定会转换成设备实际表现出来的颜色。

与DIB同时，Windows 3.0还介绍了「Windows调色盘管理器」，它让程序能够从显示的256种颜色中自订颜色。就像我们在第十六章所看到的那样，应用程序通常在显示DIB时使用「调色盘管理器」。

Microsoft在Windows 95（和Windows NT 4.0）中扩展了DIB的定义，并且在Windows 98（和

Windows NT 5.0）中再次扩展。这些扩展增加了所谓的「图像颜色管理器（ICM：Image Color Management），并允许DIB更精确地指定图像所需要的颜色。我将在第十五章简要讨论ICM。

不论DIB多么重要，在处理位图时，早期的GDI位图对象依然扮演了重要的角色。掌握位图使用方式的最好方法是按各种用法在演进发展的时间顺序来学习，先从GDI位图对象和位块传输的概念开始。

位块传输

我前面提到过，您可以把整个视讯显示器看作是一幅大位图。您在屏幕上见到的像素由储存在视讯显示卡上内存中的位来描述。任何视讯显示的矩形区域也都是一个位图，其大小是它所包含的行列数。

让我们从将图像从视讯显示的一个区域复制到另一个区域，开始我们在位图世界的旅行吧！这个是强大的BitBlt函数的工作。

Bitblt（读作「bit blit」）代表「位块传输（bit-block transfer）」。BLT起源于一条汇编语言指令，该指令在DEC PDP-10上用来传输内存块。术语「bitblt」第一次用在图像上与Xerox Palo Alto Research Center（PARC）设计的SmallTalk系统有关。在SmallTalk中，所有的图形输出操作都使用bitblt。程序写作者有时将blt用作动词，例如：「Then I wrote some code to blt the happy face to the screen and play a wave file.」

BitBlt函数移动的是像素，或者（更明确地）是一个位映像图块。您将看到，术语「传输（transfer）」与BitBlt函数不尽相同。此函数实际上对像素执行了一次位操作，而且可以产生一些有趣的结果。

简单的BitBlt

程序14-1所示的BITBLT程序用BitBlt函数将程序系统的菜单图标（位于程序Windows的左上角）复制到它的显示区域。

程序14-1 BITBLT

BITBLT.C

```
/*---------------------------------------------------------------------
BITBLT.C -- BitBlt Demonstration
(c) Charles Petzold, 1998
---------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("BitBlt") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_INFORMATION) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
```

```
  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("BitBlt Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int cxClient, cyClient, cxSource, cySource ;
  HDC hdcClient, hdcWindow ;
  int x, y ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    cxSource = GetSystemMetrics (SM_CXSIZEFRAME) +
      GetSystemMetrics (SM_CXSIZE) ;
    cySource = GetSystemMetrics (SM_CYSIZEFRAME) +
      GetSystemMetrics (SM_CYCAPTION) ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdcClient = BeginPaint (hwnd, &ps) ;
    hdcWindow = GetWindowDC (hwnd) ;

    for (y = 0 ; y < cyClient ; y += cySource)
      for (x = 0 ; x < cxClient ; x += cxSource)
      {
        BitBlt (hdcClient, x, y, cxSource, cySource,
          hdcWindow, 0, 0, SRCCOPY) ;
      }

      ReleaseDC (hwnd, hdcWindow) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

但为什么只用了一个BitBlt呢？实际上，那个BITBLT用系统菜单图标的多个副本来填满显示区域（在此情况下是信息方块中普遍使用的IDI_INFORMATION图标），如图14-1所示。

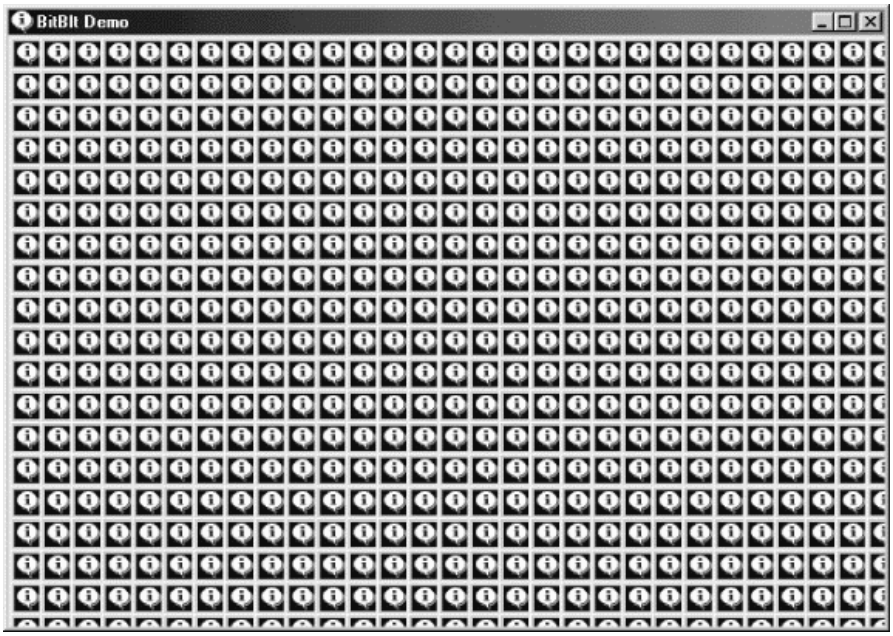


图14-1 BITBLT的屏幕显示

BitBlt函数从称为「来源」的设备内容中将一个矩形区的像素传输到称为「目的(destination)」的另一个设备内容中相同大小的矩形区。此函数的语法如下：

BitBlt (hdcDst, xDst, yDst, cx, cy, hdcSrc, xSrc, ySrc, dwROP) ;

来源和目的设备内容可以相同。

在BITBLT程序中，目的设备内容是窗口的显示区域，设备内容句柄从BeginPaint函数获得。来源设备内容是应用程序的整个窗口，此设备内容句柄从GetWindowDC获得的。很明显地，这两个设备内容指的是同一个实际设备（视讯显示器）。不过，这两个设备内容的坐标原点不同。

xSrc和ySrc参数指明了来源图像左上角的坐标位置。在BITBLT中，这两个参数设为0，表示图像从来源设备内容（也就是整个窗口）的左上角开始，cx和cy参数是图像的宽度和高度。BITBLT根据从GetSytemMetrics函数获得的信息来计算这些值。

xDst和yDst参数表示了复制图像位置左上角的坐标位置。在BITBLT中，这两个参数设定为不同的值以便多次复制图像。对于第一次BitBlt呼叫，这两个参数设制为0，将图像复制到显示区域的左上角位置。

BitBlt的最后一个参数是位映像操作型态。我将简短地讨论一下这个值。

请注意，BitBlt是从实际视讯显示内存传输像素，而不是从系统菜单图标的其它图像传输。如果您移动BITBLT窗口以使部分系统菜单图标移出屏幕，然后调整BITBLT窗口的尺寸使其重画，这时您将发现BITBLT显示区域中显示的是菜单图标的一部分。BitBlt函数不再存取整个图像。

在BitBlt函数中，来源和目的设备内容可以相同。您可以重新编写BITBLT以使WM_PAINT处理执行以下内容：

```
BitBlt (hdcClient, 0, 0, cxSource, cySource,
       hdcWindow, 0, 0, SRCCOPY) ;
for (y = 0 ; y < cyClient ; y += cySource)
for (x = 0 ; x < cxClient ; x += cxSource)
{
  if (x > 0 || y > 0)
    BitBlt (hdcClient, x, y, cxSource, cySource,
    hdcClient, 0, 0, SRCCOPY) ;
```

```
}
```

这将与前面显示的BITBLT一样产生相同的效果，只是显示区域左上角比较模糊。

在BitBlt内的最大限制是两个设备内容必须是兼容的。这意味着或者其中之一必须是单色的，或者两者的每个像素都相同的位数。总而言之，您不能用此方法将屏幕上的某些图形复制到打印机。

拉伸位图

在BitBlt函数中，目的图像与来源图像的尺寸是相同的，因为函数只有两个参数来说明宽度和高度。如果您想在复制时拉伸或者压缩图像尺寸，可以使用StretchBlt函数。StretchBlt函数的语法如下：

```
StretchBlt (hdcDst, xDst, yDst, cxDst, cyDst,
          hdcSrc, xSrc, ySrc, cxSrc, cySrc, dwROP) ;
```

此函数添加了两个参数。现在的函数就分别包含了目的和来源各自的宽度和高度。STRETCH程序展示了StretchBlt函数，如程序14-2所示。

程序14-2　STRETCH

STRETCH.C

```
/*---------------------------------------------------------------------------
STRETCH.C -- StretchBlt Demonstration
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("Stretch") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_INFORMATION) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("StretchBlt Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
```

```
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxClient, cyClient, cxSource, cySource ;
  HDC hdcClient, hdcWindow ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    cxSource = GetSystemMetrics (SM_CXSIZEFRAME) +
      GetSystemMetrics (SM_CXSIZE) ;

    cySource = GetSystemMetrics (SM_CYSIZEFRAME) +
      GetSystemMetrics (SM_CYCAPTION) ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
    hdcClient = BeginPaint (hwnd, &ps) ;
    hdcWindow = GetWindowDC (hwnd) ;

    StretchBlt (hdcClient, 0, 0, cxClient, cyClient,
      hdcWindow, 0, 0, cxSource, cySource, MERGECOPY) ;

    ReleaseDC (hwnd, hdcWindow) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

此程序只有呼叫了StretchBlt函数一次，但是利用此函数以系统菜单图标填充了整个显示区域，如图14-2所示。

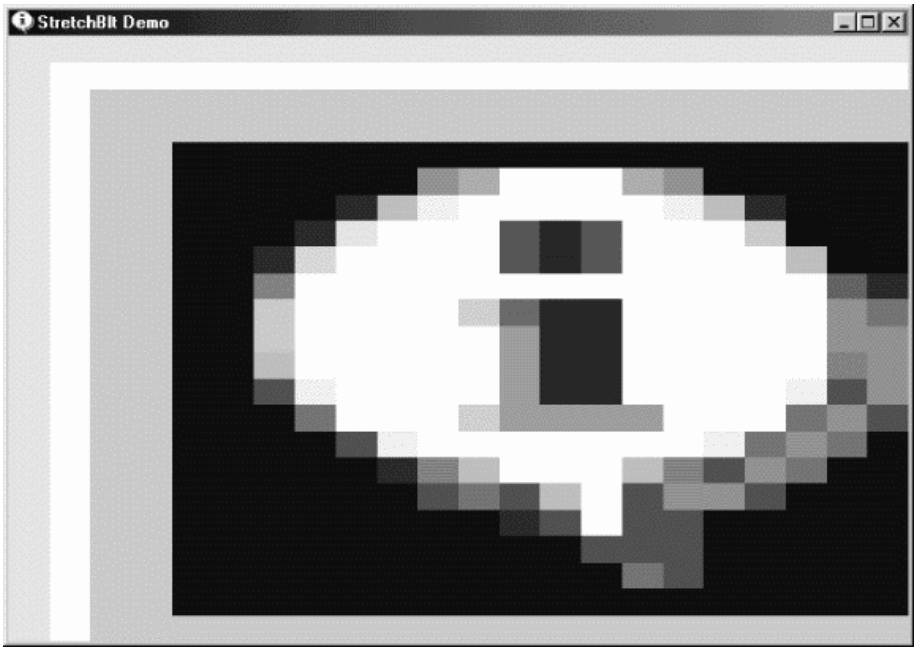


图14-2 STRETCH的屏幕显示

BitBlt和StretchBlt函数中所有的坐标与大小都是依据逻辑单位的。但是当您在BitBlt函数中定义了两个不同的设备内容，而这两个设备内容虽然参考同一个实际设备，却各自有着不同的映像模式，这时将发生什么结果呢？如果出现这种情况，呼叫BitBlt产生的结果就显得不明确了：cx和cy参数都是逻辑单位，而它们同样应用于来源设备内容和目的设备内容中的矩形区。所有的坐标和尺寸必须在实际的位传输之前转换为设备坐标。因为cx和cy值同时用于来源和目的设备内容，所以此值必须转换为设备内容自己的单位。

当来源和目的设备内容相同，或者两个设备内容都使用MM_TEXT图像模式时，设备单位下的矩形尺寸在两个设备内容中会是相同的，然后才由Windows进行像素对像素的转换。不过，如果设备单位下的矩形尺寸在两个设备内容中不同时，则Windows就把此工作转交给更通用的StretchBlt函数。

StretchBlt也允许水平或垂直翻转图像。如果cxSrc和cxDst标记（转换成设备单位以后）不同，那么StretchBlt就建立一个镜像：左右翻转。在STRETCH程序中，通过将xDst参数改为cxClient并将cxDst参数改成-cxClient，您就可以做到这一点。如果cySrc和cyDst不同，则StretchBlt会上下翻转图像。要在STRETCH程序中测试这一点，可将yDst参数改为cyClient并将cyDst参数改成-cyClient。

StretchBlt模式

使用StretchBlt会碰到一些与位图大小缩放相关的一些根本问题。在扩展一个位图时，StretchBlt必须复制像素行或列。如果放大倍数不是原图的整数倍，那么此操作会造成产生的图像有些失真。

如果目的矩形比来源矩形小，那么StretchBlt在缩小图像时就必须把两行（或列）或者多行（或列）的像素合并到一行（或列）。完成此操作有四种方法，它根据设备内容伸展模式属性来选择其中一种方法。您可使用SetStretchBltMode函数来修改这个属性。

SetStretchBltMode (hdc, iMode) ;

iMode可取下列值：

BLACKONWHITE或者STRETCH_ANDSCANS（内定）如果两个或多个像素得合并成一个像素，那么StretchBlt会对像素执行一个逻辑AND运算。这样的结果是只有全部的原始像素是白色时该像素才为白色，其实际意义是黑色像素控制了白色像素。这适用于白背景中主要是黑色的单色位图。

WHITEONBLACK或STRETCH_ORSCANS 如果两个或多个像素得合并成一个像素，那么StretchBlt执行逻辑OR运算。这样的结果是只有全部的原始像素都是黑色时才是黑色，也就是说由白色像素决定颜色。这适用于黑色背景中主要是白色的单色位图。

COLORONCOLOR或STRETCH_DELETESCANS StretchBlt简单地消除像素行或列，而没有任何逻辑组合。这是通常是处理彩色位图的最佳方法。

HALFTONE或STRETCH_HALFTONE Windows根据组合起来的来源颜色来计算目的的平均颜色。这将与半调调色盘联合使用，第十六章将展示这一程序。

Windows还包括用于取得目前伸展模式的GetStretchBltMode函数。

位映像操作

BITBLT和STRETCH程序简单地将来源位图复制给了目的位图，在过程中也可能进行了缩放。这是把SRCCOPY作为BitBlt和StretchBlt函数最后一个参数的结果。SRCCOPY只是您能在这些函数中

使用的256个位映像操作中的一个。让我们先在STRETCH程序中做一个别的实验，然后再系统地研究位映像操作。

尽量用NOTSRCCOPY来代替SRCCOPY。与它们名称一样，位映像操作在复制位图时转换其颜色。在显示区域窗口，所有的颜色转换：黑色变成白色、白色变成黑色，蓝色变成黄色。现在试一下SRCINVERT，您将得到同样效果。如果试一下BLACKNESS，正如其名称一样，整个显示区域都将变成黑色，而WHITENESS则使其变成白色。

现在试一试用下列三条叙述来代替StretchBlt呼叫：

```
SelectObject (hdcClient, CreateHatchBrush (HS_DIAGCROSS, RGB (0, 0, 0)));
StretchBlt (hdcClient, 0, 0, cxClient, cyClient,
          hdcWindow, 0, 0, cxSource, cySource, MERGECOPY) ;
DeleteObject (hdcClient, GetStockObject (WHITE_BRUSH)) ;
```

这次，您将在图像上看到一个菱形的画刷，这是什么？

我在前面说过，BitBlt和StretchBlt函数不是简单的位块传输。此函数实际在下面三种图像间执行位操作。

Source 来源位图，拉伸或压缩（如果有必要）到目的矩形的尺寸。

Destination 在BitBlt或StretchBlt呼叫之前的目的矩形。

Pattern 在目的设备内容中选择的目前画刷，水平或垂直地复制到目的矩形范围内。

结果是复制到了目的矩形中。

位映像操作与我们在第五章遇到的绘图模式在概念上相似。绘图模式采用图像对象的控件方式，例如一条线就组合成一个目的。我们知道有16种绘图模式 – 也就是说，对象中的0和1画出时，唯一结果就是目的中0和1的组合。

使用BitBlt和StretchBlt的位映像操作包含了三个对象的组合，这将产生256种位映像操作。有256种方法来组合来源位图、目的位图和图案。有15种位映像操作已经命名 – 其中一些名称其实还不能够清楚清楚说明其意义 – 它们定义在WINGDI.H里头，其余的都有数值，列在/Platform SDK/Graphics and Multimedia Services/GDI/Raster Operation Codes/Ternary Raster Operations之中。

有名称的15种ROP代码见表14-4。

表14-4

图案（P）：11110000 来源（s）：11001100 目的（D）：10101010		布尔操作	ROP代码	名称
结果：	00000000	0	0x000042	BLACKNESS
	00010001	~(S?#160;D)	0x1100A6	NOTSRCERASE
	00110011	~S	0x330008	NOTSRCCOPY
	01000100	S & ~D	0x440328	SRCERASE
	01010101	~D	0x550009	DSTINVERT
	01011010	P ^ D	0x5A0049	PATINVERT
	01100110	S ^ D	0x660046	SRCINVERT
	10001000	S & D	0x8800C6	SRCAND

	10111011	~S?#160;D	0xBB0226	MERGEPAINT
	11000000	P & S	0xC000CA	MERGECOPY
	11001100	S	0xCC0020	SRCCOPY
	11101110	S?#160;D	0xEE0086	SRCPAINT
	11110000	P	0xF00021	PATCOPY
	11111011	P?#160; ~S?#160; D	0xFB0A09	PATPAINT
	11111111	1	0xFF0062	WHITENESS

此表格对于理解和使用位映像操作非常重要，因此我们应花点时间来研究。

在这个表格中，「ROP代码」行的值将传递给BitBlt或StretchBlt的最后一个参数；在「名称」行中的值在WINGDI.H定义。ROP代码的低字组协助设备驱动程序传输位映像操作。高字组是0到255之间的数值。此数值与第2列的图案的位相同，这是在图案、来源和显示在顶部的目的之间进行位操作的结果。「布尔运算」列按C语法显示图案、来源和目的的组合方式。

要开始了解此表，最简单的办法是假定您正处理一个单色系统（每像素1位）其中0代表黑色，1代表白色。BLACKNESS操作的结果是不管是来源、目的和图案是什么，全部为零，因此目的将显示黑色。类似地，WHITENESS总导致目的呈白色。

现在假定您使用位映像操作PATCOPY。这导致结果位与图案位相同，而忽略了来源和目的位图。换句话说，PATCOPY简单地将目前图案复制给了目的矩形。

PATPAINT位映像操作包含一个更复杂的操作。其结果相同于在图案、目的和反转的来源之间进行位或操作。当来源位图是黑色（0）时，其结果总是白色（1）；当来源是白色（1）时，只要图案或目的为白色，则结果就是白色。换句话说，只有来源为白色而图案和目的都是黑色时，结果才是黑色。

彩色显示时每个像素都使用了多个位。BitBlt和StretchBlt函数对每个颜色位都分别提供了位操作。例如，如果目的是红色而来源为蓝色，SRCPAINT位映像操作把目的变成洋红色。注意，操作实际是按显示卡内储存的位执行的。这些位所对应的颜色取决于显示卡的调色盘的设定。Windows完成了此操作，以便位映像操作能达到您预计的结果。不过，如果您修改了调色盘（我将在第十六章讨论），位映像操作将产生无法预料的结果。

如要得到位映像操作较好的应用程序，请参见本章后面的「非矩形位图图像」一节。

图案Blt

除了BitBlt和StretchBlt以外，Windows还包括一个称为PatBlt （「pattern block transfer：图案块传输」）的函数。这是三个「blt」函数中最简单的。与BitBlt和StretchBlt不同，它只使用一个目的设备内容。PatBlt语法是：

PatBlt (hdc, x, y, cx, cy, dwROP) ;

x、y、cx和cy参数字于逻辑单位。逻辑点（x,y）指定了矩形的左上角。矩形宽为cx单位，高为cy单位。这是PatBlt修改的矩形区域。PatBlt在画刷与目的设备内容上执行的逻辑操作由dwROP参数决定，此参数是ROP代码的子集 – 也就是说，您可以只使用那些不包括来源目的设备内容的ROP代码。下表列出了PatBlt支持的16个位映像操作：

表14-5

图案（P）：1100	布尔操作	ROP代码	名称

目的（D）：1 0 1 0				
结果：	0 0 0 0	0	0x000042	BLACKNESS
	0 0 0 1	~(P | D)	0x0500A9	
	0 0 1 0	~P & D	0x0A0329	
	0 0 1 1	~P	0x0F0001	
	0 1 0 0	P & ~D	0x500325	
	0 1 0 1	~D	0x550009	DSTINVERT
	0 1 1 0	P ^ D	0x5A0049	PATINVERT
	0 1 1 1	~(P & D)	0x5F00E9	
	1 0 0 0	P & D	0xA000C9	
	1 0 0 1	~(P ^ D)	0xA50065	
	1 0 1 0	D	0xAA0029	
	1 0 1 1	~P | D	0xAF0229	
	1 1 0 0	P	0xF00021	PATCOPY
	1 1 0 1	P | ~D	0xF50225	
	1 1 1 0	P | D	0xFA0089	
	1 1 1 1	1	0xFF0062	WHITENESS

下面列出了PatBlt一些更常见用途。如果想画一个黑色矩形，您可呼叫

PatBlt (hdc, x, y, cx, cy, BLACKNESS) ;

要画一个白色矩形，请用

PatBlt (hdc, x, y, cx, cy, WHITENESS) ;

函数

PatBlt (hdc, x, y, cx, cy, DSTINVERT) ;

用于改变矩形的颜色。如果目前设备内容中选择了WHITE_BRUSH，那么函数

PatBlt (hdc, x, y, cx, cy, PATINVERT) ;

也改变矩形。

您可以再次呼叫FillRect函数来用画笔充满一个矩形区域：

FillRect (hdc, &rect, hBrush) ;

FillRect函数相同于下列代码：

```
hBrush = SelectObject (hdc, hBrush) ;
PatBlt (hdc, rect.left, rect.top,
       rect.right - rect.left,
       rect.bottom - rect.top, PATCOPY) ;
SelectObject (hdc, hBrush) ;
```

实际上，此程序代码是Windows用于执行FillRect函数的动作。如果您呼叫

InvertRect (hdc, &rect) ;

Windows将其转换成函数：

```
PatBlt (hdc, rect.left, rect.top,
       rect.right - rect.left,
       rect.bottom - rect.top, DSTINVERT) ;
```

在介绍PatBlt函数的语法时，我说过点（x,y）指出了矩形的左上角，而且此矩形宽度为cx单位，高度为cy单位。此叙述并不完全正确。BitBlt、PatBlt和StretchBlt是最合适的GDI画图函数，它们根据从一个角测得的逻辑宽度和高度来指定逻辑直角坐标。矩形边框用到的其它所有GDI画图函数都要求根据左上角和右下角的坐标来指定坐标。对于MM_TEXT映像模式，上面讲述的PatBlt参数就是正确的。然而对于公制的映像模式来说，就不正确。如果您使用一的cx和cy值，那么点（x,y）将是矩形的左下角。如果希望点（x,y）是矩形的左上角，那么cy参数必须设为矩形的负高度。

如果想更精确，用PatBlt修改颜色的矩形将通过cx的绝对值获得逻辑宽度，通过cy的绝对值获得逻辑高度。这两个参数可以是负值。由逻辑点（x, y）和（x + cx, y + cy）给定的两个角定义了矩形。矩形的左上角通常属于PatBlt修改的区域。右上角则超出了矩形的范围。根据映像模式和cx、cy参数的符号，矩形左上角的点应为（x, y）、（x, y + cy）、（x + cx, y）或者（x + cx, y + cy）。

如果给MM_LOENGLISH设定了映像模式，并且您想在显示区域左上角的一小块正方形上使用PatBlt，您可以使用

PatBlt (hdc, 0, 0, 100, -100, dwROP) ;

或

PatBlt (hdc, 0, -100, 100, 100, dwROP) ;

或

PatBlt (hdc, 100, 0, -100, -100, dwROP) ;

或

PatBlt (hdc, 100, -100, -100, 100, dwROP) ;

给PatBlt设定正确参数最容易的方法是将x和y设为矩形左上角。如果映像模式定义y坐标随着向上卷动显示而增加，那么请使用负的cy参数。如果映像模式定义x坐标向左增加（很少有人用），则需要使用负的cx参数。

GDI 位图对象

我在本章前面已提到过Windows从1.0开始就支持GDI位图对象。因为在Windows 3.0发表了设备无关位图，GDI位图对象有时也称为设备相关位图，或者DDB。我尽量不全部引用device-dependent bitmap的全文，因为它看上去与device-independent bitmap类似。缩写DDB会好一些，因为我们很容易把它与DIB区别开来。

对程序写作者来说，现存的两种不同型态的位图从Windows 3.0开始就更为混乱。许多有经验的Windows程序写作者都不能准确地理解DIB和DDB之间的关系。（恐怕本书的Windows 3.0版本不能澄清这个问题）。诚然，DIB和DDB在许多方面是相关的：DIB与DDB能相互转换（尽管转换程序中会丢失一些信息）。然而DIB和DDB是不可以相互替换的，并且不能简单地选择一种方法来表示同一个可视数据。

如果我们能假设说DIB一定会替代DDB，那以后就会很方便了。但现实并不是如此，DDB还在Windows中扮演着很重要角色，尤其是您在乎程序执行表现好坏时。

建立DDB

DDB是Windows图形设备接口的图形对象之一（其中还包括绘图笔、画刷、字体、metafile和调色盘）。这些图形对象储存在GDI模块内部，由应用程序软件以句柄数字的方式引用。您可以将

DDB句柄储存在一个HBITMAP（「handle to a bitmap：位图句柄」）型态的变量中，例如：

HBITMAP hBitmap ;

然后通过呼叫DDB建立的一个函数来获得句柄，例如：CreateBitmap。这些函数配置并初始化GDI内存中的一些内存来储存关于位图的信息，以及实际位图位的信息。应用程序不能直接存取这段内存。位图与设备内容无关。当程序使用完位图以后，就要清除这段内存：

DeleteObject (hBitmap) ;

如果程序执行时您使用了DDB，那么程序终止时，您可以完成上面的操作。

CreateBitmap函数用法如下：

hBitmap = CreateBitmap (cx, cy, cPlanes, cBitsPixel, bits) ;

前两个参数是位图的宽度和高度（以像素为单位），第三个参数是颜色面的数目，第四个参数是每像素的位数，第五个参数是指向一个以特定颜色格式存放的位数组的指针，数组内存放有用来初始化该DDB的图像。如果您不想用一张现有的图像来初始化DDB，可以将最后一个参数设为NULL。以后您还是可以设定该DDB内像素的内容。

使用此函数时，Windows也允许建立您喜欢的特定型态GDI位图对象。例如，假设您希望位图宽7个像素、高9个像素、5个?色位面，并且每个像素占3位，您只需要执行下面的操作：

hBitmap = CreateBitmap (7, 9, 5, 3, NULL) ;

这时Windows会好好给您一个有效的位图句柄。

在此函数呼叫期间，Windows将储存您传递给函数的信息，并为像素位配置内存。粗略的计算是此位图需要7×9×5×3，即945位，这需要比118个字节还多几个位。

然而，Windows为位图配置好内存以后，每行像素都占用许多连贯的字节，这样

iWidthBytes = 2 * ((cx * cBitsPixel + 15) / 16) ;

或者C程序写作者更倾向于写成：

iWidthBytes = (cx * cBitsPixel + 15) & ~15) >> 3 ;

因此，为DDB配置的内存就是：

iBitmapBytes = cy * cPlanes * iWidthBytes ;

本例中，iWidthBytes占4字节，iBitmapBytes占180字节。

现在，知道一张位图有5个颜色位面，每像素占3个颜色位有什么意义吗？真是见鬼了，这甚至不能把它称作一个习题作业。虽然您让GDI内部配置了些内存，并且让这些内存有一定结构的内容，但是您这张位图完全作不出任何有用的事情来。

实际上，您将用两种型态的参数来呼叫CreateBitmap。

cPlanes和cBitsPixel都等于1（表示单色位图）；或者

cPlanes和cBitsPixel都等于某个特定设备内容的值，您可以使用PLANES和BITSPIXEL索引来从GetDeviceCaps函数获得。

更现实的情况下，您只会在第一种情况下呼叫CreateBitmap。对于第二种情况，您可以用CreateCompatibleBitmap来简化问题：

hBitmap = CreateCompatibleBitmap (hdc, cx, cy) ;

此函数建立了一个与设备兼容的位图，此设备的设备内容句柄由第一个参数给出。CreateCompatibleBitmap用设备内容句柄来获得GetDeviceCaps信息，然后将此信息传递给CreateBitmap。除了与实际的设备内容有相同的内存组织之外，DDB与设备内容没有其它联系。

CreateDiscardableBitmap函数与CreateCompatibleBitmap的参数相同，并且功能上相同。在早期的Windows版本中，CreateDiscardableBitmap建立的位图可以在内存减少时由Windows将其从内存中清除，然后程序再重建位图数据。

第三个位图建立函数是CreateBitmapIndirect：

hBitmap CreateBitmapIndirect (&bitmap) ;

其中bitmap是BITMAP型态的结构。BITMAP结构定义如下：

```
typedef struct _tagBITMAP
{
  LONG bmType ;  // set to 0
  LONG bmWidth ;  // width in pixels
  LONG bmHeight ;  // height in pixels
  LONG bmWidthBytes ; // width of row in bytes
  WORD bmPlanes ;  // number of color planes
  WORD bmBitsPixel ;  // number of bits per pixel
  LPVOIDbmBits ;  // pointer to pixel bits
}
BITMAP, * PBITMAP ;
```

在呼叫CreateBitmapIndirect函数时，您不需要设定bmWidthBytes字段。Windows将为您计算，您也可以将bmBits字段设定为NULL，或者设定为初始化位图时用的像素位地址。

GetObject函数内也使用BITMAP结构，首先定义一个BITMAP型态的结构。

BITMAP bitmap ;

并呼叫函数如下：

GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

Windows将用位图信息填充BITMAP结构的字段，不过，bmBits字段等于NULL。

您最后应呼叫DeleteObject来清除程序内建立的所有位图。

位图位

用CreateBitmap或CreateBitmapIndirect来建立设备相关GDI位图对象时，您可以给位图像素位指定一个指针。或者您也可以让位图维持未初始化的状态。在建立位图以后，Windows还提供两个函数来获得并设定像素位。

要设定像素位，请呼叫：

SetBitmapBits (hBitmap, cBytes, &bits) ;

GetBitmapBits函数有相同的语法：

GetBitmapBits (hBitmap, cBytes, &bits) ;

在这两个函数中，cBytes指明要复制的字节数，bits是最少cBytes大小的缓冲区。

DDB中的像素位从顶列开始排列。我在前面说过，每列的字节数都是偶数。除此之外，没什么好说明的了。如果位图是单色的，也就是说它有1个位面并且每个像素占1位，则每个像素不是1就是0。每列最左边的像素是本列第一个字节最高位的位。我们在本章的后面讲完如何显示单色DDB之后，将做一个单色的DDB。

对于非单色位图，应避免出现您需要知道像素位含义的状况。例如，假定在8位颜色的VGA上执行Windows，您可以呼叫CreateCompatibleBitmap。通过GetDeviceCaps，您能够确定您正处理一个有1个颜色位面和每像素8位的设备。一个字节储存一个像素。但是像素值0x37是什么意思呢？很明显是某种颜色，但到底是什么颜色呢？

像素实际上并不涉及任何固定的颜色，它只是一个值。DDB没有颜色表。问题的关键在于：当DDB显示在屏幕上时，像素的颜色是什么。它肯定是某种颜色，但具体是什么颜色呢？显示的像素将与在显示卡上的调色盘查看表里的0x37索引值代表的RGB颜色有关。这就是您现在碰到的设备依赖性。

不过，不要只因为我们不知道像素值的含义，就假定非单色DDB没用。我们将简要看一下它们的用途。下一章，我们将看到SetBitmapBits和GetBitmapBits函数是如何被更有用的SetDIBits和GetDIBits函数所取代的。

因此，基本的规则是这样的：不要用CreateBitmap、CreateBitmapIndirect或SetBitmapBits来设定彩色DDB的位，您只能安全地使用这些函数来设定单色DDB的位。（如果您在呼叫GetBitmapBits期间，从其它相同格式的DDB中获得位，那么这些规则例外。）

在继续之前，让我再讨论一下SetBitmapDimensionEx和GetBitmapDimensionEx函数。这些函数让您设定（和获得）位图的测量尺寸（以0.1毫米为单位）。这些信息与位图分辨率一起储存在GDI中，但不用于任何操作。它只是您与DDB联系的一个测量尺寸标识。

内存设备内容

我们必须解决的下一个概念是内存设备内容。您需要用内存设备内容来处理GDI位图对象。

通常，设备内容指的是特殊的图形输出设备（例如视讯显示器或者打印机）及其设备驱动程序。内存设备内容只位于内存中，它不是真正的图形输出设备，但可以说与指定的真正设备「兼容」。

要建立一个内存设备内容，您必须首先有实际设备的设备内容句柄。如果是hdc，那么您可以像下面那样建立内存设备内容：

```
hdcMem = CreateCompatibleDC (hdc) ;
```

通常，函数的呼叫比这更简单。如果您将参数设为NULL，那么Windows将建立一个与视讯显示器相兼容的内存设备内容。应用程序建立的任何内存设备内容最终都通过呼叫DeleteDC来清除。

内存设备内容有一个与实际位映像设备相同的显示平面。不过，最初此显示平面非常小 – 单色、1像素宽、1像素高。显示平面就是单独1位。

当然，用1位的显示平面，您不能做更多的工作，因此下一步就是扩大显示平面。您可以通过将一个GDI位图对象选进内存设备内容来完成这项工作，例如：

```
SelectObject (hdcMem, hBitmap) ;
```

此函数与您将画笔、画刷、字体、区域和调色盘选进设备内容的函数相同。然而，内存设备内容是您可以选进位图的唯一一种设备内容型态。（如果需要，您也可以将其它GDI对象选进内存设备内容。）

只有选进内存设备内容的位图是单色的，或者与内存设备内容兼容设备有相同的色彩组织时，SelectObject才会起作用。这也是建立特殊的DDB（例如有5个位面，且每像素3位）没有用的原因。

现在情况是这样：SelectObject呼叫以后，DDB就是内存设备内容的显示平面。处理实际设备内容的每项操作，您几乎都可以用于内存设备内容。例如，如果用GDI画图函数在内存设备内容

中画图，那么图像将画在位图上。这是非常有用的。还可以将内存设备内容作为来源，把视讯设备内容作为目的来呼叫BitBlt。这就是在显示器上绘制位图的方法。如果把视讯设备内容作为来源，把内存设备内容作为目的，那么呼叫BitBlt可将屏幕上的一些内容复制给位图。我们将看到这些都是可能的。

载入位图资源

除了各种各样的位图建立函数以外，获得GDI位图对象句柄的另一个方法就是呼叫LoadBitmap函数。使用此函数，您不必担心位图格式。在程序中，您只需简单地按资源来建立位图，这与建立图标或者鼠标光标的方法类似。LoadBitmap函数的语法与LoadIcon和LoadCursor相同：

hBitmap = LoadBitmap (hInstance, szBitmapName) ;

如果想加载系统位图，那么将第一个参数设为NULL。这些不同的位图是Windows视觉接口（例如关闭方块和勾选标记）的一小部分，它们的标识符以字母OBM开始。如果位图与整数标识符而不是与名称有联系，那么第二个参数就可以使用MAKEINTRESOURCE宏。由LoadBitmap加载的所有位图最终应用DeleteObject清除。

如果位图资源是单色的，那么从LoadBitmap传回的句柄将指向一个单色的位图对象。如果位图资源不是单色，那么从LoadBitmap传回的句柄将指向一个GDI位图对象，该对象与执行程序的视讯显示器有相同的色彩组织。因此，位图始终与视讯显示器兼容，并且总是选进与视讯显示器兼容的内存设备内容中。采用LoadBitmap呼叫后，就不用担心任何色彩转换的问题了。在下一章中，我们就知道LoadBitmap的具体运作方式了。

程序14-3所示的BRICKS1程序示范了加载一小张单色位图资源的方法。此位图本身不像砖块，但当它水平和垂直重复时，就与砖墙相似了。

程序14-3　BRICKS1

BRICKS1.C

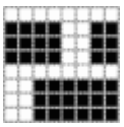
```
/*------------------------------------------------------------------------
BRICKS1.C -- LoadBitmap Demonstration
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("Bricks1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
```

```
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("LoadBitmap Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static int cxClient, cyClient, cxSource, cySource ;
  BITMAP bitmap ;
  HDC hdc, hdcMem ;
  HINSTANCE hInstance ;
  int x, y ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;

    hBitmap = LoadBitmap (hInstance, TEXT ("Bricks")) ;

    GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

    cxSource = bitmap.bmWidth ;
    cySource = bitmap.bmHeight ;

    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;

    for (y = 0 ; y < cyClient ; y += cySource)
      for (x = 0 ; x < cxClient ; x += cxSource)
      {
        BitBlt (hdc, x, y, cxSource, cySource, hdcMem, 0, 0, SRCCOPY) ;
      }

      DeleteDC (hdcMem) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;
  case WM_DESTROY:
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

BRICKS1.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Bitmap
BRICKS   BITMAP DISCARDABLE "Bricks.bmp"
```

BRICKS.BMP



在Visual C++ Developer Studio中建立位图时，应指明位图的高度和宽度都是8个像素，是单色，名称是「Bricks」。BRICKS1程序在WM_CREATE消息处理期间加载了位图并用GetObject来确定位图的像素尺寸（以便当位图不是8像素见方时程序仍能继续工作）。以后，BRICKS1将在WM_DESTROY消息中删除此位图。

在WM_PAINT消息处理期间，BRICKS1建立了一个与显示器兼容的内存设备内容，并且选进了位图。然后是从内存设备内容到显示区域设备内容一系列的BitBlt函数呼叫，再删除内存设备内容。图14-3显示了程序的执行结果。

顺便说一下，Developer Studio建立的BRICKS.BMP文件是一个设备无关位图。您可能想在Developer Studio内建立一个彩色的BRICKS.BMP文件（您可自己选定颜色），并且保证一切工作正常。

我们看到DIB能转换成与视讯显示器兼容的GDI位图对象。我们将在下一章看到这是如何操作的。

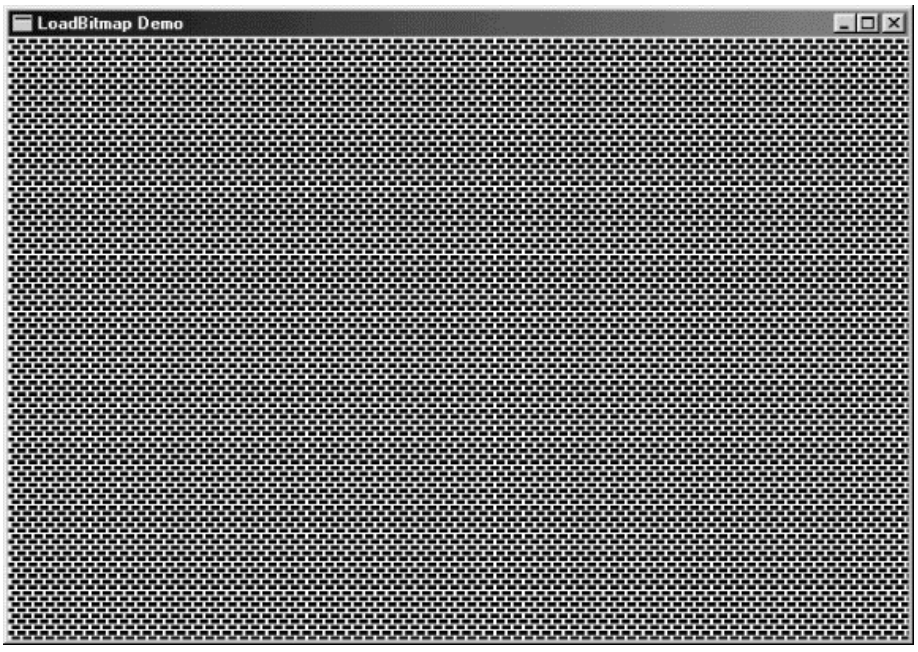
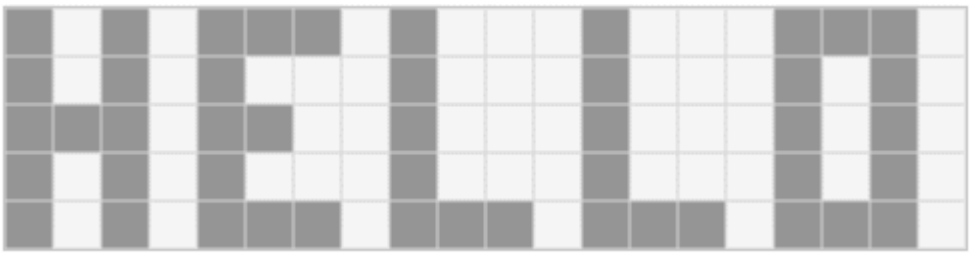


图14-3 BRICKS1的屏幕显示

单色位图格式

如果您在处理小块单色图像，那么您不必把它们当成资源来建立。与彩色位图对象不同，单色位的格式相对简单一些，而且几乎能全部从您要建立的图像中分离出来。例如，假定您要建立下图所示的位图：



您能写下一系列的位（0代表黑色，1代表白色），这些位直接对应于网格。从左到右读这些位，您能给每8字节配置一个十六进制元的字节值。如果位图的宽度不是16的倍数，在字节的右边用零填充，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

像素宽为20，扫描线高为5，字节宽为4。您可以用下面的叙述来设定此位图的BITMAP结构：

static BITMAP bitmap = { 0, 20, 5, 4, 1, 1 } ;

并且可以将位储存在BYTE数组中：

```
static BYTE bits [] = { 0x51, 0x77, 0x10, 0x00,
0x57, 0x77, 0x50, 0x00,
0x13, 0x77, 0x50, 0x00,
0x57, 0x77, 0x50, 0x00,
0x51, 0x11, 0x10, 0x00 } ;
```

用CreateBitmapIndirect来建立位图需要下面两条叙述：

bitmap.bmBits = (PSTR) bits ;

hBitmap = CreateBitmapIndirect (&bitmap) ;

另一种方法是：

hBitmap = CreateBitmapIndirect (&bitmap) ;

SetBitmapBits (hBitmap, sizeof bits, bits) ;

您也可以用一道叙述来建立位图：

hBitmap = CreateBitmap (20, 5, 1, 1, bits) ;

在程序14-4显示的BRICKS2程序利用此技术直接建立了砖块位图，而没有使用资源。

程序14-4 BRICKS2

BRICKS2.C

```
/*------------------------------------------------------------------------
BRICKS2.C -- CreateBitmap Demonstration
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
```

```
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("Bricks2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("CreateBitmap Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAP bitmap = { 0, 8, 8, 2, 1, 1 } ;
  static BYTE bits [8][2]={ 0xFF, 0, 0x0C, 0, 0x0C, 0, 0x0C, 0,
    0xFF, 0, 0xC0, 0, 0xC0, 0, 0xC0, 0 } ;
  static HBITMAP hBitmap ;
  static int cxClient, cyClient, cxSource, cySource ;
  HDC hdc, hdcMem ;
  int x, y ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    bitmap.bmBits = bits ;
    hBitmap = CreateBitmapIndirect (&bitmap) ;
    cxSource = bitmap.bmWidth ;
    cySource = bitmap.bmHeight ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
```

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

    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;

    for (y = 0 ; y < cyClient ; y += cySource)
      for (x = 0 ; x < cxClient ; x += cxSource)
      {
        BitBlt (hdc, x, y, cxSource, cySource, hdcMem, 0, 0, SRCCOPY) ;
      }

      DeleteDC (hdcMem) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;
  case WM_DESTROY:
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

您可以尝试一下与彩色位图相似的对象。例如，如果您的视讯显示器执行在256色模式下，那么您可以根据表14-2来定义彩色砖的每个像素。不过，当程序执行在其它显示模式下时，此程序代码不起作用。以设备无关方式处理彩色位图需要使用下章讨论的DIB。

位图中的画刷

BRICKS系列的最后一个项目是BRICKS3，如程序14-5所示。乍看此程序，您可能会有这种感觉：程序代码哪里去了呢?

程序14-5 BRICKS3

BRICKS3.C

```
/*------------------------------------------------------------------------
BRICKS3.C -- CreatePatternBrush Demonstration
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("Bricks3") ;
  HBITMAP hBitmap ;
  HBRUSH hBrush ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  hBitmap = LoadBitmap (hInstance, TEXT ("Bricks")) ;
  hBrush = CreatePatternBrush (hBitmap) ;
  DeleteObject (hBitmap) ;

  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 = hBrush ;
  wndclass.lpszMenuName = NULL ;
```

```
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("CreatePatternBrush Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }

  DeleteObject (hBrush) ;
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  switch (message)
  {
  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

BRICKS3.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Bitmap
BRICKS    BITMAP DISCARDABLE "Bricks.bmp"
```

此程序与BRICKS1使用同一个BRICKS.BMP文件，而且窗口看上去也相同。

正如您看到的一样，窗口消息处理程序没有更多的内容。BRICKS3实际上使用砖块图案作为窗口类别背景画刷，它在WNDCLASS结构的hbrBackground字段中定义。

您现在可能猜想GDI画刷是很小的位图，通常是8个像素见方。如果将LOGBRUSH结构的lbStyle字段设定为BS_PATTERN，然后呼叫CreatePatternBrush或CreateBrushIndirect，您就可以在位图外面来建立画刷了。此位图至少是宽高各8个像素。如果再大，Windows 98将只使用位图的左上角作为画刷。而Windows NT不受此限制，它会使用整个位图。

请记住，画刷和位图都是GDI对象，而且您应该在程序终止前删除您在程序中建立画刷和位图。如果您依据位图建立画刷，那么在用画刷画图时，Windows将复制位图位到画刷所绘制的区域内。呼叫CreatePatternBrush（或者CreateBrushIndirect）之后，您可以立即删除位图而不会影响到画笔。类似地，您也可以删除画刷而不会影响到您选进的原始位图。注意，BRICKS3在建立画刷后删除了位图，并在程序终止前删除了画刷。

绘制位图

在窗口中绘图时，我们已经将位图当成绘图来源使用过了。这要求先将位图选进内存设备内容，并呼叫BitBlt或者StretchBlt。您也可以用内存设备内容句柄作为所有实际呼叫的GDI函数中的第一参数。内存设备内容的动作与实际的设备内容相同，除非显示平面是位图。

程序14-6所示的HELLOBIT程序展示了此项技术。程序在一个小位图上显示了字符串「Hello, world!」，然后从位图到程序显示区域执行BitBlt或StretchBlt（依照选择的菜单选项而定）。

程序14-6 HELLOBIT

HELLOBIT.C

```
/*------------------------------------------------------------------------
HELLOBIT.C -- Bitmap Demonstration
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("HelloBit") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("HelloBit"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HBITMAP hBitmap ;
```

```
static HDC hdcMem ;
static int cxBitmap, cyBitmap, cxClient, cyClient, iSize = IDM_BIG ;
static TCHAR * szText = TEXT (" Hello, world! ") ;
HDC hdc ;
HMENU hMenu ;
int x, y ;
PAINTSTRUCT ps ;
SIZE size ;

switch (message)
{
case WM_CREATE:
  hdc = GetDC (hwnd) ;
  hdcMem = CreateCompatibleDC (hdc) ;

  GetTextExtentPoint32 (hdc, szText, lstrlen (szText), &size) ;
  cxBitmap = size.cx ;
  cyBitmap = size.cy ;
  hBitmap = CreateCompatibleBitmap (hdc, cxBitmap, cyBitmap) ;

  ReleaseDC (hwnd, hdc) ;

  SelectObject (hdcMem, hBitmap) ;
  TextOut (hdcMem, 0, 0, szText, lstrlen (szText)) ;
  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;
case WM_COMMAND:
  hMenu = GetMenu (hwnd) ;

  switch (LOWORD (wParam))
  {
  case IDM_BIG:
  case IDM_SMALL:
    CheckMenuItem (hMenu, iSize, MF_UNCHECKED) ;
    iSize = LOWORD (wParam) ;
    CheckMenuItem (hMenu, iSize, MF_CHECKED) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    break ;
  }
  return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    switch (iSize)
    {
    case IDM_BIG:
      StretchBlt (hdc, 0, 0, cxClient, cyClient,
        hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY) ;
      break ;
    case IDM_SMALL:
      for (y = 0 ; y < cyClient ; y += cyBitmap)
        for (x = 0 ; x < cxClient ; x += cxBitmap)
        {
          BitBlt (hdc, x, y, cxBitmap, cyBitmap,
            hdcMem, 0, 0, SRCCOPY) ;
        }
        break ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;
    case WM_DESTROY:
      DeleteDC (hdcMem) ;
      DeleteObject (hBitmap) ;
      PostQuitMessage (0) ;
      return 0 ;
```

```
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

HELLOBIT.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
HELLOBIT MENU DISCARDABLE
BEGIN
POPUP "&Size"
BEGIN
MENUITEM "&Big",    IDM_BIG, CHECKED
MENUITEM "&Small",   IDM_SMALL
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by HelloBit.rc
#define IDM_BIG   40001
#define IDM_SMALL   40002
```

程序从呼叫GetTextExtentPoint32确定字符串的像素尺寸开始。这些尺寸将成为与视讯显示兼容的位图尺寸。当此位图被选进内存设备内容（也与视讯显示兼容）后，再呼叫TextOut将文字显示在位图上。内存设备内容在程序执行期间保留。在处理WM_DESTROY信息期间，HELLOBIT删除了位图和内存设备内容。

HELLOBIT中的一条菜单选项允许您显示位图尺寸，此尺寸或者是显示区域中水平和垂直方向平铺的实际尺寸，或者是缩放成显示区域大小的尺寸，如图14-4所示。正与您所见到的一样，这不是显示大尺寸字符的好方法！它只是小字体的放大版，并带有放大时产生的锯齿线。

图14-4 HELLOBIT的屏幕显示

您可能想知道一个程序，例如HELLOBIT，是否需要处理WM_DISPLAYCHANGE消息。只要使用者（或者其它应用程序）修改了视讯显示大小或者颜色深度，应用程序就接收到此讯息。其中颜色深度的改变会导致内存设备内容和视讯设备内容不兼容。但这并不会发生，因为当显示模式修改后，Windows自动修改了内存设备内容的颜色分辨率。选进内存设备内容的位图仍然保持原样，但不会造成任何问题。

阴影位图

在内存设备内容绘图（也就是位图）的技术是执行「阴影位图（shadow bitmap）」的关键。此位图包含窗口显示区域中显示的所有内容。这样，对WM_PAINT消息的处理就简化到简单的BitBlt。

阴影位图在绘画程序中最有用。程序14-7所示的SKETCH程序并不是一个最完美的绘画程序，但它是一个开始。

程序14-7 SKETCH

SKETCH.C

```
/*---------------------------------------------------------------------------
SKETCH.C -- Shadow Bitmap Demonstration
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("Sketch") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Sketch"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  if (hwnd == NULL)
  {
    MessageBox ( NULL, TEXT ("Not enough memory to create bitmap!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  ShowWindow (hwnd, iCmdShow) ;
```

```
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void GetLargestDisplayMode (int * pcxBitmap, int * pcyBitmap)
{
  DEVMODE devmode ;
  int iModeNum = 0 ;

  * pcxBitmap = * pcyBitmap = 0 ;

  ZeroMemory (&devmode, sizeof (DEVMODE)) ;
  devmode.dmSize = sizeof (DEVMODE) ;

  while (EnumDisplaySettings (NULL, iModeNum++, &devmode))
  {
    * pcxBitmap = max (* pcxBitmap, (int) devmode.dmPelsWidth) ;
    * pcyBitmap = max (* pcyBitmap, (int) devmode.dmPelsHeight) ;
  }
}
LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL fLeftButtonDown, fRightButtonDown ;
  static HBITMAP hBitmap ;
  static HDC hdcMem ;
  static int cxBitmap, cyBitmap, cxClient, cyClient, xMouse, yMouse ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    GetLargestDisplayMode (&cxBitmap, &cyBitmap) ;

    hdc = GetDC (hwnd) ;
    hBitmap = CreateCompatibleBitmap (hdc, cxBitmap, cyBitmap) ;
    hdcMem = CreateCompatibleDC (hdc) ;
    ReleaseDC (hwnd, hdc) ;

    if (!hBitmap) // no memory for bitmap
    {
      DeleteDC (hdcMem) ;
      return -1 ;
    }

    SelectObject (hdcMem, hBitmap) ;
    PatBlt (hdcMem, 0, 0, cxBitmap, cyBitmap, WHITENESS) ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_LBUTTONDOWN:
    if (!fRightButtonDown)
      SetCapture (hwnd) ;

    xMouse = LOWORD (lParam) ;
    yMouse = HIWORD (lParam) ;
    fLeftButtonDown = TRUE ;
    return 0 ;
  case WM_LBUTTONUP:
    if (fLeftButtonDown)
      SetCapture (NULL) ;
```

```
    fLeftButtonDown = FALSE ;
    return 0 ;
  case WM_RBUTTONDOWN:
    if (!fLeftButtonDown)
      SetCapture (hwnd) ;

    xMouse = LOWORD (lParam) ;
    yMouse = HIWORD (lParam) ;
    fRightButtonDown = TRUE ;
    return 0 ;
  case WM_RBUTTONUP:
    if (fRightButtonDown)
      SetCapture (NULL) ;

    fRightButtonDown = FALSE ;
    return 0 ;
  case WM_MOUSEMOVE:
    if (!fLeftButtonDown && !fRightButtonDown)
      return 0 ;
    hdc = GetDC (hwnd) ;

    SelectObject (hdc,
      GetStockObject (fLeftButtonDown ? BLACK_PEN : WHITE_PEN)) ;

    SelectObject (hdcMem,
      GetStockObject (fLeftButtonDown ? BLACK_PEN : WHITE_PEN)) ;

    MoveToEx (hdc, xMouse, yMouse, NULL) ;
    MoveToEx (hdcMem, xMouse, yMouse, NULL) ;

    xMouse = (short) LOWORD (lParam) ;
    yMouse = (short) HIWORD (lParam) ;

    LineTo (hdc, xMouse, yMouse) ;
    LineTo (hdcMem, xMouse, yMouse) ;

    ReleaseDC (hwnd, hdc) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    BitBlt (hdc, 0, 0, cxClient, cyClient, hdcMem, 0, 0, SRCCOPY) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    DeleteDC (hdcMem) ;
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

要想在SKETCH中画线，请按下鼠标左键并拖动鼠标。要擦掉画过的东西（更确切地说，是画白线），请按下鼠标右键并拖动鼠标。要清空整个窗口，请…结束程序，然后重新加载，一切从头再来。图14-5中显示的SKETCH程序图样表达了对频果公司的麦金塔计算机早期广告的敬意。

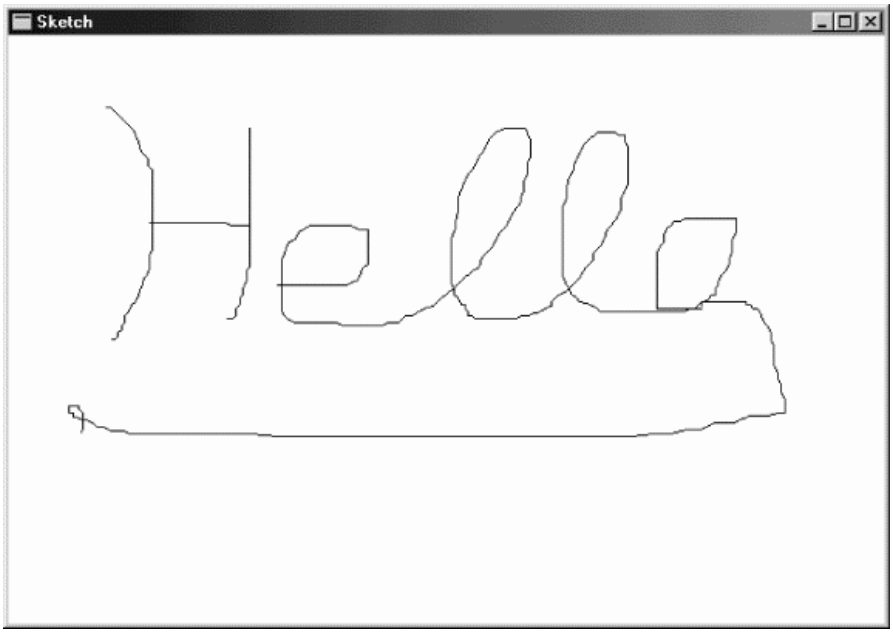


图14-5 SKETCH的屏幕显示

此阴影位图应多大？在本程序中，它应该大到能包含最大化窗口的整个显示区域。这一问题很容易根据GetSystemMetrics信息计算得出，但如果使用者修改了显示设定后再显示，进而扩大了最大化时窗口的尺寸，这时将发生什么呢？SKETCH程序在EnumDisplaySettings函数的帮助下解决了此问题。此函数使用DEVMODE结构来传回全部有效视讯显示模式的信息。第一次呼叫此函数时，应将EnumDisplaySettings的第二参数设为0，以后每次呼叫此值都增加。EnumDisplaySettings传回FALSE时完成。

与此同时，SKETCH将建立一个阴影位图，它比目前视讯显示模式的表面还多四倍，而且需要几兆字节的内存。由于如此，SKETCH将检查位图是否建立成功了，如果没有建立，就从WM_CREATE传回-1，以表示错误。

在WM_MOUSEMOVE消息处理期间，按下鼠标左键或者右键，并在内存设备内容和显示区域设备内容中画线时，SKETCH拦截鼠标。如果画线方式更复杂的话，您可能想在一个函数中实作，程序将呼叫此函数两次－一次画在视讯设备内容上，一次画在内存设备内容上。

下面是一个有趣的实验：使SKETCH窗口小于全画面尺寸。随着鼠标左键的按下，将鼠标拖出窗口的右下角。因为SKETCH拦截鼠标，所以它继续接收并处理WM_MOUSEMOVE消息。现在扩大窗口，您将看到阴影位图包含您在SKETCH窗口外所画的内容。

在菜单中使用位图

您也可以用位图在菜单上显示选项。如果您联想起菜单中文件夹、剪贴簿和资源回收筒的图片，那么不要再想那些图片了。您应该考虑一下，菜单上显示位图对画图程序用途有多大，想象一下在菜单中使用不同字体和字体大小、线宽、阴影图案以及颜色。

GRAFMENU是展示图形菜单选项的范例程序。此程序顶层菜单如图14-6所示。放大的字母来自于40×16像素的单色位图文件，该文件在Visual C++ Developer Studio建立。从菜单上选择「FONT」将弹出三个选择项－「Courier New」、「 Arial」和「Times New Roman」。它们是标准的Windows TrueType字体，并且每一个都按其相关的字体显示，如图14-7所示。这些位图在程序中用内存设备内容建立。

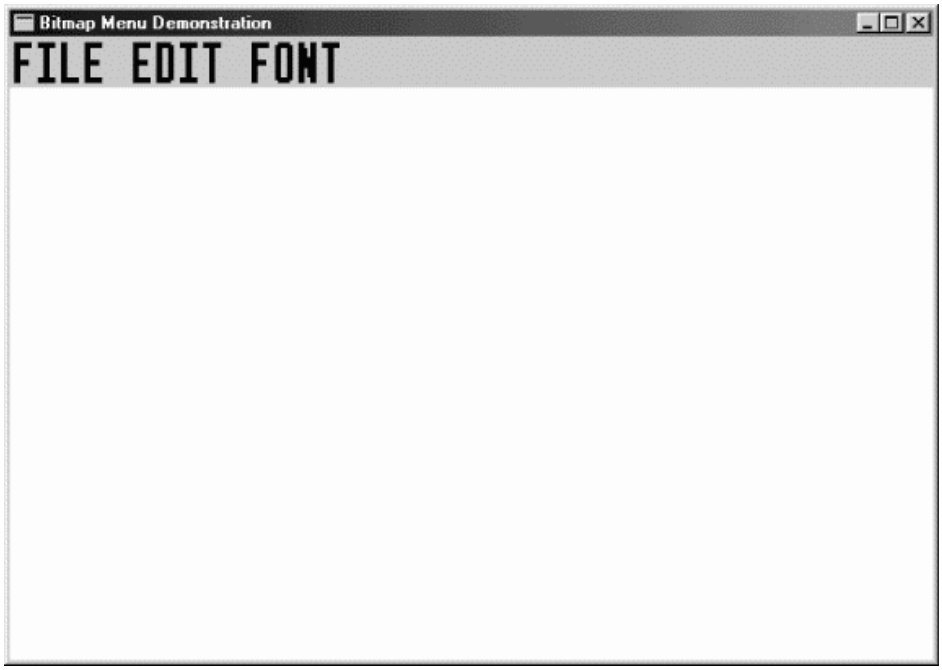


图14-6 GRAFMENU程序的顶层菜单

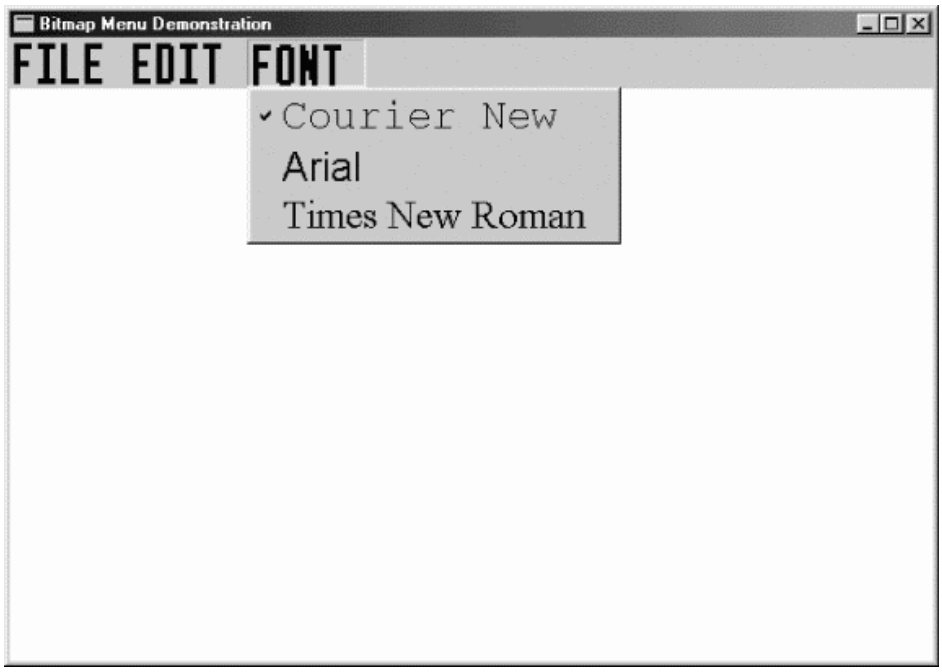


图14-7 GRAFMENU程序弹出的「FONT」菜单

最后，在拉下系统菜单时，您将获得一些「辅助」信息，用「HELP」表示了新使用者的在线求助项目（参见图14-8）。此64×64像素的单色位图是在Developer Studio中建立的。

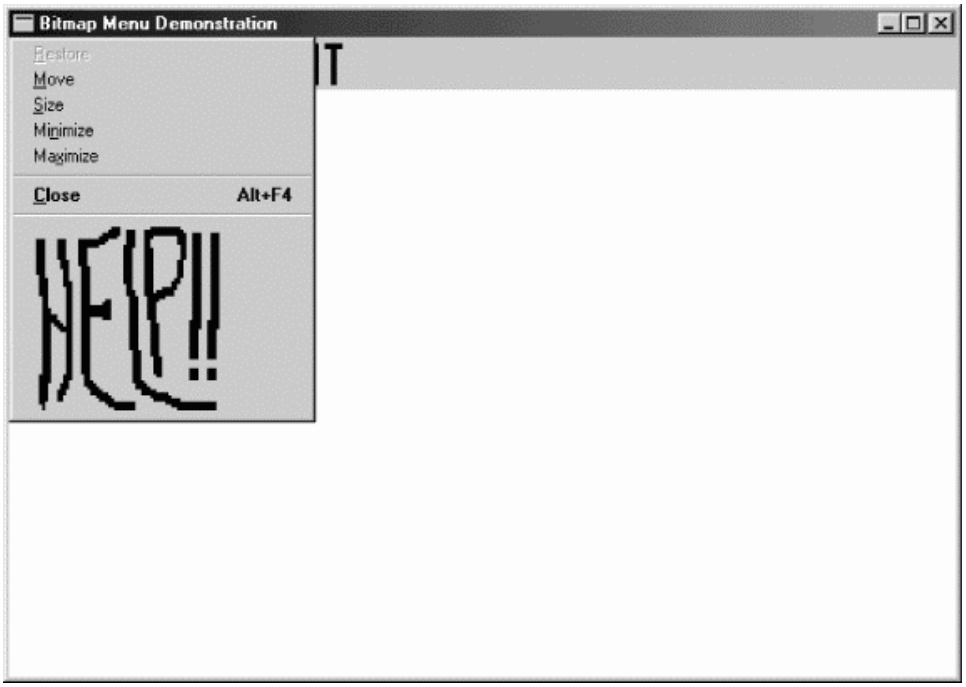


图14-8 GRAFMENU程序系统菜单

GRAFMENU程序，包括四个Developer Studio中建立的位图，如程序14-8所示。

程序14-8 GRAFMENU

GRAFMENU.C

```
/*-----------------------------------------------------------------------------
GRAFMENU.C -- Demonstrates Bitmap Menu Items
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
void AddHelpToSys (HINSTANCE, HWND) ;
HMENU CreateMyMenu (HINSTANCE) ;
HBITMAP StretchBitmap (HBITMAP) ;
HBITMAP GetBitmapFont (int) ;
void DeleteAllBitmaps (HWND) ;
TCHAR szAppName[] = TEXT ("GrafMenu") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
```

```
     szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName,TEXT ("Bitmap Menu Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,LPARAM lParam)
{
  HMENU hMenu ;
  static int iCurrentFont = IDM_FONT_COUR ;

  switch (iMsg)
  {
  case WM_CREATE:
    AddHelpToSys (((LPCREATESTRUCT) lParam)->hInstance, hwnd) ;
    hMenu = CreateMyMenu (((LPCREATESTRUCT) lParam)->hInstance) ;
    SetMenu (hwnd, hMenu) ;
    CheckMenuItem (hMenu, iCurrentFont, MF_CHECKED) ;
    return 0 ;

  case WM_SYSCOMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_HELP:
      MessageBox (hwnd, TEXT ("Help not yet implemented!"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
      return 0 ;
    }
    break ;

    case WM_COMMAND:
      switch (LOWORD (wParam))
      {
      case IDM_FILE_NEW:
      case IDM_FILE_OPEN:
      case IDM_FILE_SAVE:
      case IDM_FILE_SAVE_AS:
      case IDM_EDIT_UNDO:
      case IDM_EDIT_CUT:
      case IDM_EDIT_COPY:
      case IDM_EDIT_PASTE:
      case IDM_EDIT_CLEAR:
        MessageBeep (0) ;
        return 0 ;

      case IDM_FONT_COUR:
      case IDM_FONT_ARIAL:
      case IDM_FONT_TIMES:
        hMenu = GetMenu (hwnd) ;
        CheckMenuItem (hMenu, iCurrentFont, MF_UNCHECKED) ;
        iCurrentFont = LOWORD (wParam) ;
        CheckMenuItem (hMenu, iCurrentFont, MF_CHECKED) ;
        return 0 ;
      }
```

```
     break ;

     case WM_DESTROY:
       DeleteAllBitmaps (hwnd) ;
       PostQuitMessage (0) ;
       return 0 ;
  }
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

/*--------------------------------------------------------------------------
AddHelpToSys: Adds bitmap Help item to system menu
--------------------------------------------------------------------------*/

void AddHelpToSys (HINSTANCE hInstance, HWND hwnd)
{
  HBITMAP hBitmap ;
  HMENU hMenu ;

  hMenu = GetSystemMenu (hwnd, FALSE);
  hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapHelp"))) ;
  AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
  AppendMenu (hMenu, MF_BITMAP, IDM_HELP, (PTSTR) (LONG) hBitmap) ;
}

/*--------------------------------------------------------------------------
CreateMyMenu: Assembles menu from components
--------------------------------------------------------------------------*/

HMENU CreateMyMenu (HINSTANCE hInstance)
{
  HBITMAP hBitmap ;
  HMENU hMenu, hMenuPopup ;
  int i ;

  hMenu = CreateMenu () ;
  hMenuPopup = LoadMenu (hInstance, TEXT ("MenuFile")) ;
  hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapFile"))) ;
  AppendMenu (hMenu, MF_BITMAP | MF_POPUP, (int) hMenuPopup,
    (PTSTR) (LONG) hBitmap) ;
  hMenuPopup = LoadMenu (hInstance, TEXT ("MenuEdit")) ;
  hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapEdit"))) ;
  AppendMenu (hMenu, MF_BITMAP | MF_POPUP, (int) hMenuPopup,
    (PTSTR) (LONG) hBitmap) ;
  hMenuPopup = CreateMenu () ;
  for (i = 0 ; i < 3 ; i++)
  {
    hBitmap = GetBitmapFont (i) ;
    AppendMenu (hMenuPopup, MF_BITMAP, IDM_FONT_COUR + i,
      (PTSTR) (LONG) hBitmap) ;
  }

  hBitmap = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapFont"))) ;
  AppendMenu (hMenu, MF_BITMAP | MF_POPUP, (int) hMenuPopup,
    (PTSTR) (LONG) hBitmap) ;
  return hMenu ;
}

/*--------------------------------------------------------------------------
StretchBitmap: Scales bitmap to display resolution
--------------------------------------------------------------------------*/

HBITMAP StretchBitmap (HBITMAP hBitmap1)
{
  BITMAP bm1, bm2 ;
  HBITMAP hBitmap2 ;
  HDC hdc, hdcMem1, hdcMem2 ;
  int cxChar, cyChar ;
```

```
  // Get the width and height of a system font character

  cxChar = LOWORD (GetDialogBaseUnits ()) ;
  cyChar = HIWORD (GetDialogBaseUnits ()) ;

  // Create 2 memory DCs compatible with the display
  hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
  hdcMem1 = CreateCompatibleDC (hdc) ;
  hdcMem2 = CreateCompatibleDC (hdc) ;
  DeleteDC (hdc) ;

  // Get the dimensions of the bitmap to be stretched
  GetObject (hBitmap1, sizeof (BITMAP), (PTSTR) &bm1) ;
  // Scale these dimensions based on the system font size
  bm2 = bm1 ;
  bm2.bmWidth = (cxChar * bm2.bmWidth) / 4 ;
  bm2.bmHeight = (cyChar * bm2.bmHeight) / 8 ;
  bm2.bmWidthBytes = ((bm2.bmWidth + 15) / 16) * 2 ;

  // Create a new bitmap of larger size

  hBitmap2 = CreateBitmapIndirect (&bm2) ;
  // Select the bitmaps in the memory DCs and do a StretchBlt
  SelectObject (hdcMem1, hBitmap1) ;
  SelectObject (hdcMem2, hBitmap2) ;
  StretchBlt (hdcMem2, 0, 0, bm2.bmWidth, bm2.bmHeight,
    hdcMem1, 0, 0, bm1.bmWidth, bm1.bmHeight, SRCCOPY) ;
  // Clean up
  DeleteDC (hdcMem1) ;
  DeleteDC (hdcMem2) ;
  DeleteObject (hBitmap1) ;

  return hBitmap2 ;
}

/*-------------------------------------------------------------------------
GetBitmapFont: Creates bitmaps with font names
-------------------------------------------------------------------------*/

HBITMAP GetBitmapFont (int i)
{
  static TCHAR * szFaceName[3]= { TEXT ("Courier New"), TEXT ("Arial"),
    TEXT ("Times New Roman") } ;
  HBITMAP hBitmap ;
  HDC hdc, hdcMem ;
  HFONT hFont ;
  SIZE size ;
  TEXTMETRIC tm ;

  hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
  GetTextMetrics (hdc, &tm) ;

  hdcMem = CreateCompatibleDC (hdc) ;
  hFont = CreateFont (2 * tm.tmHeight, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    szFaceName[i]) ;

  hFont = (HFONT) SelectObject (hdcMem, hFont) ;
  GetTextExtentPoint32 (hdcMem, szFaceName[i],
    lstrlen (szFaceName[i]), &size);
  hBitmap = CreateBitmap (size.cx, size.cy, 1, 1, NULL) ;
  SelectObject (hdcMem, hBitmap) ;

  TextOut (hdcMem, 0, 0, szFaceName[i], lstrlen (szFaceName[i])) ;
  DeleteObject (SelectObject (hdcMem, hFont)) ;
  DeleteDC (hdcMem) ;
  DeleteDC (hdc) ;

  return hBitmap ;
}
```

```
/*--------------------------------------------------------------------------
DeleteAllBitmaps: Deletes all the bitmaps in the menu
--------------------------------------------------------------------------*/

void DeleteAllBitmaps (HWND hwnd)
{
  HMENU hMenu ;
  int i ;
  MENUITEMINFO mii = { sizeof (MENUITEMINFO), MIIM_SUBMENU | MIIM_TYPE } ;
  // Delete Help bitmap on system menu
  hMenu = GetSystemMenu (hwnd, FALSE);
  GetMenuItemInfo (hMenu, IDM_HELP, FALSE, &mii) ;
  DeleteObject ((HBITMAP) mii.dwTypeData) ;

  // Delete top-level menu bitmaps
  hMenu = GetMenu (hwnd) ;
  for (i = 0 ; i < 3 ; i++)
  {
    GetMenuItemInfo (hMenu, i, TRUE, &mii) ;
    DeleteObject ((HBITMAP) mii.dwTypeData) ;
  }

  // Delete bitmap items on Font menu
  hMenu = mii.hSubMenu ;;
  for (i = 0 ; i < 3 ; i++)
  {
    GetMenuItemInfo (hMenu, i, TRUE, &mii) ;
    DeleteObject ((HBITMAP) mii.dwTypeData) ;
  }
}
```

GRAFMENU.RC （摘录）

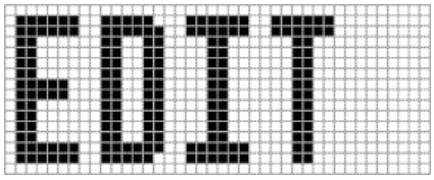
```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
MENUFILE MENU DISCARDABLE
BEGIN
MENUITEM "&New",      IDM_FILE_NEW
MENUITEM "&Open...",     IDM_FILE_OPEN
MENUITEM "&Save",     IDM_FILE_SAVE
MENUITEM "Save &As...",    IDM_FILE_SAVE_AS
END
MENUEDIT MENU DISCARDABLE
BEGIN
MENUITEM "&Undo",     IDM_EDIT_UNDO
MENUITEM SEPARATOR
MENUITEM "Cu&t",      IDM_EDIT_CUT
MENUITEM "&Copy",     IDM_EDIT_COPY
MENUITEM "&Paste",     IDM_EDIT_PASTE
MENUITEM "De&lete",   IDM_EDIT_CLEAR
END

//////////////////////////////////////////////////////////////////////////////
// Bitmap
BITMAPFONT  BITMAP  DISCARDABLE  "Fontlabl.bmp"
BITMAPHELP  BITMAP  DISCARDABLE  "Bighelp.bmp"
BITMAPEDIT  BITMAP  DISCARDABLE  "Editlabl.bmp"
BITMAPFILE  BITMAP  DISCARDABLE  "Filelabl.bmp"
```

RESOURCE.H （摘录）

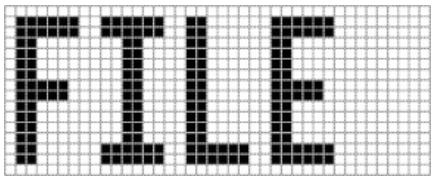
```
// Microsoft Developer Studio generated include file.
```

```
// Used by GrafMenu.rc
#define IDM_FONT_COUR    101
#define IDM_FONT_ARIAL   102
#define IDM_FONT_TIMES   103
#define IDM_HELP         104
#define IDM_EDIT_UNDO    40005
#define IDM_EDIT_CUT     40006
#define IDM_EDIT_COPY    40007
#define IDM_EDIT_PASTE   40008
#define IDM_EDIT_CLEAR   40009
#define IDM_FILE_NEW     40010
#define IDM_FILE_OPEN    40011
#define IDM_FILE_SAVE    40012
#define IDM_FILE_SAVE_AS 40013
```

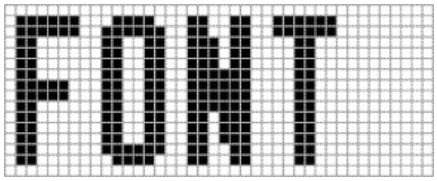
EDITLABL.BMP



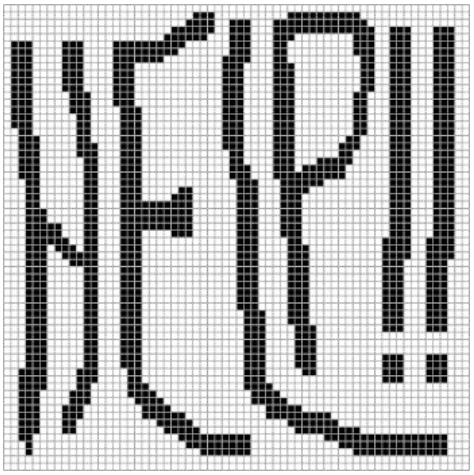
FILELABL.BMP



FONTLABL.BMP



BIGHELP.BMP



要将位图插入菜单，可以利用AppendMenu或InsertMenu。位图有两个来源：可以在Visual C++ Developer Studio建立位图，包括资源脚本中的位图文件，并在程序使用LoadBitmap时将位图资源加载到内存，然后呼叫AppendMenu或InsertMenu将位图附加到菜单上。但是用这种方法会有一些问题：位图不适于所有显示模式的分辨率和纵横比；有时您需要缩放加载的位图以解决此问题。另一种方法是：在程序内部建立位图，并将它选进内存设备内容，画出来，然后再附加到菜单中。

GRAFMENU中的GetBitmapFont函数的参数为0、1或2，传回一个位图句柄。此位图包含字符串「Courier New」、「Arial」或「Times New Roman」，而且字体是各自对应的字体，大小是正常系统字体的两倍。让我们看看GetBitmapFont是怎么做的。(下面的程序代码与GRAFMENU.C文件中的有些不同。为了清楚起见，我用「Arial」字体相应的值代替了引用szFaceName数组。)

第一步是用TEXTMETRIC结构来确定目前系统字体的大小，并建立一个与目前屏幕兼容的内存设备内容：

```
hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
GetTextMetrics (hdc, &tm) ;
hdcMem = CreateCompatibleDC (hdc) ;
```

CreateFont函数建立了一种逻辑字体，该字体高是系统字体的两倍，而且逻辑名称为「Arial」：

```
hFont = CreateFont (2 * tm.tmHeight, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                    TEXT ("Arial")) ;
```

从内存设备内容中选择该字体，然后储存内定字体句柄：

hFont = (HFONT) SelectObject (hdcMem, hFont) ;

现在，当我们向内存设备内容写一些文字时，Windows就会使用选进设备内容的TrueType Arial字体了。

但这个内存设备内容最初只有一个单像素单色设备平面。我们必须建立一个足够大的位图以容纳我们所要显示的文字。通过GetTextExtentPoint32函数，可以取得文字的大小，而用CreateBitmap可以根据这些尺寸来建立位图：

```
GetTextExtentPoint32 (hdcMem, TEXT ("Arial"), 5, &size) ;
hBitmap = CreateBitmap (size.cx, size.cy, 1, 1, NULL) ;
SelectObject (hdcMem, hBitmap) ;
```

现在这个设备内容是一个单色的显示平面，大小也是严格的文字尺寸。我们现在要做的就是书写文字：

TextOut (hdcMem, 0, 0, TEXT ("Arial"), 5) ;

除了清除，所有的工作都完成了。要清除，我们可以用SelectObject将系统字体（带有句柄hFont）重新选进设备内容，然后删除SelectObject传回的前一个字体句柄，也就是Arial字体句柄：

DeleteObject (SelectObject (hdcMem, hFont)) ;

现在可以删除两个设备内容：

DeleteDC (hdcMem) ;

DeleteDC (hdc) ;

这样，我们就获得了一个位图，该位图上有Arial字体的字符串「Arial」。

当我们需要缩放字体以适应不同显示分辨率或纵横比时，内存设备内容也能解决问题。在GRAFMENU程序中，我建立了四个位图，这些位图只适用于系统字体高8像素、宽4像素的显示。对于其它尺寸的系统字体，只能缩放位图。GRAFMENU中的StretchBitmap函数完成此功能。

第一步是获得显示的设备内容，然后取得系统字体的文字规格，接下来建立两个内存设备内容：

```
hdc = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
GetTextMetrics (hdc, &tm) ;
hdcMem1 = CreateCompatibleDC (hdc) ;
hdcMem2 = CreateCompatibleDC (hdc) ;
DeleteDC (hdc) ;
```

传递给函数的位图句柄是hBitmap1。程序能用GetObject获得位图的大小：

GetObject (hBitmap1, sizeof (BITMAP), (PSTR) &bm1) ;

此操作将尺寸复制到BITMAP型态的结构bm1中。结构bm2等于结构bm1，然后根据系统字体大小来修改某些字段：

```
bm2 = bm1 ;
bm2.bmWidth = (tm.tmAveCharWidth * bm2.bmWidth) / 4 ;
bm2.bmHeight = (tm.tmHeight * bm2.bmHeight) / 8 ;
bm2.bmWidthBytes = ((bm2.bmWidth + 15) / 16) * 2 ;
```

下一个位图带有句柄hBitmap2，可以根据动态的尺寸建立：

hBitmap2 = CreateBitmapIndirect (&bm2) ;

然后将这两个位图选进两个内存设备内容中：

SelectObject (hdcMem1, hBitmap1) ;

SelectObject (hdcMem2, hBitmap2) ;

我们想把第一个位图复制给第二个位图，并在此程序中进行拉伸。这包括StretchBlt呼叫：

```
StretchBlt (hdcMem2, 0, 0, bm2.bmWidth, bm2.bmHeight,
          hdcMem1, 0, 0, bm1.bmWidth, bm1.bmHeight, SRCCOPY) ;
```

现在第二幅图适当地缩放了，我们可将其用到菜单中。剩下的清除工作很简单：

DeleteDC (hdcMem1) ;

DeleteDC (hdcMem2) ;

DeleteObject (hBitmap1) ;

在建造菜单时，GRAFMENU中的CreateMyMenu函数呼叫了StretchBitmap和GetBitmapFont函数。GRAFMENU在资源文件中定义了两个菜单，在选择「File」和「Edit」选项时会弹出这两个菜单。函数开始先取得一个空菜单的句柄：

hMenu = CreateMenu () ;

从资源文件加载「File」的弹出式菜单（包括四个选项：「New」、「Open」、「Save」和「Save as」）：

hMenuPopup = LoadMenu (hInstance, TEXT ("MenuFile")) ;

从资源文件还加载了包含「FILE」的位图，并用StretchBitmap进行了拉伸：

hBitmapFile = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapFile"))) ;

位图句柄和弹出式菜单句柄都是AppendMenu呼叫的参数：

```
AppendMenu (hMenu, MF_BITMAP | MF_POPUP, hMenuPopup,
          (PTSTR) (LONG)hBitmapFile) ;
```

「Edit」菜单类似程序如下：

```
hMenuPopup = LoadMenu (hInstance, TEXT ("MenuEdit")) ;
hBitmapEdit = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapEdit"))) ;
AppendMenu (hMenu, MF_BITMAP | MF_POPUP, hMenuPopup,
```

```
          (PTSTR)(LONG) hBitmapEdit) ;
```

呼叫GetBitmapFont函数可以构造这三种不同字体的弹出式菜单：

```
hMenuPopup = CreateMenu () ;
for (i = 0 ; i < 3 ; i++)
{
  hBitmapPopFont [i] = GetBitmapFont (i) ;
  AppendMenu (hMenuPopup, MF_BITMAP, IDM_FONT_COUR + i,
    (PTSTR) (LONG) hMenuPopupFont [i]) ;
}
```

然后将弹出式菜单添加到菜单中：

```
hBitmapFont = StretchBitmap (LoadBitmap (hInstance, "BitmapFont")) ;
AppendMenu (hMenu, MF_BITMAP | MF_POPUP, hMenuPopup,
          (PTSTR) (LONG) hBitmapFont) ;
```

WndProc通过呼叫SetMenu，完成了窗口菜单的建立工作。

GRAFMENU还改变了AddHelpToSys函数中的系统菜单。此函数首先获得一个系统菜单句柄：

hMenu = GetSystemMenu (hwnd, FALSE) ;

这将载入「HELP」位图，并将其拉伸到适当尺寸：

hBitmapHelp = StretchBitmap (LoadBitmap (hInstance, TEXT ("BitmapHelp"))) ;

这将给系统菜单添加一条分隔线和拉伸的位图：

AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;

AppendMenu (hMenu, MF_BITMAP, IDM_HELP, (PTSTR)(LONG) hBitmapHelp) ;

GRAFMENU在退出之前呼叫一个函数来清除并删除所有位图。

下面是在菜单中使用位图的一些注意事项。

在顶层菜单中，Windows调整菜单列的高度以适应最高的位图。其它位图(或字符串)是根据菜单列的顶端对齐的。如果在顶层菜单中使用了位图，那么从使用常数SM_CYMENU的GetSystemMetrics得到的菜单列大小将不再有效。

执行GRAFMENU期间可以看到：在弹出式菜单中，您可使用带有位图菜单项的勾选标记，但勾选标记是正常尺寸。如果不满意，您可以建立一个自订的勾选标记，并使用SetMenuItemBitmaps。

在菜单中使用非文字（或者使用非系统字体的文字）的另一种方法是「拥有者绘制」菜单。

菜单的键盘接口是另一个问题。当菜单含有文字时，Windows会自动添加键盘接口。要选择一个菜单项，可以使用Alt与字符串中的一个字母的组合键。而一旦在菜单中放置了位图，就删除了键盘接口。即使位图表达了一定的含义，但Windows并不知道。

目前我们可以使用WM_MENUCHAR消息。当您按下Alt和与菜单项不相符的一个字符键的组合键时，Windows将向您的窗口消息处理程序发送一个WM_MENUCHAR消息。GRAFMENU需要截取WM_MENUCHAR消息并检查wParam的值（即按键的ASCII码）。如果这个值对应一个菜单项，那么向Windows传回双字组：其中高字组为2，低字组是与该键相关的菜单项索引值。然后由Windows处理余下的事。

非矩形位图图像

位图都是矩形，但不需要都显示成矩形。例如，假定您有一个矩形位图图像，但您却想将它显示成椭圆形。

首先，这听起来很简单。您只需将图像加载Visual C++ Developer Studio或者Windows的「画图」程序（或者更昂贵的应用程序），然后用白色的画笔将图像四周画上白色。这时将获得一幅椭圆形的图像，而椭圆的外面就成了白色。只有当背景色为白色时此位图才能正确显示，如果在其它背景色上显示，您就会发现椭圆形的图像和背景之间有一个白色的矩形。这种效果不好。

有一种非常通用的技术可解决此类问题。这种技术包括「屏蔽（mask）」位图和一些位映像操作。屏蔽是一种单色位图，它与您要显示的矩形位图图像尺寸相同。每个屏蔽的像素都对应位图图像的一个像素。屏蔽像素是1(白色)，对应着位图像素显示；是0(黑色)，则显示背景色。(或者屏蔽位图与此相反，这根据您使用的位映像操作而有一些相对应的变化。)

让我们看看BITMASK程序是如何实作这一技术的。如程序14-9所示。

程序14-9 BITMASK

BITMASK.C

```
/*-------------------------------------------------------------------------
BITMASK.C -- Bitmap Masking Demonstration
(c) Charles Petzold, 1998
-------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("BitMask") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (LTGRAY_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Bitmap Masking Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
```

```
{
  static HBITMAP hBitmapImag, hBitmapMask ;
  static HINSTANCE hInstance ;
  static int cxClient, cyClient, cxBitmap, cyBitmap ;
  BITMAP bitmap ;
  HDC hdc, hdcMemImag, hdcMemMask ;
  int x, y ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
    // Load the original image and get its size
    hBitmapImag = LoadBitmap (hInstance, TEXT ("Matthew")) ;
    GetObject (hBitmapImag, sizeof (BITMAP), &bitmap) ;
    cxBitmap = bitmap.bmWidth ;
    cyBitmap = bitmap.bmHeight ;

    // Select the original image into a memory DC
    hdcMemImag = CreateCompatibleDC (NULL) ;
    SelectObject (hdcMemImag, hBitmapImag) ;
    // Create the monochrome mask bitmap and memory DC
    hBitmapMask = CreateBitmap (cxBitmap, cyBitmap, 1, 1, NULL) ;
    hdcMemMask = CreateCompatibleDC (NULL) ;
    SelectObject (hdcMemMask, hBitmapMask) ;

    // Color the mask bitmap black with a white ellipse
    SelectObject (hdcMemMask, GetStockObject (BLACK_BRUSH)) ;
    Rectangle (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;
    SelectObject (hdcMemMask, GetStockObject (WHITE_BRUSH)) ;
    Ellipse (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;

    // Mask the original image
    BitBlt (hdcMemImag, 0, 0, cxBitmap, cyBitmap,
      hdcMemMask, 0, 0, SRCAND) ;
    DeleteDC (hdcMemImag) ;
    DeleteDC (hdcMemMask) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    // Select bitmaps into memory DCs

    hdcMemImag = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMemImag, hBitmapImag) ;

    hdcMemMask = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMemMask, hBitmapMask) ;

    // Center image

    x = (cxClient - cxBitmap) / 2 ;
    y = (cyClient - cyBitmap) / 2 ;

    // Do the bitblts

    BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemMask, 0, 0, 0x220326) ;
    BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemImag, 0, 0, SRCPAINT) ;

    DeleteDC (hdcMemImag) ;
    DeleteDC (hdcMemMask) ;
    EndPaint (hwnd, &ps) ;
```

```
    return 0 ;

  case WM_DESTROY:
    DeleteObject (hBitmapImag) ;
    DeleteObject (hBitmapMask) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

BITMASK.RC

```
// Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Bitmap
MATTHEW    BITMAP  DISCARDABLE  "matthew.bmp"
```

资源文件中的MATTHEW.BMP文件是我侄子的一幅黑白数字照片，宽200像素，高320像素，每像素8位。不过，另外制作个BITMASK只是因为此文件的内容是任何东西都可以。

注意，BITMASK将窗口背景设为亮灰色。这样就确保我们能正确地屏蔽位图，而不只是将其涂成白色。

下面让我们看一下WM_CREATE的处理程序：BITMASK用LoadBitmap函数获得hBitmapImag变量中原始图像的句柄。用GetObject函数可取得位图的宽度高度。然后将位图句柄选进句柄为hdcMemImag的内存设备内容中。

程序建立的下一个单色位图与原来的图大小相同，其句柄储存在hBitmapMask，并选进句柄为hdcMemMask的内存设备内容中。在内存设备内容中，使用GDI函数，屏蔽位图就涂成了黑色背景和一个白色的椭圆：

```
SelectObject (hdcMemMask, GetStockObject (BLACK_BRUSH)) ;
Rectangle (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;
SelectObject (hdcMemMask, GetStockObject (WHITE_BRUSH)) ;
Ellipse (hdcMemMask, 0, 0, cxBitmap, cyBitmap) ;
```

因为这是一个单色的位图，所以黑色区域的位是0，而白色区域的位是1。

然后BitBlt呼叫就按此屏蔽修改了原图像：

```
BitBlt (hdcMemImag, 0, 0, cxBitmap, cyBitmap,
       hdcMemMask, 0, 0, SRCAND) ;
```

SRCAND位映像操作在来源位（屏蔽位图）和目的位（原图像）之间执行了位AND操作。只要屏蔽位图是白色，就显示目的；只要屏蔽是黑色，则目的就也是黑色。现在原图像中就形成了一个黑色包围的椭圆区域。

现在让我们看一下WM_PAINT处理程序。此程序同时改变了选进内存设备内容中的图像位图和屏蔽位图。两次BitBlt呼叫完成了这个魔术，第一次在窗口上执行屏蔽位图的BitBlt：

BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemMask, 0, 0, 0x220326) ;

这里使用了一个没有名称的位映像操作。逻辑运算子是D & ~S。回忆来源 – 即屏蔽位图 – 是黑色（位值0）包围的一个白色（位值1）椭圆。位映像操作首先将来源反色，也就是改成白色包围的黑色椭圆。然后位操作在这个已转换的来源和目的（即窗口上）之间执行位AND操作。当目的和位值1「AND」时保持不变；与位值0「AND」时，目的将变黑。因此，BitBlt操作将在窗口上画一个黑色的椭圆。

第二次的BitBlt呼叫则在窗口中绘制图像位图：

BitBlt (hdc, x, y, cxBitmap, cyBitmap, hdcMemImag, 0, 0, SRCPAINT) ;

位映像操作在来源和目的之间执行位「OR」操作。由于来源位图的外面是黑色，因此保持目的不变；而在椭圆区域内，目的是黑色，因此图像就原封不动地复制了过来。执行结果如图14-9所示。

注意事项：

有时您需要一个很复杂的屏蔽 – 例如，抹去原始图像的整个背景。您将需要在画图程序中手工建立然后将其储存到成文件。

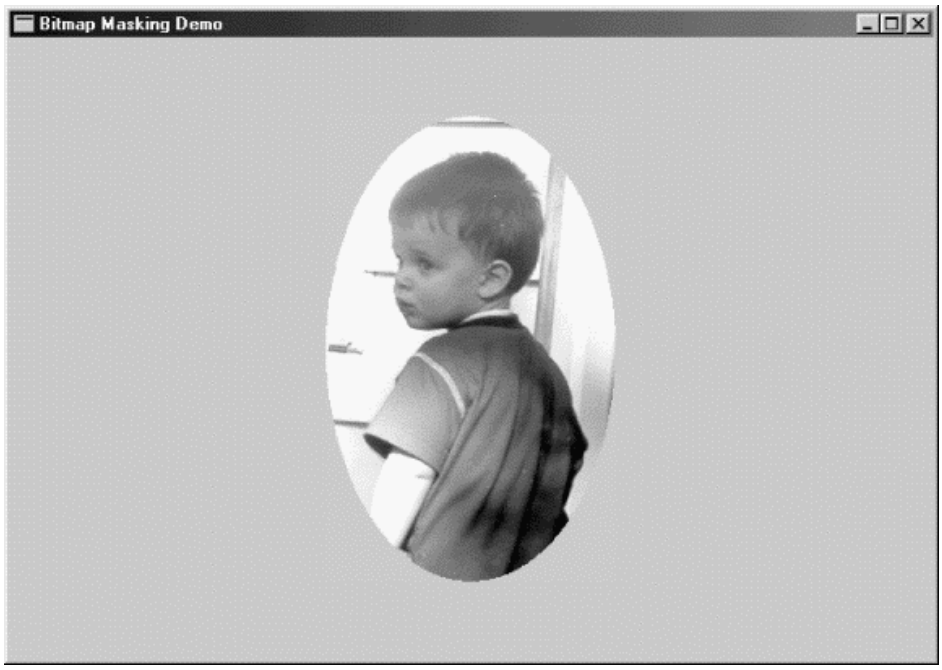


图14-9 BITMASK的屏幕显示

如果正在为Windows NT编写类似的应用程序，那么您可以使用与MASKBIT程序类似的MaskBlt函数，而只需要更少的函数呼叫。Windows NT还包括另一个类似BitBlt的函数，Windows 98不支持该函数。此函数是PlgBlt（「平行四边形位块移动：parallelogram blt」）。这个函数可以对图像进行旋转或者倾斜位图图像。

最后，如果在您的机器上执行BITMASK程序，您就只会看见黑色、白色和两个灰色的阴影，这是因为您执行的显示模式是16色或256色。对于16色模式，显示效果无法改进，但在256色模式下可以改变调色盘以显示灰阶。您将在第十六章学会如何设定调色盘。

简单的动画

小张的位图显示起来非常快，因此可以将位图和Windows定时器联合使用，来完成一些基本的动画。

现在开始这个弹球程序。

BOUNCE程序，如程序14-10所示，产生了一个在窗口显示区域弹来弹去的小球。该程序利用定时器来控制小球的行进速度。小球本身是一幅位图，程序首先通过建立位图来建立小球，将其选进内存设备内容，然后呼叫一些简单的GDI函数。程序用BitBlt从一个内存设备内容将这个位图小球画到显示器上。

程序14-10 BOUNCE

BOUNCE.C

```
/*------------------------------------------------------------------------
BOUNCE.C -- Bouncing Ball Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Bounce") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Bouncing Ball"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static int cxClient, cyClient, xCenter, yCenter, cxTotal, cyTotal,
    cxRadius, cyRadius, cxMove, cyMove, xPixel, yPixel ;
  HBRUSH hBrush ;
  HDC hdc, hdcMem ;
  int iScale ;

  switch (iMsg)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    xPixel = GetDeviceCaps (hdc, ASPECTX) ;
    yPixel = GetDeviceCaps (hdc, ASPECTY) ;
```

```
    ReleaseDC (hwnd, hdc) ;

    SetTimer (hwnd, ID_TIMER, 50, NULL) ;
    return 0 ;
  case WM_SIZE:
    xCenter = (cxClient = LOWORD (lParam)) / 2 ;
    yCenter = (cyClient = HIWORD (lParam)) / 2 ;

    iScale = min (cxClient * xPixel, cyClient * yPixel) / 16 ;

    cxRadius = iScale / xPixel ;
    cyRadius = iScale / yPixel ;

    cxMove = max (1, cxRadius / 2) ;
    cyMove = max (1, cyRadius / 2) ;

    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, ID_TIMER) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
```

```
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```

BOUNCE每次收到一个WM_SIZE消息时都重画小球。这就需要与视讯显示器兼容的内存设备内容：

hdcMem = CreateCompatibleDC (hdc) ;

小球的直径设为窗口显示区域高度或宽度中较短者的十六分之一。不过，程序构造的位图却比小球大：从位图中心到位图四个边的距离是小球半径的1.5倍：

hBitmap = CreateCompatibleBitmap (hdc, cxTotal, cyTotal) ;

将位图选进内存设备内容后，整个位图背景设成白色：

Rectangle (hdcMem, -1, -1, xTotal + 1, yTotal + 1) ;

那些不固定的坐标使矩形边框在位图之外着色。一个对角线开口的画刷选进内存设备内容，并将小球画在位图的中央：

Ellipse (hdcMem, xMove, yMove, xTotal - xMove, yTotal - yMove) ;

当小球移动时，小球边界的空白会有效地删除前一时刻的小球图像。在另一个位置重画小球只需在BitBlt呼叫中使用SRCCOPY的ROP代码：

```
BitBlt (hdc, xCenter - cxTotal / 2, yCenter - cyTotal / 2, cxTotal, cyTotal,
       hdcMem, 0, 0, SRCCOPY) ;
```

BOUNCE程序只是展示了在显示器上移动图像的最简单的方法。在一般情况下，这种方法并不能令人满意。如果您对动画感兴趣，那么除了在来源和目的之间执行或操作以外，您还应该研究其它的ROP代码（例如SRCINVERT）。其它动画技术包括Windows调色盘（以及AnimatePalette函数）和CreateDIBSection函数。对于更高级的动画您只好放弃GDI而使用DirectX接口了。

窗口外的位图

SCRAMBLE程序，如程序14-11所示，编写非常粗糙，我本来不应该展示这个程序，但它示范了一些有趣的技术，而且在交换两个显示矩形内容的BitBlt操作的程序中，用内存设备内容作为临时储存空间。

程序14-11　SCRAMBLE

SCRAMBLE.C

```
/*---------------------------------------------------------------------------
SCRAMBLE.C -- Scramble (and Unscramble) Screen
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>

#define NUM 300

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static int iKeep [NUM][4] ;
  HDC hdcScr, hdcMem ;
  int cx, cy ;
  HBITMAP hBitmap ;
  HWND hwnd ;
  int i, j, x1, y1, x2, y2 ;

  if (LockWindowUpdate (hwnd = GetDesktopWindow ()))
```

```
  {
    hdcScr = GetDCEx (hwnd, NULL, DCX_CACHE | DCX_LOCKWINDOWUPDATE) ;
    hdcMem = CreateCompatibleDC (hdcScr) ;
    cx = GetSystemMetrics (SM_CXSCREEN) / 10 ;
    cy = GetSystemMetrics (SM_CYSCREEN) / 10 ;
    hBitmap = CreateCompatibleBitmap (hdcScr, cx, cy) ;

    SelectObject (hdcMem, hBitmap) ;
    srand ((int) GetCurrentTime ()) ;

    for (i = 0 ; i < 2 ; i++)
      for (j = 0 ; j < NUM ; j++)
      {
        if (i == 0)
        {
          iKeep [j] [0] = x1 = cx * (rand () % 10) ;
          iKeep [j] [1] = y1 = cy * (rand () % 10) ;
          iKeep [j] [2] = x2 = cx * (rand () % 10) ;
          iKeep [j] [3] = y2 = cy * (rand () % 10) ;
        }
        else
        {
          x1 = iKeep [NUM - 1 - j] [0] ;
          y1 = iKeep [NUM - 1 - j] [1] ;
          x2 = iKeep [NUM - 1 - j] [2] ;
          y2 = iKeep [NUM - 1 - j] [3] ;
        }
        BitBlt (hdcMem, 0, 0, cx, cy, hdcScr, x1, y1, SRCCOPY) ;
        BitBlt (hdcScr, x1, y1, cx, cy, hdcScr, x2, y2, SRCCOPY) ;
        BitBlt (hdcScr, x2, y2, cx, cy, hdcMem, 0, 0, SRCCOPY) ;

        Sleep (10) ;
      }

      DeleteDC (hdcMem) ;
      ReleaseDC (hwnd, hdcScr) ;
      DeleteObject (hBitmap) ;

      LockWindowUpdate (NULL) ;
  }
  return FALSE ;
}
```

SCRAMBLE没有窗口消息处理程序。在WinMain中，它首先呼叫带有桌面窗口句柄的LockWindowUpdate。此函数暂时防止其它程序更新屏幕。然后SCRAMBLE通过呼叫带有参数DCX_LOCKWINDOWUPDATE的GetDCEx来获得整个屏幕的设备内容。这样就只有SCRAMBLE可以更新屏幕了。

然后SCRAMBLE确定全屏幕的尺寸，并将长宽分别除以10。程序用这个尺寸（名称是cx和cy）来建立一个位图，并将该位图选进内存设备内容。

使用C语言的rand函数，SCRAMBLE计算出四个随机值（两个坐标点）作为cx和cy的倍数。程序透过三次呼叫BitBlt函数来交换两个矩形块中显示的内容。第一次将从第一个坐标点开始的矩形复制到内存设备内容。第二次BitBlt将从第二坐标点开始的矩形复制到第一点开始的位置。第三次将内存设备内容中的矩形复制到第二个坐标点开始的区域。

此程序将有效地交换显示器上两个矩形中的内容。SCRAMBLE执行300次交换，这时的屏幕显示肯定是一团糟。但不用担心，因为SCRAMBLE记得是怎么把显示弄得这样一团糟的，接着在退出前它会按相反的次序恢复原来的桌面显示（锁定屏幕前的画面）！

您也可以用内存设备内容将一个位图复制给另一个位图。例如，假定您要建立一个位图，该位图只包含另一个位图左上角的图形。如果原来的图像句柄为hBitmap，那么您可以将其尺寸复制到一个BITMAP型态的结构中：

GetObject (hBitmap, sizeof (BITMAP), &bm) ;

然后建立一个未初始化的新位图，该位图的尺寸是原来图的1/4:

```
hBitmap2 = CreateBitmap ( bm.bmWidth / 2, bm.bmHeight / 2,
                    bm.bmPlanes, bm.bmBitsPixel, NULL) ;
```

现在建立两个内存设备内容，并将原来位图和新位图选分别进这两个内存设备内容:

```
hdcMem1 = CreateCompatibleDC (hdc) ;
hdcMem2 = CreateCompatibleDC (hdc) ;
SelectObject (hdcMem1, hBitmap) ;
SelectObject (hdcMem2, hBitmap2) ;
```

最后，将第一个位图的左上角复制给第二个:

```
BitBlt ( hdcMem2, 0, 0, bm.bmWidth / 2, bm.bmHeight / 2,
       hdcMem1, 0, 0, SRCCOPY) ;
```

剩下的只是清除工作:

DeleteDC (hdcMem1) ;

DeleteDC (hdcMem2) ;

DeleteObject (hBitmap) ;

BLOWUP.C程序，如图14-21所示，也用窗口更新锁定来在程序窗口之外显示一个捕捉的矩形。此程序允许使用者用鼠标圈选屏幕上的矩形区域，然后BLOWUP将该区域的内容复制到位图。在WM_PAINT消息处理期间，位图复制到程序的显示区域，必要时将拉伸或压缩。(参见程序14-12。)

程序14-12 BLOWUP

BLOWUP.C

```
/*------------------------------------------------------------------------
BLOWUP.C -- Video Magnifier Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <stdlib.h> // for abs definition
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("Blowup") ;
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
```

```
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Blow-Up Mouse Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

void InvertBlock (HWND hwndScr, HWND hwnd, POINT ptBeg, POINT ptEnd)
{
  HDC hdc ;
  hdc = GetDCEx (hwndScr, NULL, DCX_CACHE | DCX_LOCKWINDOWUPDATE) ;
  ClientToScreen (hwnd, &ptBeg) ;
  ClientToScreen (hwnd, &ptEnd) ;
  PatBlt (hdc, ptBeg.x, ptBeg.y, ptEnd.x - ptBeg.x, ptEnd.y - ptBeg.y,
    DSTINVERT) ;
  ReleaseDC (hwndScr, hdc) ;
}

HBITMAP CopyBitmap (HBITMAP hBitmapSrc)
{
  BITMAP bitmap ;
  HBITMAP hBitmapDst ;
  HDC hdcSrc, hdcDst ;

  GetObject (hBitmapSrc, sizeof (BITMAP), &bitmap) ;
  hBitmapDst = CreateBitmapIndirect (&bitmap) ;

  hdcSrc = CreateCompatibleDC (NULL) ;
  hdcDst = CreateCompatibleDC (NULL) ;

  SelectObject (hdcSrc, hBitmapSrc) ;
  SelectObject (hdcDst, hBitmapDst) ;

  BitBlt (hdcDst, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
    hdcSrc, 0, 0, SRCCOPY) ;
  DeleteDC (hdcSrc) ;
  DeleteDC (hdcDst) ;

  return hBitmapDst ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL bCapturing, bBlocking ;
  static HBITMAP hBitmap ;
  static HWND hwndScr ;
  static POINT ptBeg, ptEnd ;
  BITMAP bm ;
  HBITMAP hBitmapClip ;
  HDC hdc, hdcMem ;
  int iEnable ;
  PAINTSTRUCT ps ;
  RECT rect ;
```

```
switch (message)
{
case WM_LBUTTONDOWN:
  if (!bCapturing)
  {
    if (LockWindowUpdate (hwndScr = GetDesktopWindow ()))
    {
      bCapturing = TRUE ;
      SetCapture (hwnd) ;
      SetCursor (LoadCursor (NULL, IDC_CROSS)) ;
    }
    else
      MessageBeep (0) ;
  }
  return 0 ;

case WM_RBUTTONDOWN:
  if (bCapturing)
  {
    bBlocking = TRUE ;
    ptBeg.x = LOWORD (lParam) ;
    ptBeg.y = HIWORD (lParam) ;
    ptEnd = ptBeg ;
    InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
  }
  return 0 ;

case WM_MOUSEMOVE:
  if (bBlocking)
  {
    InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
    ptEnd.x = LOWORD (lParam) ;
    ptEnd.y = HIWORD (lParam) ;
    InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
  }
  return 0 ;

case WM_LBUTTONUP:
case WM_RBUTTONUP:
  if (bBlocking)
  {
    InvertBlock (hwndScr, hwnd, ptBeg, ptEnd) ;
    ptEnd.x = LOWORD (lParam) ;
    ptEnd.y = HIWORD (lParam) ;

    if (hBitmap)
    {
      DeleteObject (hBitmap) ;
      hBitmap = NULL ;
    }

    hdc = GetDC (hwnd) ;
    hdcMem = CreateCompatibleDC (hdc) ;
    hBitmap= CreateCompatibleBitmap (hdc,
      abs (ptEnd.x - ptBeg.x),
      abs (ptEnd.y - ptBeg.y)) ;

    SelectObject (hdcMem, hBitmap) ;

    StretchBlt (hdcMem, 0, 0, abs (ptEnd.x - ptBeg.x),
      abs (ptEnd.y - ptBeg.y),
      hdc, ptBeg.x, ptBeg.y, ptEnd.x - ptBeg.x,
      ptEnd.y - ptBeg.y, SRCCOPY) ;
    DeleteDC (hdcMem) ;
    ReleaseDC (hwnd, hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
  }
  if (bBlocking || bCapturing)
```

```
    {
      bBlocking = bCapturing = FALSE ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
      ReleaseCapture () ;
      LockWindowUpdate (NULL) ;
    }
    return 0 ;

  case WM_INITMENUPOPUP:
    iEnable = IsClipboardFormatAvailable (CF_BITMAP) ?
MF_ENABLED : MF_GRAYED ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_PASTE, iEnable) ;

    iEnable = hBitmap ? MF_ENABLED : MF_GRAYED ;

    EnableMenuItem ((HMENU) wParam, IDM_EDIT_CUT, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_COPY, iEnable) ;
    EnableMenuItem ((HMENU) wParam, IDM_EDIT_DELETE, iEnable) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_EDIT_CUT:
    case IDM_EDIT_COPY:
      if (hBitmap)
      {
        hBitmapClip = CopyBitmap (hBitmap) ;
        OpenClipboard (hwnd) ;
        EmptyClipboard () ;
        SetClipboardData (CF_BITMAP, hBitmapClip) ;
      }
      if (LOWORD (wParam) == IDM_EDIT_COPY)
        return 0 ;
      //fall through for IDM_EDIT_CUT
    case IDM_EDIT_DELETE:
      if (hBitmap)
      {
        DeleteObject (hBitmap) ;
        hBitmap = NULL ;
      }
      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;

    case IDM_EDIT_PASTE:
      if (hBitmap)
      {
        DeleteObject (hBitmap) ;
        hBitmap = NULL ;
      }
      OpenClipboard (hwnd) ;
      hBitmapClip = GetClipboardData (CF_BITMAP) ;

      if (hBitmapClip)
        hBitmap = CopyBitmap (hBitmapClip) ;

      CloseClipboard () ;
      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
    }
    break ;
    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      if (hBitmap)
      {
        GetClientRect (hwnd, &rect) ;
```

```
      hdcMem = CreateCompatibleDC (hdc) ;
      SelectObject (hdcMem, hBitmap) ;
      GetObject (hBitmap, sizeof (BITMAP), (PSTR) &bm) ;
      SetStretchBltMode (hdc, COLORONCOLOR) ;

      StretchBlt (hdc, 0, 0, rect.right, rect.bottom,
        hdcMem, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY) ;

      DeleteDC (hdcMem) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    if (hBitmap)
      DeleteObject (hBitmap) ;

    PostQuitMessage (0) ;
    return 0 ;
}
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

BLOWUP.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
BLOWUP MENU DISCARDABLE
BEGIN
POPUP "&Edit"
BEGIN
MENUITEM "Cu&t\tCtrl+X",  IDM_EDIT_CUT
MENUITEM "&Copy\tCtrl+C",   IDM_EDIT_COPY
MENUITEM "&Paste\tCtrl+V",    IDM_EDIT_PASTE
MENUITEM "De&lete\tDelete",   IDM_EDIT_DELETE
END
END

//////////////////////////////////////////////////////////////////////////////
// Accelerator
BLOWUP ACCELERATORS DISCARDABLE
BEGIN
"C",  IDM_EDIT_COPY,  VIRTKEY, CONTROL, NOINVERT
"V",  IDM_EDIT_PASTE, VIRTKEY, CONTROL, NOINVERT
VK_DELETE,  IDM_EDIT_DELETE, VIRTKEY, NOINVERT
"X",  IDM_EDIT_CUT,  VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Blowup.rc
#define IDM_EDIT_CUT        40001
#define IDM_EDIT_COPY       40002
#define IDM_EDIT_PASTE      40003
#define IDM_EDIT_DELETE     40004
```

图14-10 BLOWUP显示的一个范例

由于鼠标拦截的限制，所以开始使用BLOWUP时会有些困难，需要逐渐适应。下面是使用本程序的方法：

在BLOWUP显示区域按下鼠标左键不放，鼠标指针会变成「+」字型。

继续按住左键，将鼠标移到屏幕上的任何其它位置。鼠标光标的位置就是您要圈选的矩形区域的左上角。

继续按住左键，按下鼠标右键，然后拖动鼠标到您要圈选的矩形区域的右下角。释放鼠标左键和右键。(释放鼠标左、右键次序无关紧要。)

鼠标光标恢复成箭头状，这时您圈选的矩形区域已复制到了BLOWUP的显示区域，并作了适当的压缩或拉伸变化。

如果您从右上角到左下角选取的话，BLOWUP将显示矩形区域的镜像。如果从左下到右上角选取，BLOWUP将显示颠倒的图像。如果从右上角至左上角选取，程序将综合两种效果。

BLOWUP还包含将位图复制到剪贴簿，以及将剪贴簿中的位图复制到程序的处理功能。BLOWUP处理WM_INITMENUPOPUP消息来启用或禁用「Edit」菜单中的不同选项，并通过WM_COMMAND消息来处理这些菜单项。您应该对这些程序代码的结构比较熟悉，因为它们与第十二章中的复制和粘贴文字项目的处理方式在本质上是一样的。

不过，对于位图，剪贴簿对象不是整体句柄而是位图句柄。当您使用CF_BITMAP时，GetClipboardData函数传回一个HBITMAP对象，而且SetClipboardData函数接收一个HBITMAP对象。如果您想将位图传送给剪贴簿又想保留副本以供程序本身使用，那么您必须复制位图。同样，如果您从剪贴簿上粘贴了一幅位图，也应该做一个副本。BLOWUP中的CopyBitmap函数是通过取得现存位图的BITMAP结构，并在CreateBitmapIndirect函数中用这个结构建立一个新位图来完成此项操作的。(变量名的后缀Src和Dst分别代表「来源」和「目的」。) 两个位图都被选进内存设备内容，而且通过呼叫BitBlt来复制位图内容。(另一种复制位的方法，可以先按位图大小配置一块内存，然后为来源位图呼叫GetBitmapBits，为目的位图呼叫SetBitmapBits。)

我发现BLOWUP对于检查Windows及其应用程序中大量分散的小位图和图片非常有用。

第十五章 与设备无关的位图

在上一章我们了解到Windows GDI位图对象（也称为与设备相关的位图，或DDB）有许多程序设计用途。但是我并没有展示把这些位图储存到磁盘文件或把它们加载内存的方法。这是以前在Windows中使用的方法，现在根本不用了。因为位图的位格式相当依赖于设备，所以DDB不适用于图像交换。DDB内没有色彩对照表来指定位图的位与色彩之间的联系。DDB只有在Windows开机到关机的生命期内被建立和清除时才有意义。

在Windows 3.0中发表了与设备无关的位图(DIB)，提供了适用于交换的图像文件格式。正如您所知的，像.GIF或.JPEG之类的其它图像文件格式在Internet上比DIB文件更常见。这主要是因为.GIF和.JPEG格式进行了压缩，明显地减少了下载的时间。尽管有一个用于DIB的压缩方案，但极少使用。DIB内的位图几乎都没有被压缩。如果您想在程序中操作位图，这实际上是一个优点。DIB不像.GIF和.JPEG文件，Windows API直接支持DIB。如果在内存中有DIB，您就可以提供指向该DIB的指标作为某些函数的参数，来显示DIB或把DIB转化为DDB。

DIB 文件格式

有意思的是，DIB格式并不是源自于Windows。它首先定义在OS/2的1.1版中，该操作系统最初由IBM和Microsoft在八十年代中期开始开发。OS/2 1.1在1988年发布，并且是第一个包含了类似Windows的图形使用者接口的OS/2版本，该图形使用者接口被称之为「Presentation Manager（PM）」。「Presentation Manager」包含了定义位图格式的「图形程序接口」（GPI）。

然后在Windows 3.0中（发布于1990）使用了OS/2位图格式，这时称之为DIB。Windows 3.0也包含了原始DIB格式的变体，并在Windows下成为标准。在Windows 95（以及Windows NT 4.0）和Windows 98（以及Windows NT 5.0）下也定义了一些其它的增强能力，我会在本章讨论它们。

DIB首先作为一种文件格式，它的扩展名为.BMP，在极少情况下为.DIB。Windows应用程序使用的位图图像被当做DIB文件建立，并作为只读资源储存在程序的可执行文件中。图标和鼠标光标也是形式稍有不同的DIB文件。

程序能将DIB文件减去前14个字节加载连续的内存块中。这时就可以称它为「packed DIB（packed-DIB）格式的位图」。在Windows下执行的应用程序能使用packed DIB格式，通过Windows剪贴簿来交换图像或建立画刷。程序也可以完全存取DIB的内容并以任意方式修改DIB。

程序也能在内存中建立自己的DIB然后把它们存入文件。程序使用GDI函数呼叫就能「绘制」这些DIB内的图像，也能在程序中利用别的内存DIB直接设定和操作像素位。

在内存中加载了DIB后，程序也能通过几个Windows API函数呼叫来使用DIB数据，我将在本章中讨论有关内容。与DIB相关的API呼叫是很少的，并且主要与视讯显示器或打印机页面上显示DIB相关，还与转换GDI位图对象有关。

除了这些内容以外，还有许多应用程序需要完成的DIB任务，而这些任务Windows操作系统并不支持。例如，程序可能存取了24位DIB并且想把它转化为带有最佳化的256色调色盘的8位DIB，而Windows不会为您执行这些操作。但是在本章和下一章将向您显示Windows API之外的操作DIB的方式。

OS/2样式的DIB

先不要陷入太多的细节，让我们看一下与首先在OS/2 1.1中出现的位图格式兼容的Windows DIB格式。

DIB文件有四个主要部分：

文件表头

信息表头
RGB色彩对照表（不一定有）
位图像素位

您可以把前两部分看成是C的数据结构，把第三部分看成是数据结构的数组。在Windows表头文件WINGDI.H中说明了这些结构。在内存中的packed DIB格式内有三个部分：

信息表头

RGB色彩对照表（不一定有）
位图像素位

除了没有文件表头外，其它部分与储存在文件内的DIB相同。

DIB文件（不是内存中的packed DIB）以定义为如下结构的14个字节的文件表头开始：

```
typedef struct tagBITMAPFILEHEADER // bmfh
{
  WORD  bfType ; // signature word "BM" or 0x4D42
  DWORD  bfSize ; // entire size of file
  WORD  bfReserved1 ; // must be zero
  WORD  bfReserved2 ; // must be zero
  DWORD bfOffsetBits ; // offset in file of DIB pixel bits
}
BITMAPFILEHEADER, * PBITMAPFILEHEADER ;
```

在WINGDI.H内定义的结构可能与这不完全相同，但在功能上是相同的。第一个注释（就是文字「bmfh」）指出了给这种数据型态的数据变量命名时推荐的缩写。如果在我的程序内看到了名为pbmfh的变量，这可能是一个指向BITMAPFILEHEADER型态结构的指针或指向PBITMAPFILEHEADER型态变量的指针。

结构的长度为14字节，它以两个字母「BM」开头以指明是位图文件。这是一个WORD值0x4D42。紧跟在「BM」后的DWORD以字节为单位指出了包括文件表头在内的文件大小。下两个WORD字段设定为0。（在与DIB文件格式相似的鼠标光标文件内，这两个字段指出光标的「热点（hot spot）」）。结构还包含一个DWORD字段，它指出了文件中像素位开始位置的字节偏移量。此数值来自DIB信息表头中的信息，为了使用的方便提供在这里。

在OS/2样式的DIB内，BITMAPFILEHEADER结构后紧跟了BITMAPCOREHEADER结构，它提供了关于DIB图像的基本信息。紧缩的DIB（Packed DIB）开始于BITMAPCOREHEADER：

```
typedef struct tagBITMAPCOREHEADER // bmch
{
  DWORD bcSize ; // size of the structure = 12
  WORD bcWidth ; // width of image in pixels
  WORD bcHeight ; // height of image in pixels
  WORD bcPlanes ; // = 1
  WORD bcBitCount ; // bits per pixel (1, 4, 8, or 24)
}
BITMAPCOREHEADER, * PBITMAPCOREHEADER ;
```

「core（核心）」用在这里看起来有点奇特，它是指这种格式是其它由它所衍生的位图格式的

基础。

BITMAPCOREHEADER结构中的bcSize字段指出了数据结构的大小，在这种情况下是12字节。

bcWidth和bcHeight字段包含了以像素为单位的位图大小。尽管这些字段使用WORD意味着一个DIB可能为65,535像素高和宽，但是我们几乎不会用到那么大的单位。

bcPlanes字段的值始终是1。这个字段是我们在上一章中遇到的早期Windows GDI位图对象的残留物。

bcBitCount字段指出了每像素的位数。对于OS/2样式的DIB，这可能是1、4、8或24。DIB图像中的颜色数等于2bmch.bcBitCount，或用C的语法表示为：

1 << bmch.bcBitCount

这样，bcBitCount字段等于：

1代表2色DIB

4代表16色DIB
8代表256色DIB
24代表full -Color DIB

当我提到「8位DIB」时，就是说每像素占8位的DIB。

对于前三种情况（也就是位数为1、4和8时），BITMAPCOREHEADER后紧跟色彩对照表，24位DIB没有色彩对照表。色彩对照表是一个3字节RGBTRIPLE结构的数组，数组中的每个元素代表图像中的每种颜色：

```
typedef struct tagRGBTRIPLE // rgbt
{
  BYTE rgbtBlue ;  // blue level
  BYTE rgbtGreen ;  // green level
  BYTE rgbtRed ;  // red level
}
RGBTRIPLE ;
```

这样排列色彩对照表以便DIB中最重要的颜色首先显示，我们将在下一章说明原因。

WINGDI.H表头文件也定义了下面的结构：

```
typedef struct tagBITMAPCOREINFO // bmci
{
  BITMAPCOREHEADER bmciHeader ;   // core-header structure
  RGBTRIPLE  bmciColors[1] ;  // color table array
}
BITMAPCOREINFO, * PBITMAPCOREINFO ;
```

这个结构把信息表头与色彩对照表结合起来。虽然在这个结构中RGBTRIPLE结构的数量等于1，但在DIB文件内您绝对不会发现只有一个RGBTRIPLE。根据每个像素的位数，色彩对照表的大小始终是2、16或256个RGBTRIPLE结构。如果需要为8位DIB配置PBITMAPCOREINFO结构，您可以这样做：

pbmci = malloc (sizeof (BITMAPCOREINFO) + 255 * sizeof (RGBTRIPLE)) ;

然后可以这样存取RGBTRIPLE结构：

pbmci->bmciColors[i]

因为RGBTRIPLE结构的长度是3字节，许多RGBTRIPLE结构可能在DIB中以奇数地址开始。然而，因为在DIB文件内始终有偶数个的RGBTRIPLE结构，所以紧跟在色彩对照表数组后的数据块总是以WORD地址边界开始。

紧跟在色彩对照表（24位DIB中是信息表头）后的数据是像素位本身。

由下而上

像大多数位图格式一样，DIB中的像素位是以水平行组织的，用视讯显示器硬件的术语称作「扫描线」。行数等于BITMAPCOREHEADER结构的bcHeight字段。然而，与大多数位图格式不同的是，DIB从图像的底行开始，往上表示图像。

在此应定义一些术语，当我们说「顶行」和「底行」时，指的是当其正确显示在显示器或打印机的页面上时出现在虚拟图像的顶部和底部。就好像肖像的顶行是头发，底行是下巴，在DIB文件中的「第一行」指的是DIB文件的色彩对照表后的像素行，「最后行」指的是文件最末端的像素行。

因此，在DIB中，图像的底行是文件的第一行，图像的顶行是文件的最后一行。这称之为由下而上的组织。因为这种组织和直觉相反，您可能会问：为什么要这么做?

好，现在我们回到OS/2的Presentation Manager。IBM的人认为PM内的坐标系统 – 包括窗口、图形和位图 – 应该是一致的。这引起了争论：大多数人，包括在全画面文字方式下编程和窗口环境下工作的程序写作者认为应使用垂直坐标在屏幕上向下增加的坐标。然而，计算机图形程序写作者认为应使用解析几何的数学方法进行视讯显示，这是一个垂直坐标在空间中向上增加的直角（或笛卡尔）坐标系。

简而言之，数学方法赢了。PM内的所有事物都以左下角为原点（包括窗口坐标），因此DIB也就有了那种方式。

DIB像素位

DIB文件的最后部分（在大多数情况下是DIB文件的主体）由实际的DIB的像素字节成。像素位是由从图像的底行开始并沿着图像向上增长的水平行组织的。

DIB中的行数等于BITMAPCOREHEADER结构的bcHeight字段。每一行的像素数等于该结构的bcWidth字段。每一行从最左边的像素开始，直到图像的右边。每个像素的位数可以从bcBitCount字段取得，为1、4、8或24。

以字节为单位的每行长度始终是4的倍数。行的长度可以计算为：

```
RowLength = 4 * ((bmch.bcWidth * bmch.bcBitCount + 31) / 32) ;
```

或者在C内用更有效的方法：

```
RowLength = ((bmch.bcWidth * bmch.bcBitCount + 31) & ~31) >> 3 ;
```

如果需要，可通过在右边补充行（通常是用零）来完成长度。像素数据的总字节数等于RowLength和bmch.bcHeight的乘积。

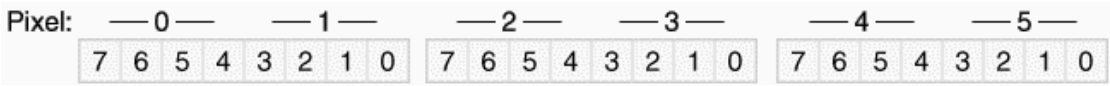
要了解像素编码的方式，让我们分别考虑四种情况。在下面的图表中，每个字节的位显示在框内并且编了号，7表示最高位，0表示最低位。像素也从行的最左端从0开始编号。

对于每像素1位的DIB，每字节对应为8像素。最左边的像素是第一个字节的最高位：

Pixel:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

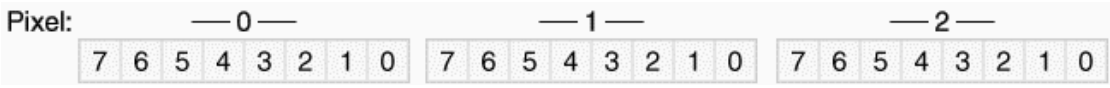
每个像素可以是0或1。0表示该像素的颜色由色彩对照表中第一个RGBTRIPLE项目给出。1表示像素的颜色由色彩对照表的第二个项目给出。

对于每像素4位的DIB，每个字节对应两个像素。最左边的像素是第一个字节的高4位，以此类推：



每像素4位的值的范围从0到15。此值是指向色彩对照表中16个项目的索引。

对于每像素8位的DIB，每个字节为1个像素：



字节的值从0到255。同样，这也是指向色彩对照表中256个项目的索引。

对于每像素24位的DIB，每个像素需要3个字节来代表红、绿和蓝的颜色值。像素位的每一行，基本上就是RGBTRIPLE结构的数组，可能需要在每行的末端补0以便该行为4字节的倍数：

每像素24位的DIB没有色彩对照表。

扩展的Windows DIB

现在我们掌握了Windows 3.0中介绍的与OS/2兼容的DIB，同时也看一看Windows中DIB的扩展版本。

这种DIB形式跟前面的格式一样，以BITMAPFILEHEADER结构开始，但是接着是BITMAPINFOHEADER结构，而不是BITMAPCOREHEADER结构：

```
typedef struct tagBITMAPINFOHEADER // bmih
{
  DWORD biSize ;  // size of the structure = 40
  LONG biWidth ;  // width of the image in pixels
  LONG biHeight ;  // height of the image in pixels
  WORD biPlanes ;  // = 1
  WORD biBitCount ;  // bits per pixel (1, 4, 8, 16, 24, or 32)
  DWORD biCompression ; // compression code
  DWORD biSizeImage ;  // number of bytes in image
  LONG biXPelsPerMeter ; // horizontal resolution
  LONG biYPelsPerMeter ; // vertical resolution
  DWORD biClrUsed ;  // number of colors used
  DWORD biClrImportant ; // number of important colors
}
BITMAPINFOHEADER, * PBITMAPINFOHEADER ;
```

您可以通过检查结构的第一字段区分与OS/2兼容的DIB和Windows DIB，前者为12，后者为40。

您将注意到，在这个结构内有六个附加的字段，但是BITMAPINFOHEADER不是简单地由BITMAPCOREHEADER加上一些新字段而成。仔细看一下：在BITMAPCOREHEADER结构中，bcWidth和bcHeight字段是16位WORD值；而在BITMAPINFOHEADER结构中它们是32位LONG值。这是一个令人讨厌的小变化，当心它会给您带来麻烦。

另一个变化是：对于使用BITMAPINFOHEADER结构的1位、4位和8位DIB，色彩对照表不是RGBTRIPLE结构的数组。相反，BITMAPINFOHEADER结构紧跟着一个RGBQUAD结构的数组：

```
typedef struct tagRGBQUAD // rgb
{
  BYTE rgbBlue ;  // blue level
  BYTE rgbGreen ; // green level
  BYTE rgbRed ;  // red level
  BYTE rgbReserved ; // = 0
}
RGBQUAD ;
```

除了包括总是设定为0的第四个字段外，与RGBTRIPLE结构相同。 WINGDI.H表头文件也定义了以下结构：

```
typedef struct tagBITMAPINFO // bmi
{
  BITMAPINFOHEADER bmiHeader ; // info-header structure
  RGBQUAD  bmiColors[1] ; // color table array
}
BITMAPINFO, * PBITMAPINFO ;
```

注意，如果BITMAPINFO结构以32位的地址边界开始，因为BITMAPINFOHEADER结构的长度是40字节，所以RGBQUAD数组内的每一个项目也以32位边界开始。这样就确保通过32位微处理器能更有效地对色彩对照表数据寻址。

尽管BITMAPINFOHEADER最初是在Windows 3.0中定义的，但是许多字段在Windows 95和Windows NT 4.0中又重新定义了，并且被带入Windows 98和Windows NT 5.0中。比如现在的文件中说：「如果biHeight是负数，则位图是由上而下的DIB，原点在左上角」。这很好，但是在1990年刚开始定义DIB格式时，如果有人做了这个决定，那会更好。我的建议是避免建立由上而下的DIB。有一些程序在编写时没有考虑这种新「特性」，在遇到负的biHeight字段时会当掉。还有如Microsoft Word 97带有的Microsoft Photo Editor在遇到由上而下的DIB时会报告「图像高度不合法」（虽然Word 97本身不会出错）。

biPlanes字段始终是1，但biBitCount字段现在可以是16或32以及1、4、8或24。这也是在Windows 95和Windows NT 4.0中的新特性。一会儿我将介绍这些附加格式工作的方式。

现在让我们先跳过biCompression和biSizeImage字段，一会儿再讨论它们。

biXPelsPerMeter和biYPelsPerMeter字段以每公尺多少像素这种笨拙的单位指出图像的实际尺寸。（「pel」--picture element（图像元素）--是IBM对像素的称呼。）Windows在内部不使用此类信息。然而，应用程序能够利用它以准确的大小显示DIB。如果DIB来源于没有方像素的设备，这些字段是很有用的。在大多数DIB内，这些字段设定为0，这表示没有建议的实际大小。每英寸72点的分辨率（有时用于视讯显示器，尽管实际分辨率依赖于显示器的大小）大约相当于每公尺2835个像素，300 DPI的普通打印机的分辨率是每公尺11,811个像素。

biClrUsed是非常重要的字段，因为它影响色彩对照表中项目的数量。对于4位和8位DIB，它能分别指出色彩对照表中包含了小于16或256个项目。虽然并不常用，但这是一种缩小DIB大小的方法。例如，假设DIB图像仅包括64个灰阶，biClrUsed字段设定为64，并且色彩对照表为256个字节大小的色彩对照表包含了64个RGBQUAD结构。像素值的范围从0x00到0x3F。DIB仍然每像素需要1字节，但每个像素字节的高2位为零。如果biClrUsed字段设定为0，意味着色彩对照表包含了由biBitCount字段表示的全部项目数。

从Windows 95开始，biClrUsed字段对于16位、24位或32位DIB可以为非零。在这些情况下，Windows不使用色彩对照表解释像素位。相反地，它指出DIB中色彩对照表的大小，程序使用该信息来设定调色盘在256色视讯显示器上显示DIB。您可能想起在OS/2兼容格式中，24位DIB没有色彩对照表。在Windows 3.0中的扩展格式中，也与这一样。而在Windows 95中，24位DIB有色彩对照表，biClrUsed字段指出了它的大小。

总结如下：

对于1位DIB，biClrUsed始终是0或2。色彩对照表始终有两个项目。

对于4位DIB，如果biClrUsed字段是0或16，则色彩对照表有16个项目。如果是从2到15的数，则指的是色彩对照表中的项目数。每个像素的最大值是小于该数的1。

对于8位DIB，如果biClrUsed字段是0或256，则色彩对照表有256个项目。如果是从2到225

的数，则指的是色彩对照表中的项目数。每个像素的最大值是小于该数的1。

对于16位、24位或32位DIB，biClrUsed字段通常为0。如果它不为0，则指的是色彩对照表中的项目数。执行于256色显示卡的应用程序能使用这些项目来为DIB设定调色盘。

另一个警告：原先使用早期DIB文件编写的程序不支持24位DIB中的色彩对照表，如果在程序使用24位DIB的色彩对照表的话，就要冒一定的风险。

biClrImportant字段实际上没有biClrUsed字段重要，它通常被设定为0以指出色彩对照表中所有的颜色都是重要的，或者它与biClrUsed有相同的值。两种方法意味着同一件事，如果它被设定为0与biClrUsed之间的值，就意味着DIB图像能仅仅通过色彩对照表中第一个biClrImportant项目合理地取得。当在256色显示卡上并排显示两个或更多8位DIB时，这是很有用的。

对于1位、4位、8位和24位的DIB，像素位的组织和OS/2兼容的DIB是相同的，一会儿我将讨论16位和32位DIB。

真实检查

当遇到一个由其它程序或别人建立的DIB时，您希望从中发现什么内容呢?

尽管在Windows3.0首次推出时，OS/2样式的DIB已经很普遍了，但最近这种格式却已经很少出现了。许多程序写作者在实际编写快速DIB例程时忽略了它们。您遇到的任何4位DIB可能是Windows的「小画家」程序使用16色视讯显示器建立的，在这些显示器上色彩对照表具有标准的16种颜色。

最普遍的DIB可能是每像素8位。8位DIB分为两类：灰阶DIB和混色DIB。不幸的是，表头信息中并没有指出8位DIB的型态。

许多灰阶DIB有一个等于64的biClrUsed字段，指出色彩对照表中的64个项目。这些项目通常以上升的灰阶层排列，也就是说色彩对照表以00-00-00、04-04-04、08-08-08、0C-0C-0C的RGB值开始，并包括F0-F0-F0、F4-F4-F4、F8-F8-F8和FC-FC-FC的RGB值。此类色彩对照表可用下列公式计算：

rgb[i].rgbRed = rgb[i].rgbGreen = rgb[i].rgbBlue = i * 256 / 64 ;

在这里rgb是RGBQUAD结构的数组，i的范围从0到63。灰阶色彩对照表可用下列公式计算：

rgb[i].rgbRed = rgb[i].rgbGreen = rgb[i].rgbBlue = i * 255 / 63 ;

因而此表以FF-FF-FF结尾。

实际上使用哪个计算公式并没有什么区别。许多视讯显示卡和显示器没有比6位更大的色彩精确度。第一个公式承认了这个事实。然而当产生小于64的灰阶时 – 可能是16或32（在此情况下公式的除数分别是15和31） – 使用第二个公式更适合，因为它确保了色彩对照表的最后一个项目是FF-FF-FF，也就是白色。

当某些8位灰阶DIB在色彩对照表内有64个项目时，其它灰阶的DIB会有256个项目。biClrUsed字段实际上可以为0（指出色彩对照表中有256个项目）或者从2到256的数。当然，biClrUsed值是2的话就没有任何意义（因为这样的8位DIB能当作1位DIB被重新编码）或者小于或等于16的值也没意义（因为它能当作4位DIB被重新编码）。任何情况下，色彩对照表中的项目数必须与biClrUsed字段相同（如果biClrUsed是0，则是256），并且像素值不能超过色彩对照表项目数减1的值。这是因为像素值是指向色彩对照表数组的索引。对于biClrUsed值为64的8位DIB，像素值的范围从0x00到0x3F。

在这里应记住一件重要的事情：当8位DIB具有由整个灰阶组成的色彩对照表（也就是说，当红

色、绿色和蓝色程度相等时），或当这些灰阶层在色彩对照表中递增（像上面描述的那样）时，像素值自身就代表了灰色的程度。也就是说，如果biClrUsed是64，那么0x00像素值为黑色，0x20的像素值是50%的灰阶，0x3F的像素值为白色。

这对于一些图像处理作业是很重要的，因为您可以完全忽略色彩对照表，仅需处理像素值。这是很有用的，如果让我回溯时光去对BITMAPINFOHEADER结构做一个简单的更改，我会添加一个旗标指出DIB映像是不是灰阶的，如果是，DIB就没有色彩对照表，并且像素值直接代表灰阶。

混色的8位DIB一般使用整个色彩对照表，它的biClrUsed字段为0或256。然而您也可能遇到较小的颜色数，如236。我们应承认一个事实：程序通常只能在Windows颜色面内更改236个项目以正确显示这些DIB，我将在下章讨论这些内容。

biXPelsPerMeter和biYPelsPerMeter很少为非零值，biClrImportant字段不为0或biClrUsed值的情况也很少。

DIB压缩

前面我没有讨论BITMAPINFOHEADER中的biCompression和biSizeImage字段，现在我们讨论一下这些值。

biCompression字段可以为四个常数之一，它们是：BI_RGB、BI_RLE8、BI_RLE4或BI_BITFIELDS。它们定义在WINGDI.H表头文件中，值分别为0到3。此字段有两个用途：对于4位和8位DIB，它指出像素位被用一种运行长度（run-length）编码方式压缩了。对于16位和32位DIB，它指出了颜色屏蔽（color masking）是否用于对像素位进行编码。这两个特性都是在Windows 95中发表的。

首先让我们看一下RLE压缩：

对于1位DIB，biCompression字段始终是BI_RGB。

对于4位DIB，biCompression字段可以是BI_RGB或BI_RLE4。
对于8位DIB，biCompression字段可以是BI_RGB或BI_RLE8。
对于24位DIB，biCompression字段始终是BI_RGB。

如果值是BI_RGB，像素位储存的方式和OS/2兼容的DIB一样，否则就使用运行长度编码压缩像素位。

运行长度编码（RLE）是一种最简单的数据压缩形式，它是根据DIB映射在一列内经常有相同的像素字符串这个事实进行的。RLE通过对重复像素的值及重复的次数编码来节省空间，而用于DIB的RLE方案只定义了很少的矩形DIB图像，也就是说，矩形的某些区域是未定义的，这能被用于表示非矩形图像。

8位DIB的运行长度编码在概念上更简单一些，因此让我们从这里入手。表15-1会帮助您理解当biCompression字段等于BI_RGB8时，像素位的编码方式。

表15-1

字节1	字节2	字节3	字节4	含义
00	00			行尾
00	01			映射尾
00	02	dx	dy	移到（x+dx，y+dy）
00	n = 03到FF			使用下面n个像

				素
n = 01到FF	像素			重复像素n次

当对压缩的DIB译码时，可成对查看DIB数据字节，例如此表内的「字节1」和「字节2」。表格以这些字节值的递增方式排列，但由下而上讨论这个表格会更有意义。

如果第一个字节非零（表格最后一行的情况），那么它就是运行长度的重复因子。下面的像素值被重复多次，例如，字节对

0x05 0x27

解碼后的像素值为：

0x27 0x27 0x27 0x27 0x27

当然DIB会有许多数据不是像素到像素的重复，表格倒数第二行处理这种情况，它指出紧跟着的像素数应逐个使用。例如：考虑序列

0x00 0x06 0x45 0x32 0x77 0x34 0x59 0x90

解碼后的像素值为：

0x45 0x32 0x77 0x34 0x59 0x90

这些序列总是以2字节的界限排列。如果第二个字节是奇数，那么序列内就有一个未使用的多余字节。例如，序列

0x00 0x05 0x45 0x32 0x77 0x34 0x59 0x00

解碼后的像素值为：

0x45 0x32 0x77 0x34 0x59

这就是运行长度编码的工作方式。很明显地，如果在DIB图像内没有重复的像素，使用此压缩技术实际上会增加了DIB文件的大小。

上面表格的前三行指出了矩形DIB图像的某些部分可以不被定义的方法。想象一下，您写的程序对已压缩的DIB进行解压缩，在这个解压缩的例程中，您将保持一对数字（x,y），开始为（0,0）。每对一个像素译码，x的值就增加1，每完成一行就将x重新设为0并且增加y的值。

当遇到跟着0x02的字节0x00时，您读取下两个字节并把它们作为无正负号的增量添加到目前的x和y值中，然后继续解碼。当遇到跟着0x00的0x00时，您就解完了一行，应将x设0并增加y值。当遇到跟着0x01的0x00时，您就完成译码了。这些代码准许DIB包含那些未定义的区域，它们用于对非矩形图像编码或在制作数字动画和电影时非常有用（因为几乎每一格影像都有来自前一格的信息而不需重新编码）。

对于4位DIB，编码一般是相同的，但更复杂，因为字节和像素之间不是一对一的关系。

如果读取的第一个字节非零，那就是一个重复因子n。第二个字节（被重复的）包含2个像素，在n个像素的被解碼的序列中交替出现。例如，字节对

0x07 0x35

被解碼为：

0x35 0x35 0x35 0x3?

其中的问号指出像素还未知，如果是上面显示的0x07 0x35对紧跟着下面的字节对：

0x05 0x24

则整个解碼的序列为：

0x35 0x35 0x35 0x32 0x42 0x42

如果字节对中的第一字节是0x00 ，第二个字节是0x03或更大，则使用第二字节指出的像素数。例如，序列

0x00 0x05 0x23 0x57 0x10 0x00

解碼为：

0x23 0x57 0x1?

注意必须填补解碼的序列使其成为偶数字节。

无论biCompression字段是BI_RLE4或BI_RLE8，biSizeImage字段都指出了字节内DIB像素数据的大小。如果biCompression字段是BI_RGB，则biSizeImage通常为0，但是它能被设定为行内位组长度的biHeight倍，就像在本章前面计算的那样。

目前文件说「由上而下的DIB不能被压缩」。由上而下的DIB是在biHeight字段为负数的情况下出现的。

颜色掩码（Color Masking）

biCompression字段也用于连结Windows 95中新出现的16位和32位DIB。对于这些DIB，biCompression字段可以是BI_RGB或BI_BITFIELDS（均定义为值3）。

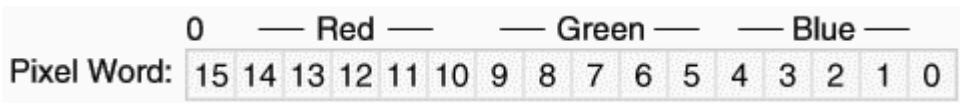
让我们看一下24位DIB的像素格式，它始终有一个等于BI_RGB的biCompression字段：

也就是说，每一行基本上都是RGBTRIPLE结构的数组，在每行末端有可能补充值以使行内的字节是4的倍数。

对于具有biCompression字段等于BI_RGB的16位DIB，每个像素需要两个字节。颜色是这样来编码的：

每种颜色使用5位。对于行内的第一个像素，蓝色值是第一个字节的最低五位。绿色值在第一和第二个字节中都有位：绿色值的两个最高位是第二个字节中的两个最低位，绿色值的三个最低位是第一个字节中的三个最高位。红色值是第二个字节中的2到6位。第二个字节的最高位是0。

当以16位字组存取像素值时，这会更加有意义。因为多个字节值的最低位首先被储存，像素字组如下：



假设在wPixel内储存了16位像素，您能用下列公式计算红色、绿色和蓝色值：

```
Red = ((0x7C00 & wPixel) >> 10) << 3 ;
Green = ((0x03E0 & wPixel) >> 5) << 3 ;
```

Blue = ((0x001F & wPixel) >> 0) << 3 ;

首先，使用屏蔽值与像素进行了位AND运算。此结果是：红色向右移动10位，绿色向右移动5位，蓝色向右移动0位。（这些移动值我称之为「右移值」）。这就产生了从0x00和0x1F的颜色值，这些值必须向左移动3位以合成从0x00到0xF8的颜色值。（这些移动值我称之为「左移值」。）

请记住：如果16位DIB的像素宽度是奇数，每行会在末端补充多余的2字节以使字节宽度能被4整除。

对于32位DIB，如果biCompression等于BI_RGB，每个像素需要4字节。蓝色值是第一个字节，绿色为第二个，红色为第三个，第四字节等于0。也可这么说，像素是RGBQUAD结构的数组。因为每个像素的长度是4字节，在列末端就不需填补字节。

若想以32位双字组存取每个像素，它就像这样：

	0							Red							Green					Blue	
Pixel Double Word:	31	30	...	25	24	23	22	...	17	16	15	14	...	9	8	7	6	...	1	0

如果dwPixel是32位双字组，

Red = ((0x00FF0000 & dwPixel) >> 16) << 0 ;
Green = ((0x0000FF00 & dwPixel) >> 8) << 0 ;
Blue = ((0x000000FF & dwPixel) >> 0) << 0 ;

左移值全为零，因为颜色值在0xFF已是最大。注意这个双字组与Windows GDI函数呼叫中用于指定RGB颜色的32位COLORREF值不一致。在COLORREF值中，红色占最低位的字节。

到目前为止，我们讨论了当biCompression字段为BI_RGB时，16位和32位DIB的内定情况。如果biCompression字段为BI_BITFIELDS，则紧跟着DIB的BITMAPINFOHEADER结构的是三个32位颜色屏蔽，第一个用于红色，第二个用于绿色，第三个用于蓝色。可以使用C的位AND运算子（&）把这些屏蔽应用于16位或32位的像素值上。然后通过右移值向右移动结果，这些值只有检查完屏蔽后才能知道。颜色屏蔽的规则应该很明确：每个颜色屏蔽位串内的1必须是连续的，并且1不能在三个屏蔽位串中重迭。

让我们来举个例子，如果您有一个16位DIB，并且biCompression字段为BI_BITFIELDS。您应该检查BITMAPINFOHEADER结构之后的前三个双字组：

0x0000F800

0x000007E0

0x0000001F

注意，因为这是16位DIB，所以只有位于底部16位的位值才能被设定为1。您可以把变量dwMask[0]、dwMask[1]和dwMask[2]设定为这些值。现在可以编写从掩码中计算右移和左移值的一些例程了：

```
int MaskToRShift (DWORD dwMask)
{
  int iShift ;
  if ( dwMask == 0)
    return 0 ;
  for (  iShift = 0 ; !(dwMask & 1) ; iShift++)
    dwMask >>= 1 ;

  return iShift ;
}
```

```
int MaskToLShift (DWORD dwMask)
{
  int iShift ;
  if ( dwMask == 0)
    return 0 ;

  while (!(dwMask & 1))
    dwMask >>= 1 ;

  for (iShift = 0 ; dwMask & 1 ; iShift++)
    dwMask >>= 1 ;

  return 8 - iShift ;
}
```

然后呼叫MaskToRShift函数三次来获得右移值：

iRShift[0] = MaskToRShift (dwMask[0]) ;

iRShift[1] = MaskToRShift (dwMask[1]) ;

iRShift[2] = MaskToRShift (dwMask[2]) ;

分别得到值11、5和0。然后呼叫MaskToLShift：

iLShift[0] = MaskToLShift (dwMask[0]) ;

iLShift[1] = MaskToLShift (dwMask[1]) ;

iLShift[2] = MaskToLShift (dwMask[2]) ;

分别得到值3、2和3。现在能从像素中提取每种颜色：

```
Red = ((dwMask[0] & wPixel) >> iRShift[0]) << iLShift[0] ;
Green = ((dwMask[1] & wPixel) >> iRShift[1]) << iLShift[1] ;
Blue = ((dwMask[2] & wPixel) >> iRShift[2]) << iLShift[2] ;
```

除了颜色标记能大于0x0000FFFF（这是16位DIB的最大屏蔽值）之外，程序与32位DIB一样。

注意：

对于16位或32位DIB，红色、绿色和蓝色值能大于255。实际上，在32位DIB中，如果屏蔽中有两个为0，第三个应为32位颜色值0xFFFFFFFF。当然，这有点荒唐，但不用担心这个问题。

不像Windows NT，Windows 95和Windows 98在使用颜色屏蔽时有许多的限制。可用的值显示在表15-2中。

表15-2

	16位DIB	16位DIB	32位DIB
红色屏蔽	0x00007C00	0x0000F800	0x00FF0000
绿色屏蔽	0x000003E0	0x000007E0	0x0000FF00
蓝色屏蔽	0x0000001F	0x0000001F	0x000000FF
速记为	5-5-5	5-6-5	8-8-8

换句话说，就是当biCompression是BI_RGB时，您能使用内定的两组屏蔽，包括前面例子中显示的屏蔽组。表格底行显示了一个速记符号来指出每像素红色、绿色和蓝色的位数。

第4版本的Header

我说过，Windows 95更改了一些原始BITMAPINFOHEADER字段的定义。Windows 95也包括了一个称为BITMAPV4HEADER的新扩展的信息表头。如果您知道Windows 95曾经称作Windows 4.0，则就会明白此结构的名称了，Windows NT 4.0也支持此结构。

```
typedef struct
{
  DWORD  bV4Size ;  // size of the structure = 120
  LONG bV4Width ;  // width of the image in pixels
  LONG  bV4Height ;  // height of the image in pixels
  WORD  bV4Planes ;  // = 1
  WORD  bV4BitCount ;  // bits per pixel (1, 4, 8, 16, 24, or 32)
  DWORD  bV4Compression ; // compression code
  DWORD  bV4SizeImage ; // number of bytes in image
  LONG  bV4XPelsPerMeter ; // horizontal resolution
  LONG  bV4YPelsPerMeter ; // vertical resolution
  DWORD  bV4ClrUsed ;  // number of colors used
  DWORD  bV4ClrImportant ; // number of important colors
  DWORD  bV4RedMask ;  // Red color mask
  DWORD  bV4GreenMask ; // Green color mask
  DWORD  bV4BlueMask ;  // Blue color mask
  DWORD  bV4AlphaMask ; // Alpha mask
  DWORD  bV4CSType ;  // color space type
  CIEXYZTRIPLE bV4Endpoints ; // XYZ values
  DWORD  bV4GammaRed ;  // Red gamma value
  DWORD  bV4GammaGreen ; // Green gamma value
  DWORD  bV4GammaBlue ; // Blue gamma value
}
BITMAPV4HEADER, * PBITMAPV4HEADER ;
```

注意前11个字段与BITMAPINFOHEADER结构中的相同，后5个字段支持Windows 95和Windows NT 4.0的图像颜色调配技术。除非使用BITMAPV4HEADER结构的后四个字段，否则您应该使用BITMAPINFOHEADER（或BITMAPV5HEADER）。

当bV4Compression字段等于BI_BITFIELDS时，bV4RedMask、bV4GreenMask和bV4BlueMask可以用于16位和32位DIB。它们作为定义在BITMAPINFOHEADER结构中的颜色屏蔽用于相同的函数，并且当使用除了明确的结构字段之外的原始结构时，它们实际上出现在DIB文件的相同位置。就我所知，bV4AlphaMask字段不被使用。

BITMAPV5HEADER结构剩余的字段包括「Windows颜色管理（Image Color Management)」，它的内容超越了本书的范围，但是了解一些背景会对您有益。

为色彩使用RGB方案的问题在于，它依赖于视讯显示器、彩色照相机和彩色扫描仪的显示技术。如果颜色指定为RGB值(255,0,0)，意味着最大的电压应该加到阴极射线管内的红色电子枪上，RGB值（128,0,0）表示使用一半电压。不同显示器会产生不同的效果。而且，打印机使用了不同的颜色表示方法，以青色、洋红色、黄色和黑色的组合表示颜色。这些方法称之为CMY（cyan-magenta-yellow：青色-洋红色-黄色）和CMYK（cyan-magenta-yellow-black：青色-洋红色-黄色-黑色）。数学公式能把RGB值转化为CMY和CMYK，但不能保证打印机颜色与显示器颜色相符合。「色彩调配技术」是把颜色与对设备无关的标准联系起来的一种尝试。

颜色的现象与可见光的波长有关，波长的范围从380nm（蓝）到780nm（红）之间。一切我们能察觉的光线是可见光谱内不同波长的组合。1931年，Commission Internationale de L'Eclairage (International Commission on Illumination)或CIE开发了一种科学度量颜色的方法。这包括使用三个颜色调配函数（名称为x、y和z），它们以其省略的形式（带有每5nm的值）发表在

CIE Publication 15.2-1986，「Colorimetry，Second Edition」的表2.1中。

颜色的光谱（S）是一组指出每个波长强度的值。如果知道光谱，就能够将与颜色相关的函数应用到光谱来计算X、Y和Z：

$$X=\sum_{\lambda=380}^{780} S(\lambda)\,\bar{x}(\lambda)$$

$$Y=\sum_{\lambda=380}^{780} S(\lambda)\,\bar{y}(\lambda)$$

$$Z=\sum_{\lambda=380}^{780} S(\lambda)\,\bar{z}(\lambda)$$

这些值称为**大X、大Y和大 Z**。y颜色匹配函数等于肉眼对范围在可见光谱内光线的反应。（他看上去像一条由380nm和780nm到0的时钟形曲线）。Y称之为CIE亮度，因为它指出了光线的总体强度。

如果使用BITMAPV5HEADER结构，bV4CSType字段就必须设定为LCS_CALIBRATED_RGB，其值为0。后四个字节必须设定为有效值。

CIEXYZTRIPLE结构按照如下方式定义：

```
typedef struct tagCIEXYZTRIPLE
{
  CIEXYZ ciexyzRed ;
  CIEXYZ ciexyzGreen ;
  CIEXYZ ciexyzBlue ;
}
CIEXYZTRIPLE, * LPCIEXYZTRIPLE ;
```

而CIEXYZ结构定义如下：

```
typedef struct tagCIEXYZ
{
  FXPT2DOT30 ciexyzX ;
  FXPT2DOT30 ciexyzY ;
  FXPT2DOT30 ciexyzZ ;
}
CIEXYZ, * LPCIEXYZ ;
```

这三个字段定义为FXPT2DOT30值，意味着它们是带有2位整数部分和30位小数部分的定点值。这样，0x40000000是1.0，0x48000000是1.5。最大值0xFFFFFFFF仅比4.0小一点点。

bV4Endpoints字段提供了三个与RGB颜色（255,0,0）、（0,255,0）和（0,0,255）相关的X、Y和Z值。这些值应该由建立DIB的应用程序插入以指明这些RGB颜色的设备无关的意义。

BITMAPV4HEADER剩余的三个字段指「伽马值」（希腊的小写字母γ），它指出颜色等级规格内的非线性。在DIB内，红、绿、蓝的范围从0到225。在显示卡上，这三个数值被转化为显示器使用的三个模拟电压，电压决定了每个像素的强度。然而，由于阴极射线管中电子枪的电子特性，像素的强度（I）并不与电压（V）线性相关，它们的关系为：

$$I = (V + \varepsilon)^{\gamma}$$

ε是由显示器的「亮度」控制设定的黑色等级（理想值为0）。指数γ由显示器的「图像」或「对比度」控制设定的。对于大多数显示器，γ大约在2.5左右。

为了对此非线性作出补偿，摄影机在线路内包含了「伽马修正」。指数0.45修正了进入摄影机的光线，这意味着视讯显示器的伽马为2.2。（视讯显示器的高伽马值增加了对比度，这通常是不需要的，因为周围的光线更适合于低对比度。）

视讯显示器的这个非线性反应实际上是很适当的，这是因为人类对光线的反应也是非线性的。我曾提过，Y被称为CIE亮度，这是线性的光线度量。CIE也定义了一个接近于人类感觉的亮度值。亮度是L* (发音为"ell star") ，通过使用如下公式从Y计算得到的：

$$L^* = \begin{cases} 903.3\dfrac{Y}{Y_n} & \dfrac{Y}{Y_n} \le 0.008856 \\ 116\left(\dfrac{Y}{Y_n}\right)^{\frac{1}{3}} - 16 & 0.008856 < \dfrac{Y}{Y_n} \end{cases}$$

在此Yn是白色等级。公式的第一部分是一个小的线性部分。一般，人类的亮度感觉是与线性亮度的立方根相关的，这由第二个公式指出。L* 的范围从0到100，每次L* 的增加都假定是人类能感觉到的亮度的最小变化。

根据知觉亮度而不是线性亮度对光线强度编码要更好一些。这使得位的数量减少到一个合理的程度并且在模拟线路上也降低了噪声。

让我们来看一下整个程序。像素值(P)范围从0到255，它被线性转化成电压等级，我们假定标准化为0.0到1.0之间的值。假设显示器的黑色级设定为0，则像素的强度为：

$$I = V^{r} = \left(\frac{P}{255}\right)^{r}$$

这里γ大约为2.5。人类感觉的亮度(L*)依赖于此强度的立方根和变化从0到100的范围，因此大约是：

$$L^* = 100\left(\frac{P}{255}\right)^{\frac{r}{3}}$$

指数值大约为0.85。如果指数值为1，那么CIE亮度与像素值完全匹配。当然不完全是那种情况，

但是如果像素值指出了线性亮度就非常接近。

BITMAPV4HEADER的最后三个字段为建立DIB的程序提供了一种为像素值指出假设的伽马值的方法。这些值由16位整数值和16位的小数值说明。例如，0x10000为1.0。如果DIB是捕捉实际影像而建立的，影像捕捉硬件就可能包含这个伽马值，并且可能是2.2（编码为0x23333）。如果DIB是由程序通过算法产生的，程序会使用一个函数将它使用的任何线性亮度转化为CIE亮度。

第5版的Header

为 Windows 98 和 Windows NT 5.0(即 Windows 2000)编写的程序能使用拥有新的BITMAPV5HEADER信息结构的DIB：

```
typedef struct
{
  DWORD  bV5Size ;  // size of the structure = 120
  LONG  bV5Width ;  // width of the image in pixels
  LONG  bV5Height ;  // height of the image in pixels
  WORD  bV5Planes ;  // = 1
  WORD  bV5BitCount ;  // bits per pixel (1,4,8,16,24,or32)
  DWORD  bV5Compression ; // compression code
  DWORD  bV5SizeImage ; // number of bytes in image
  LONG  bV5XPelsPerMeter ; // horizontal resolution
  LONG  bV5YPelsPerMeter ; // vertical resolution
  DWORD  bV5ClrUsed ;  // number of colors used
  DWORD  bV5ClrImportant ; // number of important colors
  DWORD  bV5RedMask ;  // Red color mask
  DWORD  bV5GreenMask ; // Green color mask
  DWORD  bV5BlueMask ;  // Blue color mask
  DWORD  bV5AlphaMask ; // Alpha mask
  DWORD  bV5CSType ;  // color space type
  CIEXYZTRIPLE bV5Endpoints ; // XYZ values
  DWORD  bV5GammaRed ;  // Red gamma value
  DWORD  bV5GammaGreen ; // Green gamma value
  DWORD  bV5GammaBlue ; // Blue gamma value
  DWORD  bV5Intent ;  // rendering intent
  DWORD  bV5ProfileData ; // profile data or filename
  DWORD  bV5ProfileSize ; // size of embedded data or filename
  DWORD  bV5Reserved ;
}
BITMAPV5HEADER, * PBITMAPV5HEADER ;
```

这里有四个新字段，只有其中三个有用。这些字段支持ICC Profile Format Specification，这是由「国际色彩协会（International Color Consortium）」(由Adobe、Agfa、Apple、Kodak、Microsoft、Silicon Graphics、Sun Microsystems及其它公司组成)建立的。您能在http://www.icc.org上取得这个标准的副本。基本上，每个输入（扫描仪和摄影机）、输出（打印机和胶片记录器）以及显示（显示器）设备与将原始设备相关颜色（一般为RGB或CMYK）联系到设备无关颜色规格的设定文件有关，最终依据CIE XYZ值来修正颜色。这些设定文件的扩展名是.ICM（指「图像颜色管理：image color management」）。设定文件能嵌入DIB文件中或从DIB文件连结以指出建立DIB的方式。您能在/Platform SDK/Graphics and Multimedia Services/Color Management中取得有关Windows「图像颜色管理」的详细信息。

BITMAPV5HEADER的bV5CSType字段能拥有几个不同的值。如果是LCS_CALIBRATED_RGB，那么它就与BITMAPV4HEADER结构兼容。bV5Endpoints字段和伽马字段必须有效。

如果bV5CSType字段是LCS_sRGB，就不用设定剩余的字段。预设的颜色空间是「标准」的RGB颜色空间，这是由Microsoft和Hewlett-Packard主要为Internet设计的，它包含设备无关的内容而不需要大量的设定文件。此文件位于http://www.color.org/contrib/sRGB.html。

如果bV5CSType字段是LCS_Windows_COLOR_SPACE，就不用设定剩余的字段。Windows通过API函数呼叫使用预设的颜色空间显示位图。

如果bV5CSType字段是PROFILE_EMBEDDED，则DIB文件包含一个ICC设定文件。如果字段是PROFILE_LINKED，DIB文件就包含了ICC设定文件的完整路径和文件名称。在这两种情况下，bV5ProfileData都是从BITMAPV5HEADER开始到设定文件数据或文件名称起始位置的偏移量。bV5ProfileSize字段给出了数据或文件名的大小。不必设定bV5Endpoints和伽马字段。

显示DIB信息

现在让我们来看一些程序代码。实际上我们并不未充分了解显示DIB的知识，但至少能表从头结构上显示有关DIB的信息。如程序15-1 DIBHEADS所示。

程序15-1 DIBHEADS

DIBHEADS.C

```
/*--------------------------------------------------------------------------
DIBHEADS.C -- Displays DIB Header Information
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("DibHeads") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("DIB Headers"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
```

```
  return msg.wParam ;
}

void Printf (HWND hwnd, TCHAR * szFormat, ...)
{
  TCHAR szBuffer [1024] ;
  va_list pArgList ;

  va_start (pArgList, szFormat) ;
  wvsprintf (szBuffer, szFormat, pArgList) ;
  va_end (pArgList) ;

  SendMessage (hwnd, EM_SETSEL, (WPARAM) -1, (LPARAM) -1) ;
  SendMessage (hwnd, EM_REPLACESEL, FALSE, (LPARAM) szBuffer) ;
  SendMessage (hwnd, EM_SCROLLCARET, 0, 0) ;
}
void DisplayDibHeaders (HWND hwnd, TCHAR * szFileName)
{
  static TCHAR * szInfoName []= { TEXT ("BITMAPCOREHEADER"),
    TEXT ("BITMAPINFOHEADER"),
    TEXT ("BITMAPV4HEADER"),
    TEXT ("BITMAPV5HEADER") } ;
  static TCHAR * szCompression []={TEXT ("BI_RGB"),
    TEXT ("BI_RLE8"),
    TEXT ("BI_RLE4"),
    TEXT ("BI_BITFIELDS"),
    TEXT ("unknown") } ;
  BITMAPCOREHEADER * pbmch ;
  BITMAPFILEHEADER * pbmfh ;
  BITMAPV5HEADER* pbmih ;
  BOOL bSuccess ;
  DWORD dwFileSize, dwHighSize, dwBytesRead ;
  HANDLE hFile ;
  int i ;
  PBYTE pFile ;
  TCHAR * szV ;

  // Display the file name

  Printf (hwnd, TEXT ("File: %s\r\n\r\n"), szFileName) ;
  // Open the file
  hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
  if (hFile == INVALID_HANDLE_VALUE)
  {
    Printf (hwnd, TEXT ("Cannot open file.\r\n\r\n")) ;
    return ;
  }

  // Get the size of the file
  dwFileSize = GetFileSize (hFile, &dwHighSize) ;
  if (dwHighSize)
  {
    Printf (hwnd, TEXT ("Cannot deal with >4G files.\r\n\r\n")) ;
    CloseHandle (hFile) ;
    return ;
  }
  // Allocate memory for the file
  pFile = malloc (dwFileSize) ;
  if (!pFile)
  {
    Printf (hwnd, TEXT ("Cannot allocate memory.\r\n\r\n")) ;
    CloseHandle (hFile) ;
    return ;
  }

  // Read the file
  SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
  ShowCursor (TRUE) ;
```

```
bSuccess = ReadFile (hFile, pFile, dwFileSize, &dwBytesRead, NULL) ;
ShowCursor (FALSE) ;
SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

if (!bSuccess || (dwBytesRead != dwFileSize))
{
  Printf (hwnd, TEXT ("Could not read file.\r\n\r\n")) ;
  CloseHandle (hFile) ;
  free (pFile) ;
  return ;
}

// Close the file
CloseHandle (hFile) ;
// Display file size
Printf (hwnd, TEXT ("File size = %u bytes\r\n\r\n"), dwFileSize) ;
// Display BITMAPFILEHEADER structure
pbmfh = (BITMAPFILEHEADER *) pFile ;
Printf (hwnd, TEXT ("BITMAPFILEHEADER\r\n")) ;
Printf (hwnd, TEXT ("\t.bfType = 0x%X\r\n"), pbmfh->bfType) ;
Printf (hwnd, TEXT ("\t.bfSize = %u\r\n"), pbmfh->bfSize) ;
Printf (hwnd, TEXT ("\t.bfReserved1 = %u\r\n"), pbmfh->bfReserved1) ;
Printf (hwnd, TEXT ("\t.bfReserved2 = %u\r\n"), pbmfh->bfReserved2) ;
Printf (hwnd, TEXT ("\t.bfOffBits = %u\r\n\r\n"), pbmfh->bfOffBits) ;

// Determine which information structure we have

pbmih = (BITMAPV5HEADER *) (pFile + sizeof (BITMAPFILEHEADER)) ;
switch (pbmih->bV5Size)
{
case sizeof (BITMAPCOREHEADER):i= 0 ; break ;
case sizeof (BITMAPINFOHEADER): i= 1 ; szV=
                            TEXT ("i") ; break ;
case sizeof (BITMAPV4HEADER):i= 2 ; szV=
                          TEXT ("V4") ; break ;
case sizeof (BITMAPV5HEADER):i= 3 ; szV=
                          TEXT ("V5") ; break ;
default:
  Printf (hwnd, TEXT ("Unknown header size of %u.\r\n\r\n"),
    pbmih->bV5Size) ;
  free (pFile) ;
  return ;
}

Printf (hwnd, TEXT ("%s\r\n"), szInfoName[i]) ;
// Display the BITMAPCOREHEADER fields
if (pbmih->bV5Size == sizeof (BITMAPCOREHEADER))
{
  pbmch = (BITMAPCOREHEADER *) pbmih ;
  Printf(hwnd,TEXT("\t.bcSize = %u\r\n"), pbmch->bcSize) ;
  Printf(hwnd,TEXT("\t.bcWidth = %u\r\n"), pbmch->bcWidth) ;
  Printf(hwnd,TEXT("\t.bcHeight = %u\r\n"), pbmch->bcHeight) ;
  Printf(hwnd,TEXT("\t.bcPlanes = %u\r\n"), pbmch->bcPlanes) ;
  Printf(hwnd,TEXT("\t.bcBitCount = %u\r\n\r\n"), pbmch->bcBitCount) ;
  free (pFile) ;
  return ;
}

// Display the BITMAPINFOHEADER fields
Printf(hwnd,TEXT("\t.b%sSize = %u\r\n"), szV, pbmih->bV5Size) ;
Printf(hwnd,TEXT("\t.b%sWidth = %i\r\n"), szV, pbmih->bV5Width) ;
Printf(hwnd,TEXT("\t.b%sHeight = %i\r\n"), szV, pbmih->bV5Height) ;
Printf(hwnd,TEXT("\t.b%sPlanes = %u\r\n"), szV, pbmih->bV5Planes) ;
Printf(hwnd,TEXT("\t.b%sBitCount=%u\r\n"),szV, pbmih->bV5BitCount) ;
Printf(hwnd,TEXT("\t.b%sCompression = %s\r\n"), szV,
  szCompression [min (4, pbmih->bV5Compression)]) ;
Printf(hwnd,TEXT("\t.b%sSizeImage= %u\r\n"),szV,
  pbmih->bV5SizeImage) ;
```

```
Printf(hwnd,TEXT ("\t.b%sXPelsPerMeter = %i\r\n"), szV,
  pbmih->bV5XPelsPerMeter) ;
Printf(hwnd,TEXT ("\t.b%sYPelsPerMeter = %i\r\n"), szV,
  pbmih->bV5YPelsPerMeter) ;
Printf (hwnd, TEXT ("\t.b%sClrUsed = %i\r\n"), szV,
  pbmih->bV5ClrUsed) ;
Printf (hwnd, TEXT ("\t.b%sClrImportant = %i\r\n\r\n"), szV,
  pbmih->bV5ClrImportant) ;

if (pbmih->bV5Size == sizeof (BITMAPINFOHEADER))
{
  if (pbmih->bV5Compression == BI_BITFIELDS)
  {
    Printf (hwnd,TEXT("Red Mask = %08X\r\n"), pbmih->bV5RedMask) ;
    Printf (hwnd,TEXT ("Green Mask = %08X\r\n"), pbmih->bV5GreenMask) ;
    Printf (hwnd,TEXT ("Blue Mask = %08X\r\n\r\n"), pbmih->bV5BlueMask) ;
  }
  free (pFile) ;
  return ;
}

// Display additional BITMAPV4HEADER fields
Printf (hwnd, TEXT ("\t.b%sRedMask = %08X\r\n"), szV,
  pbmih->bV5RedMask) ;
Printf (hwnd, TEXT ("\t.b%sGreenMask = %08X\r\n"), szV,
  pbmih->bV5GreenMask) ;
Printf (hwnd, TEXT ("\t.b%sBlueMask = %08X\r\n"), szV,
  pbmih->bV5BlueMask) ;
Printf (hwnd, TEXT ("\t.b%sAlphaMask = %08X\r\n"), szV,
  pbmih->bV5AlphaMask) ;
Printf (hwnd, TEXT ("\t.b%sCSType = %u\r\n"), szV,
  pbmih->bV5CSType) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzX = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzRed.ciexyzX) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzY = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzRed.ciexyzY) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzRed.ciexyzZ = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzRed.ciexyzZ) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzGreen.ciexyzX = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzGreen.ciexyzX) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzGreen.ciexyzY = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzGreen.ciexyzY) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzGreen.ciexyzZ = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzGreen.ciexyzZ) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzX = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzBlue.ciexyzX) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzY = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzBlue.ciexyzY) ;
Printf (hwnd, TEXT ("\t.b%sEndpoints.ciexyzBlue.ciexyzZ = %08X\r\n"),
  szV, pbmih->bV5Endpoints.ciexyzBlue.ciexyzZ) ;
Printf (hwnd, TEXT ("\t.b%sGammaRed = %08X\r\n"), szV,
  pbmih->bV5GammaRed) ;
Printf (hwnd, TEXT ("\t.b%sGammaGreen = %08X\r\n"), szV,
  pbmih->bV5GammaGreen) ;
Printf (hwnd, TEXT ("\t.b%sGammaBlue = %08X\r\n\r\n"), szV,
  pbmih->bV5GammaBlue) ;

if (pbmih->bV5Size == sizeof (BITMAPV4HEADER))
{
  free (pFile) ;
  return ;
}
// Display additional BITMAPV5HEADER fields
Printf (hwnd, TEXT ("\t.b%sIntent = %u\r\n"), szV, pbmih->bV5Intent) ;
Printf (hwnd, TEXT ("\t.b%sProfileData = %u\r\n"), szV,
  pbmih->bV5ProfileData) ;
Printf (hwnd, TEXT ("\t.b%sProfileSize = %u\r\n"), szV,
  pbmih->bV5ProfileSize) ;
Printf (hwnd, TEXT ("\t.b%sReserved = %u\r\n\r\n"), szV,
```

```
    pbmih->bV5Reserved) ;
  free (pFile) ;
  return ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndEdit ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[]= TEXT("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT("All Files (*.*)\0*.*\0\0") ;

  switch (message)
  {
  case WM_CREATE:
    hwndEdit = CreateWindow (TEXT ("edit"), NULL,
      WS_CHILD | WS_VISIBLE | WS_BORDER |
      WS_VSCROLL | WS_HSCROLL |
      ES_MULTILINE | ES_AUTOVSCROLL | ES_READONLY,
      0, 0, 0, 0, hwnd, (HMENU) 1,
      ((LPCREATESTRUCT) lParam)->hInstance, NULL) ;

    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("bmp") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;
    return 0 ;
  case WM_SIZE:
    MoveWindow (hwndEdit, 0, 0, LOWORD (lParam), HIWORD (lParam), TRUE) ;
    return 0 ;
  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:
      if (GetOpenFileName (&ofn))
        DisplayDibHeaders (hwndEdit, szFileName) ;

      return 0 ;
    }
    break ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIBHEADS.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
```

```
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Accelerator
DIBHEADS ACCELERATORS DISCARDABLE
BEGIN
"O",   IDM_FILE_OPEN,    VIRTKEY, CONTROL, NOINVERT
END
//////////////////////////////////////////////////////////////////////////////
// Menu
DIBHEADS MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open\tCtrl+O",  IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DibHeads.rc
#define IDM_FILE_OPEN    40001
```

此程序有一个简短的WndProc函数，它建立了一个只读的编辑窗口来填满它的显示区域，它也处理菜单上的「File Open」命令。它通过呼叫GetOpenFileName函数使用标准的「File Open」对话框，然后呼叫DisplayDibHeaders函数。此函数把整个DIB文件读入内存并逐栏地显示所有的表头信息。

显示和打印

位图是用来看的。在这一节中，我们看一看Windows在视讯显示器上或打印页面上支持显示DIB的两个函数。要得到更好的性能，您可以使用一种兜圈子的方法来显示位图，我会在本章的后面讨论该方法，但先研究这两个函数会好一些。

这两个函数称为SetDIBitsToDevice（发音为「set dee eye bits to device」）和StretchDIBits（发音为「stretch dee eye bits」）。每个函数都使用储存在内存中的DIB并能显示整个DIB或它的矩形部分。当使用SetDIBitsToDevice时，以像素为单位所显示映像的大小与DIB的像素大小相同。例如，一个640×480的DIB会占据整个标准的VGA屏幕，但在300dpi的激光打印机上它只有约2.1×1.6英寸。StretchDIBits能延伸和缩小DIB尺寸的行和列从而在输出设备上显示一个特定的大小。

了解DIB

当呼叫两个函数之一来显示DIB时，您需要几个关于图像的信息。正如我前面说过的，DIB文件包含下列部分：

DIB文件能被加载内存。如果除了文件表头外，整个文件被储存在内存的连续区块中，指向该内存块开始处（也就是信息表头的开头）的指标被称为指向packed DIB的指标（见下图）。

这是通过剪贴簿传输DIB时所用的格式，并且也是您从DIB建立画刷时所用的格式。因为整个DIB由单个指标（如pPackedDib）引用，所以packed DIB是在内存中储存DIB的方便方法，您可以把指标定义为指向BYTE的指标。使用本章前面所示的结构定义，能得到所有储存在DIB内的信息，包括色彩对照表和个别像素位。

然而，要想得到这么多信息，还需要一些程序代码。例如，您不能通过以下叙述简单地取得DIB的像素宽度：

iWidth = ((PBITMAPINFOHEADER) pPackedDib)->biWidth ;

DIB有可能是OS/2兼容格式的。在那种格式中，packed DIB以BITMAPCOREHEADER结构开始，并且DIB的像素宽度和高度以16位WORD，而不是32位LONG储存。因此，首先必须检查DIB是否为旧的格式，然后进行相对应的操作：

```
if (((PBITMAPCOREHEADER) pPackedDib)->bcSize == sizeof (BITMAPCOREHEADER))
  iWidth = ((PBITMAPCOREHEADER) pPackedDib)->bcWidth ;
else
  iWidth = ((PBITMAPINFOHEADER) pPackedDib)->biWidth ;
```

当然，这不很糟，但它不如我们所喜好的清晰。

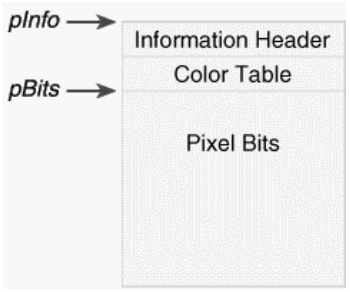
现在有一个很有趣的实验：给定一个指向packed DIB的指标，我们要找出位于坐标（5,27）的像素值。即使假定DIB不是OS/2兼容的格式，您也需要了解DIB的宽度、高度和位数。您需要计算每一列像素的位组长度，确定色彩对照表内的项目数，以及色彩对照表是否包括三个32位的颜色屏蔽。您还需检查DIB是否被压缩，在这种情况下像素是不能直接由地址得到的。

如果您需要直接存取所有的DIB像素（就像许多图形处理工作一样），这可能会增加一点处理时间。由于这个原因，储存一个指向packed DIB的指标就很方便了，不过这并不是一种有效率的解决方式。另一个漂亮的解决方法是为DIB定义一个包含足够成员数据的C++类别，从而允许快速随机地存取DIB像素。然而，我曾经答应读者在本书内无需了解C++，我将在下一章说明一个C的解决方法。

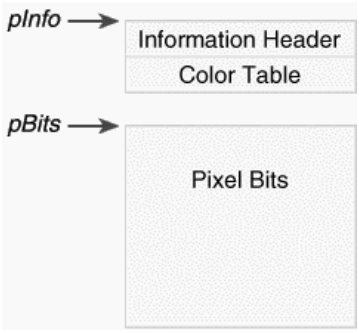
对于SetDIBitsToDevice和StretchDIBits函数，需要的信息包括一个指向DIB的BITMAPINFO结构的指针。您应回想起，BITMAPINFO结构由BITMAPINFOHEADER结构和色彩对照表组成。因此这仅是一个指向packed DIB的指标。

函数也需要一个指向像素位的指针。尽管程序代码写得很不漂亮，但这个指针还是可以从信息表头内的信息推出。注意，当您存取BITMAPFILEHEADER结构的bfOffBits字段时，这个指标能很容易地计算出。bfOffBits字段指出了从DIB文件的开头到像素位的偏移量。您可以简单地把此偏移量加到BITMAPINFO指标中，然后减去BITMAPFILEHEADER结构的大小。然而，当您从剪贴簿上得到指向packed DIB的指标时，这并不起作用，因为没有BITMAPFILEHEADER结构。

此图表显示了两个所需的指标：



SetDIBitsToDevice和StretchDIBits函数需要两个指向DIB的指标，因为这两个部分不在连续的内存块内。您可能有如下所示的两块内存：



确实，把DIB分成两个内存块是很有用的，只是我们更喜欢与整个DIB储存在单个内存块的packed DIB打交道。

除了这两个指标，SetDIBitsToDevice和StretchDIBits函数通常也需要DIB的像素宽度和高度。如只想显示DIB的一部分，就不必明确地知道这些值，但它们会定义您在DIB像素位数组内定义的矩形的上限。

点对点像素显示

SetDIBitsToDevice函数显示没有延伸和缩小的DIB。DIB的每个像素对应到输出设备的一个像素上，而且DIB中的图像一定会被正确显示出来 – 也就是说，图像的顶列在上方。任何会影响设备内容的坐标转换都影响了显示DIB的开始位置，但不影响显示出来的图片大小和方向。该函数如下：

```
iLines = SetDIBitsToDevice (
                        hdc, // device context handle
                        xDst, // x destination coordinate
                        yDst, // y destination coordinate
                        cxSrc, // source rectangle width
                        cySrc, // source rectangle height
                        xSrc, // x source coordinate
                        ySrc, // y source coordinate
                        yScan, // first scan line to draw
                        cyScans, // number of scan lines to draw
                        pBits, // pointer to DIB pixel bits
                        pInfo, // pointer to DIB information
                        fClrUse) ; // color use flag
```

不要对参数的数量感到厌烦，在多数情况下，函数用起来比看起来要简单。不过在其它用途上来说，它的用法真的是乱七八糟，不过我们将学会怎么用它。

和GDI显示函数一样，SetDIBitsToDevice的第一个参数是设备内容句柄，它指出显示DIB的设备。下面两个参数xDst和yDst，是输出设备的逻辑坐标，并指出了显示DIB图像左上角的坐标（「上端」指的是视觉上的上方，并不是DIB像素的第一行）。注意，这些都是逻辑坐标，因此它们附属于实际上起作用的任何坐标转换方式或 – 在Windows NT的情况下 – 设定的任何空间转换。在内定的

MM_TEXT映像方式下，可以把这些参数设为0，从显示平面上向左向上显示DIB图像。

您可以显示整个DIB图像或仅显示其中的一部分，这就是后四个参数的作用。但是DIB像素数据的由上而下的方向产生了许多误解，待会儿会谈到这些。现在应该清楚当显示整个DIB时，应把xSrc和ySrc设定为0，并且cxSrc和cySrc应分别等于DIB的像素宽度和高度。注意，因为BITMAPINFOHEADER结构的biHeight字段对于由上而下的DIB来说是负的，cySrc应设定为biHeight字段的绝对值。

此函数的文件（/Platform SDK/Graphics and Multimedia Services/GDI/Bitmaps/Bitmap Reference/Bitmap Functions/SetDIBitsToDevice）中说xSrc、ySrc、cxSrc和cySrc参数是逻辑单位。这是不正确的，它们是像素的坐标和尺寸。对于DIB内的像素，拥有逻辑坐标和单位是没有什么意义的。而且，不管是什么映像方式，在输出设备上显示的DIB始终是cxSrc像素宽和cySrc像素高。

现在先不详细讨论这两个参数yScan和cyScan。这些参数在您从磁盘文件或通过调制解调器读取数据时，透过每次显示DIB的一小部分减少对内存的需求。通常，yScan设定为0，cyScan设定为DIB的高度。

pBits参数是指向DIB像素位的指针。pInfo参数是指向DIB的BITMAPINFO结构的指针。虽然BITMAPINFO结构的地址与BITMAPINFOHEADER结构的地址相同，但是SetDIBitsToDevice结构被定义为使用BITMAPINFO结构，暗示着：对于1位、4位和8位DIB，位图信息表头后必须跟着色彩对照表。尽管pInfo参数被定义为指向BITMAPINFO结构的指针，它也是指向BITMAPCOREINFO、BITMAPV4HEADER或BITMAPV5HEADER结构的指针。

最后一个参数是DIB_RGB_COLORS或DIB_PAL_COLORS，在WINGDI.H内分别定义为0和1。如果您使用DIB_RGB_COLORS，这意味着DIB包含了色彩对照表。DIB_PAL_COLORS旗标指出，DIB内的色彩对照表已经被指向在设备内容内选定并识别的调色盘的16位索引代替。在下一章我们将学习这个选项。现在先使用DIB_RGB_COLORS，或者是0。

SetDIBitsToDevice函数传回所显示的扫描行的数目。

因此，要呼叫SetDIBitsToDevice来显示整个DIB图像，您需要下列信息：

hdc目的表面的设备内容句柄

xDst和yDst图像左上角的目的坐标

cxDib和cyDibDIB的像素宽度和高度，在这里，cyDib是BITMAPINFOHEADER结构内biHeight字段的绝对值。

pInfo和pBits指向位图信息部分和像素位的指针

然后用下列方法呼叫SetDIBitsToDevice：

```
SetDIBitsToDevice (
                hdc, // device context handle
                xDst, // x destination coordinate
                yDst, // y destination coordinate
                cxDib, // source rectangle width
                cyDib, // source rectangle height
                0, // x source coordinate
                0, // y source coordinate
                0, // first scan line to draw
                cyDib, // number of scan lines to draw
                pBits, // pointer to DIB pixel bits
                pInfo, // pointer to DIB information
                0) ; // color use flag
```

因此，在DIB的12个参数中，四个设定为0，一个是重复的。

程序15-2 SHOWDIB1通过使用SetDIBitsToDevice函数显示DIB。

程序15-2 SHOWDIB1

SHOWDIB1.C

```
/*------------------------------------------------------------------------
SHOWDIB1.C -- Shows a DIB in the client area
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "dibfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib1") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #1"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAPFILEHEADER * pbmfh ;
  static BITMAPINFO * pbmi ;
  static BYTE * pBits ;
  static int cxClient, cyClient, cxDib, cyDib ;
```

```
static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
BOOL bSuccess ;
HDC hdc ;
PAINTSTRUCT ps ;

switch (message)
{
case WM_CREATE:
  DibFileInitialize (hwnd) ;
  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;

case WM_INITMENUPOPUP:
  EnableMenuItem ((HMENU) wParam, IDM_FILE_SAVE,
    pbmfh ? MF_ENABLED : MF_GRAYED) ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_FILE_OPEN:
    // Show the File Open dialog box

    if (!DibFileOpenDlg (hwnd, szFileName, szTitleName))
      return 0 ;

    // If there's an existing DIB, free the memory

    if (pbmfh)
    {
      free (pbmfh) ;
      pbmfh = NULL ;
    }
    // Load the entire DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pbmfh = DibLoadImage (szFileName) ;
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    // Invalidate the client area for later update

    InvalidateRect (hwnd, NULL, TRUE) ;

    if (pbmfh == NULL)
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
      return 0 ;
    }
    // Get pointers to the info structure & the bits

    pbmi = (BITMAPINFO *) (pbmfh + 1) ;
    pBits = (BYTE *) pbmfh + pbmfh->bfOffBits ;

    // Get the DIB width and height

    if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    {
      cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
      cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
    }
    else
```

```
     {
       cxDib = pbmi->bmiHeader.biWidth ;
       cyDib = abs(pbmi->bmiHeader.biHeight) ;
     }
     return 0 ;

   case IDM_FILE_SAVE:
     // Show the File Save dialog box

     if (!DibFileSaveDlg (hwnd, szFileName, szTitleName))
       return 0 ;

     // Save the DIB to memory

     SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
     ShowCursor (TRUE) ;

     bSuccess = DibSaveImage (szFileName, pbmfh) ;
     ShowCursor (FALSE) ;
     SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

     if (!bSuccess)
       MessageBox ( hwnd, TEXT ("Cannot save DIB file"),
       szAppName, 0) ;
     return 0 ;
   }
   break ;

   case WM_PAINT:
     hdc = BeginPaint (hwnd, &ps) ;

     if (pbmfh)
       SetDIBitsToDevice (hdc,
       0, // xDst
       0, // yDst
       cxDib, // cxSrc
       cyDib, // cySrc
       0, // xSrc
       0, // ySrc
       0, // first scan line
       cyDib, // number of scan lines
       pBits,
       pbmi,
       DIB_RGB_COLORS) ;
     EndPaint (hwnd, &ps) ;
     return 0 ;

   case WM_DESTROY:
     if (pbmfh)
       free (pbmfh) ;

     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIBFILE.H

```
/*----------------------------------------------------------------------------
DIBFILE.H -- Header File for DIBFILE.C
----------------------------------------------------------------------------*/
void DibFileInitialize (HWND hwnd) ;
BOOL DibFileOpenDlg(HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName) ;
BOOL DibFileSaveDlg(HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName) ;
BITMAPFILEHEADER * DibLoadImage (PTSTR pstrFileName) ;
BOOL DibSaveImage(PTSTR pstrFileName, BITMAPFILEHEADER *) ;
```

DIBFILE.C

```
/*----------------------------------------------------------------------------
DIBFILE.C -- DIB File Functions
----------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "dibfile.h"

static OPENFILENAME ofn ;
void DibFileInitialize (HWND hwnd)
{
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0") \
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  ofn.lStructSize = sizeof (OPENFILENAME) ;
  ofn.hwndOwner = hwnd ;
  ofn.hInstance = NULL ;
  ofn.lpstrFilter = szFilter ;
  ofn.lpstrCustomFilter = NULL ;
  ofn.nMaxCustFilter = 0 ;
  ofn.nFilterIndex = 0 ;
  ofn.lpstrFile = NULL ; // Set in Open and Close functions
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = NULL ; // Set in Open and Close functions
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.lpstrInitialDir = NULL ;
  ofn.lpstrTitle = NULL ;
  ofn.Flags = 0 ; // Set in Open and Close functions
  ofn.nFileOffset = 0 ;
  ofn.nFileExtension = 0 ;
  ofn.lpstrDefExt = TEXT ("bmp") ;
  ofn.lCustData = 0 ;
  ofn.lpfnHook = NULL ;
  ofn.lpTemplateName = NULL ;
}
BOOL DibFileOpenDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
  ofn.hwndOwner = hwnd ;
  ofn.lpstrFile = pstrFileName ;
  ofn.lpstrFileTitle = pstrTitleName ;
  ofn.Flags = 0 ;

  return GetOpenFileName (&ofn) ;
}

BOOL DibFileSaveDlg (HWND hwnd, PTSTR pstrFileName, PTSTR pstrTitleName)
{
  ofn.hwndOwner = hwnd ;
  ofn.lpstrFile = pstrFileName ;
  ofn.lpstrFileTitle = pstrTitleName ;
  ofn.Flags = OFN_OVERWRITEPROMPT ;

  return GetSaveFileName (&ofn) ;
}

BITMAPFILEHEADER * DibLoadImage (PTSTR pstrFileName)
{
  BOOL bSuccess ;
  DWORD dwFileSize, dwHighSize, dwBytesRead ;
  HANDLE hFile ;
  BITMAPFILEHEADER * pbmfh ;

  hFile = CreateFile ( pstrFileName, GENERIC_READ, FILE_SHARE_READ,NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN,NULL) ;

  if ( hFile == INVALID_HANDLE_VALUE)
    return NULL ;
```

```
  dwFileSize = GetFileSize (hFile, &dwHighSize) ;

  if (dwHighSize)
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  pbmfh = malloc (dwFileSize) ;
  if (!pbmfh)
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  bSuccess = ReadFile (hFile, pbmfh, dwFileSize, &dwBytesRead, NULL) ;
  CloseHandle (hFile) ;

  if (!bSuccess || (dwBytesRead != dwFileSize)
    || (pbmfh->bfType != * (WORD *) "BM")
    || (pbmfh->bfSize != dwFileSize))
  {
    free (pbmfh) ;
    return NULL ;
  }
  return pbmfh ;
}

BOOL DibSaveImage (PTSTR pstrFileName, BITMAPFILEHEADER * pbmfh)
{
  BOOL bSuccess ;
  DWORD dwBytesWritten ;
  HANDLE hFile ;

  hFile = CreateFile ( pstrFileName, GENERIC_WRITE, 0, NULL,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;

  if (hFile == INVALID_HANDLE_VALUE)
    return FALSE ;
  bSuccess = WriteFile (hFile, pbmfh, pbmfh->bfSize, &dwBytesWritten, NULL) ;
  CloseHandle (hFile) ;

  if (!bSuccess || (dwBytesWritten != pbmfh->bfSize))
  {
    DeleteFile (pstrFileName) ;
    return FALSE ;
  }
  return TRUE ;
}
```

SHOWDIB1.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB1 MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open...", IDM_FILE_OPEN
MENUITEM "&Save...", IDM_FILE_SAVE
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib1.rc
#define IDM_FILE_OPEN   40001
#define IDM_FILE_SAVE  40002
```

DIBFILE.C文件包含了显示「File Open」和「File Save」对话框的例程，以及把整个DIB文件（拥有BITMAPFILEHEADER结构）加载单个内存块的例程。程序也会将这样一个内存区写出到文件。

当在SHOWDIB1.C内执行「File Open」命令加载DIB文件后，程序计算内存块中BITMAPINFOHEADER结构和像素位的偏移量，程序也获得DIB的像素宽度和高度。所有信息都储存在静态变量中。在处理WM_PAINT消息处理期间，程序通过呼叫SetDIBitsToDevice显示DIB。

当然，SHOWDIB1还缺少一些功能。例如，如果DIB对显示区域来说太大，则没有滚动条可用来移动查看。在下一章的末尾将修改这些缺陷。

DIB的颠倒世界

我们将得到一个重要的教训，它不仅在生活中重要，而且在操作系统的应用程序接口的设计中也重要。这个教训是：覆水难收。

回到OS/2 Presentation Manager那由下而上的DIB像素位的定义处，这样的定义是有点道理的，因为PM内的任何坐标系都有一个内定的左下角原点。例如：在PM窗口内，内定的（0,0）原点是窗口的左下角。（如果您觉得这很古怪，很多人和您的感觉一样。如果您不觉得古怪，那您可能是位数学家。）位图的绘制函数也根据左下角指定目的地。

因此，在OS/2内如果给位图指定了目的坐标(0,0)，则图像将从窗口的左下角向上向右显示，如图15-1所示。

图15-1 在OS/2中以（0，0）为目的点显示的位图

在够慢的机器上，您能实际看到计算机由下而上绘制位图。

尽管OS/2坐标系统显得很古怪，但它的优点是高度的一致。位图的 (0,0)原点是位图文件中第一行的第一个像素，并且此像素被映像到在位图绘制函数中指定的目的坐标上。

Windows存在的问题是不能保持内部的一致性。当您只要显示整个DIB图像中的一小块矩形时，就要使用参数xSrc、ySrc、cxSrc和cySrc。这些来源坐标和大小与DIB数据的第一行（图像的最后一行）相关。这方面与OS/2相似，与OS/2不同的是，Windows在目的坐标上显示图像的顶列。因此，如果显示整个DIB图像，显示在（xDst,yDst）的像素是位于坐标（0,cyDib - 1）处的像素。DIB数据的最后一列就是图形的顶列。如果仅显示图像的一部分，则在（xDst,yDst）显示的像素是位于坐标（xSrc, ySrc + cySrc - 1）处的DIB像素。

图15-2显示的图表将帮助您理解这方面的内容。您可以把下面显示的DIB当成是储存在内存中的－就是说，上下颠倒。坐标的原点与DIB像素数据的第一个位是一致的。SetDIBitsToDevice的xSrc参数是以DIB的左边为基准，并且cxSrc是xSrc右边的图像宽度，这很直观。ySrc参数以DIB数据的首列（也就是图像的底部）为基准，并且cySrc是从ySrc到数据的末列（图像的顶端）的图像高度。

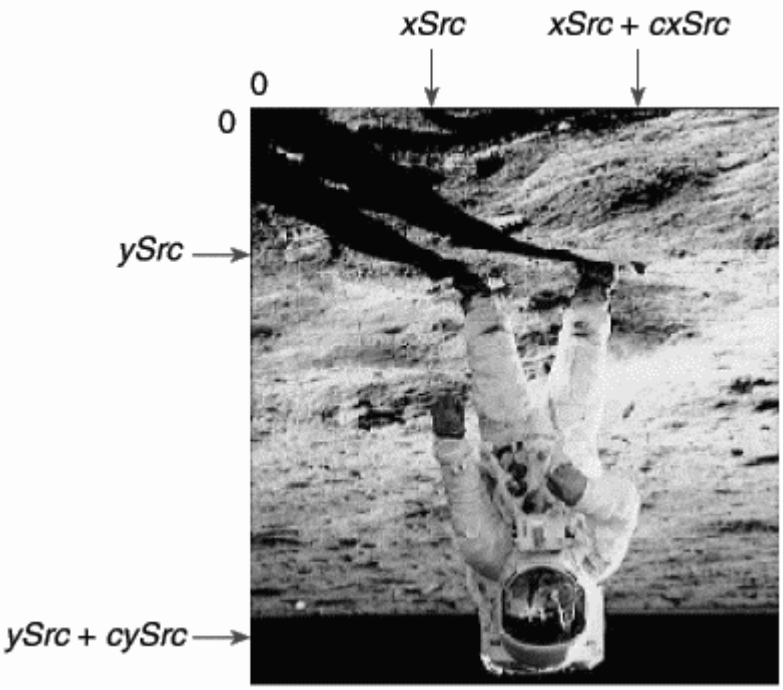


图15-2 正常DIB（由下而上）的坐标

如果目的设备内容具有使用MM_TEXT映像方式的内定像素坐标，来源矩形和目的矩形角落坐标之间的关系显示在表15-3中。

表15-3

来源矩形	目的矩形
(xSrc, ySrc)	(xDst, yDst + cySrc - 1)
(xSrc + cxSrc - 1, ySrc)	(xDst + cxSrc - 1, yDst + cySrc - 1)
(xSrc, ySrc + cySrc - 1)	(xDst, yDst)
(xSrc + cxSrc - 1, ySrc + cySrc - 1)	(xDst + cxSrc - 1, yDst)

(xSrc,ySrc)不映射到(xDst,yDst)，使得表格显得很混乱。在其它映像方式中，点(xSrc,ySrc + cySrc - 1)总是映像到逻辑点(xDst,yDst)，图像也与MM_TEXT所显示的一样。

到目前为止，我们讨论了当BITMAPINFOHEADER结构的biHeight字段是正值时的正常情况。如果biHeight字段是负值，则DIB数据会以合理的由上而下的方式排列。您可能会认为这样将解决

所有问题，如果您真地这样认为，那您就错了。

很明显地，有人会认为如果把DIB上下倒置，旋转每一行，然后给biHeight设定一个正值，它将像正常的由下而上的DIB一样操作，所有与DIB矩形相关的现存程序代码就不必修改。我认为这是一个合理的目的，但它忘记了一个事实，程序需要修改以处理由上而下的DIB，这样就不会使用一个负高度。

而且，此决定的结果意味着由上而下的DIB的来源坐标在DIB数据的最后一列有一个原点，它也是图像的底列。这与我们遇到的情况完全不同。位于(0,0)原点的DIB像素不再是pBits指标引用的第一个像素，也不是DIB文件的最后一个像素，它位于两者之间。

图15-3显示的图表说明了在由上而下的DIB中指定矩形的方法，也是它储存在文件或内存中的样子。

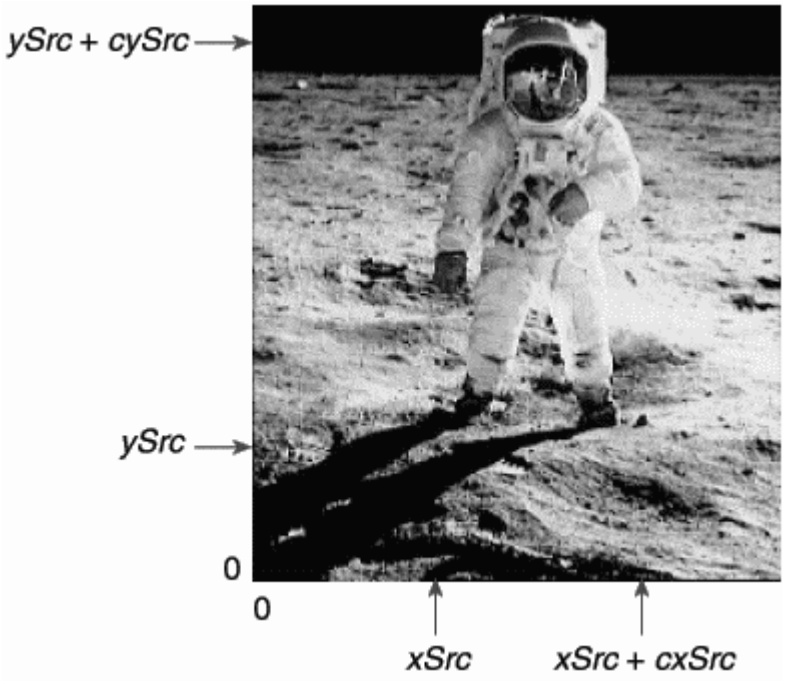


图15-3 指定由上而下的DIB的坐标

无论如何，这个方案的实际优点是SetDIBitsToDevice函数的参数与DIB数据的方向无关。如果有显示了同一图像的两个DIB（一个由下而上，另一个由上而下。表示在两个DIB文件内的列顺序相反），您可以使用相同的参数呼叫SetDIBitsToDevice来选择显示图像的相同部分。

如程序15-3 APOLLO11中所示。

程序15-3 APOLLO11

APOLLO11.C

```
/*---------------------------------------------------------------------------
APOLLO11.C -- Program for screen captures
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "dibfile.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Apollo11") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
```

```
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLAS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow (szAppName, TEXT ("Apollo 11"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAPFILEHEADER * pbmfh [2] ;
  static BITMAPINFO * pbmi [2] ;
  static BYTE * pBits [2] ;
  static int cxClient, cyClient, cxDib[2], cyDib[2] ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    pbmfh[0] = DibLoadImage (TEXT ("Apollo11.bmp")) ;
    pbmfh[1] = DibLoadImage (TEXT ("ApolloTD.bmp")) ;

    if (pbmfh[0] == NULL || pbmfh[1] == NULL)
    {
      MessageBox (hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
      return 0 ;
    }
    // Get pointers to the info structure & the bits

    pbmi [0] = (BITMAPINFO *) (pbmfh[0] + 1) ;
    pbmi [1] = (BITMAPINFO *) (pbmfh[1] + 1) ;
    pBits [0] = (BYTE *) pbmfh[0] + pbmfh[0]->bfOffBits ;
    pBits [1] = (BYTE *) pbmfh[1] + pbmfh[1]->bfOffBits ;

    // Get the DIB width and height (assume BITMAPINFOHEADER)
```

```
  // Note that cyDib is the absolute value of the header value!!!

  cxDib [0] = pbmi[0]->bmiHeader.biWidth ;
  cxDib [1] = pbmi[1]->bmiHeader.biWidth ;

  cyDib [0] = abs (pbmi[0]->bmiHeader.biHeight) ;
  cyDib [1] = abs (pbmi[1]->bmiHeader.biHeight) ;
  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;

case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;

  // Bottom-up DIB full size

  SetDIBitsToDevice (hdc,
    0, // xDst
    cyClient / 4, // yDst
    cxDib[0], // cxSrc
    cyDib[0], // cySrc
    0, // xSrc
    0, // ySrc
    0, // first scan line
    cyDib[0], // number of scan lines
    pBits[0],
    pbmi[0],
    DIB_RGB_COLORS) ;

  // Bottom-up DIB partial

  SetDIBitsToDevice (hdc,
    240, // xDst
    cyClient / 4, // yDst
    80, // cxSrc
    166, // cySrc
    80, // xSrc
    60, // ySrc
    0, // first scan line
    cyDib[0], // number of scan lines
    pBits[0],
    pbmi[0],
    DIB_RGB_COLORS) ;

  // Top-down DIB full size

  SetDIBitsToDevice (hdc,
    340, // xDst
    cyClient / 4, // yDst
    cxDib[0], // cxSrc
    cyDib[0], // cySrc
    0, // xSrc
    0, // ySrc
    0, // first scan line
    cyDib[0], // number of scan lines
    pBits[0],
    pbmi[0],
    DIB_RGB_COLORS) ;

  // Top-down DIB partial

  SetDIBitsToDevice (hdc,
    580, // xDst
    cyClient / 4, // yDst
    80, // cxSrc
    166, // cySrc
```

```
     80, // xSrc
     60, // ySrc
     0, // first scan line
     cyDib[1], // number of scan lines
     pBits[1],
     pbmi[1],
     DIB_RGB_COLORS) ;

   EndPaint (hwnd, &ps) ;
   return 0 ;

 case WM_DESTROY:
   if (pbmfh[0])
     free (pbmfh[0]) ;
   if (pbmfh[1])
     free (pbmfh[1]) ;

   PostQuitMessage (0) ;
   return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

程序加载了名为APOLLO11.BMP（由下而上版本）和APOLLOTD.BMP（由上而下版本）的两个DIB。它们都是220像素宽和240像素高。注意，在程序从表头信息结构中确定DIB的宽度和高度时，它使用abs函数得到biHeight字段的绝对值。当以全部大小或范围显示DIB时，不管显示位图的种类，xSrc、ySrc、cxSrc和cySrc坐标都是相同的。结果如图15-4所示。

图15-4 APOLLO11的屏幕显示

注意，「第一条扫描线」和「扫描线数目」参数保持不变，我将在以后简短说明。pBits参数也不变，不要只为了使它指向您需要显示的区域而试图更改pBits。

我在这个问题上花了这么多时间，并不是因为要让那些试图跟API定义中有问题的部分妥协的Windows程序写作者难堪，而是想让您不至于因为这个令人混淆的问题而紧张起来。这个问题之所以令人困惑，是因为它本身早就被搞混了。

我也想让您留意Windows文件中的某些叙述，例如对SetDIBitsToDevice，文件说：「由下而上DIB的原点是位图的左下角；由上而下DIB的原点是左上角」。这不仅模糊，而且是错误的。我可以用更好的方式来讲述：由下而上DIB的原点是位图图像的左下角，它是位图资料的第一列的第一个

像素。由上而下DIB的原点也是位图图像的左下角，但在这种情况下，左下角是位图数据的最后一列的第一个像素。

如果要撰写存取DIB个别位的函数，问题会变的更糟。这应该与您为显示部分DIB映像而指定的坐标一致，我的解决方法是（我将在第十六章的DIB链接库中使用）以统一的手法参考DIB像素和坐标，就像在图像被正确显示时（0,0）原点所指的是DIB图像顶行的最左边的像素一样。

循序显示

拥有海量存储器能确保程序更容易地执行。要显示磁盘文件内的DIB，可以分为两个独立的工作：将DIB加载内存，然后显示它。

然而，您也可能在不把整个文件加载内存的情况下显示DIB。即使有足够的物理内存提供给DIB，把DIB移入内存也会迫使Windows的虚拟内存系统把内存中别的数据和程序代码移到磁盘上。如果DIB仅用于显示并立即从内存中消除，这就非常讨厌。

还有另一个问题：假设DIB位于例如软盘的慢速储存媒体上，或由调制解调器传输过来，或者来自扫描仪或视频截取程序取得像素数据的转换例程。您是否得等到整个DIB被加载内存后才显示它？还是从磁盘或电话线或扫描仪上得到DIB时，就开始显示它？

解决这些问题是SetDIBitsToDevice函数中yScan和cyScans参数的目的。要使用这个功能，需要多次呼叫SetDIBitsToDevice，大多数情况下使用同样的参数。然而对于每次呼叫，pBits参数指向位图像素总体排列的不同部分。yScans参数指出了pBits指向像素资料的行，cyScans参数是被pBits引用的行数。这大量地减少了内存需求。您仅需要为储存DIB的信息部分（BITMAPINFOHEADER结构和色彩对照表）和至少一行像素数据配置足够的内存。

例如，假设DIB有23行像素，您希望每次最多5行的分段显示这个DIB。您可能想配置一个由变量pInfo引用的内存块来储存DIB的BITMAPINFO部分，然后从文件中读取该DIB。在检查完此结构的字段后，能够计算出一行的位组长度。乘以5并配置该大小的另一个内存块（pBits）。现在读取前5行，呼叫您正常使用的函数，把yScan设定为0，把cyScans设定为5。现在从文件中读取下5行，这一次将yScan设定为5，继续将yScan设定为10，然后为15。最后，将最后3行读入pBits指向的内存块，并将yScan设定为20，将cyScans设定为3，以呼叫SetDIBitsToDevice。

现在有一个不好的消息。首先，使用SetDIBitsToDevice的这个功能要求程序的数据取得和数据显示元素之间结合得相当紧密。这通常是不理想的，因为您必须在获得数据和显示数据之间切换。首先，您将延缓整个程序；第二，SetDIBitsToDevice是唯一具有这个功能的位图显示函数。StretchDIBits函数不包括这个功能，因此您不能使用它以不同像素大小显示发表的DIB。您必须呼叫StretchDIBits多次，每次更改BITMAPINFOHEADER结构中的信息，并在屏幕的不同区域显示结果。

程序15-4 SEQDISP展示了这个功能的使用方法。

程序15-4 SEQDISP

SEQDISP.C

```
/*---------------------------------------------------------------------------
SEQDISP.C -- Sequential Display of DIBs
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("SeqDisp") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
```

```
                PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("DIB Sequential Display"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BITMAPINFO * pbmi ;
  static BYTE * pBits ;
  static int cxDib, cyDib, cBits ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[]= TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  BITMAPFILEHEADER bmfh ;
  BOOL bSuccess, bTopDown ;
  DWORD dwBytesRead ;
  HANDLE hFile ;
  HDC hdc ;
  HMENU hMenu ;
  int iInfoSize, iBitsSize, iRowLength, y ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
```

```
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("bmp") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;
    return 0 ;

  case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:
      // Display File Open dialog
      if (!GetOpenFileName (&ofn))
        return 0 ;

      // Get rid of old DIB

      if (pbmi)
      {
        free (pbmi) ;
        pbmi = NULL ;
      }

      if (pBits)
      {
        free (pBits) ;
        pBits = NULL ;
      }

      // Generate WM_PAINT message to erase background

      InvalidateRect (hwnd, NULL, TRUE) ;
      UpdateWindow (hwnd) ;

      // Open the file

      hFile = CreateFile (szFileName, GENERIC_READ,
        FILE_SHARE_READ, NULL, OPEN_EXISTING,
        FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;

      if (hFile == INVALID_HANDLE_VALUE)
      {
        MessageBox ( hwnd, TEXT ("Cannot open file."),
          szAppName, MB_ICONWARNING | MB_OK) ;
        return 0 ;
      }

      // Read in the BITMAPFILEHEADER

      bSuccess = ReadFile (hFile,&bmfh, sizeof (BITMAPFILEHEADER),
        &dwBytesRead, NULL) ;

      if (!bSuccess || dwBytesRead != sizeof (BITMAPFILEHEADER))
      {
```

```
      MessageBox (hwnd, TEXT ("Cannot read file."),
        szAppName, MB_ICONWARNING | MB_OK) ;
      CloseHandle (hFile) ;
      return 0 ;
    }

    // Check that it's a bitmap

    if (bmfh.bfType != * (WORD *) "BM")
    {
      MessageBox (hwnd, TEXT ("File is not a bitmap."),
        szAppName, MB_ICONWARNING | MB_OK) ;
      CloseHandle (hFile) ;
      return 0 ;
    }

    // Allocate memory for header and bits

    iInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
    iBitsSize = bmfh.bfSize - bmfh.bfOffBits ;

    pbmi = malloc (iInfoSize) ;
    pBits = malloc (iBitsSize) ;

    if (pbmi == NULL || pBits == NULL)
    {
      MessageBox (hwnd, TEXT ("Cannot allocate memory."),
        szAppName, MB_ICONWARNING | MB_OK) ;
      if (pbmi)
        free (pbmi) ;
      if (pBits)
        free (pBits) ;
      CloseHandle (hFile) ;
      return 0 ;
    }

    // Read in the Information Header

    bSuccess = ReadFile (hFile, pbmi, iInfoSize, &dwBytesRead, NULL) ;

    if (!bSuccess || (int) dwBytesRead != iInfoSize)
    {
      MessageBox (hwnd, TEXT ("Cannot read file."),
        szAppName, MB_ICONWARNING | MB_OK) ;
      if (pbmi)
        free (pbmi) ;
      if (pBits)
        free (pBits) ;
      CloseHandle (hFile) ;
      return 0 ;
    }

    // Get the DIB width and height

    bTopDown = FALSE ;

    if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    {
      cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
      cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
      cBits = ((BITMAPCOREHEADER *) pbmi)->bcBitCount ;
    }
    else
    {
      if (pbmi->bmiHeader.biHeight < 0)
        bTopDown = TRUE ;

      cxDib = pbmi->bmiHeader.biWidth ;
      cyDib = abs (pbmi->bmiHeader.biHeight) ;
```

```
      cBits = pbmi->bmiHeader.biBitCount ;

      if (pbmi->bmiHeader.biCompression != BI_RGB &&
        pbmi->bmiHeader.biCompression != BI_BITFIELDS)
      {
        MessageBox (hwnd, TEXT ("File is compressed."),
          szAppName, MB_ICONWARNING | MB_OK) ;
        if (pbmi)
          free (pbmi) ;
        if (pBits)
          free (pBits) ;
        CloseHandle (hFile) ;
        return 0 ;
      }
    }

    // Get the row length

    iRowLength = ((cxDib * cBits + 31) & ~31) >> 3 ;

    // Read and display
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    hdc = GetDC (hwnd) ;

    for (y = 0 ; y < cyDib ; y++)
    {
      ReadFile (hFile, pBits + y * iRowLength,iRowLength,&dwBytesRead, NULL) ;
      SetDIBitsToDevice (hdc,
        0, // xDst
        0, // yDst
        cxDib, // cxSrc
        cyDib, // cySrc
        0, // xSrc
        0, // ySrc
        bTopDown ? cyDib - y - 1 : y,
        // first scan line
        1, // number of scan lines
        pBits + y * iRowLength,
        pbmi,
        DIB_RGB_COLORS) ;
    }
    ReleaseDC (hwnd, hdc) ;
    CloseHandle (hFile) ;
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
    return 0 ;
}
break ;

case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;

  if (pbmi && pBits)
    SetDIBitsToDevice (hdc,
    0, // xDst
    0, // yDst
    cxDib, // cxSrc
    cyDib, // cySrc
    0, // xSrc
    0, // ySrc
    0, // first scan line
    cyDib, // number of scan lines
    pBits,
    DIB_RGB_COLORS) ;
  EndPaint (hwnd, &ps) ;
  return 0 ;
```

```
 case WM_DESTROY:
   if (pbmi)
     free (pbmi) ;

   if (pBits)
     free (pBits) ;

   PostQuitMessage (0) ;
   return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SEQDISP.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Accelerator
SEQDISP ACCELERATORS DISCARDABLE
BEGIN
"O", IDM_FILE_OPEN, VIRTKEY, CONTROL, NOINVERT
END

//////////////////////////////////////////////////////////////////////////////
// Menu
SEQDISP MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open...\tCtrl+O", IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by SeqDisp.rc
#define IDM_FILE_OPEN    40001
```

在处理「File Open」菜单命令期间，在SEQDISP.C内的所有文件I/O都会发生。在处理WM_COMMAND的最后，程序进入读取单行像素并用SetDIBitsToDevice显示该行像素的循环。整个DIB储存在内存中以便在处理WM_PAINT期间也能显示它。

缩放到合适尺寸

SetDIBitsToDevice完成了将DIB的像素对点送入输出设备的显示程序。这对于打印DIB用处不大。打印机的分辨率越高，得到的图像就越小，您最终会得到如邮票大小的图像。

要通过缩小或放大DIB，在输出设备上以特定的大小显示它，可以使用StretchDIBits：

```
iLines = StretchDIBits (
                    hdc, // device context handle
                    xDst, // x destination coordinate
                    yDst, // y destination coordinate
                    cxDst, // destination rectangle width
                    cyDst, // destination rectangle height
                    xSrc, // x source coordinate
                    ySrc, // y source coordinate
                    cxSrc, // source rectangle width
                    cySrc, // source rectangle height
                    pBits, // pointer to DIB pixel bits
                    pInfo, // pointer to DIB information
                    fClrUse, // color use flag
```

```
                dwRop) ; // raster operation
```

函数参数除了下列三个方面，均与SetDIBitsToDevice相同。

目的坐标包括逻辑宽度(cxDst)和高度(cyDst)，以及开始点。

不能通过持续显示DIB来减少内存需求。

最后一个参数是位映像操作方式，它指出了DIB像素与输出设备像素结合的方式，在最后一章将学到这些内容。现在我们为此参数设定为SRCCOPY。

还有另一个更细微的差别。如果查看SetDIBitsToDevice的声明，您会发现cxSrc和cySrc是DWORD，这是32位无正负号长整数型态。在StretchDIBits中，cxSrc和cySrc（以及cxDst和cyDst）定义为带正负号的整数型态，这意味着它们可以为负数，实际上等一下就会看到，它们确实能为负数。如果您已经开始检查是否别的参数也可以为负数，就让我声明一下：在两个函数中，xSrc和ySrc均定义为int，但这是错的，这些值始终是非负数。

DIB内的来源矩形被映射到目的矩形的坐标显示如表15-4所示。

表15-4

来源矩形	目的矩形
(xSrc, ySrc)	(xDst, yDst + cyDst - 1)
(xSrc + cxSrc - 1, ySrc)	(xDst + cxDst - 1, yDst + cyDst - 1)
(xSrc, ySrc + cySrc - 1)	(xDst, yDst)
(xSrc + cxSrc - 1, ySrc + cySrc - 1)	(xDst + cxDst - 1, yDst)

右列中的-1项是不精确的，因为放大的程度（以及映像方式和其它变换）能产生略微不同的结果。

例如，考虑一个2×2的DIB，这里StretchDIBits的xSrc和ySrc参数均为0，cxSrc和cySrc均为2。假定我们显示到的设备内容具有MM_TEXT映像方式并且不进行变换。如果xDst和yDst均为0，cxDst和cyDst均为4，那么我们将以倍数2放大DIB。每个来源像素（x,y）将映射到下面所示的四个目的像素上：

(0,0) --> (0,2) and (1,2) and (0,3) and (1,3)

(1,0) --> (2,2) and (3,2) and (2,3) and (3,3)

(0,1) --> (0,0) and (1,0) and (0,1) and (1,1)

(1,1) --> (2,0) and (3,0) and (2,1) and (3,1)

上表正确地指出了目的的角，(0,3)、(3,3)、(0,0)和(3,0)。在其它情况下，坐标可能是个大概值。

目的设备内容的映像方式对SetDIBitsToDevice的影响仅是由于xDst和yDst是逻辑坐标。StretchDIBits完全受映像方式的影响。例如，如果您设定了y值向上递增的一种度量映像方式，DIB就会颠倒显示。

您可以通过把cyDst设定为负数来弥补这种情况。实际上，您可以将任何参数的宽度和高度变为负值来水平或垂直翻转DIB。在MM_TEXT映像方式下，如果cySrc和cyDst符号相反，DIB会沿着水平轴翻转并颠倒显示。如果cxSrc和cxDst符号相反，DIB会沿着垂直轴翻转并显示它的镜面图像。

下面是总结这些内容的表达式，xMM和yMM指出映像方式的方向，如果x值向右增长，则xMM

值为1；如果x值向左增长，则值为-1。同样，如果y值向下增长，则yMM值为1；如果y值向上增长，则值为-1。Sign函数对于正值传回TURE，对于负值传回FALSE。

if (!Sign (xMM × cxSrc × cxDst))

DIB is flipped on its vertical axis (mirror image)

if (!Sign (yMM × cySrc × cyDst))

DIB is flipped on its horizontal axis (upside down)

若有疑问，请查阅表15-4。

程序15-5 SHOWDIB以实际尺寸显示DIB、放大至显示区域窗口的大小、打印DIB以及把DIB传输到剪贴簿。

程序15-5 SHOWDIB

SHOWDIB2.C

```
/*-------------------------------------------------------------------------
SHOWDIB2.C -- Shows a DIB in the client area
(c) Charles Petzold, 1998
-------------------------------------------------------------------------*/
#include <windows.h>
#include "dibfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #2"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
```

```
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

int ShowDib (HDC hdc, BITMAPINFO * pbmi, BYTE * pBits, int cxDib, int cyDib,
           int cxClient, int cyClient, WORD wShow)
{
  switch (wShow)
  {
  case IDM_SHOW_NORMAL:
    return SetDIBitsToDevice (hdc, 0, 0, cxDib, cyDib, 0, 0,
      0, cyDib, pBits, pbmi, DIB_RGB_COLORS) ;

  case IDM_SHOW_CENTER:
    return SetDIBitsToDevice (hdc, (cxClient - cxDib) / 2,
      (cyClient - cyDib) / 2,
      cxDib, cyDib, 0, 0, 0, cyDib, pBits, pbmi, DIB_RGB_COLORS) ;

  case IDM_SHOW_STRETCH:
    SetStretchBltMode (hdc, COLORONCOLOR) ;
    return StretchDIBits(hdc,0, 0, cxClient, cyClient, 0, 0, cxDib, cyDib,
      pBits, pbmi, DIB_RGB_COLORS, SRCCOPY) ;

  case IDM_SHOW_ISOSTRETCH:
    SetStretchBltMode (hdc, COLORONCOLOR) ;
    SetMapMode (hdc, MM_ISOTROPIC) ;
    SetWindowExtEx (hdc, cxDib, cyDib, NULL) ;
    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
    SetWindowOrgEx (hdc, cxDib / 2, cyDib / 2, NULL) ;
    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;

    return StretchDIBits(hdc,0, 0, cxDib, cyDib, 0, 0, cxDib, cyDib,
      pBits, pbmi, DIB_RGB_COLORS, SRCCOPY) ;
  }
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAPFILEHEADER * pbmfh ;
  static BITMAPINFO * pbmi ;
  static BYTE * pBits ;
  static DOCINFO di = {sizeof (DOCINFO),
    TEXT ("ShowDib2: Printing") } ;
  static int cxClient, cyClient, cxDib, cyDib ;
  static PRINTDLG printdlg = { sizeof (PRINTDLG) } ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static WORD wShow = IDM_SHOW_NORMAL ;
  BOOL bSuccess ;
  HDC hdc, hdcPrn ;
  HGLOBAL hGlobal ;
  HMENU hMenu ;
  int cxPage, cyPage, iEnable ;
  PAINTSTRUCT ps ;
  BYTE * pGlobal ;

  switch (message)
  {
  case WM_CREATE:
    DibFileInitialize (hwnd) ;
    return 0 ;
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
```

```
case WM_INITMENUPOPUP:
  hMenu = GetMenu (hwnd) ;

  if (pbmfh)
    iEnable = MF_ENABLED ;
  else
    iEnable = MF_GRAYED ;

  EnableMenuItem (hMenu, IDM_FILE_SAVE, iEnable) ;
  EnableMenuItem (hMenu, IDM_FILE_PRINT, iEnable) ;
  EnableMenuItem (hMenu, IDM_EDIT_CUT, iEnable) ;
  EnableMenuItem (hMenu, IDM_EDIT_COPY, iEnable) ;
  EnableMenuItem (hMenu, IDM_EDIT_DELETE, iEnable) ;
  return 0 ;

case WM_COMMAND:
  hMenu = GetMenu (hwnd) ;

  switch (LOWORD (wParam))
  {
  case IDM_FILE_OPEN:
    // Show the File Open dialog box

    if (!DibFileOpenDlg (hwnd, szFileName, szTitleName))
      return 0 ;

    // If there's an existing DIB, free the memory
    if (pbmfh)
    {
      free (pbmfh) ;
      pbmfh = NULL ;
    }
    // Load the entire DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pbmfh = DibLoadImage (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    // Invalidate the client area for later update

    InvalidateRect (hwnd, NULL, TRUE) ;

    if (pbmfh == NULL)
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;
    }
    // Get pointers to the info structure & the bits

    pbmi = (BITMAPINFO *) (pbmfh + 1) ;
    pBits = (BYTE *) pbmfh + pbmfh->bfOffBits ;

    // Get the DIB width and height

    if (pbmi->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    {
      cxDib = ((BITMAPCOREHEADER *) pbmi)->bcWidth ;
      cyDib = ((BITMAPCOREHEADER *) pbmi)->bcHeight ;
    }
    else
    {
      cxDib = pbmi->bmiHeader.biWidth ;
      cyDib = abs (pbmi->bmiHeader.biHeight) ;
    }
```

```
    return 0 ;

  case IDM_FILE_SAVE:
    // Show the File Save dialog box

    if (!DibFileSaveDlg (hwnd, szFileName, szTitleName))
      return 0 ;
    // Save the DIB to a disk file

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    bSuccess = DibSaveImage (szFileName, pbmfh) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (!bSuccess)
      MessageBox ( hwnd, TEXT ("Cannot save DIB file"),
      szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

  case IDM_FILE_PRINT:
    if (!pbmfh)
      return 0 ;

    // Get printer DC

    printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

    if (!PrintDlg (&printdlg))
      return 0 ;

    if (NULL == (hdcPrn = printdlg.hDC))
    {
      MessageBox (hwnd, TEXT ("Cannot obtain Printer DC"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;
    }

    // Check whether the printer can print bitmaps

    if (!(RC_BITBLT & GetDeviceCaps (hdcPrn, RASTERCAPS)))
    {
      DeleteDC (hdcPrn) ;
      MessageBox ( hwnd, TEXT ("Printer cannot print bitmaps"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;
    }
    // Get size of printable area of page

    cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
    cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

    bSuccess = FALSE ;
    // Send the DIB to the printer

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
    {
      ShowDib ( hdcPrn, pbmi, pBits, cxDib, cyDib,
        cxPage, cyPage, wShow) ;

      if (EndPage (hdcPrn) > 0)
      {
        bSuccess = TRUE ;
        EndDoc (hdcPrn) ;
```

```
      }
    }
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    DeleteDC (hdcPrn) ;

    if (!bSuccess)
      MessageBox (hwnd, TEXT ("Could not print bitmap"),
      szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

  case IDM_EDIT_COPY:
  case IDM_EDIT_CUT:
    if (!pbmfh)
      return 0 ;

    // Make a copy of the packed DIB

    hGlobal = GlobalAlloc (GHND | GMEM_SHARE, pbmfh->bfSize -
      sizeof (BITMAPFILEHEADER)) ;

    pGlobal = GlobalLock (hGlobal) ;

    CopyMemory ( pGlobal, (BYTE *) pbmfh + sizeof (BITMAPFILEHEADER),
      pbmfh->bfSize - sizeof (BITMAPFILEHEADER)) ;

    GlobalUnlock (hGlobal) ;

    // Transfer it to the clipboard

    OpenClipboard (hwnd) ;
    EmptyClipboard () ;
    SetClipboardData (CF_DIB, hGlobal) ;
    CloseClipboard () ;

    if (LOWORD (wParam) == IDM_EDIT_COPY)
      return 0 ;
    // fall through if IDM_EDIT_CUT
  case IDM_EDIT_DELETE:
    if (pbmfh)
    {
      free (pbmfh) ;
      pbmfh = NULL ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

  case IDM_SHOW_NORMAL:
  case IDM_SHOW_CENTER:
  case IDM_SHOW_STRETCH:
  case IDM_SHOW_ISOSTRETCH:
    CheckMenuItem (hMenu, wShow, MF_UNCHECKED) ;
    wShow = LOWORD (wParam) ;
    CheckMenuItem (hMenu, wShow, MF_CHECKED) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
}
break ;

case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;

  if (pbmfh)
    ShowDib ( hdc, pbmi, pBits, cxDib, cyDib,
    cxClient, cyClient, wShow) ;

  EndPaint (hwnd, &ps) ;
  return 0 ;
```

```
 case WM_DESTROY:
   if (pbmfh)
     free (pbmfh) ;

   PostQuitMessage (0) ;
   return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB2.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB2 MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open...\tCtrl+O",IDM_FILE_OPEN
MENUITEM "&Save...\tCtrl+S", IDM_FILE_SAVE
MENUITEM SEPARATOR
MENUITEM "&Print\tCtrl+P", IDM_FILE_PRINT
END
POPUP "&Edit"
BEGIN
MENUITEM  "Cu&t\tCtrl+X", IDM_EDIT_CUT
MENUITEM  "&Copy\tCtrl+C", IDM_EDIT_COPY
MENUITEM  "&Delete\tDelete", IDM_EDIT_DELETE
END
POPUP "&Show"
BEGIN
MENUITEM  "&Actual Size", IDM_SHOW_NORMAL, CHECKED
MENUITEM  "&Center",  IDM_SHOW_CENTER
MENUITEM  "&Stretch to Window", IDM_SHOW_STRETCH
MENUITEM  "Stretch &Isotropically", IDM_SHOW_ISOSTRETCH
END
END

//////////////////////////////////////////////////////////////////////////////
// Accelerator
SHOWDIB2 ACCELERATORS DISCARDABLE
BEGIN
"C", IDM_EDIT_COPY,  VIRTKEY, CONTROL, NOINVERT
"O", IDM_FILE_OPEN, VIRTKEY, CONTROL, NOINVERT
"P", IDM_FILE_PRINT, VIRTKEY, CONTROL, NOINVERT
"S", IDM_FILE_SAVE,  VIRTKEY, CONTROL, NOINVERT
VK_DELETE, IDM_EDIT_DELETE, VIRTKEY, NOINVERT
"X", IDM_EDIT_CUT,  VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib2.rc
#define IDM_FILE_OPEN   40001
#define IDM_SHOW_NORMAL  40002
#define IDM_SHOW_CENTER  40003
#define IDM_SHOW_STRETCH  40004
#define IDM_SHOW_ISOSTRETCH 40005
#define IDM_FILE_PRINT  40006
#define IDM_EDIT_COPY  40007
#define IDM_EDIT_CUT   40008
```

```
#define IDM_EDIT_DELETE  40009
#define IDM_FILE_SAVE   40010
```

有意思的是ShowDib函数，它依赖于菜单选择以四种不同的方式之一在程序的显示区域显示DIB。可以使用SetDIBitsToDevice从显示区域的左上角或在显示区域的中心显示DIB。程序也有两个使用StretchDIBits的选项，DIB能放大填充整个显示区域。在此情况下它可能会变形，或它能等比例显示，也就是说不会变形。

把DIB复制到剪贴簿包括：在整体共享内存中制作packed DIB内存块的副本。剪贴簿数据型态为CF_DIB。程序没有列出从剪贴簿复制DIB的方法，因为在仅有指向packed DIB的指标的情况下这样做需要更多步骤来确定像素位的偏移量。我将在下一章的末尾示范如何做到这点的方法。

您可能注意到了SHOWDIB2中的一些不足之处。如果您以256色显示模式执行Windows，就会看到显示除了单色或4位DIB以外的其它图形出现的问题，您看不到真正的颜色。存取那些颜色需要使用调色盘，在下一章会做这些工作。您也可能注意到速度问题，尤其在Windows NT下执行SHOWDIB2时。在下一章packed DIB和位图时，我会展示处理的方法。我也给DIB显示添加滚动条，这样也能以实际尺寸查看大于屏幕的DIB。

色彩转换、调色盘和显示效能

记得在虎豹小霸王编剧William Goldman的另一出电影剧本《All the President's Men》中，Deep Throat告诉Bob Woodward揭开水门秘密的关键是「跟着钱走」。那么在位图显示中获得高级性能的关键就是「跟着像素位走」以及理解色彩转换发生的时机。DIB是设备无关的格式，视讯显示器内存几乎总是与像素格式不同。在SetDIBitsToDevice或StretchDIBits函数呼叫期间，每个像素（可能有几百万个）必须从设备无关的格式转换成设备相关格式。

在许多情况下，这种转换是很繁琐的。例如，在24位视讯显示器上显示24位DIB，显示驱动程序最多是切换红、绿、蓝的字节顺序而已。在24位设备上显示16位DIB就需要位的搬移和修剪了。在24位设备上显示4位或8位DIB要求在DIB色彩对照表内查找DIB像素位，然后对字节重新排列。

但是要在4位或8位视讯显示器上显示16位、24位或32位DIB时，会发生什么事情呢？一种完全不一样的颜色转换发生了。对于DIB内的每个像素，设备驱动程序必须在像素和显示器上可用的颜色之间「找寻最接近的色彩」，这包括循环和计算。（GDI函数GetNearestColor进行「最接近色彩搜寻」。）

整个RGB色彩的三维数组可用立方体表示。曲线内任意两点之间的距离是：

$$\sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

在这里两个颜色是R1G1B1和R2G2B2。执行最接近色彩搜寻包括从一种颜色到其它颜色集合中找寻最短距离。幸运的是，在RGB颜色立方体中「比较」距离时，并不需要计算平方根部分。但是需转换的每个像素必须与设备的所有颜色相比较以发现最接近的颜色。这是个工作量相当大的工作。（尽管在8位设备上显示8位DIB也得进行最接近色彩搜寻，但它不必对每个像素都进行，它仅需对DIB色彩对照表中的每种颜色进行寻找。）

正是由于以上原因，应该避免使用SetDIBitsToDevice或StretchDIBits在8位视讯显示卡上显示16位、24位或32位DIB。DIB应转换为8位DIB，或者8位DDB，以求得更好的显示效能。实际上，您

可以经由将DIB转换为DDB并使用BitBlt和StretchBlt显示图像，来加快显示任何DIB的速度。

如果在8位视讯显示器上执行Windows（或仅仅切换到8位模式来观察在显示True-ColorDIB时的效能变化），您可能会注意到另一个问题：DIB不会使用所有颜色来显示。任何在8位视讯显示器上的DIB刚好限制在以20种颜色显示。如何获得多于20种颜色是「调色盘管理器」的任务，这将在下一章提到。

最后，如果在同一台机器上执行Windows 98和Windows NT，您可能会注意到：对于同样的显示模式，Windows NT显示大型DIB花费的时间较长。这是Windows NT的客户/服务器体系结构的结果，它使大量数据在传输给API函数时耗费更多时间。解决方法是将DIB转换为DDB。而我等一下将谈到的CreateDIBSection函数对这种情况特别有用。

DIB 和 DDB 的结合

您可以做许多事情去发掘DIB的格式，并呼叫两个DIB绘图函数：SetDIBitsToDevice和StretchDIBits。您可以直接存取DIB中的各个位、字节和像素，且一旦您有了一堆能让您以结构化的方式检查和更改数据的函数，您要怎么处理DIB就没人管了。

实际上，我们发现还是有一些限制。在上一章，我们了解了使用GDI函数在DDB上绘制图像的方法。到目前为止，还没有在DIB上绘图的方法。另一个问题是SetDIBitsToDevice和StretchDIBits没有BitBlt和StretchBlt速度快，尤其在Windows NT环境下以及执行许多最接近颜色搜寻（例如，在8位视频卡上显示24位DIB）时。

因此，在DIB和DDB之间进行转换是有好处的。例如，如果我们有一个需要在屏幕上显示许多次的DIB，那么把DIB转换为DDB就很有意义，这样我们就能够使用快速的BitBlt和StretchBlt函数来显示它了。

从DIB建立DDB

从DIB中建立GDI位图对象可能吗？基本上我们已经知道了方法：如果有DIB，您就能够使用CreateCompatibleBitmap来建立与DIB大小相同并与视讯显示器兼容的GDI位图对象。然后将该位图对象选入内存设备内容并呼叫SetDIBitsToDevice在那个内存DC上绘图。结果就是DDB具有与DIB相同的图像，但具有与视讯显示器兼容的颜色组织。

您也可以通过呼叫CreateDIBitmap用几个步骤完成上述工作。函数的语法为：

```
hBitmap = CreateDIBitmap (
                      hdc, // device context handle
                      pInfoHdr, // pointer to DIB information header
                      fInit, // 0 or CBM_INIT
                      pBits, // pointer to DIB pixel bits
                      pInfo, // pointer to DIB information
                      fClrUse) ; // color use flag
```

请注意pInfoHdr和pInfo这两个参数，它们分别定义为指向BITMAPINFOHEADER结构和BITMAPINFO结构的指针。正如我们所知，BITMAPINFO结构是后面紧跟色彩对照表的BITMAPINFOHEADER结构。我们一会儿会看到这种区别所起的作用。最后一个参数是DIB_RGB_COLORS（等于0）或DIB_PAL_COLORS，它们在SetDIBitsToDevice函数中使用。下一章我将讨论更多这方面的内容。

理解Windows中位图函数的作用是很重要的。不要考虑CreateDIBitmap函数的名称，它不建立与「设备无关的位图」，它从设备无关的规格中建立「设备相关的位图」。注意该函数传回GDI位图对象的句柄，CreateBitmap、CreateBitmapIndirect和CreateCompatibleBitmap也与它一

样。

呼叫CreateDIBitmap函数最简单的方法是：

hBitmap = CreateDIBitmap (NULL, pbmih, 0, NULL, NULL, 0) ;

唯一的参数是指向BITMAPINFOHEADER结构（不带色彩对照表）的指标。在这个形式中，函数建立单色GDI位图对象。第二种简单的方法是：

hBitmap = CreateDIBitmap (hdc, pbmih, 0, NULL, NULL, 0) ;

在这个形式中，函数建立了与设备内容兼容的DDB，该设备内容由hdc参数指出。到目前为止，我们都是透过CreateBitmap（建立单色位图）或CreateCompatibleBitmap（建立与视讯显示器兼容的位图）来完成一些工作。

在CreateDIBitmap的这两个简化模式中，像素还未被初始化。如果CreateDIBitmap的第三个参数是CBM_INIT，Windows就会建立DDB并使用最后三个参数初始化位图位。pInfo参数是指向包括色彩对照表的BITMAPINFO结构的指针。pBits参数是指向由BITMAPINFO结构指出的色彩格式中的位数组的指针，根据色彩对照表这些位被转换为设备的颜色格式，这与SetDIBitsToDevice的情况相同。实际上，整个CreateDIBitmap函数可以用下列程序代码来实作：

```
HBITMAP CreateDIBitmap ( HDC hdc, CONST BITMAPINFOHEADER * pbmih,
                    DWORD fInit, CONST VOID * pBits,
                    CONST BITMAPINFO * pbmi, UINT fUsage)
{
  HBITMAP hBitmap ;
  HDC hdc ;
  int cx, cy, iBitCount ;
  if (pbmih->biSize == sizeof (BITMAPCOREHEADER))
  {
    cx = ((PBITMAPCOREHEADER) pbmih)->bcWidth ;
    cy = ((PBITMAPCOREHEADER) pbmih)->bcHeight ;
    iBitCount = ((PBITMAPCOREHEADER) pbmih)->bcBitCount ;
  }
  else
  {
    cx = pbmih->biWidth ;
    cy = pbmih->biHeight ;
    iBitCount = pbmih->biBitCount ;
  }
  if (hdc)
    hBitmap = CreateCompatibleBitmap (hdc, cx, cy) ;
  else
    hBitmap = CreateBitmap (cx, cy, 1, 1, NULL) ;
  if (fInit == CBM_INIT)
  {
    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;
    SetDIBitsToDevice ( hdcMem, 0, 0, cx, cy, 0, 0, 0 cy,
      pBits, pbmi, fUsage) ;
    DeleteDC (hdcMem) ;
  }

  return hBitmap ;
}
```

如果仅需显示DIB一次，并担心SetDIBitsToDevice显示太慢，则呼叫CreateDIBitmap并使用BitBlt或StretchBlt来显示DDB就没有什么意义。因为SetDIBitsToDevice和CreateDIBitmap都执行颜色转换，这两个工作会占用同样长的时间。只有在多次显示DIB时（例如在处理WM_PAINT消息时）进行这种转换才有意义。

程序15-6 DIBCONV展示了利用SetDIBitsToDevice把DIB文件转换为DDB的方法。

程序15-6 DIBCONV

DIBCONV.C

```
/*---------------------------------------------------------------------------
DIBCONV.C -- Converts a DIB to a DDB
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("DibConv") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("DIB to DDB Conversion"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;

  }
  return msg.wParam ;
}
HBITMAP CreateBitmapObjectFromDibFile (HDC hdc, PTSTR szFileName)
{
  BITMAPFILEHEADER * pbmfh ;
  BOOL bSuccess ;
  DWORD dwFileSize, dwHighSize, dwBytesRead ;
  HANDLE hFile ;
  HBITMAP hBitmap ;

  // Open the file: read access, prohibit write access

  hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
```

```
  if (hFile == INVALID_HANDLE_VALUE)
    return NULL ;

  // Read in the whole file

  dwFileSize = GetFileSize (hFile, &dwHighSize) ;

  if (dwHighSize)
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  pbmfh = malloc (dwFileSize) ;

  if (!pbmfh)
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  bSuccess = ReadFile (hFile, pbmfh, dwFileSize, &dwBytesRead, NULL) ;
  CloseHandle (hFile) ;

  // Verify the file
  if (!bSuccess || (dwBytesRead != dwFileSize)
    || (pbmfh->bfType != * (WORD *) "BM")
    || (pbmfh->bfSize != dwFileSize))
  {
    free (pbmfh) ;
    return NULL ;
  }
  // Create the DDB
  hBitmap = CreateDIBitmap (hdc,
    (BITMAPINFOHEADER *) (pbmfh + 1),
    CBM_INIT,
    (BYTE *) pbmfh + pbmfh->bfOffBits,
    (BITMAPINFO *) (pbmfh + 1),
    DIB_RGB_COLORS) ;
  free (pbmfh) ;
  return hBitmap ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[]=TEXT("Bitmap Files(*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  BITMAP bitmap ;
  HDC hdc, hdcMem ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
```

```
  ofn.lpstrInitialDir = NULL ;
  ofn.lpstrTitle = NULL ;
  ofn.Flags = 0 ;
  ofn.nFileOffset = 0 ;
  ofn.nFileExtension = 0 ;
  ofn.lpstrDefExt = TEXT ("bmp") ;
  ofn.lCustData = 0 ;
  ofn.lpfnHook = NULL ;
  ofn.lpTemplateName = NULL ;

  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_FILE_OPEN:

    // Show the File Open dialog box
    if (!GetOpenFileName (&ofn))
      return 0 ;

    // If there's an existing DIB, delete it
    if (hBitmap)
    {
      DeleteObject (hBitmap) ;
      hBitmap = NULL ;
    }
    // Create the DDB from the DIB
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    hdc = GetDC (hwnd) ;
    hBitmap = CreateBitmapObjectFromDibFile (hdc, szFileName) ;
    ReleaseDC (hwnd, hdc) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    // Invalidate the client area for later update
    InvalidateRect (hwnd, NULL, TRUE) ;
    if (hBitmap == NULL)
    {
      MessageBox (hwnd, TEXT ("Cannot load DIB file"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    }
    return 0 ;
  }
  break ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hBitmap)
    {
      GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

      hdcMem = CreateCompatibleDC (hdc) ;
      SelectObject (hdcMem, hBitmap) ;

      BitBlt (hdc,0, 0, bitmap.bmWidth, bitmap.bmHeight,
        hdcMem, 0, 0, SRCCOPY) ;

      DeleteDC (hdcMem) ;
    }
```

```
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    if (hBitmap)
      DeleteObject (hBitmap) ;

    PostQuitMessage (0) ;
    return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIBCONV.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
DIBCONV MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open",  IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DibConv.rc
#define IDM_FILE_OPEN  40001
```

DIBCONV.C本身就是完整的，并不需要前面的文件。在对它仅有的菜单命令（「File Open」）的响应中，WndProc呼叫程序的CreateBitmapObjectFromDibFile函数。此函数将整个文件读入内存并将指向内存块的指针传递给CreateDIBitmap函数，函数传回位图的句柄，然后包含DIB的内存块被释放。在WM_PAINT消息处理期间，WndProc将位图选入兼容的内存设备内容并使用BitBlt（不是SetDIBitsToDevice）在显示区域显示位图。它通过使用位图句柄呼叫带有BITMAP结构的GetObject函数来取得位图的宽度和高度。

在从CreateDIBitmap建立位图时不必初始化DDB像素位，之后您可以呼叫SetDIBits初始化像素位。该函数的语法如下：

```
iLines = SetDIBits (
                hdc, // device context handle
                hBitmap, // bitmap handle
                yScan, // first scan line to convert
                cyScans, // number of scan lines to convert
                pBits, // pointer to pixel bits
                pInfo, // pointer to DIB information
                fClrUse) ; // color use flag
```

函数使用了BITMAPINFO结构中的色彩对照表把位转换为设备相关的格式。只有在最后一个参数设定为DIB_PAL_COLORS时，才需要设备内容句柄。

从DDB到DIB

与SetDIBits函数相似的函数是GetDIBits，您可以使用此函数把DDB转化为DIB：

```
int WINAPI GetDIBits (
                  hdc, // device context handle
                  hBitmap, // bitmap handle
                  yScan, // first scan line to convert
                  cyScans, // number of scan lines to convert
```

```
                pBits, // pointer to pixel bits (out)
                pInfo, // pointer to DIB information (out)
                fClrUse) ; // color use flag
```

然而，此函数产生的恐怕不是SetDIBits的反运算结果。在一般情况下，如果使用CreateDIBitmap和SetDIBits将DIB转换为DDB，然后使用GetDIBits把DDB转换回DIB，您就不会得到原来的图像。这是因为在DIB被转换为设备相关的格式时，有一些信息遗失了。遗失的信息数量取决于进行转换时Windows所执行的显示模式。

您可能会发现没有使用GetDIBits的必要性。考虑一下：在什么环境下您的程序发现自身带有位图句柄，但没有用于在起始的位置建立位图的数据？剪贴簿？但是剪贴簿为DIB提供了自动的转换。GetDIBits函数的一个例子是在捕捉屏幕显示内容的情况下，例如第十四章中BLOWUP程序所做的。我不示范这个函数，但在Microsoft网站的Knowledge Base文章Q80080中有一些信息。

DIB区块

我希望您已经对设备相关和设备无关位图的区别有了清晰的概念。DIB能拥有几种色彩组织中的一种，DDB必须是单色的或是与真实输出设备相同的格式。DIB是一个文件或内存块；DDB是GDI位图对象并由位图句柄表示。DIB能被显示或转换为DDB并转换回DIB，但是这里包含了设备无关位和设备相关位之间的转换程序。

现在您将遇到一个函数，它打破了这些规则。该函数在32位Windows版本中发表，称为CreateDIBSection，语法为：

```
hBitmap = CreateDIBSection (
                        hdc, // device context handle
                        pInfo, // pointer to DIB information
                        fClrUse, // color use flag
                        ppBits, // pointer to pointer variable
                        hSection, // file-mapping object handle
                        dwOffset) ; // offset to bits in file-mapping object
```

CreateDIBSection是Windows API中最重要的函数之一（至少在使用位图时），然而您会发现它很深奥并难以理解。

让我们从它的名称开始，我们知道DIB是什么，但「DIB section」到底是什么呢？当您第一次检查CreateDIBSection时，可能会寻找该函数与DIB区块工作的方式。这是正确的，CreateDIBSection所做的就是建立了DIB的一部分（位图像素位的内存块）。

现在我们看一下传回值，它是GDI位图对象的句柄，这个传回值可能是该函数呼叫最会拐人的部分。传回值似乎暗示着CreateDIBSection在功能上与CreateDIBitmap相同。事实上，它只是相似但完全不同。实际上，从CreateDIBSection传回的位图句柄与我们在本章和上一章遇到的所有位图建立函数传回的位图句柄在本质上不同。

一旦理解了CreateDIBSection的真实特性，您可能觉得奇怪为什么不把传回值定义得有所区别。您也可能得出结论：CreateDIBSection应该称之为CreateDIBitmap，并且如同我前面所指出的CreateDIBitmap应该称之为CreateDDBitmap。

首先让我们检查一下如何简化CreateDIBSection，并正确地使用它。首先，把最后两个参数hSection和dwOffset，分别设定为NULL和0，我将在本章最后讨论这些参数的用法。第二，仅在fColorUse参数设定为DIB_ PAL_COLORS时，才使用hdc参数，如果fColorUse为DIB_RGB_COLORS（或0），hdc将被忽略（这与CreateDIBitmap不同，hdc参数用于取得与DDB兼容的设备的色彩格式）。

因此，CreateDIBSection最简单的形式仅需要第二和第四个参数。第二个参数是指向BITMAPINFO结构的指针，我们以前曾使用过。我希望指向第四个参数的指标定义的指标不会使您

困惑，它实际上很简单。

假设要建立每像素24位的384×256位DIB，24位格式不需要色彩对照表，因此它是最简单的，所以我们可以为BITMAPINFO参数使用BITMAPINFOHEADER结构。

您需要定义三个变量：BITMAPINFOHEADER结构、BYTE指针和位图句柄：

```
BITMAPINFOHEADER bmih ;
BYTE  * pBits ;
HBITMAP  hBitmap ;
```

现在初始化BITMAPINFOHEADER结构的字段

```
bmih->biSize = sizeof (BITMAPINFOHEADER) ;
bmih->biWidth = 384 ;
bmih->biHeight = 256 ;
bmih->biPlanes = 1 ;
bmih->biBitCount = 24 ;
bmih->biCompression = BI_RGB ;
bmih->biSizeImage = 0 ;
bmih->biXPelsPerMeter = 0 ;
bmih->biYPelsPerMeter = 0 ;
bmih->biClrUsed = 0 ;
bmih->biClrImportant = 0 ;
```

在基本准备后，我们呼叫该函数：

hBitmap = CreateDIBSection (NULL, (BITMAPINFO *) &bmih, 0, &pBits, NULL, 0) ;

注意，我们为第二个参数赋予BITMAPINFOHEADER结构的地址。这是常见的，但一个BYIE指针pBits的地址，就不常见了。这样，第四个参数是函数需要的指向指标的指标。

这是函数呼叫所做的：CreateDIBSection检查BITMAPINFOHEADER结构并配置足够的内存块来加载DIB像素位。（在这个例子里，内存块的大小为384×256×3字节。）它在您提供的pBits参数中储存了指向此内存块的指针。函数传回位图句柄，正如我说的，它与CreateDIBitmap和其它位图建立函数传回的句柄不一样。

然而，我们还没有做完，位图像素是未初始化的。如果正在读取DIB文件，可以简单地把pBits参数传递给ReadFile函数并读取它们。或者可以使用一些程序代码「人工」设定。

程序15-7 DIBSECT除了呼叫CreateDIBSection而不是CreateDIBitmap之外，与DIBCONV程序相似。

程序15-7 DIBSECT

DIBSECT.C

```
/*-------------------------------------------------------------------------
DIBSECT.C -- Displays a DIB Section in the client area
(c) Charles Petzold, 1998
-------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "resource.h"
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName[] = TEXT ("DIBsect") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("DIB Section Display"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;

  }
  return msg.wParam ;
}
HBITMAP CreateDIBsectionFromDibFile (PTSTR szFileName)
{
  BITMAPFILEHEADER bmfh ;
  BITMAPINFO * pbmi ;
  BYTE * pBits ;
  BOOL bSuccess ;
  DWORD dwInfoSize, dwBytesRead ;
  HANDLE hFile ;
  HBITMAP hBitmap ;

  // Open the file: read access, prohibit write access

  hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ,
    NULL, OPEN_EXISTING, 0, NULL) ;
  if (hFile == INVALID_HANDLE_VALUE)
    return NULL ;
  // Read in the BITMAPFILEHEADER
  bSuccess = ReadFile (hFile, &bmfh, sizeof (BITMAPFILEHEADER),
    &dwBytesRead, NULL) ;

  if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
    || (bmfh.bfType != * (WORD *) "BM"))
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  // Allocate memory for the BITMAPINFO structure & read it in
  dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
  pbmi = malloc (dwInfoSize) ;
  bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;
  if (!bSuccess || (dwBytesRead != dwInfoSize))
  {
    free (pbmi) ;
    CloseHandle (hFile) ;
    return NULL ;
```

```
  }
  // Create the DIB Section
  hBitmap = CreateDIBSection (NULL, pbmi, DIB_RGB_COLORS, &pBits, NULL, 0) ;
  if (hBitmap == NULL)
  {
    free (pbmi) ;
    CloseHandle (hFile) ;
    return NULL ;
  }

  // Read in the bitmap bits
  ReadFile (hFile, pBits, bmfh.bfSize - bmfh.bfOffBits, &dwBytesRead, NULL) ;
  free (pbmi) ;
  CloseHandle (hFile) ;

  return hBitmap ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  BITMAP bitmap ;
  HDC hdc, hdcMem ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("bmp") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

      // Show the File Open dialog box

      if (!GetOpenFileName (&ofn))
```

```
      return 0 ;

    // If there's an existing bitmap, delete it

    if (hBitmap)
    {
      DeleteObject (hBitmap) ;
      hBitmap = NULL ;
    }
    // Create the DIB Section from the DIB file

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    hBitmap = CreateDIBsectionFromDibFile (szFileName) ;
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    // Invalidate the client area for later update

    InvalidateRect (hwnd, NULL, TRUE) ;

    if (hBitmap == NULL)
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    }
    return 0 ;
  }
  break ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hBitmap)
    {
      GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

      hdcMem = CreateCompatibleDC (hdc) ;
      SelectObject (hdcMem, hBitmap) ;

      BitBlt ( hdc, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
        hdcMem, 0, 0, SRCCOPY) ;

      DeleteDC (hdcMem) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    if (hBitmap)
      DeleteObject (hBitmap) ;

    PostQuitMessage (0) ;
    return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIBSECT.RC（摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
DIBSECT MENU DISCARDABLE
```

```
BEGIN
   POPUP "&File"
   BEGIN
         MENUITEM "&Open",     IDM_FILE_OPEN
   END
END
```

RESOURCE.H（摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DIBsect.rc
#define IDM_FILE_OPEN     40001
```

注意DIBCONV中的CreateBitmapObjectFromDibFile函数和DIBSECT中的CreateDIbsectionFromDibFile函数之间的区别。DIBCONV读入整个文件，然后把指向DIB内存块的指针传递给CreateDIBitmap函数。DIBSECT首先读取BITMAPFILEHEADER结构中的信息，然后确定BITMAPINFO结构的大小，为此配置内存，并在第二个ReadFile呼叫中将它读入内存。然后，函数把指向BITMAPINFO结构和指针变量pBits的指针传递给CreateDIBSection。函数传回位图句柄并设定pBits指向函数将要读取DIB像素位的内存块。

pBits指向的内存块归系统所有。当通过呼叫DeleteObject删除位图时，内存会被自动释放。然而，程序能利用该指针直接改变DIB位。当应用程序透过API传递海量存储器块时，只要系统拥有这些内存块，在WINDOWS NT下就不会影响速度。

我之前曾说过，当在视讯显示器上显示DIB时，某些时候必须进行从设备无关像素到设备相关像素的转换，有时这些格式转换可能相当费时。来看一看三种用于显示DIB的方法：

当使用SetDIBitsToDevice或StretchDIBits来把DIB直接显示在屏幕上，格式转换在SetDIBitsToDevice或StretchDIBits呼叫期间发生。

当使用CreateDIBitmap和（可能是）SetDIBits把DIB转换为DDB，然后使用BitBlt或StretchBlt来显示它时，如果设定了CBM_INIT旗标，格式转换在CreateDIBitmap或SetDIBits期间发生。

当使用CreateDIBSection建立DIB区块，然后使用BitBlt或StretchBlt显示它时，格式转换在BitBlt对StretchBlt的呼叫期间发生。

再读一下上面这些叙述，确定您不会误解它的意思。这是从CreateDIBSection传回的位图句柄不同于我们所遇到的其它位图句柄的一个地方。此位图句柄实际上指向储存在内存中由系统维护但应用程序能存取的DIB。在需要的时候，DIB会转化为特定的色彩格式，通常是在用BitBlt或StretchBlt显示位图时。

您也可以将位图句柄选入内存设备内容并使用GDI函数来绘制。在 pBits 变量指向的DIB像素内将反映出结果。因为Windows NT下的GDI函数分批呼叫，在内存设备背景上绘制之后和「人为」的存取位之前会呼叫GdiFlush。

在DIBSECT，我们清除pBits变量，因为程序不再需要这个变量了。您会使用CreateDIBSection的主要原因在于您有需要直接更改位值。在CreateDIBSection呼叫之后似乎就没有别的方法来取得位指针了。

DIB区块的其它区别

从CreateDIBitmap传回的位图句柄与函数的hdc参数引用的设备有相同的平面和像素字节织。您能通过具有BITMAP结构的GetObject呼叫来检验这一点。

CreateDIBSection就不同了。如果以该函数传回的位图句柄的BITMAP结构呼叫GetObject，

您会发现位图具有的色彩组织与BITMAPINFOHEADER结构的字段指出的色彩组织相同。您能将这个句柄选入与视讯显示器兼容的内存设备内容。这与上一章关于DDB的内容相矛盾，但这也就是我说此DIB区块位图句柄不同的原因。

另一个奇妙之处是：你可能还记得，DIB中像素数据行的位组长度始终是4的倍数。GDI位图对象中行的位组长度，就是使用GetObject从BITMAP结构的bmWidthBytes字段中得到的长度，始终是2的倍数。如果用每像素24位和宽度2像素设定BITMAPINFOHEADER结构并随后呼叫GetObject，您就会发现bmWidthBytes字段是8而不是6。

使用从CreateDIBSection传回的位图句柄，也可以使用DIBSECTION结构呼叫GetObject：

GetObject (hBitmap, sizeof (DIBSECTION), &dibsection) ;

此函数不能处理其它位图建立函数传回的位图句柄。DIBSECTION结构定义如下：

```
typedef struct tagDIBSECTION // ds
{
  BITMAP  dsBm ; // BITMAP structure
  BITMAPINFOHEADER dsBmih ;  // DIB information header
  DWORD  dsBitfields [3] ; // color masks
  HANDLE  dshSection ; // file-mapping object handle
  DWORD  dsOffset ; // offset to bitmap bits
}
DIBSECTION, * PDIBSECTION ;
```

此结构包含BITMAP结构和BITMAPINFOHEADER结构。最后两个字段是传递给CreateDIBSection的最后两个参数，等一下将会讨论它们。

DIBSECTION结构中包含除了色彩对照表以外有关位图的许多内容。当把DIB区块位图句柄选入内存设备内容时，可以通过呼叫GetDIBColorTable来得到色彩对照表：

```
hdcMem = CreateCompatibleDC (NULL) ;
SelectObject (hdcMem, hBitmap) ;
GetDIBColorTable (hdcMem, uFirstIndex, uNumEntries, &rgb) ;
DeleteDC (hdcMem) ;
```

同样，您可以通过呼叫SetDIBColorTable来设定色彩对照表中的项目。

文件映像选项

我们还没有讨论CreateDIBSection的最后两个参数，它们是文件映像对象的句柄和文件中位图位开始的偏移量。文件映像对象使您能够像文件位于内存中一样处理文件。也就是说，可以通过使用内存指针来存取文件，但文件不需要整个加载内存中。

在大型DIB的情况下，此技术对于减少内存需求是很有帮助的。DIB像素位能够储存在磁盘上，但仍然可以当作位于内存中一样进行存取，虽然会影响程序执行效能。问题是，当像素位实际上储存在磁盘上时，它们不可能是实际DIB文件的一部分。它们必须位于其它的文件内。

为了展示这个程序，下面显示的函数除了不把像素位读入内存以外，与DIBSECT中建立DIB区块的函数很相似。然而，它提供了文件映像对象和传递给CreateDIBSection函数的偏移量：

```
HBITMAP CreateDIBsectionMappingFromFile (PTSTR szFileName)
{
  BITMAPFILEHEADER bmfh ;
  BITMAPINFO * pbmi ;
  BYTE * pBits ;
  BOOL bSuccess ;
  DWORD dwInfoSize, dwBytesRead ;
  HANDLE hFile, hFileMap ;
  HBITMAP hBitmap ;
  hFile = CreateFile (szFileName, GENERIC_READ | GENERIC_WRITE,
    0, // No sharing!
    NULL, OPEN_EXISTING, 0, NULL) ;
```

```
  if (hFile == INVALID_HANDLE_VALUE)
    return NULL ;
  bSuccess = ReadFile ( hFile, &bmfh, sizeof (BITMAPFILEHEADER),
    &dwBytesRead, NULL) ;

  if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
    || (bmfh.bfType != * (WORD *) "BM"))
  {
    CloseHandle (hFile) ;
    return NULL ;
  }
  dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
  pbmi = malloc (dwInfoSize) ;
  bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;

  if (!bSuccess || (dwBytesRead != dwInfoSize))
  {
    free (pbmi) ;
    CloseHandle (hFile) ;
    return NULL ;
  }
  hFileMap = CreateFileMapping (hFile, NULL, PAGE_READWRITE, 0, 0, NULL) ;
  hBitmap = CreateDIBSection ( NULL, pbmi, DIB_RGB_COLORS, &pBits,hFileMap, bmfh.bfOffBits) ;
  free (pbmi) ;
  return hBitmap ;
}
```

啊哈！这个程序不会动。CreateDIBSection的文件指出「dwOffset [函数的最后一个参数]必须是DWORD大小的倍数」。尽管信息表头的大小始终是4的倍数并且色彩对照表的大小也始终是4的倍数，但位图文件表头却不是，它是14字节。因此bmfh.bfOffBits永远不会是4的倍数。

总结

如果您有小型的DIB并且需要频繁地操作像素位，您可以使用SetDIBitsToDevice和StretchDIBits来显示它们。然而，对于大型的DIB，此技术会遇到显示效能的问题，尤其在8位视讯显示器上和Windows NT环境下。

您可以使用CreateDIBitmap和SetDIBits把DIB转化为DDB。现在，显示位图可以使用快速的BitBlt和StretchBlt函数来进行了。然而，您不能直接存取这些与设备无关的像素位。

CreateDIBSection是一个很好的折衷方案。在Windows NT下通过BitBlt和StretchBlt使用位图句柄比使用SetDIBitsToDevice和StretchDIBits（但没有DDB的缺陷）会得到更好的效能。您仍然可以存取DIB像素位。

下一章，在讨论「Windows调色盘管理器」之后会进入位图的探索。

第十六章 调色盘管理器

如果硬件允许，本章就没有存在的必要。尽管许多现代的显示卡提供24位颜色（也称「true color」或「数百万色」）或16位颜色（「增强色」或「数万种颜色」），一些显示卡 – 尤其是在便携式计算机上或高分辨率模式中 – 每个像素只允许8位。这意味着仅有256种颜色。

我们用256种颜色能做什么呢？很明显，要显示真实世界的图像，仅16种颜色是不够的，至少要使用数千或数百万种颜色，256种颜色位于中间状态。是的，用256种颜色来显示真实世界的图像足够了，但需要根据特定的图像来指定这些颜色。这意味着操作系统不能简单地选择「标准」系列的256种颜色，就希望它们对每个应用程序都是理想的颜色。

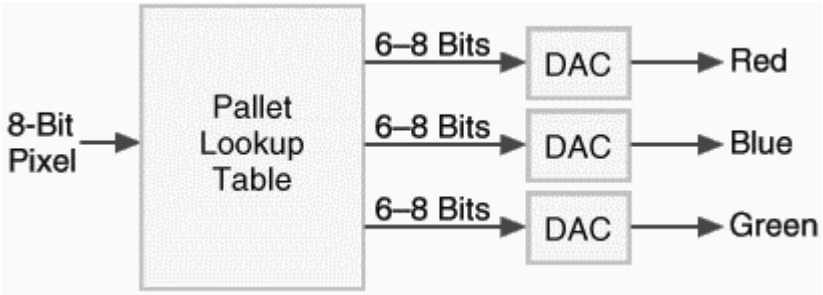
这就是Windows调色盘管理器所要涉及的全部内容。它用于指定程序在8位显示模式下执行时所需要的颜色。如果知道程序肯定不会在8位显示模式下执行，那么您也不需要使用调色盘管理器。不过，由于补充了位图的一些细节，所以本章还是包含重要信息的。

使用调色盘

传统上讲，调色盘是画家用来混合颜色的板子。这个词也可以指画家在绘画过程中使用的所有颜色。在计算机图形中，调色盘是在图形输出设备（例如视讯显示器）上可用的颜色范围。这个名词也可以指支持256色模式的显示卡上的对照表。

视频硬件

显示卡上的调色盘对照表运作过程如下图所示：



在8位显示模式中，每个像素占8位。像素值查询包含256RGB值的对照表的地址。这些RGB值可以正好24位宽，或者小一点，通常是18位宽（即主要的红、绿和蓝各6位）。每种颜色的值都输入到数字模拟转换器，以得到发送给监视器的红、绿和蓝三个模拟信号。

通常，软件可以用任意值来加载调色盘对照表，但这对设备无关的窗口接口，例如Microsoft Windows，会有一些干扰。首先，Windows必须提供软件接口，以便在不直接干扰硬件的情况下，应用程序就可以存取调色盘管理器。第二个问题更严重：因为所有的应用程序都共享同一个视讯显示器，而且同时执行，所以一个应用程序使用了调色盘对照表可能会影响其它程序的使用。

这时就需要使用Windows调色盘管理器（在Windows 3.0中提出）了。Windows保留了256种颜色中的20种，而允许应用程序修改其余的236种。（在某些情况下，应用程序最多可以改变256

种颜色中的254种 – 只有黑色和白色除外 – 但这有一点麻烦)。Windows为系统保留的20种颜色(有时称为20种「静态」颜色)如表16-1所示。

表16-1 256种颜色显示模式中的20种保留的颜色

像素位	RGB值	颜色名称	像素位	RGB值	颜色名称
00000000	00 00 00	黑	11111111	FF FF FF	白
00000001	80 00 00	暗红	11111110	00 FF FF	青
00000010	00 80 00	暗绿	11111101	FF 00 FF	洋红
00000011	80 80 00	暗黄	11111100	00 00 FF	蓝
00000100	00 00 80	暗蓝	11111011	FF FF 00	黄
00000101	80 00 80	暗洋红	11111010	00 FF 00	绿
00000110	00 80 80	暗青	11111001	FF 00 00	红
00000111	C0 C0 C0	亮灰	11111000	80 80 80	暗灰
00001000	C0 DC C0	美元绿	11110111	A0 A0 A4	中性灰
00001001	A6 CA F0	天蓝	11110110	FF FB F0	乳白色

在256种颜色显示模式下执行时,由Windows维护系统调色盘,此调色盘与显示卡上的硬件调色盘对照表相同。内定的系统调色盘如表16-1所示。应用程序可以通过指定「逻辑调色盘(logical palettes)」来修改其余236种颜色。如果有多个应用程序使用逻辑调色盘,那么Windows就给活动窗口最高优先权(我们知道,活动窗口有高亮显示标题栏,并且显示在其它所有窗口的前面)。我们将用一些简单的范例程序来检查它是如何工作的。

要执行本章其它部分的程序,您可能需要将显示卡切换成256色模式。在桌面上单擎鼠标右键,从菜单中选择「属性」,然后选择「设定」页面标签。

显示灰阶

程序16-1所示的GRAYS1程序没有使用Windows调色盘管理器,而尝试用正常显示的65级种阶作为从黑到白的多种彩色的「来源」。

程序16-1 GRAYS1

GRAYS1.C

```
/*------------------------------------------------------------------------
GRAYS1.C -- Gray Shades
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Grays1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Shades of Gray #1"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int cxClient, cyClient ;
  HBRUSH hBrush ;
  HDC hdc ;
  int i ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    // Draw the fountain of grays

    for (i = 0 ; i < 65 ; i++)
    {
      rect.left = i * cxClient / 65 ;
      rect.top = 0 ;
      rect.right = (i + 1) * cxClient / 65 ;
      rect.bottom = cyClient ;

      hBrush = CreateSolidBrush (RGB(min (255, 4 * i),
        min (255, 4 * i),
        min (255, 4 * i))) ;
      FillRect (hdc, &rect, hBrush) ;
      DeleteObject (hBrush) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
```

```
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在WM_PAINT消息处理期间，程序呼叫了65次FillRect函数，每次都使用不同灰阶建立的画刷。灰阶值是RGB值（0,0,0）、（4,4,4）、（8,8,8）等等，直到最后一个值（255,255,255）。最后一个值来自CreateSolidBrush函数中的min宏。

如果在256色显示模式下执行该程序，您将看到从黑到白的65种灰阶，而且它们几乎都用混色着色。纯颜色只有黑色、暗灰色（128,128,128）、亮灰色（192,192,192）和白色。其它颜色是混合了这些纯颜色的多位模式。如果我们在显示行或文字，而不是用这65种灰阶填充区域，Windows将不使用混色而只使用这四种纯色。如果我们正在显示位图，则图像将用20种标准Windows颜色近似。这时正如同您在执行最后一章中的程序的同时又加载了彩色或灰阶DIB所见到的一样。通常，Windows在位图中不使用混色。

程序16-2所示的GRAYS2程序用较少的外部程序代码验证了调色盘管理器中最重要的函数和消息。

程序16-2 GRAYS2

GRAYS2.C

```
/*------------------------------------------------------------------------
GRAYS2.C -- Gray Shades Using Palette Manager
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Grays2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (! RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Shades of Gray #2"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
```

```
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  HBRUSH hBrush ;
  HDC hdc ;
  int i ;
  LOGPALETTE * plp ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    // Set up a LOGPALETTE structure and create a palette

    plp = (LOGPALETTE*)malloc(sizeof (LOGPALETTE) + 64 * sizeof (PALETTEENTRY)) ;

    plp->palVersion = 0x0300 ;
    plp->palNumEntries = 65 ;

    for (i = 0 ; i < 65 ; i++)
    {
      plp->palPalEntry[i].peRed = (BYTE) min (255, 4 * i) ;
      plp->palPalEntry[i].peGreen = (BYTE) min (255, 4 * i) ;
      plp->palPalEntry[i].peBlue = (BYTE) min (255, 4 * i) ;
      plp->palPalEntry[i].peFlags = 0 ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    // Select and realize the palette in the device context

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    // Draw the fountain of grays

    for (i = 0 ; i < 65 ; i++)
    {
      rect.left = i * cxClient / 64 ;
      rect.top = 0 ;
      rect.right = (i + 1) * cxClient / 64 ;
      rect.bottom = cyClient ;

      hBrush = CreateSolidBrush (PALETTERGB( min (255, 4 * i),
        min (255, 4 * i),
        min (255, 4 * i))) ;
      FillRect (hdc, &rect, hBrush) ;
      DeleteObject (hBrush) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
```

```
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

  case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
      break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

  case WM_DESTROY:
    DeleteObject (hPalette) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

通常，使用调色盘管理器的第一步就是呼叫CreatePalette函数来建立逻辑调色盘。逻辑调色盘包含程序所需要的全部颜色－即236种颜色。GRAYS1程序在WM_CREATE消息处理期间处理此作业。它初始化LOGPALETTE（「logical palette：逻辑调色盘」）结构的字段，并将这个结构的指针传递给CreatePalette函数。CreatePalette传回逻辑调色盘的句柄，并将此句柄储存在静态变量hPalette中。

LOGPALETTE结构定义如下：

```
typedef struct
{
  WORD  palVersion ;
  WORD  palNumEntries ;
  PALETTEENTRY palPalEntry[1] ;
}
LOGPALETTE, * PLOGPALETTE ;
```

第一个字段通常设为0x0300，表示兼容Windows 3.0。第二个字段设定为调色盘表中的项目数。LOGPALETTE结构中的第三个字段是一个PALETTEENTRY结构的数组，此结构也是一个调色盘项目。PALETTEENTRY结构定义如下：

```
typedef struct
{
  BYTE peRed ;
  BYTE peGreen ;
  BYTE peBlue ;
  BYTE peFlags ;
}
PALETTEENTRY, * PPALETTEENTRY ;
```

每个PALETTEENTRY结构都定义了一个我们要在调色盘中使用的RGB颜色值。

注意，LOGPALETTE中只能定义一个PALETTEENTRY结构的数组。您需要为LOGPALETTE结构和附加的PALETTEENTRY结构配置足够大的内存空间。GRAYS2需要65种灰阶，因此它为LOGPALETTE结构和64个附加的PALETTEENTRY结构配置了足够大的内存空间。GRAYS2将

palNumEntries字段设定为65，然后从0到64循环，计算灰阶等级（一般是循环索引的4倍，但不超过255），将结构中的peRed、peGreen和peBlue字段设定为此灰阶等级。peFlags字段设为0。程序将指向这个内存块的指针传递给CreatePalette，在一个静态变量中储存该调色盘句柄，然后释放内存。

逻辑调色盘是GDI对象。程序应该删除它们建立的所有逻辑调色盘。WndProc透过在WM_DESTROY消息处理期间呼叫DeleteObject，仔细地删除了逻辑调色盘。

注意逻辑调色盘是独立的设备内容。在真正使用之前，必须确保将其选进设备内容。在WM_PAINT消息处理期间，SelectPalette将逻辑调色盘选进设备内容。除了含有第三个参数以外，此函数与SelectObject函数相似。通常，第三个参数设为FALSE。如果SelectPalette的第三个参数设为TRUE，那么调色盘将始终是「背景调色盘」，这意味着当其它所有程序都显现了各自的调色盘之后，该调色盘才可以获得仍位于系统调色盘中的一个未使用项目。

在任何时候都只有一个逻辑调色盘能选进设备内容。函数将传回前一个选进设备内容的逻辑调色盘句柄。如果您希望将此逻辑调色盘重新选进设备内容，则可以储存此句柄。

通过将颜色映像到系统调色盘，RealizePalette函数使Windows在设备内容中「显现」逻辑调色盘，而系统调色盘是与显示卡实际的实际调色盘相对应。实际工作在此函数呼叫期间进行。Windows必须决定呼叫函数的窗口是活动的还是非活动的，并尽可能将系统调色盘已改变通知给其它窗口（我们将简要说明一下通知的程序）。

回忆一下GRAYS1，它用RGB宏来指定纯色画刷的颜色。RGB宏建构一个32位长整数（记作COLORREF值），其中高字节是0，3个低字节是红、绿和蓝的亮度。

使用Windows调色盘管理器的程序可以继续使用RGB颜色值来指定颜色。不过，这些RGB颜色值将不能存取逻辑调色盘中的附加颜色。它们的作用与没有使用调色盘管理器相同。要在逻辑调色盘中使用附加的颜色，就要用到PALETTERGB宏。除了COLORREF值的高字节设为2而不是0以外，「调色盘RGB」颜色与RGB颜色很相似。

下面是重要的规则：

为了使用逻辑调色盘中的颜色，请用调色盘RGB值或调色盘索引来指定（我将简要讨论调色盘索引）。不要使用常规的RGB值。如果使用了常规的RGB值，您将得到一种标准颜色，而不是逻辑调色盘中的颜色。

没有将调色盘选进设备内容时，不要使用调色盘RGB值或调色盘索引。

尽管可以使用调色盘RGB值来指定逻辑调色盘中没有的颜色，但您还是要从逻辑调色盘获得颜色。

例如，在GRAYS2中处理WM_PAINT期间，当您选择并显现了逻辑调色盘之后，如果试图显示红色，则将显示灰阶。您必须用RGB颜色值来选择不在逻辑调色盘中的颜色。

注意，GRAYS2从不检查视讯显示驱动程序是否支持调色盘管理程序。在不支持调色盘管理程序的显示模式（即所有非256种颜色的显示模式）下执行GRAYS2时，GRAYS2的功能与GRASY1相同。

调色盘信息

如果程序在逻辑调色盘中指定一种颜色，该颜色又是20种保留颜色之一，那么Windows将把逻辑调色盘项目映像给该颜色。另外，如果两个或多个应用程序都在它们的逻辑调色盘中指定了同一种颜色，那么这些应用程序将共享系统调色盘项目。程序可以通过将PALETTEENTRY结构的peFlags字段指定为常数PC_NOCOLLAPSE来忽略该内定状态（其余两个可能的标记是

PC_EXPLICIT（用于显示系统调色盘）和PC_RESERVED（用于调色盘动画），我将在本章的后面展示这两个标记）。

要帮助组织系统调色盘，Windows调色盘管理器含有两个发送给主窗口的消息。

第一个是QM_QUERYNEWPALETTE。当主窗口活动时，该消息发送给主窗口。如果程序在您的窗口上绘画时使用了调色盘管理器，则它必须处理该消息。GRAYS2展示具体的作法。程序获得设备内容句柄，并选进调色盘，呼叫RealizePalette，然后使窗口失效以产生WM_PAINT消息。如果显现了逻辑调色盘，则窗口消息处理程序从该消息传回TRUE，否则传回FALSE。

当系统调色盘改成与WM_QUERYNEWPALETTE消息的结果相同时，Windows将WM_PALETTECHANGED消息发送给由目前活动的窗口来启动并终止处理窗口链的所有主窗口。这允许前台窗口有优先权。传递给窗口消息处理程序的wParam值是活动窗口的句柄。只有当wParam不等于程序的窗口句柄时，使用调色盘管理器的程序才会处理该消息。

通常，在处理WM_PALETTECHANGED时，使用自订调色盘的任何程序都呼叫SelectPalette和RealizePalette。后续的窗口在消息处理期间呼叫RealizePalette时，Windows首先检查逻辑调色盘中的RGB颜色是否与已加载到系统调色盘中的RGB颜色相匹配。如果两个程序需要相同的颜色，那么这两个程序就共同使用一个系统调色盘项目。接下来，Windows检查未使用的系统调色盘项目。如果都已使用，则逻辑调色盘中的颜色从20种保留项目映像到最近的颜色。

如果不关心程序非活动时显示区域的外观，那么您不必处理WM_PALETTECHANGED消息。否则，您有两个选择。GRAYS2显示其中之一：在处理WM_QUERYNEWPALETTE消息时，它获得设备内容，选进调色盘，然后呼叫RealizePalette。这时就可以在处理WM_QUERYNEWPALETTE时呼叫InvalidateRect了。相反地，GRAYS2呼叫UpdateColors。这个函数通常比重新绘制窗口更有效，同时它改变窗口中像素的值来帮助保护以前的颜色。

使用调色盘管理器的许多程序都将让WM_QUERYNEWPALETTE和WM_PALETTECHANGED消息用GRAYS2所显示的方法来处理。

调色盘索引方法

程序16-3所示的GRAYS3程序与GRAYS2非常相似，只是在处理WM_PAINT期间使用了呼叫PALETTEINDEX的宏，而不是PALETTERGB。

程序16-3 GRAYS3

GRAYS3.C

```
/*------------------------------------------------------------------------
GRAYS3.C -- Gray Shades Using Palette Manager
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Grays3") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Shades of Gray #3"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  HBRUSH hBrush ;
  HDC hdc ;
  int i ;
  LOGPALETTE * plp ;
  PAINTSTRUCT ps ;
  RECT rect ;
  switch (message)
  {
  case WM_CREATE:
    // Set up a LOGPALETTE structure and create a palette

    plp = (LOGPALETTE*)malloc (sizeof (LOGPALETTE) + 64 * sizeof (PALETTEENTRY)) ;

    plp->palVersion = 0x0300 ;
    plp->palNumEntries = 65 ;

    for (i = 0 ; i < 65 ; i++)
    {
      plp->palPalEntry[i].peRed = (BYTE) min (255, 4 * i) ;
      plp->palPalEntry[i].peGreen = (BYTE) min (255, 4 * i) ;
      plp->palPalEntry[i].peBlue = (BYTE) min (255, 4 * i) ;
      plp->palPalEntry[i].peFlags = 0 ;
    }
    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
```

```
    // Select and realize the palette in the device context

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    // Draw the fountain of grays

    for (i = 0 ; i < 65 ; i++)
    {
      rect.left = i * cxClient / 64 ;
      rect.top = 0 ;
      rect.right = (i + 1) * cxClient / 64 ;
      rect.bottom = cyClient ;

      hBrush = CreateSolidBrush (PALETTEINDEX (i)) ;

      FillRect (hdc, &rect, hBrush) ;
      DeleteObject (hBrush) ;
    }

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, FALSE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

  case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
      break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

「调色盘」索引的颜色不同于调色盘RGB颜色，其高字节是1，而低字节的值是目前在设备内容中选择的、逻辑调色盘中的索引。在GRAYS3中，逻辑调色盘有65个项目，用于这些项目的索引从0到64。值

PALETTEINDEX (0)

指黑色，

PALETTEINDEX (32)

指灰色，而

PALETTEINDEX (64)

指白色。

因为Windows不需要执行最近颜色的搜索，所以使用调色盘索引比使用RGB值更有效。

查询调色盘支持

您可以容易地验证：当Windows在16位或24位显示模式下执行时，GRAYS2和GRAYS3程序执行良好。但是在某些情况下，要使用调色盘管理器的Windows应用程序可能要先确定设备驱动程序是否支持它。这时，您可以呼叫GetDeviceCaps，并以视讯显示的设备内容句柄和PASTERCAPS作为参数。函数将传回由一系列旗标组成的整数。通过在传回值和常数RC_PALETTE之间执行位操作来检验支持的调色盘：

RC_PALETTE & GetDeviceCaps (hdc, RASTERCAPS)

如果此值非零，则视讯显示器设备驱动程序将支持调色盘操作。在这种情况之下，来自GetDeviceCaps的其它三个重要项目也是可用的。函数呼叫

GetDeviceCaps (hdc, SIZEPALETTE)

将传回在显示卡上调色盘表的总尺寸。这与同时显示的颜色总数相同。因为调色盘管理器只用于每像素8位的视讯显示模式，所以此值将是256。

函数呼叫

GetDeviceCaps (hdc, NUMRESERVED)

传回在调色盘表中的颜色数，该表是设备驱动程序为系统保留的，此值是20。不呼叫调色盘管理器，这些只是Windows应用程序在256色显示模式下使用的纯色。要使用其余的236种颜色，程序必须使用调色盘管理器函数。

一个附加项目也可用：

GetDeviceCaps (hdc, COLORRES)

此值告诉您加载到硬件调色盘表的RGB颜色值分辨率（以位计）。这些是进入数字模拟转换器的位。某些视讯显示卡只使用6位ADC，所以该值是18。其余使用8位的ADC，所以值是24。

Windows程序注意颜色分辨率并因此采取一些动作是很有用的。例如，如果该颜色分辨率是18，那么程序将不可能要求到128种灰阶，因为只有64个离散的灰阶可用。要求到128种灰阶就不必用多余的项目来填充硬件调色盘表。

系统调色盘

我在前面提过，Windows系统调色盘直接与显示卡上的硬件调色盘查询表相符（然而，硬件调色盘查询表可能比系统调色盘的颜色分辨率低）。程序可以通过呼叫下面的函数来获得系统调色盘中的某些或全部的RGB项目：

GetSystemPaletteEntries (hdc, uStart, uNum, &pe) ;

只有显示卡模式支持调色盘操作时，该函数才能执行。第二个和第三个参数是无正负号整数，显示第一个调色盘项目的索引和调色盘项目数。最后一个参数是指向PALETTEENTRY型态的指针。

您可以在几种情况下使用该函数。程序可以定义PALETTEENTRY结构如下：

PALETTEENTRY pe ;

然后可按下面的方法多次呼叫GetSystemPaletteEntries：

GetSystemPaletteEntries (hdc, i, 1, &pe) ;

其中的i从0到某个值，该值小于从GetDeviceCaps（带有SIZEPALETTE索引255）传回的值。或者，程序要获得所有的系统调色盘项目，可以通过定义指向PALETTEENTRY结构的指针，然后重新配置足够的内存块，以储存与调色盘大小指定同样多的PALETTEENTRY结构。

GetSystemPaletteEntries函数确实允许您检验硬件调色盘表。系统调色盘中的项目按像素值增加的顺序排列，这些值用于表示视讯显示缓冲区中的颜色。我将简单地讨论一下具体作法。

其它调色盘函数

我们在前面看过，Windows程序能够改变系统调色盘，但只是间接改变：第一步建立逻辑调色盘，它基本上是程序要使用的RGB颜色值数组。CreatePalette函数不会导致系统调色盘或者显示卡调色盘表的任何变化。逻辑调色盘必须在任何事情发生之前就选进设备内容并显现。

程序可以通过呼叫

GetPaletteEntries (hPalette, uStart, uNum, &pe) ;

来查询逻辑调色盘中的RGB颜色值。您可以按使用GetSystemPaletteEntries的方法来使用此函数。但是要注意，第一个参数是逻辑调色盘的句柄，而不是设备内容的句柄。

建立逻辑调色盘以后，让您改变其中的值的相应函数是：

SetPaletteEntries (hPalette, uStart, uNum, &pe) ;

另外，记住呼叫此函数不引起系统调色盘的任何变化 – 即使目前调色盘选进了设备内容。此函数也不改变逻辑调色盘的尺寸。要改变逻辑调色盘的尺寸，请使用ResizePalette。

下面的函数接受RGB颜色引用值作为最后的参数，并将索引传回给逻辑调色盘，该逻辑调色盘与和它最接近的RGB颜色值相对应：

iIndex = GetNearestPaletteIndex (hPalette, cr) ;

第二个参数是COLORREF值。如果希望的话，呼叫GetPaletteEntries就可以获得逻辑调色盘中实际的RGB颜色值。

如果程序在8位显示模式下需要多于236种自订颜色，则可以呼叫GetSystemPaletteUse。这允许程序设定254种自订颜色；系统仅保留黑色和白色。不过，程序仅在最大化充满全屏幕时才允许这样，而且它还将一些系统颜色设为黑色和白色，以便标题栏和菜单等仍然可见。

位映像操作问题

从第五章可以了解到，GDI允许使用不同的「绘画模式」或「位映像操作」来画线并填充区域。用SetROP2设定绘画模式，其中的「2」表示两个对象之间的二元（binary）位映像操作。三元位映像操作用于处理BitBlt和类似功能。这些位映像操作决定了正在画的对象像素与表面像素的结合方式。例如，您可以画一条直线，以便在线的像素与显示的像素按位异或的方式相结合。

位映像操作就是在像素位上照着各个位的顺序进行操作。改变调色盘会影响到这些位映像操作。位映像操作的操作对象是像素位，而这些像素位可能与实际颜色没有关联。

透过执行GRAYS2或GRAYS3程序，您自己就可以得出这个结论。调整尺寸时，拖动顶部或底部的边界穿过窗口，Windows利用反转背景像素位的位映像操作来显示拖动尺寸的边界，其目的是使拖动尺寸边界总是可见的。但在GRAYS2和GRAYS3程序中，您将看到各种随机变换的颜色，这些颜色恰好与对应于调色盘表中未使用的项目，那是反转显示像素位的结果。可视颜色没有反转 – 只有像素位反转了。

正如您在表16-1中所看到的一样，20种标准保留颜色位于系统调色盘的顶部和底部，以便位映像操作的结果仍然正常。然而，一旦您开始修改调色盘 – 尤其是替换了保留颜色 – 那么颜色对象的位映像操作就变得没有意义了。

唯一保证的是位映像操作将用黑色和白色运作。黑色是系统调色盘中的第一个项目（所有的像素位都设为0），而白色是最后的项目（所有的像素位都设为1）。这两个项目不能改变。如果需要预知在颜色对象上进行位映像操作的结果，则可以先获得系统调色盘表，然后查看不同像素位值的RGB颜色值。

查看系统调色盘

在Windows下执行的程序将处理逻辑调色盘，为使逻辑调色盘更好地服务于所有使用逻辑调色盘的程序，Windows将在系统调色盘中设定颜色。该系统调色盘复制了显示卡的硬件对照表内容。这样，查看系统调色盘有助于调适调色盘应用程序。

因为对于这个问题有三种截然不同的处理方式，所以我将向您展示三个程序，以显示系统调色盘的内容。

SYSPAL1程序，如程序16-4所示，使用了前面所讲的GetSystemPaletteEntries函数。

程序16-4 SYSPAL1

SYSPAL1.C

```
/*------------------------------------------------------------------------
SYSPAL1.C -- Displays system palette
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName [] = TEXT ("SysPal1") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("System Palette #1"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
```

```
  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

BOOL CheckDisplay (HWND hwnd)
{
  HDC hdc ;
  int iPalSize ;

  hdc = GetDC (hwnd) ;
  iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
  ReleaseDC (hwnd, hdc) ;

  if (iPalSize != 256)
  {
    MessageBox (hwnd,TEXT ("This program requires that the video ")
      TEXT ("display mode have a 256-color palette."),
      szAppName, MB_ICONERROR) ;
    return FALSE ;
  }
  return TRUE ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int cxClient, cyClient ;
  static SIZE sizeChar ;
  HDC hdc ;
  HPALETTE hPalette ;
  int i, x, y ;
  PAINTSTRUCT ps ;
  PALETTEENTRY pe [256] ;
  TCHAR szBuffer [16] ;

  switch (message)
  {
  case WM_CREATE:
    if (!CheckDisplay (hwnd))
      return -1 ;
    hdc = GetDC (hwnd) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
    GetTextExtentPoint32 (hdc, TEXT ("FF-FF-FF"), 10, &sizeChar) ;
    ReleaseDC (hwnd, hdc) ;
    return 0 ;

  case WM_DISPLAYCHANGE:
    if (!CheckDisplay (hwnd))
      DestroyWindow (hwnd) ;

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
```

```
    GetSystemPaletteEntries (hdc, 0, 256, pe) ;

    for (i = 0, x = 0, y = 0 ; i < 256 ; i++)
    {
      wsprintf ( szBuffer, TEXT ("%02X-%02X-%02X"),
        pe[i].peRed, pe[i].peGreen, pe[i].peBlue) ;

      TextOut (hdc, x, y, szBuffer, lstrlen (szBuffer)) ;

      if (( x += sizeChar.cx) + sizeChar.cx > cxClient)
      {
        x = 0 ;

        if (( y += sizeChar.cy) > cyClient)
          break ;
      }
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_PALETTECHANGED:
    InvalidateRect (hwnd, NULL, FALSE) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

与SYSPAL系列中的其它程序一样，除非带有SIZEPALETTE参数的GetDeviceCaps传回值为256，否则SYSPAL1不会执行。

注意无论SYSPAL1的显示区域什么时候收到WM_PALETTECHANGED消息，它都是无效的。在合并WM_PAINT消息处理期间，SYSPAL1呼叫GetSystemPaletteEntries，并用一个含256个PALETTEENTRY结构的数组作为参数。RGB值作为文字字符串显示在显示区域。程序执行时，注意20种保留颜色是RGB值列表中的前10个和后10个，这与表16-1所示相同。

当SYSPAL1显示有用的信息时，它与实际看到的256种颜色不同。那就是SYSPAL2的作业，如程序16-5所示。

程序16-5 SYSPAL2

SYSPAL2.C

```
/*------------------------------------------------------------------------
SYSPAL2.C -- Displays system palette
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName [] = TEXT ("SysPal2") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("System Palette #2"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;
  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

BOOL CheckDisplay (HWND hwnd)
{
  HDC hdc ;
  int iPalSize ;

  hdc = GetDC (hwnd) ;
  iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
  ReleaseDC (hwnd, hdc) ;

  if (iPalSize != 256)
  {
    MessageBox (hwnd, TEXT("This program requires that the video ")
      TEXT ("display mode have a 256-color palette."),
      szAppName, MB_ICONERROR) ;
    return FALSE ;
  }
  return TRUE ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  HBRUSH hBrush ;
  HDC hdc ;
  int i, x, y ;
  LOGPALETTE * plp ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    if (!CheckDisplay (hwnd))
      return -1 ;

    plp = (LOGPALETTE*)malloc (sizeof (LOGPALETTE) + 255 * sizeof (PALETTEENTRY)) ;
```

```
    plp->palVersion = 0x0300 ;
    plp->palNumEntries = 256 ;

    for (i = 0 ; i < 256 ; i++)
    {
      plp->palPalEntry[i].peRed = i ;
      plp->palPalEntry[i].peGreen = 0 ;
      plp->palPalEntry[i].peBlue = 0 ;
      plp->palPalEntry[i].peFlags = PC_EXPLICIT ;
    }

    hPalette = CreatePalette (plp) ;
    free (plp) ;
    return 0 ;

  case WM_DISPLAYCHANGE:
    if (!CheckDisplay (hwnd))
      DestroyWindow (hwnd) ;

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    for (y = 0 ; y < 16 ; y++)
      for (x = 0 ; x < 16 ; x++)
      {
        hBrush = CreateSolidBrush (PALETTEINDEX (16 * y + x)) ;
        SetRect (&rect, x * cxClient /16, y * cyClient / 16,
          (x + 1) * cxClient / 16,(y+1) * cyClient / 16);
        FillRect (hdc, &rect, hBrush) ;
        DeleteObject (hBrush) ;
      }
      EndPaint (hwnd, &ps) ;
      return 0 ;

  case WM_PALETTECHANGED:
    if ((HWND) wParam != hwnd)
      InvalidateRect (hwnd, NULL, FALSE) ;

    return 0 ;

  case WM_DESTROY:
    DeleteObject (hPalette) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SYSPAL2在WM_CREATE消息处理期间建立了逻辑调色盘。但是请注意：逻辑调色盘中所有的256个值都是从0到255的调色盘索引，并且peFlags字段是PC_EXPLICIT。该旗标是这样定义的：「逻辑调色盘项目的较低字组指定了一个硬件调色盘索引。此旗标允许应用程序显示硬件调色盘的内容。」该旗标就是专为我们要做的这件事情而设计的。

在WM_PAINT消息处理期间，SYSPAL2将该调色盘选进设备内容并显现它。这不会引起系统调色盘的任何重组，而是允许程序使用PALETTEINDEX宏来指定系统调色盘中的颜色。按此方法，

SYSPAL2显示了256个矩形。另外，当您执行该程序时，注意顶行和底行的前10种和后10种颜色是20种保留颜色，如表16-1所示。当您执行使用自己逻辑调色盘的程序时，显示就改变了。

如果您既喜欢看SYSPAL2中的颜色，又喜欢RGB的值，那么请与第八章的WHATCLR程序同时执行。

SYSPAL系列中的第三版使用的技术对我来说是最近才出现的 – 从我开始研究Windows调色盘管理器七年多后，才出现了那些技术。

事实上，所有的GDI函数都直接或间接地指定颜色作为RGB值。在GDI内部，这将转换成与那个颜色相关的像素位。在某些显示模式中（例如，16位或24位颜色模式），这些转换是相当直接的。在其它显示模式中（4位或8位颜色），这可能涉及最接近颜色的搜索。

然而，有两个GDI函数让您直接指定像素位中的颜色。当然在这种方式中使用的这两个函数都与设备高度相关。它们太依赖设备了，以至于它们可以直接显示视讯显示卡上实际的调色盘对照表。这两个函数是BitBlt和StretchBlt。

程序16-6所示的SYSPAL3程序显示了使用StretchBlt显示系统调色盘中颜色的方法。

程序16-6 SYSPAL3

SYSPAL3.C

```
/*------------------------------------------------------------------------
SYSPAL3.C -- Displays system palette
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("SysPal3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("System Palette #3"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

BOOL CheckDisplay (HWND hwnd)
{
  HDC hdc ;
  int iPalSize ;

  hdc = GetDC (hwnd) ;
  iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
  ReleaseDC (hwnd, hdc) ;

  if (iPalSize != 256)
  {
    MessageBox (hwnd,TEXT("This program requires that the video ")
      TEXT("display mode have a 256-color palette."),
      szAppName, MB_ICONERROR) ;
    return FALSE ;
  }
  return TRUE ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static int cxClient, cyClient ;
  BYTE bits [256] ;
  HDC hdc, hdcMem ;
  int i ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    if (! CheckDisplay (hwnd))
      return -1 ;

    for ( i = 0 ; i < 256 ; i++)
      bits [i] = i ;

    hBitmap = CreateBitmap (16, 16, 1, 8, &bits) ;
    return 0 ;

  case WM_DISPLAYCHANGE:
    if (!CheckDisplay (hwnd))
      DestroyWindow (hwnd) ;

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    hdcMem = CreateCompatibleDC (hdc) ;
    SelectObject (hdcMem, hBitmap) ;

    StretchBlt (hdc, 0, 0, cxClient, cyClient,
      hdcMem, 0, 0, 16, 16, SRCCOPY) ;

    DeleteDC (hdcMem) ;
```

```
    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_DESTROY:
    DeleteObject (hBitmap) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在WM_CREATE消息处理期间，SYSPAL3使用CreateBitmap来建立16×16的每像素8位的位图。该函数的最后一个参数是包括数值0到255的256字节数组。这些是256种可能的像素位值。在处理WM_PAINT消息的程序中，程序将这个位图选进内存设备内容，用StretchBlt来显示并填充该显示区域。Windows仅将位图中的像素位传输到视讯显示器硬件，从而允许这些像素位存取调色盘对照表中的256个项目。程序的显示区域甚至不必使接收WM_PALETTECHANGED消息无效 – 对于对照表的任何修改都会立即影响到SYSPAL3的显示。

调色盘动画

在本节的标题中看到「动画」一词，并开始考虑屏幕周围执行的「计算机宠物」时，您的眼前可能会为之一亮。是的，您可以使用Windows调色盘管理器作一些动画，而且是有一定专业水平的动画。

通常，Windows下的动画就是快速连续地显示一系列位图。调色盘动画与这种方法有很大的区别。您透过在屏幕上绘制您所需要的每件东西开始，然后您处理调色盘来改变这些对象的颜色，可能是画一些相对于屏幕背景来说是不可见的图像。您用这种方法就可以获得动画效果，而不必重画任何东西。调色盘动画的速度是相当快的。

对于调色盘动画，最初的建立工作与我们前面看见的有些不同：对于动画期间要修改的每种RGB颜色值，PALETTEENTRY结构的peFlags字段必须设定为PC_RESERVED。

通常，就像我们所看到的一样，在建立逻辑调色盘时，您将peFlags标记设为0。这允许GDI将多个逻辑调色盘中同样的颜色映像到相同的系统调色盘项目。例如，假设两个Windows程序都建立了包含RGB项目10-10-10的逻辑调色盘，那么在系统调色盘表中，Windows只需要一个10-10-10项目。但如果这两个程序中的一个使用调色盘动画，那您就不要再让GDI使用调色盘了。调色盘动画意味着速度非常快 – 而且如果不重画，它也只可能提高速度。当使用调色盘动画的程序修改调色盘时，它不会影响其它程序，或者迫使GDI重组系统调色盘表。PC_RESERVED的peFlags值为单个逻辑调色盘储存系统调色盘项目。

使用调色盘动画时，通常您可以在WM_PAINT消息处理期间呼叫SelectPalette和RealizePalette，使用PALETTEINDEX宏来指定颜色。该宏将一个索引带进逻辑调色盘表。

对于动画，您可能要通过改变调色盘来响应WM_TIMER消息。要改变逻辑调色盘中的RGB颜色值，请使用一个PALETTEENTRY结构的数组来呼叫函数AnimatePalette。此函数速度很快，因为它只需要改变系统调色盘以及显示卡硬件调色盘表中的项目。

跳动的球

程序16-7显示了BOUNCE程序的组件，但还有一个程序可显示跳动的球。为了简单起见，根据显示区域的大小将球画成了椭圆形。因为本章有几个调色盘动画程序，所以PALANIM.C（「调色盘动画」）文件包含一些通用内容。

程序16-7 BOUNCE

PALANIM.C

```
/*------------------------------------------------------------------------
PALANIM.C -- Palette Animation Shell Program
s(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
extern HPALETTE CreateRoutine (HWND) ;
extern void PaintRoutine (HDC, int, int) ;
extern void TimerRoutine (HDC, HPALETTE) ;
extern void DestroyRoutine (HWND, HPALETTE) ;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

extern TCHAR szAppName [] ;
extern TCHAR szTitle [] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

BOOL CheckDisplay (HWND hwnd)
{
  HDC hdc ;
  int iPalSize ;

  hdc = GetDC (hwnd) ;
  iPalSize = GetDeviceCaps (hdc, SIZEPALETTE) ;
  ReleaseDC (hwnd, hdc) ;
```

```
  if (iPalSize != 256)
  {
    MessageBox (hwnd, TEXT ("This program requires that the video ")
      TEXT ("display mode have a 256-color palette."),
      szAppName, MB_ICONERROR) ;
    return FALSE ;
  }
  return TRUE ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    if (!CheckDisplay (hwnd))
      return -1 ;

    hPalette = CreateRoutine (hwnd) ;
    return 0 ;

  case WM_DISPLAYCHANGE:
    if (!CheckDisplay (hwnd))
      DestroyWindow (hwnd) ;

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;

    PaintRoutine (hdc, cxClient, cyClient) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;
  case WM_TIMER:
    hdc = GetDC (hwnd) ;

    SelectPalette (hdc, hPalette, FALSE) ;

    TimerRoutine (hdc, hPalette) ;

    ReleaseDC (hwnd, hdc) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;
```

```
  case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
      break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;

  case WM_DESTROY:
    DestroyRoutine (hwnd, hPalette) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

BOUNCE.C

```
/*------------------------------------------------------------------------
BOUNCE.C -- Palette Animation Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Bounce") ;
TCHAR szTitle [] = TEXT ("Bounce: Palette Animation Demo") ;

static LOGPALETTE * plp ;
HPALETTE CreateRoutine (HWND hwnd)
{
  HPALETTE hPalette ;
  int i ;

  plp = malloc (sizeof (LOGPALETTE) + 33 * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = 34 ;

  for (i = 0 ; i < 34 ; i++)
  {
    plp->palPalEntry[i].peRed = 255 ;
    plp->palPalEntry[i].peGreen = (i == 0 ? 0 : 255) ;
    plp->palPalEntry[i].peBlue = (i == 0 ? 0 : 255) ;
    plp->palPalEntry[i].peFlags = (i == 33 ? 0 : PC_RESERVED) ;
  }
  hPalette = CreatePalette (plp) ;
  SetTimer (hwnd, ID_TIMER, 50, NULL) ;
  return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
  HBRUSH hBrush ;
  int i, x1, x2, y1, y2 ;
  RECT rect ;
  // Draw window background using palette index 33

  SetRect (&rect, 0, 0, cxClient, cyClient) ;
  hBrush = CreateSolidBrush (PALETTEINDEX (33)) ;
  FillRect (hdc, &rect, hBrush) ;
  DeleteObject (hBrush) ;

  // Draw the 33 balls
  SelectObject (hdc, GetStockObject (NULL_PEN)) ;
  for (i = 0 ; i < 33 ; i++)
```

```
  {
    x1 = i * cxClient / 33 ;
    x2 = (i + 1)* cxClient / 33 ;

    if (i < 9)
    {
      y1 = i * cyClient / 9 ;
      y2 = (i + 1) * cyClient / 9 ;
    }
    else if (i < 17)
    {
      y1 = (16 - i) * cyClient / 9 ;
      y2 = (17 - i) * cyClient / 9 ;
    }
    else if (i < 25)
    {
      y1 = (i - 16) * cyClient / 9 ;
      y2 = (i - 15) * cyClient / 9 ;
    }
    else
    {
      y1 = (32 - i) * cyClient / 9 ;
      y2 = (33 - i) * cyClient / 9 ;
    }

    hBrush = CreateSolidBrush (PALETTEINDEX (i)) ;
    SelectObject (hdc, hBrush) ;
    Ellipse (hdc, x1, y1, x2, y2) ;
    DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
  }
  return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
  static BOOL bLeftToRight = TRUE ;
  static int iBall ;

  // Set old ball to white
  plp->palPalEntry[iBall].peGreen = 255 ;
  plp->palPalEntry[iBall].peBlue = 255 ;

  iBall += (bLeftToRight ? 1 : -1) ;
  if ( iBall == (bLeftToRight ? 33 : -1))
  {
    iBall = (bLeftToRight ? 31 : 1) ;
    bLeftToRight ^= TRUE ;
  }

  // Set new ball to red

  plp->palPalEntry[iBall].peGreen = 0 ;
  plp->palPalEntry[iBall].peBlue = 0 ;

  // Animate the palette

  AnimatePalette (hPalette, 0, 33, plp->palPalEntry) ;
  return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
  KillTimer (hwnd, ID_TIMER) ;
  DeleteObject (hPalette) ;
  free (plp) ;
  return ;
}
```

除非Windows处于支持调色盘的显示模式下，否则调色盘动画将不能工作。因此，PALANIM.C通过呼叫CheckDisplay函数（与SYSPAL程序中的函数相同）来开始处理WM_CREATE。

PALANIM.C呼叫BOUNCE.C中的四个函数：在WM_CREATE消息处理期间呼叫CreateRoutine（在BOUNCE中用于建立逻辑调色盘）；在WM_PAINT消息处理期间呼叫PaintRoutine；在WM_TIMER消息处理期间呼叫TimerRoutine；在WM_DESTROY消息处理期间呼叫DestroyRoutine（在BOUNCE中用于清除）。在呼叫PaintRoutine和TimerRoutine之前，PALANIM.C获得设备内容，并将其选进逻辑调色盘。在呼叫PaintRoutine之前，它也显现调色盘。PALANIM.C期望TimerRoutine呼叫AnimatePalette。尽管AnimatePalette需要从设备内容中选择调色盘，但它不需要呼叫RealizePalette。

BOUNCE中的球按「W」路线在显示区域中来回跳动。显示区域背景是白色，球是红色。任何时候，都可以在33个不重迭的位置之一看见球。这需要34个调色盘项目：一个用于背景，其它33个用于不同位置的球。在CreateRoutine中，BOUNCE初始化PALETTEENTRY结构的一个数组，将第一个调色盘项目（与球在左上角的位置对应）设定为红色，其它的设定为白色。注意，对于除背景以外的所有项目，peFlags字段都设定为PC_RESERVED（背景是最后的一个调色盘项目）。BOUNCE通过将Windows定时器的间隔设定为50毫秒来终止CreateRoutine。

BOUNCE在PaintRoutine完成所有的绘画工作。窗口背景用一个实心画刷和调色盘索引33所指定的颜色来绘制。33个球的颜色是依据从0到32的调色盘索引的颜色。当BOUNCE第一次在显示区域内绘画时，0的调色盘索引映像成红色，其它调色盘索引映像到白色。这导致球出现在左上角。

当WndProc处理WM_TIMER消息并呼叫TimerRoutine时，动画就发生了。TimerRoutine通过呼叫AnimatePalette来结束，语法如下：

```
AnimatePalette (hPalette, uStart, uNum, &pe) ;
```

其中，第一个参数是调色盘句柄，最后一个参数是指向数组的指针，该数组由一个或多个PALETTEENTRY结构组成。该函数改变逻辑调色盘中从uStart项目到uNum项目之间的若干项目。逻辑调色盘中新的uStart项目是PALETTEENTRY结构中的第一个成员。当心！uStart参数是进入原始逻辑调色盘表的索引，而不是进入PALETTEENTRY数组的索引。

为了方便起见，BOUNCE使用PALETTEENTRY结构的数组，该结构是建立逻辑调色盘时使用的LOGPALETTE结构的一部分。球的目前位置（从0到32）储存在静态变量iBall中。在TimerRoutine期间，BOUNCE将PALETTEENTRY成员设为白色。然后计算球的下一个位置，并将该元素设为红色。用下面的呼叫来改变调色盘：

```
AnimatePalette (hPalette, 0, 33, plp->palPalEntry) ;
```

GDI改变33逻辑调色盘项目中的第一个（尽管实际上只改变了两个），使它与系统调色盘表中的变化相对应，然后修改显示卡上的硬件调色盘表。这样，不用重画球就开始移动了。

BOUNCE执行时，您会发现同时执行SYSPAL2或SYSPAL3效果会更好。

尽管AnimatePalette执行得非常快，但是当只有一两个项目改变时，您还应该尽量避免改变所有的逻辑调色盘项目。这在BOUNCE中有点复杂，因为球要来回地跳 – iBall要先增加，然后再减少。一种方法是使用两个变量：分别称为iBallOld（设定球的目前位置）和iBallMin（iBall和iBallOld中较小的）。然后您就可以像下面这样呼叫AnimatePalette来改变两个项目了：

```
iBallMin = min (iBall, iBallOld) ;

AnimatePalette (hPal, iBallMin, 2, plp->palPalEntry + iBallMin) ;
```

还有另一种方法：我们先假定您定义了一个PALETTEENTRY结构：

PALETTEENTRY pe ;

在TimerRoutine期间，您将PALETTEENTRY字段设为白色，并呼叫AnimatePalette来改变逻辑调色盘中iBall位置的一个项目：

```
pe.peRed = 255 ;
pe.peGreen = 255 ;
pe.peBlue = 255 ;
pe.peFlags = PC_RESERVED ;
AnimatePalette (hPalette, iBall, 1, &pe) ;
```

然后计算显示在BOUNCE中的iBall的新值，将PALETTEENTRY结构的字段定义为红色，然后再次呼叫AnimatePalette：

```
pe.peRed = 255 ;
pe.peGreen = 0 ;
pe.peBlue = 0 ;
pe.peFlags = PC_RESERVED ;
AnimatePalette (hPalette, iBall, 1, &pe) ;
```

尽管跳动的球是对动画的一个传统的简单说明，但它实际上并不适合调色盘动画，因为必须先画出球的所有可能位置。调色盘动画更适合于显示运动的重复图案。

一个项目的调色盘动画

调色盘动画中一个更有趣的方面就是，可以只使用一个调色盘项目来完成一些有趣的技术。例如程序16-8所示的FADER程序。这个程序也需要前面的PALANIM.C文件。

程序16-8 FADER

FADER.C

```
/*------------------------------------------------------------------------
FADER.C -- Palette Animation Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Fader") ;
TCHAR szTitle [] = TEXT ("Fader: Palette Animation Demo") ;

static LOGPALETTE lp ;
HPALETTE CreateRoutine (HWND hwnd)
{
  HPALETTE hPalette ;
  lp.palVersion = 0x0300 ;
  lp.palNumEntries = 1 ;
  lp.palPalEntry[0].peRed = 255 ;
  lp.palPalEntry[0].peGreen = 255 ;
  lp.palPalEntry[0].peBlue = 255 ;
  lp.palPalEntry[0].peFlags = PC_RESERVED ;

  hPalette = CreatePalette (&lp) ;
  SetTimer (hwnd, ID_TIMER, 50, NULL) ;
  return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
  static TCHAR szText [] = TEXT (" Fade In and Out ") ;
  int x, y ;
  SIZE sizeText ;

  SetTextColor (hdc, PALETTEINDEX (0)) ;
  GetTextExtentPoint32 (hdc, szText, lstrlen (szText), &sizeText) ;
```

```
  for (x = 0 ; x < cxClient ; x += sizeText.cx)
    for (y = 0 ; y < cyClient ; y += sizeText.cy)
    {
      TextOut (hdc, x, y, szText, lstrlen (szText)) ;
    }

    return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
  static BOOL bFadeIn = TRUE ;
  if (bFadeIn)
  {
    lp.palPalEntry[0].peRed -= 4 ;
    lp.palPalEntry[0].peGreen -= 4 ;

    if ( lp.palPalEntry[0].peRed == 3)
      bFadeIn = FALSE ;
  }
  else
  {
    lp.palPalEntry[0].peRed += 4 ;
    lp.palPalEntry[0].peGreen += 4 ;

    if (lp.palPalEntry[0].peRed == 255)
      bFadeIn = TRUE ;
  }

  AnimatePalette (hPalette, 0, 1, lp.palPalEntry) ;
  return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
  KillTimer (hwnd, ID_TIMER) ;
  DeleteObject (hPalette) ;
  return ;
}
```

FADER在显示区域上显示满了文字字符串「Fade In And Out」。文字首先显示为白色，这对于白色背景的窗口来说是看不出来的。通过使用调色盘动画，FADER慢慢地将文字的颜色改为蓝色，然后再改回白色，这样一遍一遍地重复。文字就有渐现渐隐的显示效果了。

FADER用CreateRoutine函数建立了逻辑调色盘，它只需要一个调色盘项目，并将颜色初始化为白色－红色、绿色和蓝色值都设为255。在PaintRoutine中（您可能想起，当逻辑调色盘选进设备内容并显现以后，PALANIM呼叫过此函数），FADER呼叫SetTextColor将文字颜色设定为PALETTEINDEX(0)。这意味着文字颜色设定为调色盘表格中的第一个项目，此项目初始为白色。然后FADER用「Fade In And Out」文字字符串填充显示区域。这时，窗口背景是白色，文字也是白色，所以文字不可见。

在TimerRoutine函数中，FADER通过改变PALETTEENTRY结构并将其传递给AnimatePalette来完成调色盘动画。最初，对每一个WM_TIMER消息，程序都将红色和绿色值减4，直到等于3；然后将这些值加4，直到等于255。这将使文字颜色逐渐从白色变到蓝色，然后又回到白色。

程序16-9所示的ALLCOLOR程序只用了逻辑调色盘的一个项目来显示显示卡可以着色的所有颜色。当然，程序不是同时显示这些颜色，而是连续显示。如果显示卡有18位的分辨率（这时能有262144种不同的颜色），那么在两种颜色间隔55毫秒的速度下，只需要4小时就可以在屏幕上看到所有的颜色。

程序16-9 ALLCOLOR

ALLCOLOR.C

```
/*------------------------------------------------------------------------
ALLCOLOR.C -- Palette Animation Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1

TCHAR szAppName [] = TEXT ("AllColor") ;
TCHAR szTitle [] = TEXT ("AllColor: Palette Animation Demo") ;

static int iIncr ;
static PALETTEENTRY pe ;

HPALETTE CreateRoutine (HWND hwnd)
{
  HDC hdc ;
  HPALETTE hPalette ;
  LOGPALETTE lp ;

  // Determine the color resolution and set iIncr
  hdc = GetDC (hwnd) ;
  iIncr = 1 << (8 - GetDeviceCaps (hdc, COLORRES) / 3) ;
  ReleaseDC (hwnd, hdc) ;

  // Create the logical palette
  lp.palVersion = 0x0300 ;
  lp.palNumEntries = 1 ;
  lp.palPalEntry[0].peRed = 0 ;
  lp.palPalEntry[0].peGreen = 0 ;
  lp.palPalEntry[0].peBlue = 0 ;
  lp.palPalEntry[0].peFlags = PC_RESERVED ;

  hPalette = CreatePalette (&lp) ;
  // Save global for less typing
  pe = lp.palPalEntry[0] ;
  SetTimer (hwnd, ID_TIMER, 10, NULL) ;
  return hPalette ;
}

void DisplayRGB (HDC hdc, PALETTEENTRY * ppe)
{
  TCHAR szBuffer [16] ;
  wsprintf (szBuffer, TEXT (" %02X-%02X-%02X "),
    ppe->peRed, ppe->peGreen, ppe->peBlue) ;
  TextOut (hdc, 0, 0, szBuffer, lstrlen (szBuffer)) ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
  HBRUSH hBrush ;
  RECT rect ;

  // Draw Palette Index 0 on entire window

  hBrush = CreateSolidBrush (PALETTEINDEX (0)) ;
  SetRect (&rect, 0, 0, cxClient, cyClient) ;
  FillRect (hdc, &rect, hBrush) ;
  DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
  // Display the RGB value
  DisplayRGB (hdc, &pe) ;
  return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
  static BOOL bRedUp = TRUE, bGreenUp = TRUE, bBlueUp = TRUE ;
  // Define new color value
```

```
  pe.peBlue += (bBlueUp ? iIncr : -iIncr) ;
  if ( pe.peBlue == (BYTE) (bBlueUp ? 0 : 256 - iIncr))
  {
    pe.peBlue = (bBlueUp ? 256 - iIncr : 0) ;
    bBlueUp ^= TRUE ;
    pe.peGreen += (bGreenUp ? iIncr : -iIncr) ;

    if ( pe.peGreen == (BYTE) (bGreenUp ? 0 : 256 - iIncr))
    {
      pe.peGreen = (bGreenUp ? 256 - iIncr : 0) ;
      bGreenUp ^= TRUE ;
      pe.peRed += (bRedUp ? iIncr : -iIncr) ;

      if ( pe.peRed == (BYTE) (bRedUp ? 0 : 256 - iIncr))
      {
        pe.peRed = (bRedUp ? 256 - iIncr : 0) ;
        bRedUp ^= TRUE ;
      }
    }
  }

  // Animate the palette
  AnimatePalette (hPalette, 0, 1, &pe) ;
  DisplayRGB (hdc, &pe) ;
  return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
  KillTimer (hwnd, ID_TIMER) ;
  DeleteObject (hPalette) ;
  return ;
}
```

在结构上，ALLCOLOR与FADER非常相似。在CreateRoutine中，ALLCOLOR只用一个设为黑色的调色盘项目（PALETTEENTRY结构的red、green和blue字段设为0）来建立调色盘。在PaintRoutine中，ALLCOLOR用PALETTEINDEX(0)建立实心画刷，并呼叫FillRect来用此画刷为整个显示区域着色。

在TimerRoutine中，ALLCOLOR通过改变PALETTEENTRY颜色并呼叫AnimatePalette来启动调色盘。我编写ALLCOLOR程序，以便颜色变化顺畅。首先，蓝色值渐渐增加。达到最大时，绿色值增加，而蓝色值渐渐减少。红色、绿色和蓝色值的增加和减少取决于iIncr变量。在CreateRoutine期间，这将根据用COLORRES参数从GetDeviceCaps传回的值来计算。例如，如果GetDeviceCaps传回18，那么iIncr设为4 – 获得所有颜色所需要的最小值。

ALLCOLOR还在显示区域的左上角显示目前的RGB颜色值。我最初添加这个程序代码是出于测试目的，但是现在证明它是有用的，所以我保留了它。

工程应用程序

在工程应用程序中，动画对于显示机械或电的作用过程很有用。在计算机屏幕上显示内燃引擎虽然简单，但是动画可以使它变得更加生动，且更清楚地显示其工作程序。

使用调色盘动画的一个好范例就是显示流体通过管子的过程。这是一个例子，图像不必十分精确 – 实际上，如果图像很精确（就像看透明的管子），则很难说明管子里的流体是如何运动的。这时用符号会更好一些。程序16-10所示的PIPES程序是此技术的简单示范：在显示区域有两个水平的管子，流体在上面的管子里从左向右流动，而在下面的管子里从右向左移动。

程序16-10 PIPES

PIPES.C

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

```
PIPES.C -- Palette Animation Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Pipes") ;
TCHAR szTitle [] = TEXT ("Pipes: Palette Animation Demo") ;

static LOGPALETTE * plp ;

HPALETTE CreateRoutine (HWND hwnd)
{
  HPALETTE hPalette ;
  int i ;

  plp = (LOGPALETTE*)malloc (sizeof (LOGPALETTE) + 32 * sizeof (PALETTEENTRY)) ;
  // Initialize the fields of the LOGPALETTE structure
  plp->palVersion = 0x300 ;
  plp->palNumEntries = 16 ;

  for (i = 0 ; i <= 8 ; i++)
  {
    plp->palPalEntry[i].peRed = (BYTE) min (255, 0x20 * i) ;
    plp->palPalEntry[i].peGreen = 0 ;
    plp->palPalEntry[i].peBlue = (BYTE) min (255, 0x20 * i) ;
    plp->palPalEntry[i].peFlags = PC_RESERVED ;

    plp->palPalEntry[16 - i] = plp->palPalEntry[i] ;
    plp->palPalEntry[16 + i] = plp->palPalEntry[i] ;
    plp->palPalEntry[32 - i] = plp->palPalEntry[i] ;
  }

  hPalette = CreatePalette (plp) ;
  SetTimer (hwnd, ID_TIMER, 100, NULL) ;
  return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
  HBRUSH hBrush ;
  int i ;
  RECT rect ;

  // Draw window background

  SetRect (&rect, 0, 0, cxClient, cyClient) ;
  hBrush = (HBRUSH)SelectObject (hdc, GetStockObject (WHITE_BRUSH)) ;
  FillRect (hdc, &rect, hBrush) ;

  // Draw the interiors of the pipes
  for (i = 0 ; i < 128 ; i++)
  {
    hBrush = CreateSolidBrush (PALETTEINDEX (i % 16)) ;
    SelectObject (hdc, hBrush) ;

    rect.left = (127 - i) * cxClient / 128 ;
    rect.right = (128 - i) * cxClient / 128 ;
    rect.top = 4 * cyClient / 14 ;
    rect.bottom = 5 * cyClient / 14 ;

    FillRect (hdc, &rect, hBrush) ;

    rect.left = i * cxClient / 128 ;
    rect.right = ( i + 1) * cxClient / 128 ;
    rect.top = 9 * cyClient / 14 ;
    rect.bottom = 10 * cyClient / 14 ;

    FillRect (hdc, &rect, hBrush) ;
```

```
    DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
  }

  // Draw the edges of the pipes
  MoveToEx (hdc, 0, 4 * cyClient / 14, NULL) ;
  LineTo (hdc, cxClient, 4 * cyClient / 14) ;

  MoveToEx (hdc, 0, 5 * cyClient / 14, NULL) ;
  LineTo (hdc, cxClient,5 * cyClient / 14) ;

  MoveToEx (hdc, 0, 9 * cyClient / 14, NULL) ;
  LineTo (hdc, cxClient, 9 * cyClient / 14) ;

  MoveToEx (hdc, 0, 10 * cyClient / 14, NULL) ;
  LineTo (hdc, cxClient, 10 * cyClient / 14) ;
  return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
  static int iIndex ;
  AnimatePalette (hPalette, 0, 16, plp->palPalEntry + iIndex) ;
  iIndex = (iIndex + 1) % 16 ;

  return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
  KillTimer (hwnd, ID_TIMER) ;
  DeleteObject (hPalette) ;
  free (plp) ;
  return ;
}
```

PIPES为动画使用了16个调色盘项目，而您可能会使用更少的项目。最小化时，真正需要的是有足够的项目来显示流动的方向。用三个调色盘项目要比用一个静态箭头好。

程序16-11所示的TUNNEL程序是这组程序中最贪心的程序，它为动画使用了128个调色盘项目，但是从效果来看，值得这样做。

程序16-11 TUNNEL

TUNNEL.C

```
/*------------------------------------------------------------------------
TUNNEL.C -- Palette Animation Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
TCHAR szAppName [] = TEXT ("Tunnel") ;
TCHAR szTitle [] = TEXT ("Tunnel: Palette Animation Demo") ;

static LOGPALETTE * plp ;
HPALETTE CreateRoutine (HWND hwnd)
{
  BYTE byGrayLevel ;
  HPALETTE hPalette ;
  int i ;

  plp = (LOGPALETTE*)malloc (sizeof (LOGPALETTE) + 255 * sizeof (PALETTEENTRY)) ;
  // Initialize the fields of the LOGPALETTE structure
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = 128 ;

  for (i = 0 ; i < 128 ; i++)
  {
```

```
    if (i < 64)
      byGrayLevel = (BYTE) (4 * i) ;
    else
      byGrayLevel = (BYTE) min (255, 4 * (128 - i)) ;
    plp->palPalEntry[i].peRed = byGrayLevel ;
    plp->palPalEntry[i].peGreen = byGrayLevel ;
    plp->palPalEntry[i].peBlue = byGrayLevel ;
    plp->palPalEntry[i].peFlags = PC_RESERVED ;

    plp->palPalEntry[i + 128].peRed = byGrayLevel ;
    plp->palPalEntry[i + 128].peGreen = byGrayLevel ;
    plp->palPalEntry[i + 128].peBlue = byGrayLevel ;
    plp->palPalEntry[i + 128].peFlags = PC_RESERVED ;
  }

  hPalette = CreatePalette (plp) ;
  SetTimer (hwnd, ID_TIMER, 50, NULL) ;
  return hPalette ;
}

void PaintRoutine (HDC hdc, int cxClient, int cyClient)
{
  HBRUSH hBrush ;
  int i ;
  RECT rect ;

  for (i = 0 ; i < 127 ; i++)
  {
    // Use a RECT structure for each of 128 rectangles
    rect.left = i * cxClient / 255 ;
    rect.top = i * cyClient / 255 ;
    rect.right = cxClient - i * cxClient / 255 ;
    rect.bottom = cyClient - i * cyClient / 255 ;

    hBrush = CreateSolidBrush (PALETTEINDEX (i)) ;
    // Fill the rectangle and delete the brush

    FillRect (hdc, &rect, hBrush) ;
    DeleteObject (hBrush) ;
  }
  return ;
}

void TimerRoutine (HDC hdc, HPALETTE hPalette)
{
  static int iLevel ;
  iLevel = (iLevel + 1) % 128 ;
  AnimatePalette (hPalette, 0, 128, plp->palPalEntry + iLevel) ;
  return ;
}

void DestroyRoutine (HWND hwnd, HPALETTE hPalette)
{
  KillTimer (hwnd, ID_TIMER) ;
  DeleteObject (hPalette) ;
  free (plp) ;
  return ;
}
```

TUNNEL在128个调色盘项目中使用64种移动的灰阶 – 从黑到白，再从白到黑 – 表现在隧道旅行的效果。

调色盘和真实世界图像

当然，尽管我们已经完成了许多有趣的事：连续显示色彩的底纹、做了调色盘动画，但调色盘管理器的真正目的是允许在8位显示模式下显示真实世界中的图像。对于本章的其余部分，我们正好研究一下。正如您所期望的，在使用packed DIB、GDI位图对象和DIB区块时，必须按照不同的方法来使用调色盘。下面的六个程序阐明了用调色盘来处理位图的各种技术。

调色盘和Packed DIB

下面三个程序，有助于我们建立处理packed DIB内存块的一系列函数。这些函数都在程序16-12所示的PACKEDIB文件中。

程序16-12 PACKEDIB文件

PACKEDIB.H

```
/*------------------------------------------------------------------------
PACKEDIB.H --  Header file for PACKEDIB.C
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

BITMAPINFO * PackedDibLoad (PTSTR szFileName) ;
int PackedDibGetWidth (BITMAPINFO * pPackedDib) ;
int PackedDibGetHeight (BITMAPINFO * pPackedDib) ;
int PackedDibGetBitCount (BITMAPINFO * pPackedDib) ;
int PackedDibGetRowLength (BITMAPINFO * pPackedDib) ;
int PackedDibGetInfoHeaderSize (BITMAPINFO * pPackedDib) ;
int PackedDibGetColorsUsed (BITMAPINFO * pPackedDib) ;
int PackedDibGetNumColors (BITMAPINFO * pPackedDib) ;
int PackedDibGetColorTableSize (BITMAPINFO * pPackedDib) ;
RGBQUAD * PackedDibGetColorTablePtr (BITMAPINFO * pPackedDib) ;
RGBQUAD * PackedDibGetColorTableEntry (BITMAPINFO * pPackedDib, int i) ;
BYTE * PackedDibGetBitsPtr (BITMAPINFO * pPackedDib) ;
int PackedDibGetBitsSize (BITMAPINFO * pPackedDib) ;
HPALETTE PackedDibCreatePalette (BITMAPINFO * pPackedDib) ;
```

PACKEDIB.C

```
/*------------------------------------------------------------------------
PACKEDIB.C -- Routines for using packed DIBs
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

/*------------------------------------------------------------------------
PackedDibLoad: Load DIB File as Packed-Dib Memory Block
------------------------------------------------------------------------*/

BITMAPINFO * PackedDibLoad (PTSTR szFileName)
{
  BITMAPFILEHEADER bmfh ;
  BITMAPINFO * pbmi ;
  BOOL bSuccess ;
  DWORD dwPackedDibSize, dwBytesRead ;
  HANDLE hFile ;
  // Open the file: read access, prohibit write access

  hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
  if (hFile == INVALID_HANDLE_VALUE)
    return NULL ;
  // Read in the BITMAPFILEHEADER
```

```
  bSuccess = ReadFile (hFile, &bmfh, sizeof (BITMAPFILEHEADER),
    &dwBytesRead, NULL) ;

  if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
    || (bmfh.bfType != * (WORD *) "BM"))
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  // Allocate memory for the packed DIB & read it in
  dwPackedDibSize = bmfh.bfSize - sizeof (BITMAPFILEHEADER) ;
  pbmi = (BITMAPINFO*)malloc (dwPackedDibSize) ;
  bSuccess = ReadFile (hFile, pbmi, dwPackedDibSize, &dwBytesRead, NULL) ;
  CloseHandle (hFile) ;

  if (!bSuccess || (dwBytesRead != dwPackedDibSize))
  {
    free (pbmi) ;
    return NULL ;
  }

  return pbmi ;
}

/*-------------------------------------------------------------------------
Functions to get information from packed DIB
-------------------------------------------------------------------------*/

int PackedDibGetWidth (BITMAPINFO * pPackedDib)
{
  if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return ((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcWidth ;
  else
    return pPackedDib->bmiHeader.biWidth ;
}

int PackedDibGetHeight (BITMAPINFO * pPackedDib)
{
  if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return ((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcHeight ;
  else
    return abs (pPackedDib->bmiHeader.biHeight) ;
}

int PackedDibGetBitCount (BITMAPINFO * pPackedDib)
{
  if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return ((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcBitCount ;
  else
    return pPackedDib->bmiHeader.biBitCount ;
}

int PackedDibGetRowLength (BITMAPINFO * pPackedDib)
{
  return (( PackedDibGetWidth (pPackedDib) *
    PackedDibGetBitCount (pPackedDib) + 31) & ~31) >> 3 ;
}

/*-------------------------------------------------------------------------
PackedDibGetInfoHeaderSize includes possible color masks!
-------------------------------------------------------------------------*/

int PackedDibGetInfoHeaderSize (BITMAPINFO * pPackedDib)
{
  if ( pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return ((PBITMAPCOREINFO)pPackedDib)->bmciHeader.bcSize ;

  else if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPINFOHEADER))
```

```
    return pPackedDib->bmiHeader.biSize +
    (pPackedDib->bmiHeader.biCompression ==
    BI_BITFIELDS ? 12 : 0) ;
  else return pPackedDib->bmiHeader.biSize ;
}

/*-------------------------------------------------------------------------
PackedDibGetColorsUsed returns value in information header;
could be 0 to indicate non-truncated color table!
-------------------------------------------------------------------------*/

int PackedDibGetColorsUsed (BITMAPINFO * pPackedDib)
{
  if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return 0 ;
  else
    return pPackedDib->bmiHeader.biClrUsed ;
}

/*-------------------------------------------------------------------------
PackedDibGetNumColors is actual number of entries in color table
-------------------------------------------------------------------------*/

int PackedDibGetNumColors (BITMAPINFO * pPackedDib)
{
  int iNumColors ;
  iNumColors = PackedDibGetColorsUsed (pPackedDib) ;
  if ( iNumColors == 0 && PackedDibGetBitCount (pPackedDib) < 16)
    iNumColors =1 << PackedDibGetBitCount (pPackedDib) ;

  return iNumColors ;
}

int PackedDibGetColorTableSize (BITMAPINFO * pPackedDib)
{
  if (pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return PackedDibGetNumColors (pPackedDib) * sizeof (RGBTRIPLE) ;
  else
    return PackedDibGetNumColors (pPackedDib) * sizeof (RGBQUAD) ;
}

RGBQUAD * PackedDibGetColorTablePtr (BITMAPINFO * pPackedDib)
{
  if (PackedDibGetNumColors (pPackedDib) == 0)
    return 0 ;
  return (RGBQUAD *) (((BYTE *)pPackedDib) +
    PackedDibGetInfoHeaderSize (pPackedDib)) ;
}

RGBQUAD * PackedDibGetColorTableEntry (BITMAPINFO * pPackedDib, int i)
{
  if ( PackedDibGetNumColors (pPackedDib) == 0)
    return 0 ;

  if ( pPackedDib->bmiHeader.biSize == sizeof (BITMAPCOREHEADER))
    return (RGBQUAD *)
    (((RGBTRIPLE *) PackedDibGetColorTablePtr (pPackedDib)) + i) ;
  else
    return PackedDibGetColorTablePtr (pPackedDib) + i ;
}

/*-------------------------------------------------------------------------
PackedDibGetBitsPtr finally!
-------------------------------------------------------------------------*/

BYTE * PackedDibGetBitsPtr (BITMAPINFO * pPackedDib)
{
  return ((BYTE *) pPackedDib)+PackedDibGetInfoHeaderSize (pPackedDib) +
    PackedDibGetColorTableSize (pPackedDib) ;
```

```
}

/*-------------------------------------------------------------------------
PackedDibGetBitsSize can be calculated from the height and row length
if it's not explicitly in the biSizeImage field
-------------------------------------------------------------------------*/

int PackedDibGetBitsSize (BITMAPINFO * pPackedDib)
{
  if ((pPackedDib->bmiHeader.biSize != sizeof (BITMAPCOREHEADER)) &&
    (pPackedDib->bmiHeader.biSizeImage != 0))
    return pPackedDib->bmiHeader.biSizeImage ;

  return PackedDibGetHeight (pPackedDib) *
    PackedDibGetRowLength (pPackedDib) ;
}

/*-------------------------------------------------------------------------
PackedDibCreatePalette creates logical palette from PackedDib
-------------------------------------------------------------------------*/
HPALETTE PackedDibCreatePalette (BITMAPINFO * pPackedDib)
{
  HPALETTE hPalette ;
  int i, iNumColors ;
  LOGPALETTE * plp ;
  RGBQUAD * prgb ;

  if (0 == ( iNumColors = PackedDibGetNumColors (pPackedDib)))
    return NULL ;
  plp = (LOGPALETTE*)malloc (sizeof (LOGPALETTE) *
    (iNumColors - 1) * sizeof (PALETTEENTRY)) ;

  plp->palVersion = 0x0300 ;
  plp->palNumEntries = iNumColors ;
  for (i = 0 ; i < iNumColors ; i++)
  {
    prgb = PackedDibGetColorTableEntry (pPackedDib, i) ;
    plp->palPalEntry[i].peRed = prgb->rgbRed ;
    plp->palPalEntry[i].peGreen = prgb->rgbGreen ;
    plp->palPalEntry[i].peBlue = prgb->rgbBlue ;
    plp->palPalEntry[i].peFlags = 0 ;
  }

  hPalette = CreatePalette (plp) ;
  free (plp) ;

  return hPalette ;
}
```

第一个函数是PackedDibLoad，它将唯一的参数作为文件名，并传回指向内存中packed DIB的指针。其它所有函数都将这个packed DIB指标作为它们的第一个参数并传回有关DIB的信息。这些函数按「由下而上」顺序排列到文件中。每个函数都使用从前面函数获得的信息。

我不倾向于说这是在处理packed DIB时有用的「完整」函数集。而且，我也不想汇编一个真正的扩展集，因为我不认为这是处理packed DIB的一个好方法。在写类似下面的函数时，您会很明显地发现这一点：

dwPixel = PackedDibGetPixel (pPackedDib, x, y) ;

这种函数包括太多的巢状函数呼叫，以致于效率非常低而且很慢。本章的后面将讨论一种我认为更好的方法。

另外，您将注意到，其中许多函数都需要对OS/2兼容的DIB采取不同的处理程序；这样，函数将频繁地检查BITMAPINFO结构的第一个字段是否与BITMAPCOREHEADER结构的大小相同。

特别注意最后一个函数PackedDibCreatePalette。这个函数用DIB中的颜色表来建立调色盘。

如果DIB中没有颜色表（这意味着DIB的每像素有16、24或32位），那么就不建立调色盘。我们有时会将从DIB颜色表建立的调色盘称为DIB **自己的**调色盘。

PACKEDIB文件都放在SHOWDIB3，如程序16-13所示。

程序16-13 SHOWDIB3

SHOWDIB3.C

```
/*------------------------------------------------------------------------
SHOWDIB3.C -- Displays DIB with native palette
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #3: Native Palette"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAPINFO * pPackedDib ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
```

```
  TEXT ("All Files (*.*)\0*.*\0\0") ;
HDC hdc ;
PAINTSTRUCT ps ;

switch (message)
{
case WM_CREATE:
  ofn.lStructSize = sizeof (OPENFILENAME) ;
  ofn.hwndOwner = hwnd ;
  ofn.hInstance = NULL ;
  ofn.lpstrFilter = szFilter ;
  ofn.lpstrCustomFilter = NULL ;
  ofn.nMaxCustFilter = 0 ;
  ofn.nFilterIndex = 0 ;
  ofn.lpstrFile = szFileName ;
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = szTitleName ;
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.lpstrInitialDir = NULL ;
  ofn.lpstrTitle = NULL ;
  ofn.Flags = 0 ;
  ofn.nFileOffset = 0 ;
  ofn.nFileExtension = 0 ;
  ofn.lpstrDefExt = TEXT ("bmp") ;
  ofn.lCustData = 0 ;
  ofn.lpfnHook = NULL ;
  ofn.lpTemplateName = NULL ;

  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_FILE_OPEN:

    // Show the File Open dialog box

    if (!GetOpenFileName (&ofn))
      return 0 ;

    // If there's an existing packed DIB, free the memory

    if (pPackedDib)
    {
      free (pPackedDib) ;
      pPackedDib = NULL ;
    }

    // If there's an existing logical palette, delete it

    if (hPalette)
    {
      DeleteObject (hPalette) ;
      hPalette = NULL ;
    }
    // Load the packed DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pPackedDib = PackedDibLoad (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
```

```
    if (pPackedDib)
    {
      // Create the palette from the DIB color table

      hPalette = PackedDibCreatePalette (pPackedDib) ;
    }
    else
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  }
  break ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;
    }

    if (pPackedDib)
      SetDIBitsToDevice (hdc, 0,0,PackedDibGetWidth (pPackedDib),
      PackedDibGetHeight (pPackedDib),
      0,0,0,PackedDibGetHeight (pPackedDib),
      PackedDibGetBitsPtr (pPackedDib),
      pPackedDib,
      DIB_RGB_COLORS) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

  case WM_PALETTECHANGED:
    if (!hPalette || (HWND) wParam == hwnd)
      break ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;

    ReleaseDC (hwnd, hdc) ;
    break ;


  case WM_DESTROY:
    if (pPackedDib)
      free (pPackedDib) ;

    if (hPalette)
      DeleteObject (hPalette) ;

    PostQuitMessage (0) ;
```

```
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB3.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB3 MENU DISCARDABLE
BEGIN
   POPUP "&File"
   BEGIN
      MENUITEM "&Open",    IDM_FILE_OPEN
   END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib3.rc
#define IDM_FILE_OPEN     40001
```

SHOWDIB3中的窗口消息处理程序将packed DIB指针作为静态变量来维护，窗口消息处理程序在「File Open」命令期间呼叫PACKEDIB.C中的PackedDibLoad函数时获得了此指标。在处理此命令的过程中，SHOWDIB3也呼叫PackedDibCreatePalette来获得可能用于DIB的调色盘。注意，无论SHOWDIB3什么时候准备加载新的DIB，都应先释放前一个DIB的内存，并删除前一个DIB的调色盘。在处理WM_DESTROY消息的程序中，最后的DIB最后释放，最后的调色盘最后删除。

处理WM_PAINT消息很简单：如果存在调色盘，则SHOWDIB3将它选进设备内容并显现它。然后它呼叫SetDIBitsToDevice，并传递有关DIB的函数信息（例如宽、高和指向DIB像素位的指针），这些信息从PACKEDIB中的函数获得。

另外，请记住SHOWDIB3依据DIB中的颜色表建立了调色盘。如果在DIB中没有颜色表 – 通常是16位、24位和32位DIB的情况 – 就不建立调色盘。在8位显示模式下显示DIB时，它只能用标准保留的20种颜色显示。

对这个问题有两种解决方法：第一种是简单地使用「通用」调色盘，这种调色盘适用于许多图形。您也可以自己建立调色盘。第二种解决方法是分析DIB的像素位，并决定要显示图像的最佳颜色。很明显，第二种方法将涉及更多的工作（对于程序写作者和处理器都是如此），但是我将在本章结束之前告诉您如何使用第二种方法。

「通用」调色盘

程序16-14所示的SHOWDIB4程序建立了一个通用的调色盘，它用于显示加载到程序中的所有DIB。另外，SHOWDIB4与SHOWDIB3非常相似。

程序16-14 SHOWDIB4

SHOWDIB4.C

```
/*------------------------------------------------------------------------
SHOWDIB4.C -- Displays DIB with "all-purpose" palette
(c) Charles Petzold, 1998
```

```
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\ShowDib3\\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib4") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #4: All-Purpose Palette"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

/*------------------------------------------------------------------------
CreateAllPurposePalette: Creates a palette suitable for a wide variety
of images; the palette has 247 entries, but 15 of them are
duplicates or match the standard 20 colors.
------------------------------------------------------------------------*/

HPALETTE CreateAllPurposePalette (void)
{
  HPALETTE hPalette ;
  int i, incr, R, G, B ;
  LOGPALETTE * plp ;

  plp = malloc (sizeof (LOGPALETTE) + 246 * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = 247 ;

  // The following loop calculates 31 gray shades, but 3 of them
  // will match the standard 20 colors

  for (i = 0, G = 0, incr = 8 ; G <= 0xFF ; i++, G += incr)
  {
```

```
    plp->palPalEntry[i].peRed = (BYTE) G ;
    plp->palPalEntry[i].peGreen = (BYTE) G ;
    plp->palPalEntry[i].peBlue = (BYTE) G ;
    plp->palPalEntry[i].peFlags = 0 ;

    incr = (incr == 9 ? 8 : 9) ;
  }

  // The following loop is responsible for 216 entries, but 8 of
  // them will match the standard 20 colors, and another
  // 4 of them will match the gray shades above.

  for (R = 0 ; R <= 0xFF ; R += 0x33)
    for (G = 0 ; G <= 0xFF ; G += 0x33)
      for (B = 0 ; B <= 0xFF ; B += 0x33)
      {
        plp->palPalEntry [i].peRed = (BYTE) R ;
        plp->palPalEntry [i].peGreen = (BYTE) G ;
        plp->palPalEntry [i].peBlue = (BYTE) B ;
        plp->palPalEntry [i].peFlags = 0 ;

        i++ ;
      }
     hPalette = CreatePalette (plp) ;
     free (plp) ;
     return hPalette ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAPINFO * pPackedDib ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  switch (message)
  {
  case WM_CREATE:
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("bmp") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;

    // Create the All-Purpose Palette

    hPalette = CreateAllPurposePalette () ;
    return 0 ;

  case WM_SIZE:
```

```
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

      // Show the File Open dialog box

      if (!GetOpenFileName (&ofn))
        return 0 ;

      // If there's an existing packed DIB, free the memory
      if (pPackedDib)
      {
        free (pPackedDib) ;
        pPackedDib = NULL ;
      }

      // Load the packed DIB into memory

      SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
      ShowCursor (TRUE) ;

      pPackedDib = PackedDibLoad (szFileName) ;

      ShowCursor (FALSE) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

      if (!pPackedDib)
      {
        MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
          szAppName, 0) ;
      }
      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
    }
    break ;

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      if (pPackedDib)
      {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;

        SetDIBitsToDevice (hdc,0,0,PackedDibGetWidth (pPackedDib),
          PackedDibGetHeight (pPackedDib),
          0,0,0,PackedDibGetHeight (pPackedDib),
          PackedDibGetBitsPtr (pPackedDib),
          pPackedDib,
          DIB_RGB_COLORS) ;
      }
      EndPaint (hwnd, &ps) ;
      return 0 ;

    case WM_QUERYNEWPALETTE:
      hdc = GetDC (hwnd) ;
      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;
      InvalidateRect (hwnd, NULL, TRUE) ;

      ReleaseDC (hwnd, hdc) ;
      return TRUE ;

    case WM_PALETTECHANGED:
```

```
     if ((HWND) wParam != hwnd)

       hdc = GetDC (hwnd) ;
     SelectPalette (hdc, hPalette, FALSE) ;
     RealizePalette (hdc) ;
     UpdateColors (hdc) ;

     ReleaseDC (hwnd, hdc) ;
     break ;

   case WM_DESTROY:
     if (pPackedDib)
       free (pPackedDib) ;

     DeleteObject (hPalette) ;

     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB4.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB4 MENU DISCARDABLE
BEGIN
POPUP "&Open"
BEGIN
MENUITEM "&File",         IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib4.rc
#define IDM_FILE_OPEN   40001
```

在处理WM_CREATE消息时，SHOWDIB4将呼叫CreateAllPurposePalette，并在程序中保留该调色盘，而在WM_DESTROY消息处理期间删除它。因为程序知道调色盘一定存在，所以在处理WM_PAINT、WM_QUERYNEWPALETTE或WM_PALETTECHANGED消息时，不必检查调色盘的存在。

CreateAllPurposePalette函数似乎是用247个项目来建立逻辑调色盘，它超出了系统调色盘中允许程序正常存取的236个项目。的确如此，不过这样做很方便。这些项目中有15个被复制或者映射到20种标准的保留颜色中。

CreateAllPurposePalette从建立31种灰阶开始，即0x00、0x09、0x11、0x1A、0x22、0x2B、0x33、0x3C、0x44、0x4D、0x55、0x5E、0x66、0x6F、0x77、0x80、0x88、0x91、0x99、0xA2、0xAA、0xB3、0xBB、0xC4、0xCC、0xD5、0xDD、0xE6、0xEE、0xF9和0xFF的红色、绿色和蓝色值。注意，第一个、最后一个和中间的项目都在标准的20种保留颜色中。下一个函数用红色、绿色和蓝色值的所有组合建立了颜色0x00、0x33、0x66、0x99、0xCC和0xFF。这样就共有216种颜色，但是其中8种颜色复制了标准的20种保留颜色，而另外4个复制了前面计算的灰阶。如果将PALETTEENTRY结构的peFlags字段设为0，则Windows将不把复制的项目放进系统调色盘。

显然地，实际的程序不希望计算16位、24位或者32位DIB的最佳调色盘，程序将继续使用DIB颜色表来显示8位DIB。SHOWDIB4不完成这项工作，它只对每件事都使用通用调色盘。因为SHOWDIB4是一个展示程序，而且您可以与SHOWDIB3显示的8位DIB进行比较。如果看一些人像的彩色DIB，那么您可能会得出这样的结论：SHOWDIB4没有足够的颜色来精确地表示鲜艳的色调。

如果用SHOWDIB4中的CreateAllPurposePalette函数来试验（可能是通过将逻辑调色盘的大小减少到只有几个项目的方法），您将发现当调色盘选进设备内容时，Windows将只使用调色盘中的颜色，而不使用标准的20种颜色调色盘的颜色。

中间色调色盘

Windows API包括一个通用调色盘，程序可以通过呼叫CreateHalftonePalette来获得该调色盘。使用此调色盘的方法与使用从SHOWDIB4中的CreateAllPurposePalette获得调色盘的方法相同，或者您也可以与位图缩放模式中的HALFTONE设定 – 用SetStretchBltMode设定 – 一起使用。程序16-15所示的SHOWDIB5程序展示了使用中间色调色盘的方法。

程序16-15　SHOWDIB5

SHOWDIB5.C

```
/*--------------------------------------------------------------------------
SHOWDIB5.C -- Displays DIB with halftone palette
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\ShowDib3\\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib5") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #5: Halftone Palette"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BITMAPINFO * pPackedDib ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("bmp") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;

    // Create the All-Purpose Palette

    hdc = GetDC (hwnd) ;
    hPalette = CreateHalftonePalette (hdc) ;
    ReleaseDC (hwnd, hdc) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

      // Show the File Open dialog box

      if (!GetOpenFileName (&ofn))
        return 0 ;

      // If there's an existing packed DIB, free the memory

      if (pPackedDib)
```

```
    {
      free (pPackedDib) ;
      pPackedDib = NULL ;
    }

    // Load the packed DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pPackedDib = PackedDibLoad (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (!pPackedDib)
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  }
  break ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (pPackedDib)
    {
      // Set halftone stretch mode

      SetStretchBltMode (hdc, HALFTONE) ;
      SetBrushOrgEx (hdc, 0, 0, NULL) ;

      // Select and realize halftone palette

      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;

      // StretchDIBits rather than SetDIBitsToDevice
      StretchDIBits (hdc,0,0,PackedDibGetWidth (pPackedDib),
        PackedDibGetHeight (pPackedDib),
        0,0,PackedDibGetWidth (pPackedDib),
        PackedDibGetHeight (pPackedDib),
        PackedDibGetBitsPtr (pPackedDib),
        pPackedDib,
        DIB_RGB_COLORS,
        SRCCOPY) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

  case WM_PALETTECHANGED:
    if ((HWND) wParam != hwnd)

      hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    UpdateColors (hdc) ;
```

```
     ReleaseDC (hwnd, hdc) ;
     break ;

   case WM_DESTROY:
     if (pPackedDib)
       free (pPackedDib) ;

     DeleteObject (hPalette) ;

     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB5.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB5 MENU DISCARDABLE
BEGIN
POPUP "&Open"
BEGIN
MENUITEM "&File",      IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib5.rc
#define IDM_FILE_OPEN   40001
```

SHOWDIB5程序类似于SHOWDIB4，SHOWDIB4中不使用DIB中的颜色表，而使用适用于图像范围更大的调色盘。为此，SHOWDIB5使用了由Windows支持的逻辑调色盘，其句柄可以从CreateHalftonePalette函数获得。

中间色调色盘并不比SHOWDIB4中的CreateAllPurposePalette函数所建立的调色盘更复杂。的确，如果只是拿来自用，结果是相似的。然而，如果您呼叫下面两个函数：

```
SetStretchBltMode (hdc, HALFTONE) ;
SetBrushOrgEx (hdc, x, y, NULL) ;
```

其中，x和y是DIB左上角的设备坐标，并且如果您用StretchDIBits而不是SetDIBitsToDevice来显示DIB，那么结果会让您吃惊：颜色色调要比不设定位图缩放模式来使用CreateAllPurposePalette或者CreateHalftonePalette更精确。Windows使用一种混色图案来处理中间色调色盘上的颜色，以使其更接近8位显示卡上原始图像的颜色。与您所想象的一样，这样做的缺点是需要更多的处理时间。

索引调色盘颜色

现在开始处理SetDIBitsToDevice、StretchDIBits、CreateDIBitmap、SetDIBits、GetDIBits和CreateDIBSection的fClrUse参数。通常，您将这个参数设定为DIB_RGB_COLORS（等于0）。不过，您也能将它设定为DIB_PAL_COLORS。在这种情况下，假定BITMAPINFO结构中的颜色表不包括RGB颜色值，而是包括逻辑调色盘中颜色项目的16位索引。逻辑调色盘是作为第一个参数传递给函数的设备内容中目前选择的那个。实际上，在CreateDIBSection中，之所以需要指定一个非

NULL的设备内容句柄作为第一个参数，只是因为使用了DIB_PAL_COLORS。

DIB_PAL_COLORS能为您做些什么呢？它可能提高一些性能。考虑一下在8位显示模式下呼叫SetDIBitsToDevice显示的8位DIB。Windows首先必须在DIB颜色表的所有颜色中搜索与设备可用颜色最接近的颜色。然后设定一个小表，以便将DIB像素值映像到设备像素。也就是说，最多需要搜索256次最接近的颜色。但是如果DIB颜色表中含有从设备内容中选择颜色的逻辑调色盘项目索引，那么就可能跳过搜索。

除了使用调色盘索引以外，程序16-16所示的SHOWDIB6程序与SHOWDIB3相似。

程序16-16 SHOWDIB6

SHOWDIB6.C

```
/*------------------------------------------------------------------------
SHOWDIB6.C -- Display DIB with palette indices
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\ShowDib3\\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib6") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #6: Palette Indices"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
```

```
{
  static BITMAPINFO * pPackedDib ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  HDC hdc ;
  int i, iNumColors ;
  PAINTSTRUCT ps ;
  WORD * pwIndex ;

  switch (message)
  {
  case WM_CREATE:
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("bmp") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:

      // Show the File Open dialog box
      if (!GetOpenFileName (&ofn))
        return 0 ;

      // If there's an existing packed DIB, free the memory

      if (pPackedDib)
      {
        free (pPackedDib) ;
        pPackedDib = NULL ;
      }

      // If there's an existing logical palette, delete it

      if (hPalette)
      {
        DeleteObject (hPalette) ;
        hPalette = NULL ;
      }
```

```
    // Load the packed DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pPackedDib = PackedDibLoad (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (pPackedDib)
    {
      // Create the palette from the DIB color table

      hPalette = PackedDibCreatePalette (pPackedDib) ;

      // Replace DIB color table with indices

      if (hPalette)
      {
        iNumColors = PackedDibGetNumColors (pPackedDib) ;
        pwIndex = (WORD *)
          PackedDibGetColorTablePtr (pPackedDib) ;

        for (i = 0 ; i < iNumColors ; i++)
          pwIndex[i] = (WORD) i ;
      }
    }
    else
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  }
  break ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;
    }

    if (pPackedDib)
      SetDIBitsToDevice (hdc,0,0,PackedDibGetWidth (pPackedDib),
      PackedDibGetHeight (pPackedDib),
      0,0,0,PackedDibGetHeight (pPackedDib),
      PackedDibGetBitsPtr (pPackedDib),
      pPackedDib,
      DIB_PAL_COLORS) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;
```

```
   case WM_PALETTECHANGED:
     if (!hPalette || (HWND) wParam == hwnd)
       break ;

     hdc = GetDC (hwnd) ;
     SelectPalette (hdc, hPalette, FALSE) ;
     RealizePalette (hdc) ;
     UpdateColors (hdc) ;

     ReleaseDC (hwnd, hdc) ;
     break ;


   case WM_DESTROY:
     if (pPackedDib)
       free (pPackedDib) ;

     if (hPalette)
       DeleteObject (hPalette) ;

     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB6.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB6 MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open",            IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib6.rc
//
#define IDM_FILE_OPEN   40001
```

SHOWDIB6将DIB加载到内存并由此建立了调色盘以后，SHOWDIB6简单地用以0开始的WORD索引替换了DIB颜色表中的颜色。PackedDibGetNumColors函数将表示有多少种颜色，而PackedDibGetColorTablePtr函数传回指向DIB颜色表起始位置的指针。

注意，只有直接从DIB颜色表来建立调色盘时，此技术才可行。如果使用通用调色盘，则必须搜索最接近的颜色，以获得放入DIB的索引。

如果要使用调色盘索引，那么请在将DIB储存到磁盘之前，确实替换掉DIB中的颜色表。另外，不要将包含调色盘索引的DIB放入剪贴簿。实际上，在显示之前，将调色盘索引放入DIB，然后将RGB颜色值放回，会更安全一些。

调色盘和位图对象

程序16-17中的SHOWDIB7程序显示了如何使用与DIB相关联的调色盘，这些DIB是使用

CreateDIBitmap函数转换成GDI位图对象的。

程序16-17 SHOWDIB7

SHOWDIB7.C

```
/*------------------------------------------------------------------------
SHOWDIB7.C -- Shows DIB converted to DDB
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\ShowDib3\\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib7") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #7: Converted to DDB"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
  static OPENFILENAME ofn ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  BITMAP bitmap ;
  BITMAPINFO * pPackedDib ;
```

```
HDC hdc, hdcMem ;
PAINTSTRUCT ps ;

switch (message)
{
case WM_CREATE:
  ofn.lStructSize = sizeof (OPENFILENAME) ;
  ofn.hwndOwner = hwnd ;
  ofn.hInstance = NULL ;
  ofn.lpstrFilter = szFilter ;
  ofn.lpstrCustomFilter = NULL ;
  ofn.nMaxCustFilter = 0 ;
  ofn.nFilterIndex = 0 ;
  ofn.lpstrFile = szFileName ;
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = szTitleName ;
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.lpstrInitialDir = NULL ;
  ofn.lpstrTitle = NULL ;
  ofn.Flags = 0 ;
  ofn.nFileOffset = 0 ;
  ofn.nFileExtension = 0 ;
  ofn.lpstrDefExt = TEXT ("bmp") ;
  ofn.lCustData = 0 ;
  ofn.lpfnHook = NULL ;
  ofn.lpTemplateName = NULL ;

  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_FILE_OPEN:

    // Show the File Open dialog box

    if (!GetOpenFileName (&ofn))
      return 0 ;

    // If there's an existing packed DIB, free the memory

    if (hBitmap)
    {
      DeleteObject (hBitmap) ;
      hBitmap = NULL ;
    }

    // If there's an existing logical palette, delete it

    if (hPalette)
    {
      DeleteObject (hPalette) ;
      hPalette = NULL ;
    }

    // Load the packed DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    pPackedDib = PackedDibLoad (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
```

```
    if (pPackedDib)
    {
      // Create palette from the DIB and select it into DC

      hPalette = PackedDibCreatePalette (pPackedDib) ;

      hdc = GetDC (hwnd) ;

      if (hPalette)
      {
        SelectPalette (hdc, hPalette, FALSE) ;
        RealizePalette (hdc) ;
      }
      // Create the DDB from the DIB
      hBitmap = CreateDIBitmap(hdc,(PBITMAPINFOHEADER) pPackedDib,
        CBM_INIT,PackedDibGetBitsPtr (pPackedDib),
        pPackedDib, DIB_RGB_COLORS) ;
      ReleaseDC (hwnd, hdc) ;

      // Free the packed-DIB memory

      free (pPackedDib) ;
    }
    else
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  }
  break ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;
    }
    if (hBitmap)
    {
      GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

      hdcMem = CreateCompatibleDC (hdc) ;
      SelectObject (hdcMem, hBitmap) ;

      BitBlt (hdc,0,0,bitmap.bmWidth, bitmap.bmHeight,
        hdcMem,0, 0, SRCCOPY) ;

      DeleteDC (hdcMem) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;

  case WM_PALETTECHANGED:
```

```
     if (!hPalette || (HWND) wParam == hwnd)
       break ;

     hdc = GetDC (hwnd) ;
     SelectPalette (hdc, hPalette, FALSE) ;
     RealizePalette (hdc) ;
     UpdateColors (hdc) ;

     ReleaseDC (hwnd, hdc) ;
     break ;


   case WM_DESTROY:
     if (hBitmap)
       DeleteObject (hBitmap) ;

     if (hPalette)
       DeleteObject (hPalette) ;

     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB7.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB7 MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open",   IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib7.rc
#define IDM_FILE_OPEN     40001
```

与前面的程序一样，SHOWDIB7获得了一个指向packed DIB的指标，该DIB回应菜单的「File」、「Open」命令。程序从packed DIB建立了调色盘，然后－还是在WM_COMMAND消息的处理过程中－获得了用于视讯显示的设备内容，并选进调色盘，显现调色盘。然后SHOWDIB7呼叫CreateDIBitmap以便从DIB建立DDB。如果调色盘没有选进设备内容并显现，那么CreateDIBitmap建立的DDB将不使用逻辑调色盘中的附加颜色。

呼叫CreateDIBitmap以后，该程序将释放packed DIB占用的内存空间。pPackedDib变量不是静态变量。相反的，SHOWDIB7按静态变量保留了位图句柄（hBitmap）和逻辑调色盘句柄（hPalette）。

在WM_PAINT消息处理期间，调色盘再次选进设备内容并显现。GetObject函数可获得位图的宽度和高度。然后，程序通过建立兼容的内存设备内容在显示区域显示位图，选进位图，并执行BitBlt。显示DDB时所用的调色盘，必须与从CreateDIBitmap呼叫建立时所用的一样。

如果将位图复制到剪贴簿，则最好使用packed DIB格式。然后Windows可以将位图对象提供

给希望使用这些位图的程序。然而，如果需要将位图对象复制到剪贴簿，则首先要获得视讯设备内容并显现调色盘。这允许Windows依据目前的系统调色盘将DDB转换为DIB。

调色盘和DIB区块

最后，程序16-18所示的SHOWDIB8说明了如何使用带有DIB区块的调色盘。

程序16-18 SHOWDIB8

SHOWDIB8.C

```
/*-----------------------------------------------------------------------------
SHOWDIB8.C -- Shows DIB converted to DIB section
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\ShowDib3\\PackeDib.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("ShowDib8") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Show DIB #8: DIB Section"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HBITMAP hBitmap ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient ;
```

```
static OPENFILENAME ofn ;
static PBYTE pBits ;
static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
static TCHAR szFilter[] = TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
  TEXT ("All Files (*.*)\0*.*\0\0") ;
BITMAP bitmap ;
BITMAPINFO * pPackedDib ;
HDC hdc, hdcMem ;
PAINTSTRUCT ps ;

switch (message)
{
case WM_CREATE:
  ofn.lStructSize = sizeof (OPENFILENAME) ;
  ofn.hwndOwner = hwnd ;
  ofn.hInstance = NULL ;
  ofn.lpstrFilter = szFilter ;
  ofn.lpstrCustomFilter = NULL ;
  ofn.nMaxCustFilter = 0 ;
  ofn.nFilterIndex = 0 ;
  ofn.lpstrFile = szFileName ;
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = szTitleName ;
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.lpstrInitialDir = NULL ;
  ofn.lpstrTitle = NULL ;
  ofn.Flags = 0 ;
  ofn.nFileOffset = 0 ;
  ofn.nFileExtension = 0 ;
  ofn.lpstrDefExt = TEXT ("bmp") ;
  ofn.lCustData = 0 ;
  ofn.lpfnHook = NULL ;
  ofn.lpTemplateName = NULL ;

  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_FILE_OPEN:

    // Show the File Open dialog box

    if (!GetOpenFileName (&ofn))
      return 0 ;

    // If there's an existing packed DIB, free the memory

    if (hBitmap)
    {
      DeleteObject (hBitmap) ;
      hBitmap = NULL ;
    }

    // If there's an existing logical palette, delete it

    if (hPalette)
    {
      DeleteObject (hPalette) ;
      hPalette = NULL ;
    }
    // Load the packed DIB into memory

    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
```

```
    ShowCursor (TRUE) ;

    pPackedDib = PackedDibLoad (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    if (pPackedDib)
    {
      // Create the DIB section from the DIB

      hBitmap = CreateDIBSection (NULL,pPackedDib,DIB_RGB_COLORS,&pBits,NULL, 0) ;

      // Copy the bits

      CopyMemory (pBits, PackedDibGetBitsPtr (pPackedDib),
        PackedDibGetBitsSize (pPackedDib)) ;

      // Create palette from the DIB

      hPalette = PackedDibCreatePalette (pPackedDib) ;

      // Free the packed-DIB memory

      free (pPackedDib) ;
    }
    else
    {
      MessageBox ( hwnd, TEXT ("Cannot load DIB file"),
        szAppName, 0) ;
    }
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  }
  break ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hPalette)
    {
      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;
    }
    if (hBitmap)
    {
      GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;

      hdcMem = CreateCompatibleDC (hdc) ;
      SelectObject (hdcMem, hBitmap) ;

      BitBlt ( hdc, 0, 0, bitmap.bmWidth, bitmap.bmHeight,
        hdcMem, 0, 0, SRCCOPY) ;

      DeleteDC (hdcMem) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_QUERYNEWPALETTE:
    if (!hPalette)
      return FALSE ;

    hdc = GetDC (hwnd) ;
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
    InvalidateRect (hwnd, NULL, TRUE) ;

    ReleaseDC (hwnd, hdc) ;
    return TRUE ;
```

```
   case WM_PALETTECHANGED:
     if (!hPalette || (HWND) wParam == hwnd)
       break ;

     hdc = GetDC (hwnd) ;
     SelectPalette (hdc, hPalette, FALSE) ;
     RealizePalette (hdc) ;
     UpdateColors (hdc) ;

     ReleaseDC (hwnd, hdc) ;
     break ;


   case WM_DESTROY:
     if (hBitmap)
       DeleteObject (hBitmap) ;

     if (hPalette)
       DeleteObject (hPalette) ;

     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

SHOWDIB8.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
SHOWDIB8 MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open",         IDM_FILE_OPEN
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by ShowDib8.rc
#define IDM_FILE_OPEN      40001
```

在SHOWDIB7和SHOWDIB8中的WM_PAINT处理是一样的：两个程序都将位图句柄（hBitmap）和逻辑调色盘句柄（hPalette）作为静态变量。调色盘被选进设备内容并显现，位图的宽度和高度从GetObject函数获得，程序建立内存设备内容并选进位图，然后通过呼叫BitBlt将位图显示到显示区域。

两个程序之间最大的差别在于处理「File」、「Open」菜单命令的程序。在获得指向packed DIB的指标并建立了调色盘以后，SHOWDIB7必须将调色盘选进视讯设备内容，并在呼叫CreateDIBitmap之前显现。SHOWDIB8在获得packed DIB指标以后呼叫CreateDIBSection。不必将调色盘选进设备内容，这是因为CreateDIBSection不将DIB转换成设备相关的格式。的确，CreateDIBSection的第一个参数（即设备内容句柄）的唯一用途在于您是否使用DIB_PAL_COLORS旗标。

呼叫CreateDIBSection以后，SHOWDIB8将像素位从packed DIB复制到从

CreateDIBSection函数传回的内存位置，然后呼叫PackedDibCreatePalette。尽管此函数便于程序使用，但是SHOWDIB8将依据从GetDIBColorTable函数传回的信息建立调色盘。

DIB 处理链接库

就是现在－经过我们长时间地学习GDI位图对象、设备无关位图、DIB区块和Windows调色盘管理器之后－我们才做好了开发一套有助于处理位图的函数的准备。

前面的PACKEDIB文件展示了一种可能的方法：内存中的packed DIB只用指向它的指标表示。程序所的有关DIB的全部信息都可以从存取表头信息结构的函数获得。然而，实际上到「get pixel」和「set pixel」例程时，这种方法就会产生严重的执行问题。图像处理任务当然需要存取位图位，并且这些函数也应该尽可能地快。

可能的C++的解决方式中包括建立DIB类别，这时指向packed DIB的指标正好是一个成员变量。其它成员变量和成员函数有助于更快地执行获得和设定DIB中的像素的例程。不过，因为我在第一章已经指出，对于本书您只需要了解C，使用C++将是其它书的范围。

当然，用C++能做的事情用C也能做。一个好的例子就是许多Windows函数都使用句柄。除了将句柄当作数值以外，应用程序对它还了解什么呢？程序知道句柄引用特殊的函数对象，还知道函数用于处理现存的对象。显然，操作系统按某种方式用句柄来引用对象的内部信息。句柄可以与结构指针一样简单。

例如，假设有一个函数集，这些函数都使用一个称为HDIB的句柄。HDIB是什么呢？它可能在某个表头文件中定义如下：

typedef void * HDIB ;

此定义用「不关您的事」回答了「HDIB是什么」这个问题。

然而，实际上HDIB可能是结构指针，该结构不仅包括指向packed DIB的指针，还包括其它信息：

```
typedef struct
{
  BITMAPINFO * pPackedDib ;
  int cx, cy, cBitsPerPixel, cBytesPerRow ;
  BYTE * pBits ;
}
DIBSTRUCTURE, * PDIBSTRUCTURE ;
```

此结构的其它五个字段包括从packed DIB中引出的信息。当然，结构中这些值允许更快速地存取它们。不同的DIB链接库函数都可以处理这个结构，而不是pPackedDib指标。可以按下面的方法来执行DibGetPixelPointer函数：

```
BYTE * DibGetPixelPointer (HDIB hdib, int x, int y)
{
  PDIBSTRUCTURE pdib = hdib ;
  return pdib->pBits + y * pdib->cBytesPerRow +
    x * pdib->cBitsPerPixel / 8 ;
}
```

当然，这种方法可能要比PACKEDIB.C中执行「get pixel」例程快。

由于这种方法非常合理，所以我决定放弃packed DIB，并改用处理DIB区块的DIB链接库。这实际上使我们对packed DIB的处理有更大的弹性（也就是说，能够在设备无关的方式下操纵DIB像素位），而且在Windows NT下执行时将更有效。

DIBSTRUCT结构

DIBHELP.C文件 – 如此命名是因为对处理DIB提供帮助 – 有上千行，并在几个小部分中显示。但是首先让我们看一下DIBHELP函数所处理的结构，该结构在DIBHELP.C中定义如下：

```
typedef struct
{
  PBYTE * ppRow ; // array of row pointers
  int iSignature ; // = "Dib "
  HBITMAP hBitmap ; // handle returned from CreateDIBSection
  BYTE * pBits ; // pointer to bitmap bits
  DIBSECTION ds ; // DIBSECTION structure
  int iRShift[3] ; // right-shift values for color masks
  int iLShift[3] ; // left-shift values for color masks
}
DIBSTRUCT, * PDIBSTRUCT ;
```

现在跳过第一个字段。它之所以为第一个字段是因为它使某些宏更易于使用 – 在讨论完其它字段以后再来理解第一个字段就更容易了。

在DIBHELP.C中，当DIB建立的函数首先设定了此结构时，第二个字段就设定为文字字符串「Dib」的二进制值。通过一些DIBHELP函数，第二个字段将用于结构有效指针的一个标记。

第三个字段，即hBitmap，是从CreateDIBSection函数传回的位图句柄。您将想起该句柄可有多种使用方式，它与我们在第十四章遇到的GDI位图对象的句柄用法一样。不过，从CreateDIBSection传回的句柄将涉及按设备无关格式储存的位图，该位图格式一直储存到通过呼叫BitBlt和StretchBlt来将位图画到输出设备。

DIBSTRUCT的第四个字段是指向位图位的指针。此值也可由CreateDIBSection函数设定。您将想起，操作系统将控制这个内存块，但应用程序有存取它的权限。在删除位图句柄时，内存块将自动释放。

DIBSTRUCT的第五个字段是DIBSECTION结构。如果您有从CreateDIBSection传回的位图句柄，那么您可以将句柄传递给GetObject函数以获得有关DIBSECTION结构中的位图信息：

GetObject (hBitmap, sizeof (DIBSECTION), &ds) ;

作为提示，DIBSECTION结构在WINGDI.H中定义如下：

```
typedef struct tagDIBSECTION {
  BITMAP dsBm ;
  BITMAPINFOHEADER dsBmih ;
  DWORD dsBitfields[3] ; // Color masks
  HANDLE dshSection ;
  DWORD dsOffset ;
}
DIBSECTION, * PDIBSECTION ;
```

第一个字段是BITMAP结构，它与CreateBitmapIndirect一起建立位图对象，与GetObject一起传回关于DDB的信息。第二个字段是BITMAPINFOHEADER结构。不管位图信息结构是否传递给CreateDIBSection函数，DIBSECTION结构总有BITMAPINFOHEADER结构而不是其它结构，例如BITMAPCOREHEADER结构。这意味着在存取此结构时，DIBHELP.C中的许多函数都不必检查与OS/2兼容的DIB。

对于16位和32位的DIB，如果BITMAPINFOHEADER结构的biCompression字段是BI_BITFIELDS，那么在信息表头结构后面通常有三个屏蔽值。这些屏蔽值决定如何将16位和32位像素值转换成RGB颜色。屏蔽储存在DIBSECTION结构的第三个字段中。

DIBSECTION结构的最后两个字段指的是DIB区块，此区块由文件映射建立。DIBHELP不使用CreateDIBSection的这个特性，因此可以忽略这些字段。

DIBSTRUCT的最后两个字段储存左右移位值，这些值用于处理16位和32位DIB的颜色屏蔽。我们将在第十五章讨论这些移位值。

让我们再回来看一下DIBSTRUCT的第一个字段。正如我们所看到的一样，在开始建立DIB时，此字段设定为指向一个指针数组的指针，该数组中的每个指针都指向DIB中的一行像素。这些指标允许以更快的方式来获得DIB像素位，同时也被定义，以便顶行可以首先引用DIB像素位。此数组的最后一个元素 – 引用DIB图像的最底行 – 通常等于DIBSTRUCT的pBits字段。

信息函数

DIBHELP.C以定义DIBSTRUCT结构开始，然后提供一个函数集，此函数集允许应用程序获得有关DIB区块的信息。程序16-19显示了DIBHELP.C的第一部分。

程序16-19 DIBHELP.C文件的第一部分

DIBHELP.C （第一部分）

```
/*----------------------------------------------------------------------------
DIBHELP.C -- DIB Section Helper Routines
(c) Charles Petzold, 1998
----------------------------------------------------------------------------*/
#include <windows.h>
#include "dibhelp.h"

#define HDIB_SIGNATURE (* (int *) "Dib ")
typedef struct
{
  PBYTE * ppRow ; // must be first field for macros!
  int iSignature ;
  HBITMAP hBitmap ;
  BYTE * pBits ;
  DIBSECTION ds ;
  int  iRShift[3] ;
  int  iLShift[3] ;
}
DIBSTRUCT, * PDIBSTRUCT ;

/*----------------------------------------------------------------------------
DibIsValid: Returns TRUE if hdib points to a valid DIBSTRUCT
----------------------------------------------------------------------------*/

BOOL DibIsValid (HDIB hdib)
{
  PDIBSTRUCT pdib = hdib ;
  if (pdib == NULL)
    return FALSE ;
  if (IsBadReadPtr (pdib, sizeof (DIBSTRUCT)))
    return FALSE ;
  if (pdib->iSignature != HDIB_SIGNATURE)
    return FALSE ;
  return TRUE ;
}

/*----------------------------------------------------------------------------
DibBitmapHandle: Returns the handle to the DIB section bitmap object
----------------------------------------------------------------------------*/

HBITMAP DibBitmapHandle (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return NULL ;
  return ((PDIBSTRUCT) hdib)->hBitmap ;
}

/*----------------------------------------------------------------------------
DibWidth: Returns the bitmap pixel width
```

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

int DibWidth (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return 0 ;
  return ((PDIBSTRUCT) hdib)->ds.dsBm.bmWidth ;
}

/*-----------------------------------------------------------------------------
DibHeight: Returns the bitmap pixel height
-----------------------------------------------------------------------------*/

int DibHeight (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return 0 ;
  return ((PDIBSTRUCT) hdib)->ds.dsBm.bmHeight ;
}

/*-----------------------------------------------------------------------------
DibBitCount: Returns the number of bits per pixel
-----------------------------------------------------------------------------*/

int DibBitCount (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return 0 ;

  return ((PDIBSTRUCT) hdib)->ds.dsBm.bmBitsPixel ;
}

/*-----------------------------------------------------------------------------
DibRowLength: Returns the number of bytes per row of pixels
-----------------------------------------------------------------------------*/

int DibRowLength (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return 0 ;
  return 4 * ((DibWidth (hdib) * DibBitCount (hdib) + 31) / 32) ;
}

/*-----------------------------------------------------------------------------
DibNumColors: Returns the number of colors in the color table
-----------------------------------------------------------------------------*/

int DibNumColors (HDIB hdib)
{
  PDIBSTRUCT pdib = hdib ;

  if (!DibIsValid (hdib))
    return 0 ;

  if (pdib->ds.dsBmih.biClrUsed != 0)
  {
    return pdib->ds.dsBmih.biClrUsed ;
  }
  else if (DibBitCount (hdib) <= 8)
  {
    return 1 << DibBitCount (hdib) ;
  }
  return 0 ;
}
/*-----------------------------------------------------------------------------
DibMask: Returns one of the color masks
-----------------------------------------------------------------------------*/

DWORD DibMask (HDIB hdib, int i)
```

```
{
  PDIBSTRUCT pdib = hdib ;

  if (!DibIsValid (hdib) || i < 0 || i > 2)
    return 0 ;

  return pdib->ds.dsBitfields[i] ;
}

/*-------------------------------------------------------------------------
DibRShift: Returns one of the right-shift values
-------------------------------------------------------------------------*/

int DibRShift (HDIB hdib, int i)
{
  PDIBSTRUCT pdib = hdib ;

  if (!DibIsValid (hdib) || i < 0 || i > 2)
    return 0 ;

  return pdib->iRShift[i] ;
}

/*-------------------------------------------------------------------------
DibLShift: Returns one of the left-shift values
-------------------------------------------------------------------------*/

int DibLShift (HDIB hdib, int i)
{
  PDIBSTRUCT pdib = hdib ;

  if (!DibIsValid (hdib) || i < 0 || i > 2)
    return 0 ;

  return pdib->iLShift[i] ;
}

/*-------------------------------------------------------------------------
DibCompression: Returns the value of the biCompression field
-------------------------------------------------------------------------*/
int DibCompression (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return 0 ;

  return ((PDIBSTRUCT) hdib)->ds.dsBmih.biCompression ;
}

/*-------------------------------------------------------------------------
DibIsAddressable: Returns TRUE if the DIB is not compressed
-------------------------------------------------------------------------*/

BOOL DibIsAddressable (HDIB hdib)
{
  int iCompression ;

  if (!DibIsValid (hdib))
    return FALSE ;

  iCompression = DibCompression (hdib) ;

  if ( iCompression == BI_RGB || iCompression == BI_BITFIELDS)
    return TRUE ;

  return FALSE ;
}

/*-------------------------------------------------------------------------
These functions return the sizes of various components of the DIB section
```

```
AS THEY WOULD APPEAR in a packed DIB. These functions aid in converting
the DIB section to a packed DIB and in saving DIB files.
-------------------------------------------------------------------------*/

DWORD DibInfoHeaderSize (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return 0 ;
  return ((PDIBSTRUCT) hdib)->ds.dsBmih.biSize ;
}

DWORD DibMaskSize (HDIB hdib)
{
  PDIBSTRUCT pdib = hdib ;
  if (!DibIsValid (hdib))
    return 0 ;

  if (pdib->ds.dsBmih.biCompression == BI_BITFIELDS)
    return 3 * sizeof (DWORD) ;

  return 0 ;
}

DWORD DibColorSize (HDIB hdib)
{
  return DibNumColors (hdib) * sizeof (RGBQUAD) ;
}

DWORD DibInfoSize (HDIB hdib)
{
  return DibInfoHeaderSize(hdib) + DibMaskSize(hdib) + DibColorSize(hdib) ;
}

DWORD DibBitsSize (HDIB hdib)
{
  PDIBSTRUCT pdib = hdib ;

  if (!DibIsValid (hdib))
    return 0 ;

  if (pdib->ds.dsBmih.biSizeImage != 0)
  {
    return pdib->ds.dsBmih.biSizeImage ;
  }
  return DibHeight (hdib) * DibRowLength (hdib) ;
}

DWORD DibTotalSize (HDIB hdib)
{
  return DibInfoSize (hdib) + DibBitsSize (hdib) ;
}

/*-------------------------------------------------------------------------
These functions return pointers to the various components of the DIB
section.
-------------------------------------------------------------------------*/
BITMAPINFOHEADER * DibInfoHeaderPtr (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return NULL ;
  return & (((PDIBSTRUCT) hdib)->ds.dsBmih) ;
}

DWORD * DibMaskPtr (HDIB hdib)
{
  PDIBSTRUCT pdib = hdib ;
  if (!DibIsValid (hdib))
    return 0 ;
  return pdib->ds.dsBitfields ;
```

```
}

void * DibBitsPtr (HDIB hdib)
{
  if (!DibIsValid (hdib))
    return NULL ;
  return ((PDIBSTRUCT) hdib)->pBits ;
}

/*-------------------------------------------------------------------------
DibSetColor: Obtains entry from the DIB color table
-------------------------------------------------------------------------*/

BOOL DibGetColor (HDIB hdib, int index, RGBQUAD * prgb)
{
  PDIBSTRUCT pdib = hdib ;
  HDC  hdcMem ;
  int  iReturn ;

  if (!DibIsValid (hdib))
    return 0 ;
  hdcMem = CreateCompatibleDC (NULL) ;
  SelectObject (hdcMem, pdib->hBitmap) ;
  iReturn = GetDIBColorTable (hdcMem, index, 1, prgb) ;
  DeleteDC (hdcMem) ;
  return iReturn ? TRUE : FALSE ;
}

/*-------------------------------------------------------------------------
DibGetColor: Sets an entry in the DIB color table
-------------------------------------------------------------------------*/

BOOL DibSetColor (HDIB hdib, int index, RGBQUAD * prgb)
{
  PDIBSTRUCT pdib = hdib ;
  HDC hdcMem ;
  int iReturn ;

  if (!DibIsValid (hdib))
    return 0 ;
  hdcMem = CreateCompatibleDC (NULL) ;
  SelectObject (hdcMem, pdib->hBitmap) ;
  iReturn = SetDIBColorTable (hdcMem, index, 1, prgb) ;
  DeleteDC (hdcMem) ;

  return iReturn ? TRUE : FALSE ;
}
```

DIBHELP.C中的大部分函数是不用解释的。DibIsValid函数能有助于保护整个系统。在试图引用DIBSTRUCT中的信息之前，其它函数都呼叫DibIsValid。所有这些函数都有（而且通常是只有）HDIB型态的第一个参数，（我们将立即看到）该参数在DIBHELP.H中定义为空指标。这些函数可以将此参数储存到PDIBSTRUCT，然后再存取结构中的字段。

注意传回BOOL值的DibIsAddressable函数。DibIsNotCompressed函数也可以呼叫此函数。传回值表示独立的DIB像素能否寻址。

以DibInfoHeaderSize开始的函数集将取得DIB区块中不同组件出现在packed DIB中的大小。与我们所看到的一样，这些函数有助于将DIB区块转换成packed DIB，并储存DIB文件。这些函数的后面是获得指向不同DIB组件的指针的函数集。

尽管DIBHELP.C包括名称为DibInfoHeaderPtr的函数，而且该函数将获得指向BITMAPINFOHEADER结构的指针，但还是没有函数可以获得BITMAPINFO结构指针－即接在DIB颜色表后面的信息结构。这是因为在处理DIB区块时，应用程序并不直接存取这种型态的结构。BITMAPINFOHEADER结构和颜色屏蔽都在DIBSECTION结构中有效，而且从CreateDIBSection函

数传回指向像素位的指针，这时通过呼叫GetDIBColorTable和SetDIBColorTable，就只能间接存取DIB颜色表。这些功能都封装到DIBHELP的DibGetColor和DibSetColor函数里头了。

在DIBHELP.C的后面，文件DibCopyToInfo配置一个指向BITMAPINFO结构的指针，并填充信息，但是那与获得指向内存中现存结构的指针不完全相同。

读、写像素

应用程序维护packed DIB或DIB区块的一个引人注目的优点是能够直接操作DIB像素位。程序16-20所示的DIBHELP.C第二部分列出了提供此功能的函数。

程序16-20 DIBHELP.C文件的第二部分

DIBHELP.C （第二部分）

```
/*-------------------------------------------------------------------------
DibPixelPtr: Returns a pointer to the pixel at position (x, y)
-------------------------------------------------------------------------*/
BYTE * DibPixelPtr (HDIB hdib, int x, int y)
{
  if (!DibIsAddressable (hdib))
    return NULL ;
  if (x < 0 || x >= DibWidth (hdib) || y < 0 || y >= DibHeight (hdib))
    return NULL ;
  return (((PDIBSTRUCT) hdib)->ppRow)[y] + (x * DibBitCount (hdib) >> 3) ;
}

/*-------------------------------------------------------------------------
DibGetPixel: Obtains a pixel value at (x, y)
-------------------------------------------------------------------------*/

DWORD DibGetPixel (HDIB hdib, int x, int y)
{
  PBYTE pPixel ;
  if (!(pPixel = DibPixelPtr (hdib, x, y)))
    return 0 ;
  switch (DibBitCount (hdib))
  {
  case 1: return 0x01 & (* pPixel >> (7 - (x & 7))) ;
  case 4: return 0x0F & (* pPixel >> (x & 1 ? 0 : 4)) ;
  case 8: return * pPixel ;
  case 16: return * (WORD *) pPixel ;
  case 24: return 0x00FFFFFF & * (DWORD *) pPixel ;
  case 32: return * (DWORD *) pPixel ;
  }
  return 0 ;
}

/*-------------------------------------------------------------------------
DibSetPixel: Sets a pixel value at (x, y)
-------------------------------------------------------------------------*/

BOOL DibSetPixel (HDIB hdib, int x, int y, DWORD dwPixel)
{
  PBYTE pPixel ;
  if (!(pPixel = DibPixelPtr (hdib, x, y)))
    return FALSE ;
  switch (DibBitCount (hdib))
  {
  case 1: * pPixel &= ~(1 << (7 - (x & 7))) ;
    * pPixel |= dwPixel << (7 - (x & 7)) ;
    break ;
  case 4: * pPixel &= 0x0F << (x & 1 ? 4 : 0) ;
    * pPixel |= dwPixel << (x & 1 ? 0 : 4) ;
    break ;
  case 8: * pPixel = (BYTE) dwPixel ;
    break ;
```

```
  case 16: * (WORD *) pPixel = (WORD) dwPixel ;
    break ;

  case 24: * (RGBTRIPLE *) pPixel = * (RGBTRIPLE *) &dwPixel ;
    break ;

  case 32: * (DWORD *) pPixel = dwPixel ;
    break ;
  default:
    return FALSE ;
  }
  return TRUE ;
}

/*-------------------------------------------------------------------------
DibGetPixelColor: Obtains the pixel color at (x, y)
-------------------------------------------------------------------------*/

BOOL DibGetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb)
{
  DWORD dwPixel ;
  int iBitCount ;
  PDIBSTRUCT pdib = hdib ;

  // Get bit count; also use this as a validity check

  if (0 == (iBitCount = DibBitCount (hdib)))
    return FALSE ;
  // Get the pixel value
  dwPixel = DibGetPixel (hdib, x, y) ;
  // If the bit-count is 8 or less, index the color table
  if (iBitCount <= 8)
    return DibGetColor (hdib, (int) dwPixel, prgb) ;
  // If the bit-count is 24, just use the pixel
  else if (iBitCount == 24)
  {
    * (RGBTRIPLE *) prgb = * (RGBTRIPLE *) & dwPixel ;
    prgb->rgbReserved = 0 ;
  }

  // If the bit-count is 32 and the biCompression field is BI_RGB,
  // just use the pixel

  else if (iBitCount == 32 &&
    pdib->ds.dsBmih.biCompression == BI_RGB)
  {
    * prgb = * (RGBQUAD *) & dwPixel ;
  }

  // Otherwise, use the mask and shift values
  // (for best performance, don't use DibMask and DibShift functions)
  else
  {
    prgb->rgbRed = (BYTE)(((pdib->ds.dsBitfields[0] & dwPixel)
      >> pdib->iRShift[0]) << pdib->iLShift[0]) ;

    prgb->rgbGreen=(BYTE((pdib->ds.dsBitfields[1] & dwPixel)
      >> pdib->iRShift[1]) << pdib->iLShift[1]) ;

    prgb->rgbBlue=(BYTE)(((pdib->ds.dsBitfields[2] & dwPixel)
      >> pdib->iRShift[2]) << pdib->iLShift[2]) ;
  }
  return TRUE ;
}

/*-------------------------------------------------------------------------
DibSetPixelColor: Sets the pixel color at (x, y)
-------------------------------------------------------------------------*/
```

```
BOOL DibSetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb)
{
  DWORD dwPixel ;
  int iBitCount ;
  PDIBSTRUCT pdib = hdib ;

  // Don't do this function for DIBs with color tables

  iBitCount = DibBitCount (hdib) ;
  if (iBitCount <= 8)
    return FALSE ;
  // The rest is just the opposite of DibGetPixelColor
  else if (iBitCount == 24)
  {
    * (RGBTRIPLE *) & dwPixel = * (RGBTRIPLE *) prgb ;
    dwPixel &= 0x00FFFFFF ;
  }
  else if (iBitCount == 32 &&
    pdib->ds.dsBmih.biCompression == BI_RGB)
  {
    * (RGBQUAD *) & dwPixel = * prgb ;
  }

  else
  {
    dwPixel = (((DWORD) prgb->rgbRed >> pdib->iLShift[0])
      << pdib->iRShift[0]) ;

    dwPixel |= (((DWORD) prgb->rgbGreen >> pdib->iLShift[1])
      << pdib->iRShift[1]) ;

    dwPixel |= (((DWORD) prgb->rgbBlue >> pdib->iLShift[2])
      << pdib->iRShift[2]) ;
  }

  DibSetPixel (hdib, x, y, dwPixel) ;
  return TRUE ;
}
```

这部分DIBHELP.C从DibPixelPtr函数开始，该函数获得指向储存（或部分储存）有特殊像素的字节的指针。回想一下DIBSTRUCT结构的ppRow字段，那是个指向DIB中由顶行开始排列的像素行地址的指针。这样，

((PDIBSTRUCT) hdib)->pprow)[0]

就是指向DIB顶行最左端像素的指标，而

(((PDIBSTRUCT) hdib)->ppRow)[y] + (x * DibBitCount (hdib) >> 3)

是指向位于(x,y)的像素的指标。注意，如果DIB中的像素不可被寻址（即如果已压缩），或者如果函数的x和y参数是负数或相对位于DIB外面的区域，则函数将传回NULL。此检查降低了函数（和所有依赖于DibPixelPtr的函数）的执行速度，下面我将讲述一些更快的例程。

文件后面的DibGetPixel和DibSetPixel函数利用了DibPixelPtr。对于8位、16位和32位DIB，这些函数只记录指向合适数据尺寸的指针，并存取像素值。对于1位和4位的DIB，则需要屏蔽和移位角度。

DibGetColor函数按RGBQUAD结构获得像素颜色。对于1位、4位和8位DIB，这包括使用像素值来从DIB颜色表获得颜色。对于16位、24位和32位DIB，通常必须将像素值屏蔽和移位以得到RGB颜色。DibSetPixel函数则相反，它允许从RGBQUAD结构设定像素值。该函数只为16位、24位和32位DIB定义。

建立和转换

程序16-21所示的DIBHELP第三部分和最后部分展示了如何建立DIB区块，以及如何将DIB区块与packed DIB相互转换。

程序16-21 DIBHELP.C文件的第三部分和最后部分

DIBHELP.C （第三部分）

```
/*-------------------------------------------------------------------------
Calculating shift values from color masks is required by the
DibCreateFromInfo function.
-------------------------------------------------------------------------*/
static int MaskToRShift (DWORD dwMask)
{
  int iShift ;
  if (dwMask == 0)
    return 0 ;
  for (iShift = 0 ; !(dwMask & 1) ; iShift++)
    dwMask >>= 1 ;
  return iShift ;
}

static int MaskToLShift (DWORD dwMask)
{
  int iShift ;
  if (dwMask == 0)
    return 0 ;
  while (!(dwMask & 1))
    dwMask >>= 1 ;
  for (iShift = 0 ; dwMask & 1 ; iShift++)
    dwMask >>= 1 ;
  return 8 - iShift ;
}
/*-------------------------------------------------------------------------
DibCreateFromInfo: All DIB creation functions ultimately call this one.
This function is responsible for calling CreateDIBSection, allocating
memory for DIBSTRUCT, and setting up the row pointer.
-------------------------------------------------------------------------*/

HDIB DibCreateFromInfo (BITMAPINFO * pbmi)
{
  BYTE  * pBits ;
  DIBSTRUCT * pdib ;
  HBITMAP  hBitmap ;
  int   i, iRowLength, cy, y ;

  hBitmap = CreateDIBSection (NULL, pbmi, DIB_RGB_COLORS, &pBits, NULL, 0) ;
  if (hBitmap == NULL)
    return NULL ;
  if (NULL == (pdib = malloc (sizeof (DIBSTRUCT))))
  {
    DeleteObject (hBitmap) ;
    return NULL ;
  }

  pdib->iSignature = HDIB_SIGNATURE ;
  pdib->hBitmap = hBitmap ;
  pdib->pBits = pBits ;

  GetObject (hBitmap, sizeof (DIBSECTION), &pdib->ds) ;
  // Notice that we can now use the DIB information functions
  // defined above.

  // If the compression is BI_BITFIELDS, calculate shifts from masks

  if (DibCompression (pdib) == BI_BITFIELDS)
  {
    for (i = 0 ; i < 3 ; i++)
    {
      pdib->iLShift[i] = MaskToLShift (pdib->ds.dsBitfields[i]) ;
```

```
    pdib->iRShift[i] = MaskToRShift (pdib->ds.dsBitfields[i]) ;
   }
  }

  // If the compression is BI_RGB, but bit-count is 16 or 32,
  // set the bitfields and the masks
  else if (DibCompression (pdib) == BI_RGB)
  {
   if (DibBitCount (pdib) == 16)
   {
    pdib->ds.dsBitfields[0] = 0x00007C00 ;
    pdib->ds.dsBitfields[1] = 0x000003E0 ;
    pdib->ds.dsBitfields[2] = 0x0000001F ;

    pdib->iRShift [0] = 10 ;
    pdib->iRShift [1] = 5 ;
    pdib->iRShift [2] = 0 ;

    pdib->iLShift [0] = 3 ;
    pdib->iLShift [1] = 3 ;
    pdib->iLShift [2] = 3 ;
   }
   else if (DibBitCount (pdib) == 24 || DibBitCount (pdib) == 32)
   {
    pdib->ds.dsBitfields[0] = 0x00FF0000 ;
    pdib->ds.dsBitfields[1] = 0x0000FF00 ;
    pdib->ds.dsBitfields[2] = 0x000000FF ;

    pdib->iRShift [0] = 16 ;
    pdib->iRShift [1] = 8 ;
    pdib->iRShift [2] = 0 ;

    pdib->iLShift [0] = 0 ;
    pdib->iLShift [1] = 0 ;
    pdib->iLShift [2] = 0 ;
   }
  }
  // Allocate an array of pointers to each row in the DIB
  cy = DibHeight (pdib) ;
  if (NULL == (pdib->ppRow = malloc (cy * sizeof (BYTE *))))
  {
   free (pdib) ;
   DeleteObject (hBitmap) ;
   return NULL ;
  }

  // Initialize them.
  iRowLength = DibRowLength (pdib) ;
  if (pbmi->bmiHeader.biHeight > 0)  // ie, bottom up
  {
   for (y = 0 ; y < cy ; y++)
    pdib->ppRow[y] = pBits + (cy - y - 1) * iRowLength ;
  }
  else
   // top down
  {
   for (y = 0 ; y < cy ; y++)
    pdib->ppRow[y] = pBits + y * iRowLength ;
  }
  return pdib ;
}

/*-------------------------------------------------------------------------
DibDelete: Frees all memory for the DIB section
-------------------------------------------------------------------------*/

BOOL DibDelete (HDIB hdib)
{
  DIBSTRUCT * pdib = hdib ;
```

```
  if (!DibIsValid (hdib))
    return FALSE ;
  free (pdib->ppRow) ;
  DeleteObject (pdib->hBitmap) ;
  free (pdib) ;
  return TRUE ;
}

/*-------------------------------------------------------------------------
DibCreate: Creates an HDIB from explicit arguments
-------------------------------------------------------------------------*/

HDIB DibCreate (int cx, int cy, int cBits, int cColors)
{
  BITMAPINFO * pbmi ;
  DWORD  dwInfoSize ;
  HDIB   hDib ;
  int   cEntries ;

  if (cx <= 0 || cy <= 0 ||
    ((cBits != 1) && (cBits != 4) && (cBits != 8) &&
    (cBits != 16) && (cBits != 24) && (cBits != 32)))
  {
    return NULL ;
  }

  if ( cColors != 0)
    cEntries = cColors ;
  else if (cBits <= 8)
    cEntries = 1 << cBits ;

  dwInfoSize = sizeof (BITMAPINFOHEADER) + (cEntries - 1) * sizeof (RGBQUAD);

  if (NULL == (pbmi = malloc (dwInfoSize)))
  {
    return NULL ;
  }

  ZeroMemory (pbmi, dwInfoSize) ;

  pbmi->bmiHeader.biSize  = sizeof (BITMAPINFOHEADER) ;
  pbmi->bmiHeader.biWidth  = cx ;
  pbmi->bmiHeader.biHeight  = cy ;
  pbmi->bmiHeader.biPlanes  = 1 ;
  pbmi->bmiHeader.biBitCount  = cBits ;
  pbmi->bmiHeader.biCompression = BI_RGB ;
  pbmi->bmiHeader.biSizeImage  = 0 ;
  pbmi->bmiHeader.biXPelsPerMeter = 0 ;
  pbmi->bmiHeader.biYPelsPerMeter = 0 ;
  pbmi->bmiHeader.biClrUsed  = cColors ;
  pbmi->bmiHeader.biClrImportant = 0 ;

  hDib = DibCreateFromInfo (pbmi) ;
  free (pbmi) ;

  return hDib ;
}

/*-------------------------------------------------------------------------
DibCopyToInfo: Builds BITMAPINFO structure.
Used by DibCopy and DibCopyToDdb
-------------------------------------------------------------------------*/

static BITMAPINFO * DibCopyToInfo (HDIB hdib)
{
  BITMAPINFO * pbmi ;
  int   i, iNumColors ;
  RGBQUAD  * prgb ;
  if (!DibIsValid (hdib))
```

```
    return NULL ;
  // Allocate the memory
  if (NULL == (pbmi = malloc (DibInfoSize (hdib))))
    return NULL ;
  // Copy the information header
  CopyMemory (pbmi, DibInfoHeaderPtr (hdib), sizeof (BITMAPINFOHEADER));

  // Copy the possible color masks

  prgb = (RGBQUAD *) ((BYTE *) pbmi + sizeof (BITMAPINFOHEADER)) ;
  if (DibMaskSize (hdib))
  {
    CopyMemory (prgb, DibMaskPtr (hdib), 3 * sizeof (DWORD)) ;
    prgb = (RGBQUAD *) ((BYTE *) prgb + 3 * sizeof (DWORD)) ;
  }
  // Copy the color table
  iNumColors = DibNumColors (hdib) ;
  for (i = 0 ; i < iNumColors ; i++)
    DibGetColor (hdib, i, prgb + i) ;
  return pbmi ;
}

/*-------------------------------------------------------------------------
DibCopy: Creates a new DIB section from an existing DIB section,
possibly swapping the DIB width and height.
-------------------------------------------------------------------------*/

HDIB DibCopy (HDIB hdibSrc, BOOL fRotate)
{
  BITMAPINFO * pbmi ;
  BYTE  * pBitsSrc, * pBitsDst ;
  HDIB hdibDst ;

  if (!DibIsValid (hdibSrc))
    return NULL ;
  if (NULL == (pbmi = DibCopyToInfo (hdibSrc)))
    return NULL ;
  if (fRotate)
  {
    pbmi->bmiHeader.biWidth = DibHeight (hdibSrc) ;
    pbmi->bmiHeader.biHeight = DibWidth (hdibSrc) ;
  }

  hdibDst = DibCreateFromInfo (pbmi) ;
  free (pbmi) ;

  if ( hdibDst == NULL)
    return NULL ;

  // Copy the bits

  if (!fRotate)
  {
    pBitsSrc = DibBitsPtr (hdibSrc) ;
    pBitsDst = DibBitsPtr (hdibDst) ;

    CopyMemory (pBitsDst, pBitsSrc, DibBitsSize (hdibSrc)) ;
  }
  return hdibDst ;
}

/*-------------------------------------------------------------------------
DibCopyToPackedDib is generally used for saving DIBs and for
transferring DIBs to the clipboard. In the second case, the second
argument should be set to TRUE so that the memory is allocated
with the GMEM_SHARE flag.
-------------------------------------------------------------------------*/

BITMAPINFO * DibCopyToPackedDib (HDIB hdib, BOOL fUseGlobal)
```

```
{
  BITMAPINFO * pPackedDib ;
  BYTE  * pBits ;
  DWORD  dwDibSize ;
  HDC  hdcMem ;
  HGLOBAL hGlobal ;
  int  iNumColors ;
  PDIBSTRUCT pdib = hdib ;
  RGBQUAD * prgb ;

  if (!DibIsValid (hdib))
    return NULL ;
  // Allocate memory for packed DIB
  dwDibSize = DibTotalSize (hdib) ;
  if (fUseGlobal)
  {
    hGlobal = GlobalAlloc (GHND | GMEM_SHARE, dwDibSize) ;
    pPackedDib = GlobalLock (hGlobal) ;
  }
  else
  {
    pPackedDib = malloc (dwDibSize) ;
  }

  if (pPackedDib == NULL)
    return NULL ;
  // Copy the information header
  CopyMemory (pPackedDib, &pdib->ds.dsBmih, sizeof (BITMAPINFOHEADER)) ;
  prgb = (RGBQUAD *) ((BYTE *) pPackedDib + sizeof (BITMAPINFOHEADER)) ;
  // Copy the possible color masks
  if (pdib->ds.dsBmih.biCompression == BI_BITFIELDS)
  {
    CopyMemory (prgb, pdib->ds.dsBitfields, 3 * sizeof (DWORD)) ;
    prgb = (RGBQUAD *) ((BYTE *) prgb + 3 * sizeof (DWORD)) ;
  }
  // Copy the color table
  if (iNumColors = DibNumColors (hdib))
  {
    hdcMem = CreateCompatibleDC (NULL) ;
    SelectObject (hdcMem, pdib->hBitmap) ;
    GetDIBColorTable (hdcMem, 0, iNumColors, prgb) ;
    DeleteDC (hdcMem) ;
  }

  pBits = (BYTE *) (prgb + iNumColors) ;
  // Copy the bits
  CopyMemory (pBits, pdib->pBits, DibBitsSize (pdib)) ;
  // If last argument is TRUE, unlock global memory block and
  // cast it to pointer in preparation for return

  if (fUseGlobal)
  {
    GlobalUnlock (hGlobal) ;
    pPackedDib = (BITMAPINFO *) hGlobal ;
  }
  return pPackedDib ;
}

/*-------------------------------------------------------------------------
DibCopyFromPackedDib is generally used for pasting DIBs from the
clipboard.
-------------------------------------------------------------------------*/

HDIB DibCopyFromPackedDib (BITMAPINFO * pPackedDib)
{
  BYTE  * pBits ;
  DWORD  dwInfoSize, dwMaskSize, dwColorSize ;
  int   iBitCount ;
  PDIBSTRUCT pdib ;
```

```
  // Get the size of the information header and do validity check

  dwInfoSize = pPackedDib->bmiHeader.biSize ;
  if ( dwInfoSize != sizeof (BITMAPCOREHEADER) &&
    dwInfoSize != sizeof (BITMAPINFOHEADER) &&
    dwInfoSize != sizeof (BITMAPV4HEADER) &&
    dwInfoSize != sizeof (BITMAPV5HEADER))
  {
    return NULL ;
  }
  // Get the possible size of the color masks

  if (dwInfoSize == sizeof (BITMAPINFOHEADER) &&
    pPackedDib->bmiHeader.biCompression == BI_BITFIELDS)
  {
    dwMaskSize = 3 * sizeof (DWORD) ;
  }
  else
  {
    dwMaskSize = 0 ;
  }
  // Get the size of the color table
  if (dwInfoSize == sizeof (BITMAPCOREHEADER))
  {
    iBitCount = ((BITMAPCOREHEADER *) pPackedDib)->bcBitCount ;

    if (iBitCount <= 8)
    {
      dwColorSize = (1 << iBitCount) * sizeof (RGBTRIPLE) ;
    }
    else
      dwColorSize = 0 ;
  }
  else   // all non-OS/2 compatible DIBs
  {
    if (pPackedDib->bmiHeader.biClrUsed > 0)
    {
      dwColorSize = pPackedDib->bmiHeader.biClrUsed * sizeof (RGBQUAD);
    }
    else if (pPackedDib->bmiHeader.biBitCount <= 8)
    {
      dwColorSize = (1 << pPackedDib->bmiHeader.biBitCount) * sizeof (RGBQUAD) ;
    }
    else
    {
      dwColorSize = 0 ;
    }
  }
  // Finally, get the pointer to the bits in the packed DIB
  pBits = (BYTE *) pPackedDib + dwInfoSize + dwMaskSize + dwColorSize ;
  // Create the HDIB from the packed-DIB pointer
  pdib = DibCreateFromInfo (pPackedDib) ;
  // Copy the pixel bits
  CopyMemory (pdib->pBits, pBits, DibBitsSize (pdib)) ;
  return pdib ;
}

/*-------------------------------------------------------------------------
DibFileLoad: Creates a DIB section from a DIB file
-------------------------------------------------------------------------*/
HDIB DibFileLoad (const TCHAR * szFileName)
{
  BITMAPFILEHEADER bmfh ;
  BITMAPINFO  * pbmi ;
  BOOL    bSuccess ;
  DWORD    dwInfoSize, dwBitsSize, dwBytesRead ;
  HANDLE   hFile ;
  HDIB    hDib ;
```

```
  // Open the file: read access, prohibit write access

  hFile = CreateFile (szFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL) ;
  if (hFile == INVALID_HANDLE_VALUE)
    return NULL ;
  // Read in the BITMAPFILEHEADER
  bSuccess = ReadFile ( hFile, &bmfh, sizeof (BITMAPFILEHEADER),
    &dwBytesRead, NULL) ;

  if (!bSuccess || (dwBytesRead != sizeof (BITMAPFILEHEADER))
    || (bmfh.bfType != * (WORD *) "BM"))
  {
    CloseHandle (hFile) ;
    return NULL ;
  }
  // Allocate memory for the information structure & read it in
  dwInfoSize = bmfh.bfOffBits - sizeof (BITMAPFILEHEADER) ;
  if (NULL == (pbmi = malloc (dwInfoSize)))
  {
    CloseHandle (hFile) ;
    return NULL ;
  }

  bSuccess = ReadFile (hFile, pbmi, dwInfoSize, &dwBytesRead, NULL) ;
  if (!bSuccess || (dwBytesRead != dwInfoSize))
  {
    CloseHandle (hFile) ;
    free (pbmi) ;
    return NULL ;
  }
  // Create the DIB
  hDib = DibCreateFromInfo (pbmi) ;
  free (pbmi) ;

  if (hDib == NULL)
  {
    CloseHandle (hFile) ;
    return NULL ;
  }
  // Read in the bits
  dwBitsSize = bmfh.bfSize - bmfh.bfOffBits ;
  bSuccess = ReadFile ( hFile, ((PDIBSTRUCT) hDib)->pBits,
    dwBitsSize, &dwBytesRead, NULL) ;
  CloseHandle (hFile) ;
  if (!bSuccess || (dwBytesRead != dwBitsSize))
  {
    DibDelete (hDib) ;
    return NULL ;
  }
  return hDib ;
}

/*-------------------------------------------------------------------------
DibFileSave: Saves a DIB section to a file
-------------------------------------------------------------------------*/

BOOL DibFileSave (HDIB hdib, const TCHAR * szFileName)
{
  BITMAPFILEHEADER bmfh ;
  BITMAPINFO  * pbmi ;
  BOOL  bSuccess ;
  DWORD  dwTotalSize, dwBytesWritten ;
  HANDLE  hFile ;

  hFile = CreateFile (szFileName, GENERIC_WRITE, 0, NULL,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;
  if (hFile == INVALID_HANDLE_VALUE)
```

```
    return FALSE ;
  dwTotalSize  = DibTotalSize (hdib) ;
  bmfh.bfType  = * (WORD *) "BM" ;
  bmfh.bfSize  = sizeof (BITMAPFILEHEADER) + dwTotalSize ;
  bmfh.bfReserved1 = 0 ;
  bmfh.bfReserved2 = 0 ;
  bmfh.bfOffBits = bmfh.bfSize - DibBitsSize (hdib) ;

  // Write the BITMAPFILEHEADER

  bSuccess = WriteFile (hFile, &bmfh, sizeof (BITMAPFILEHEADER),
    &dwBytesWritten, NULL) ;

  if (!bSuccess || (dwBytesWritten != sizeof (BITMAPFILEHEADER)))
  {
    CloseHandle (hFile) ;
    DeleteFile (szFileName) ;
    return FALSE ;
  }
  // Get entire DIB in packed-DIB format
  if (NULL == (pbmi = DibCopyToPackedDib (hdib, FALSE)))
  {
    CloseHandle (hFile) ;
    DeleteFile (szFileName) ;
    return FALSE ;
  }
  // Write out the packed DIB
  bSuccess = WriteFile (hFile, pbmi, dwTotalSize, &dwBytesWritten, NULL) ;
  CloseHandle (hFile) ;
  free (pbmi) ;

  if (!bSuccess || (dwBytesWritten != dwTotalSize))
  {
    DeleteFile (szFileName) ;
    return FALSE ;
  }
  return TRUE ;
}

/*-------------------------------------------------------------------------
DibCopyToDdb: For more efficient screen displays
-------------------------------------------------------------------------*/
HBITMAP DibCopyToDdb (HDIB hdib, HWND hwnd, HPALETTE hPalette)
{
  BITMAPINFO * pbmi ;
  HBITMAP  hBitmap ;
  HDC   hdc ;

  if (!DibIsValid (hdib))
    return NULL ;
  if (NULL == (pbmi = DibCopyToInfo (hdib)))
    return NULL ;
  hdc = GetDC (hwnd) ;
  if (hPalette)
  {
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
  }

  hBitmap = CreateDIBitmap (hdc, DibInfoHeaderPtr (hdib), CBM_INIT,
    DibBitsPtr (hdib), pbmi, DIB_RGB_COLORS) ;

  ReleaseDC (hwnd, hdc) ;
  free (pbmi) ;

  return hBitmap ;
}
```

这部分的DIBHELP.C文件从两个小函数开始，这两个函数根据16位和32位DIB的颜色屏蔽得到

左、右移位值。这些函数在第十五章「颜色屏蔽」一节说明。

DibCreateFromInfo函数是DIBHELP中唯一呼叫CreateDIBSection并为DIBSTRUCT结构配置内存的函数。其它所有建立和复制函数都重复此函数。DibCreateFromInfo唯一的参数是指向BITMAPINFO结构的指针。此结构的颜色表必须存在，但是它不必用有效的值填充。呼叫CreateDIBSection之后，该函数将初始化DIBSTRUCT结构的所有字段。注意，在设定DIBSTRUCT结构的ppRow字段的值时（指向DIB行地址的指针），DIB有由下而上和由上而下的不同储存方式。ppRow开头的元素就是DIB的顶行。

DibDelete删除DibCreateFromInfo中建立的位图，同时释放在该函数中配置的内存。

DibCreate可能比DibCreateFromInfo更像一个从应用程序呼叫的函数。前三个参数提供像素的宽度、高度和每像素的位数。最后一个参数可以设定为0（用于颜色表的内定尺寸），或者设定为非0（表示比每像素位数所需要的颜色表更小的颜色表）。

DibCopy函数根据现存的DIB区块建立新的DIB区块，并用DibCreateInfo函数为BITMAPINFO结构配置了内存，还填了所有的数据。DibCopy函数的一个BOOL参数指出是否在建立新的DIB时交换了DIB的宽度和高度。我们将在后面看到此函数的用法。

DibCopyToPackedDib和DibCopyFromPackedDib函数的使用通常与透过剪贴簿传递DIB相关。DibFileLoad函数从DIB文件建立DIB区块；DibFileSave函数将数据储存到DIB文件。

最后，DibCopyToDdb函数根据DIB建立GDI位图对象。注意，该函数需要目前调色盘的句柄和程序窗口的句柄。程序窗口句柄用于获得选进并显现调色盘的设备内容。只有这样，函数才可以呼叫CreateDIBitmap。这曾在本章前面的SHOWDIB7中展示。

DIBHELP表头文件和宏

DIBHELP.H表头文件如程序16-22所示。

程序16-22 DIBHELP.H文件

DIBHELP.H

```
/*----------------------------------------------------------------------------
DIBHELP.H header file for DIBHELP.C
----------------------------------------------------------------------------*/
typedef void * HDIB ;
// Functions in DIBHELP.C
BOOL DibIsValid (HDIB hdib) ;
HBITMAP DibBitmapHandle (HDIB hdib) ;
int DibWidth (HDIB hdib) ;
int DibHeight (HDIB hdib) ;
int DibBitCount (HDIB hdib) ;
int DibRowLength (HDIB hdib) ;
int DibNumColors (HDIB hdib) ;
DWORD DibMask (HDIB hdib, int i) ;
int DibRShift (HDIB hdib, int i) ;
int DibLShift (HDIB hdib, int i) ;
int DibCompression (HDIB hdib) ;
BOOL DibIsAddressable (HDIB hdib) ;
DWORD DibInfoHeaderSize (HDIB hdib) ;
DWORD DibMaskSize (HDIB hdib) ;
DWORD DibColorSize (HDIB hdib) ;
DWORD DibInfoSize (HDIB hdib) ;
DWORD DibBitsSize (HDIB hdib) ;
DWORD DibTotalSize (HDIB hdib) ;
BITMAPINFOHEADER * DibInfoHeaderPtr (HDIB hdib) ;
DWORD * DibMaskPtr (HDIB hdib) ;
void * DibBitsPtr (HDIB hdib) ;
BOOL DibGetColor (HDIB hdib, int index, RGBQUAD * prgb) ;
BOOL DibSetColor (HDIB hdib, int index, RGBQUAD * prgb) ;
```

```
BYTE * DibPixelPtr (HDIB hdib, int x, int y) ;
DWORD DibGetPixel (HDIB hdib, int x, int y) ;
BOOL DibSetPixel (HDIB hdib, int x, int y, DWORD dwPixel) ;
BOOL DibGetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb) ;
BOOL DibSetPixelColor (HDIB hdib, int x, int y, RGBQUAD * prgb) ;
HDIB DibCreateFromInfo (BITMAPINFO * pbmi) ;
BOOL DibDelete (HDIB hdib) ;
HDIB DibCreate (int cx, int cy, int cBits, int cColors) ;
HDIB DibCopy (HDIB hdibSrc, BOOL fRotate) ;
BITMAPINFO * DibCopyToPackedDib (HDIB hdib, BOOL fUseGlobal) ;
HDIB DibCopyFromPackedDib (BITMAPINFO * pPackedDib) ;
HDIB DibFileLoad (const TCHAR * szFileName) ;
BOOL DibFileSave (HDIB hdib, const TCHAR * szFileName) ;
HBITMAP DibCopyToDdb (HDIB hdib, HWND hwnd, HPALETTE hPalette) ;
HDIB DibCreateFromDdb (HBITMAP hBitmap) ;

/*---------------------------------------------------------------------------
Quickie no-bounds-checked pixel gets and sets
---------------------------------------------------------------------------*/

#define DibPixelPtr1(hdib, x, y) (((* (PBYTE **) hdib) [y]) + ((x) >> 3))
#define DibPixelPtr4(hdib, x, y) (((* (PBYTE **) hdib) [y]) + ((x) >> 1))
#define DibPixelPtr8(hdib, x, y) (((* (PBYTE **) hdib) [y]) + (x) )
#define DibPixelPtr16(hdib, x, y) \
((WORD *) (((* (PBYTE **) hdib) [y]) + (x) * 2))

#define DibPixelPtr24(hdib, x, y) \
((RGBTRIPLE *) (((* (PBYTE **) hdib) [y]) + (x) * 3))
#define DibPixelPtr32(hdib, x, y) \
((DWORD *) (((* (PBYTE **) hdib) [y]) + (x) * 4))

#define DibGetPixel1(hdib, x, y) \
(0x01 & (* DibPixelPtr1 (hdib, x, y) >> (7 - ((x) & 7))))

#define DibGetPixel4(hdib, x, y) \
(0x0F & (* DibPixelPtr4 (hdib, x, y) >> ((x) & 1 ? 0 : 4)))

#define DibGetPixel8(hdib, x, y) (* DibPixelPtr8 (hdib, x, y))
#define DibGetPixel16(hdib, x, y) (* DibPixelPtr16 (hdib, x, y))
#define DibGetPixel24(hdib, x, y) (* DibPixelPtr24 (hdib, x, y))
#define DibGetPixel32(hdib, x, y) (* DibPixelPtr32 (hdib, x, y))

#define DibSetPixel1(hdib, x, y, p)  \
((* DibPixelPtr1 (hdib, x, y) &= ~( 1<< (7 - ((x) & 7)))), \
 (* DibPixelPtr1 (hdib, x, y) |= ((p) << (7 - ((x) & 7)))))

#define DibSetPixel4(hdib, x, y, p)  \
 ((* DibPixelPtr4 (hdib, x, y) &= (0x0F << ((x) & 1 ? 4 : 0))), \
 (* DibPixelPtr4 (hdib, x, y) |= ((p) << ((x) & 1 ? 0 : 4))))

#define DibSetPixel8(hdib, x, y, p) (* DibPixelPtr8 (hdib, x, y) = p)
#define DibSetPixel16(hdib, x, y, p) (* DibPixelPtr16 (hdib, x, y) = p)
#define DibSetPixel24(hdib, x, y, p) (* DibPixelPtr24 (hdib, x, y) = p)
#define DibSetPixel32(hdib, x, y, p) (* DibPixelPtr32 (hdib, x, y) = p)
```

这个表头文件将HDIB定义为空指标(void*)。应用程序的确不需要了解HDIB所指结构的内部结构。此表头文件还包括DIBHELP.C中所有函数的说明，还有一些宏 – 非常特殊的宏。

如果再看一看DIBHELP.C中的DibPixelPtr、DibGetPixel和DibSetPixel函数，并试图提高它们的执行速度表现，那么您将看到两种可能的解决方法。第一种，可以删除所有的检查保护，并相信应用程序不会使用无效参数呼叫函数。还可以删除一些函数呼叫，例如DibBitCount，并使用指向DIBSTRUCT结构内部的指针来直接获得信息。

提高执行速度表现另一项较不明显的方法是删除所有对每像素位数的处理方式，同时分离出处理不同DIB函数 – 例如DibGetPixel1、DibGetPixel4、DibGetPixel8等等。下一个最佳化步骤是删除整个函数呼叫，将其处理动作透过inline function或宏中进行合并。

DIBHELP.H采用宏的方法。它依据DibPixelPtr、DibGetPixel和DibSetPixel函数提出了三套宏。这些宏都明确对应于特殊的像素位数。

DIBBLE程序

DIBBLE程序，如程序16-23所示，使用DIBHELP函数和宏工作。尽管DIBBLE是本书中最长的程序，它确实只是一些作业的粗略范例，这些作业可以在简单的数字影像处理程序中找到。对DIBBLE的明显改进是转换成了多重文件接口（MDI：multiple document interface），我们将在第十九章学习有关多重文件接口的知识。

程序16-23 DIBBLE

DIBBLE.C

```
/*------------------------------------------------------------------------
DIBBLE.C -- Bitmap and Palette Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "dibhelp.h"
#include "dibpal.h"
#include "dibconv.h"
#include "resource.h"

#define WM_USER_SETSCROLLS (WM_USER + 1)
#define WM_USER_DELETEDIB (WM_USER + 2)
#define WM_USER_DELETEPAL (WM_USER + 3)
#define WM_USER_CREATEPAL (WM_USER + 4)

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Dibble") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, szAppName,
    WS_OVERLAPPEDWINDOW | WM_VSCROLL | WM_HSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

/*--------------------------------------------------------------------------
DisplayDib: Displays or prints DIB actual size or stretched
depending on menu selection
--------------------------------------------------------------------------*/
int DisplayDib ( HDC hdc, HBITMAP hBitmap, int x, int y,
              int cxClient, int cyClient,
              WORD wShow, BOOL fHalftonePalette)
{
  BITMAP bitmap ;
  HDC hdcMem ;
  int cxBitmap, cyBitmap, iReturn ;
  GetObject (hBitmap, sizeof (BITMAP), &bitmap) ;
  cxBitmap = bitmap.bmWidth ;
  cyBitmap = bitmap.bmHeight ;

  SaveDC (hdc) ;
  if (fHalftonePalette)
    SetStretchBltMode (hdc, HALFTONE) ;
  else
    SetStretchBltMode (hdc, COLORONCOLOR) ;
  hdcMem = CreateCompatibleDC (hdc) ;
  SelectObject (hdcMem, hBitmap) ;

  switch (wShow)
  {
  case IDM_SHOW_NORMAL:
    if (fHalftonePalette)
      iReturn = StretchBlt (hdc, 0, 0, min (cxClient, cxBitmap - x),
      min (cyClient, cyBitmap - y),
      hdcMem, x, y, min (cxClient, cxBitmap - x),
      min (cyClient, cyBitmap - y),
      SRCCOPY);
    else
      iReturn = BitBlt (hdc,0, 0, min (cxClient, cxBitmap - x),
      min (cyClient, cyBitmap - y),
      hdcMem, x, y, SRCCOPY) ;
    break ;

  case IDM_SHOW_CENTER:
    if (fHalftonePalette)
      iReturn = StretchBlt (hdc, (cxClient - cxBitmap) / 2,
      (cyClient - cyBitmap) / 2,
      cxBitmap, cyBitmap,
      hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY);
    else
      iReturn = BitBlt (hdc, (cxClient - cxBitmap) / 2,
      cyClient - cyBitmap) / 2,
      cxBitmap, cyBitmap,
      hdcMem, 0, 0, SRCCOPY) ;
    break ;
  case IDM_SHOW_STRETCH:
    iReturn = StretchBlt (hdc, 0, 0, cxClient, cyClient,
      hdcMem, 0, 0, cxBitmap, cyBitmap, SRCCOPY) ;
    break ;

  case IDM_SHOW_ISOSTRETCH:
    SetMapMode (hdc, MM_ISOTROPIC) ;
```

```
    SetWindowExtEx (hdc, cxBitmap, cyBitmap, NULL) ;
    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
    SetWindowOrgEx (hdc, cxBitmap / 2, cyBitmap / 2, NULL) ;
    SetViewportOrgEx (hdc, cxClient / 2, cyClient / 2, NULL) ;

    iReturn = StretchBlt (hdc, 0, 0, cxBitmap, cyBitmap,
      hdcMem,0, 0, cxBitmap, cyBitmap, SRCCOPY) ;
    break ;
  }
  DeleteDC (hdcMem) ;
  RestoreDC (hdc, -1) ;
  return iReturn ;
}

/*------------------------------------------------------------------------
DibFlipHorizontal: Calls non-optimized DibSetPixel and DibGetPixel
------------------------------------------------------------------------*/
HDIB DibFlipHorizontal (HDIB hdibSrc)
{
  HDIB hdibDst ;
  int cx, cy, x, y ;

  if (!DibIsAddressable (hdibSrc))
    return NULL ;
  if (NULL == (hdibDst = DibCopy (hdibSrc, FALSE)))
    return NULL ;
  cx = DibWidth (hdibSrc) ;
  cy = DibHeight (hdibSrc) ;

  for (x = 0 ; x < cx ; x++)
    for (y = 0 ; y < cy ; y++)
    {
      DibSetPixel (hdibDst, x, cy - 1 - y, DibGetPixel (hdibSrc, x, y)) ;
    }

    return hdibDst ;
}

/*------------------------------------------------------------------------
DibRotateRight: Calls optimized DibSetPixelx and DibGetPixelx
------------------------------------------------------------------------*/
HDIB DibRotateRight (HDIB hdibSrc)
{
  HDIB hdibDst ;
  int cx, cy, x, y ;

  if (!DibIsAddressable (hdibSrc))
    return NULL ;

  if (NULL == (hdibDst = DibCopy (hdibSrc, TRUE)))
    return NULL ;

  cx = DibWidth (hdibSrc) ;
  cy = DibHeight (hdibSrc) ;

  switch (DibBitCount (hdibSrc))
  {
  case 1:
    for ( x = 0 ; x < cx ; x++)
      for ( y = 0 ; y < cy ; y++)
        DibSetPixel1 (hdibDst, cy - y - 1, x,
        DibGetPixel1 (hdibSrc, x, y)) ;
      break ;

  case 4:
    for ( x = 0 ; x < cx ; x++)
      for ( y = 0 ; y < cy ; y++)
        DibSetPixel4 (hdibDst, cy - y - 1, x,
        DibGetPixel4 (hdibSrc, x, y)) ;
```

```
    break ;

  case 8:
    for ( x = 0 ; x < cx ; x++)
      for ( y = 0 ; y < cy ; y++)
        DibSetPixel8 (hdibDst, cy - y - 1, x,
        DibGetPixel8 (hdibSrc, x, y)) ;
    break ;
  case 16:
    for (x = 0 ; x < cx ; x++)
      for (y = 0 ; y < cy ; y++)
        DibSetPixel16 (hdibDst, cy - y - 1, x,
        DibGetPixel16 (hdibSrc, x, y)) ;
    break ;

  case 24:
    for (x = 0 ; x < cx ; x++)
      for (y = 0 ; y < cy ; y++)
        DibSetPixel24 (hdibDst, cy - y - 1, x,
        DibGetPixel24 (hdibSrc, x, y)) ;
    break ;

  case 32:
    for (x = 0 ; x < cx ; x++)
      for (y = 0 ; y < cy ; y++)
        DibSetPixel32 (hdibDst, cy - y - 1, x,
        DibGetPixel32 (hdibSrc, x, y)) ;
    break ;
  }
  return hdibDst ;
}

/*--------------------------------------------------------------------------
PaletteMenu: Uncheck and check menu item on palette menu
--------------------------------------------------------------------------*/
void PaletteMenu (HMENU hMenu, WORD wItemNew)
{
  static WORD wItem = IDM_PAL_NONE ;
  CheckMenuItem (hMenu, wItem, MF_UNCHECKED) ;
  wItem = wItemNew ;
  CheckMenuItem (hMenu, wItem, MF_CHECKED) ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static BOOL fHalftonePalette ;
  static DOCINFO di = {sizeof(DOCINFO),TEXT("Dibble:Printing")} ;
  static HBITMAP hBitmap ;
  static HDIB hdib ;
  static HMENU hMenu ;
  static HPALETTE hPalette ;
  static int cxClient, cyClient, iVscroll, iHscroll ;
  static OPENFILENAME ofn ;
  static PRINTDLG printdlg = { sizeof (PRINTDLG) } ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[]= TEXT ("Bitmap Files (*.BMP)\0*.bmp\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  static TCHAR * szCompression[] = {
    TEXT("BI_RGB"),TEXT("BI_RLE8"),TEXT("BI_RLE4"),
      TEXT("BI_BITFIELDS"),TEXT("Unknown")} ;
    static WORD wShow = IDM_SHOW_NORMAL ;
    BOOL fSuccess ;
    BYTE * pGlobal ;
    HDC hdc, hdcPrn ;
    HGLOBAL hGlobal ;
    HDIB hdibNew ;
    int iEnable, cxPage, cyPage, iConvert ;
    PAINTSTRUCT ps ;
    SCROLLINFO si ;
```

```
    TCHAR szBuffer [256] ;

    switch (message)
    {
    case WM_CREATE:

      // Save the menu handle in a static variable

      hMenu = GetMenu (hwnd) ;

      // Initialize the OPENFILENAME structure for the File Open
      // and File Save dialog boxes.

      ofn.lStructSize = sizeof (OPENFILENAME) ;
      ofn.hwndOwner = hwnd ;
      ofn.hInstance = NULL ;
      ofn.lpstrFilter = szFilter ;
      ofn.lpstrCustomFilter = NULL ;
      ofn.nMaxCustFilter = 0 ;
      ofn.nFilterIndex = 0 ;
      ofn.lpstrFile = szFileName ;
      ofn.nMaxFile = MAX_PATH ;
      ofn.lpstrFileTitle = szTitleName ;
      ofn.nMaxFileTitle = MAX_PATH ;
      ofn.lpstrInitialDir = NULL ;
      ofn.lpstrTitle = NULL ;
      ofn.Flags = OFN_OVERWRITEPROMPT ;
      ofn.nFileOffset = 0 ;
      ofn.nFileExtension = 0 ;
      ofn.lpstrDefExt = TEXT ("bmp") ;
      ofn.lCustData = 0 ;
      ofn.lpfnHook = NULL ;
      ofn.lpTemplateName = NULL ;
      return 0 ;

    case WM_DISPLAYCHANGE:
      SendMessage (hwnd, WM_USER_DELETEPAL, 0, 0) ;
      SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;
      return 0 ;

    case WM_SIZE:
      // Save the client area width and height in static variables.

      cxClient = LOWORD (lParam) ;
      cyClient = HIWORD (lParam) ;

      wParam = FALSE ;
      // fall through
      // WM_USER_SETSCROLLS: Programmer-defined Message!
      // Set the scroll bars. If the display mode is not normal,
      // make them invisible. If wParam is TRUE, reset the
      // scroll bar position.

    case WM_USER_SETSCROLLS:
      if (hdib == NULL || wShow != IDM_SHOW_NORMAL)
      {
        si.cbSize = sizeof (SCROLLINFO) ;
        si.fMask = SIF_RANGE ;
        si.nMin = 0 ;
        si.nMax = 0 ;
        SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
        SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
      }
      else
      {
        // First the vertical scroll

        si.cbSize = sizeof (SCROLLINFO) ;
        si.fMask = SIF_ALL ;
```

```
    GetScrollInfo (hwnd, SB_VERT, &si) ;
    si.nMin = 0 ;
    si.nMax = DibHeight (hdib) ;
    si.nPage = cyClient ;
    if ((BOOL) wParam)
      si.nPos = 0 ;

    SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;

    iVscroll = si.nPos ;

    // Then the horizontal scroll

    GetScrollInfo (hwnd, SB_HORZ, &si) ;
    si.nMin = 0 ;
    si.nMax = DibWidth (hdib) ;
    si.nPage = cxClient ;

    if ((BOOL) wParam)
      si.nPos = 0 ;

    SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;

    iHscroll = si.nPos ;
  }
  return 0 ;

  // WM_VSCROLL: Vertically scroll the DIB

case WM_VSCROLL:
  si.cbSize = sizeof (SCROLLINFO) ;
  si.fMask = SIF_ALL ;
  GetScrollInfo (hwnd, SB_VERT, &si) ;

  iVscroll = si.nPos ;

  switch (LOWORD (wParam))
  {
  case SB_LINEUP: si.nPos - = 1 ; break ;
  case SB_LINEDOWN: si.nPos + = 1 ; break ;
  case SB_PAGEUP: si.nPos - = si.nPage ;break ;
  case SB_PAGEDOWN: si.nPos + = si.nPage ;break ;
  case SB_THUMBTRACK:si.nPos = si.nTrackPos ;break ;
  default: break ;
  }
  si.fMask = SIF_POS ;
  SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
  GetScrollInfo (hwnd, SB_VERT, &si) ;
  if (si.nPos != iVscroll)
  {
    ScrollWindow (hwnd, 0, iVscroll - si.nPos, NULL, NULL) ;
    iVscroll = si.nPos ;
    UpdateWindow (hwnd) ;
  }
  return 0 ;

  // WM_HSCROLL: Horizontally scroll the DIB

  case WM_HSCROLL:
    si.cbSize = sizeof (SCROLLINFO) ;
    si.fMask = SIF_ALL ;
    GetScrollInfo (hwnd, SB_HORZ, &si) ;

    iHscroll = si.nPos ;

    switch (LOWORD (wParam))
```

```
      {
      case SB_LINELEFT: si.nPos -=1 ; break ;
      case SB_LINERIGHT: si.nPos +=1 ; break ;
      case SB_PAGELEFT: si.nPos -=si.nPage ;break ;
      case SB_PAGERIGHT: si.nPos +=si.nPage ;break ;
      case SB_THUMBTRACK:si.nPos =si.nTrackPos ;break ;
      default: break ;
      }

      si.fMask = SIF_POS ;
      SetScrollInfo (hwnd, SB_HORZ, &si, TRUE) ;
      GetScrollInfo (hwnd, SB_HORZ, &si) ;

      if (si.nPos != iHscroll)
      {
        ScrollWindow (hwnd, iHscroll - si.nPos, 0, NULL, NULL) ;
        iHscroll = si.nPos ;
        UpdateWindow (hwnd) ;
      }
      return 0 ;

      // WM_INITMENUPOPUP: Enable or Gray menu items

      case WM_INITMENUPOPUP:
        if (hdib)
          iEnable = MF_ENABLED ;
        else
          iEnable = MF_GRAYED ;
        EnableMenuItem (hMenu, IDM_FILE_SAVE, iEnable) ;
        EnableMenuItem (hMenu, IDM_FILE_PRINT, iEnable) ;
        EnableMenuItem (hMenu, IDM_FILE_PROPERTIES, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_CUT, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_COPY, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_DELETE, iEnable) ;

        if (DibIsAddressable (hdib))
          iEnable = MF_ENABLED ;
        else
          iEnable = MF_GRAYED ;

        EnableMenuItem (hMenu, IDM_EDIT_ROTATE, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_FLIP, iEnable) ;
        EnableMenuItem (hMenu, IDM_CONVERT_01, iEnable) ;
        EnableMenuItem (hMenu, IDM_CONVERT_04, iEnable) ;
        EnableMenuItem (hMenu, IDM_CONVERT_08, iEnable) ;
        EnableMenuItem (hMenu, IDM_CONVERT_16, iEnable) ;
        EnableMenuItem (hMenu, IDM_CONVERT_24, iEnable) ;
        EnableMenuItem (hMenu, IDM_CONVERT_32, iEnable) ;

        switch (DibBitCount (hdib))
        {
        case 1: EnableMenuItem (hMenu, IDM_CONVERT_01, MF_GRAYED) ; break ;
        case 4: EnableMenuItem (hMenu, IDM_CONVERT_04, MF_GRAYED) ; break ;
        case 8: EnableMenuItem (hMenu, IDM_CONVERT_08, MF_GRAYED) ; break ;
        case 16: EnableMenuItem (hMenu, IDM_CONVERT_16, MF_GRAYED) ; break ;
        case 24: EnableMenuItem (hMenu, IDM_CONVERT_24, MF_GRAYED) ; break ;
        case 32: EnableMenuItem (hMenu, IDM_CONVERT_32, MF_GRAYED) ; break ;
        }

        if (hdib && DibColorSize (hdib) > 0)
          iEnable = MF_ENABLED ;
        else
          iEnable = MF_GRAYED ;

        EnableMenuItem (hMenu, IDM_PAL_DIBTABLE, iEnable) ;
        if (DibIsAddressable (hdib) && DibBitCount (hdib) > 8)
          iEnable = MF_ENABLED ;
        else
          iEnable = MF_GRAYED ;
```

```
        EnableMenuItem (hMenu, IDM_PAL_OPT_POP4, iEnable) ;
        EnableMenuItem (hMenu, IDM_PAL_OPT_POP5, iEnable) ;
        EnableMenuItem (hMenu, IDM_PAL_OPT_POP6, iEnable) ;
        EnableMenuItem (hMenu, IDM_PAL_OPT_MEDCUT, iEnable) ;
        EnableMenuItem (hMenu, IDM_EDIT_PASTE,
          IsClipboardFormatAvailable (CF_DIB) ? MF_ENABLED : MF_GRAYED) ;

        return 0 ;

        // WM_COMMAND: Process all menu commands.
        case WM_COMMAND:
          iConvert = 0 ;

          switch (LOWORD (wParam))
          {
          case IDM_FILE_OPEN:

            // Show the File Open dialog box
            if (!GetOpenFileName (&ofn))
              return 0 ;

            // If there's an existing DIB and palette, delete them
            SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;

            // Load the DIB into memory
            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            hdib = DibFileLoad (szFileName) ;

            ShowCursor (FALSE) ;
            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

            // Reset the scroll bars
            SendMessage (hwnd, WM_USER_SETSCROLLS, TRUE, 0) ;

            // Create the palette and DDB
            SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;

            if (!hdib)
            {
              MessageBox (hwnd, TEXT ("Cannot load DIB file!"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
            }
            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;

          case IDM_FILE_SAVE:

            // Show the File Save dialog box
            if (! GetSaveFileName (&ofn))
              return 0 ;

            // Save the DIB to memory
            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            fSuccess = DibFileSave (hdib, szFileName) ;

            ShowCursor (FALSE) ;
            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

            if (!fSuccess)
              MessageBox ( hwnd, TEXT ("Cannot save DIB file!"),
              szAppName, MB_OK | MB_ICONEXCLAMATION) ;
            return 0 ;

          case IDM_FILE_PRINT:
```

```
            if (!hdib)
              return 0 ;

            // Get printer DC
            printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

            if (!PrintDlg (&printdlg))
              return 0 ;

            if (NULL == (hdcPrn = printdlg.hDC))
            {
              MessageBox( hwnd, TEXT ("Cannot obtain Printer DC"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
              return 0 ;
            }
            // Check if the printer can print bitmaps
            if (!(RC_BITBLT & GetDeviceCaps (hdcPrn, RASTERCAPS)))
            {
              DeleteDC (hdcPrn) ;
              MessageBox ( hwnd, TEXT ("Printer cannot print bitmaps"),
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
              return 0 ;
            }
            // Get size of printable area of page
            cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
            cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

            fSuccess = FALSE ;

            // Send the DIB to the printer
            SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
            ShowCursor (TRUE) ;

            if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
            {
              DisplayDib (hdcPrn, DibBitmapHandle (hdib), 0, 0,
                cxPage, cyPage, wShow, FALSE) ;

              if (EndPage (hdcPrn) > 0)
              {
                fSuccess = TRUE ;
                EndDoc (hdcPrn) ;
              }
            }
            ShowCursor (FALSE) ;
            SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

            DeleteDC (hdcPrn) ;

            if ( !fSuccess)
              MessageBox ( hwnd, TEXT ("Could not print bitmap"),
              szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return 0 ;

          case IDM_FILE_PROPERTIES:
            if (!hdib)
              return 0 ;

            wsprintf (szBuffer, TEXT ("Pixel width:\t%i\n")
              TEXT ("Pixel height:\t%i\n")
              TEXT ("Bits per pixel:\t%i\n")
              TEXT ("Number of colors:\t%i\n")
              TEXT ("Compression:\t%s\n"),
              DibWidth (hdib), DibHeight (hdib),
              DibBitCount (hdib), DibNumColors (hdib),
              szCompression [min (3, DibCompression (hdib))]) ;

            MessageBox ( hwnd, szBuffer, szAppName,
              MB_ICONEXCLAMATION | MB_OK) ;
```

```
            return 0 ;

          case IDM_APP_EXIT:
            SendMessage (hwnd, WM_CLOSE, 0, 0) ;
            return 0 ;

          case IDM_EDIT_COPY:
          case IDM_EDIT_CUT:
            if (!(hGlobal = DibCopyToPackedDib (hdib, TRUE)))
              return 0 ;

            OpenClipboard (hwnd) ;
            EmptyClipboard () ;
            SetClipboardData (CF_DIB, hGlobal) ;
            CloseClipboard () ;

            if (LOWORD (wParam) == IDM_EDIT_COPY)
              return 0 ;
            // fall through for IDM_EDIT_CUT
          case IDM_EDIT_DELETE:
            SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;
            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;

          case IDM_EDIT_PASTE:
            OpenClipboard (hwnd) ;

            hGlobal = GetClipboardData (CF_DIB) ;
            pGlobal = GlobalLock (hGlobal) ;

            // If there's an existing DIB and palette,delete them.
            // Then convert the packed DIB to an HDIB.
            if (pGlobal)
            {
              SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;
              hdib = DibCopyFromPackedDib ((BITMAPINFO *) pGlobal) ;
              SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;
            }

            GlobalUnlock (hGlobal) ;
            CloseClipboard () ;
            // Reset the scroll bars
            SendMessage (hwnd, WM_USER_SETSCROLLS, TRUE, 0) ;
            InvalidateRect (hwnd, NULL, TRUE) ;
            return 0 ;

          case IDM_EDIT_ROTATE:
            if (hdibNew = DibRotateRight (hdib))
            {
              DibDelete (hdib) ;
              DeleteObject (hBitmap) ;
              hdib = hdibNew ;
              hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;
              SendMessage (hwnd, WM_USER_SETSCROLLS, TRUE, 0) ;
              InvalidateRect (hwnd, NULL, TRUE) ;
            }
            else
            {
              MessageBox ( hwnd, TEXT ("Not enough memory"),
                szAppName, MB_OK | MB_ICONEXCLAMATION) ;
            }
            return 0 ;

          case IDM_EDIT_FLIP:
            if (hdibNew = DibFlipHorizontal (hdib))
            {
              DibDelete (hdib) ;
              DeleteObject (hBitmap) ;
              hdib = hdibNew ;
```

```
            hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;
            InvalidateRect (hwnd, NULL, TRUE) ;
          }
          else
          {
            MessageBox ( hwnd, TEXT ("Not enough memory"),
              szAppName, MB_OK | MB_ICONEXCLAMATION) ;
          }
          return 0 ;

        case IDM_SHOW_NORMAL:
        case IDM_SHOW_CENTER:
        case IDM_SHOW_STRETCH:
        case IDM_SHOW_ISOSTRETCH:
          CheckMenuItem (hMenu, wShow, MF_UNCHECKED) ;
          wShow = LOWORD (wParam) ;
          CheckMenuItem (hMenu, wShow, MF_CHECKED) ;
          SendMessage (hwnd, WM_USER_SETSCROLLS, FALSE, 0) ;

          InvalidateRect (hwnd, NULL, TRUE) ;
          return 0 ;

        case IDM_CONVERT_32: iConvert += 8 ;
        case IDM_CONVERT_24: iConvert += 8 ;
        case IDM_CONVERT_16: iConvert += 8 ;
        case IDM_CONVERT_08: iConvert += 4 ;
        case IDM_CONVERT_04: iConvert += 3 ;
        case IDM_CONVERT_01: iConvert += 1 ;
          SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
          ShowCursor (TRUE) ;

          hdibNew = DibConvert (hdib, iConvert) ;

          ShowCursor (FALSE) ;
          SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

          if (hdibNew)
          {
            SendMessage (hwnd, WM_USER_DELETEDIB, 0, 0) ;
            hdib = hdibNew ;
            SendMessage (hwnd, WM_USER_CREATEPAL, TRUE, 0) ;
            InvalidateRect (hwnd, NULL, TRUE) ;
          }
          else
          {
            MessageBox ( hwnd, TEXT ("Not enough memory"),
              szAppName, MB_OK | MB_ICONEXCLAMATION) ;
          }
          return 0 ;

        case IDM_APP_ABOUT:
          MessageBox ( hwnd, TEXT ("Dibble (c) Charles Petzold, 1998"),
            szAppName, MB_OK | MB_ICONEXCLAMATION) ;
          return 0 ;
}

// All the other WM_COMMAND messages are from the palette
// items. Any existing palette is deleted, and the cursor
// is set to the hourglass.
SendMessage (hwnd, WM_USER_DELETEPAL, 0, 0) ;
SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
ShowCursor (TRUE) ;
// Notice that all messages for palette items are ended
// with break rather than return. This is to allow
// additional processing later on.
switch (LOWORD (wParam))
{
case IDM_PAL_DIBTABLE:
  hPalette = DibPalDibTable (hdib) ;
```

```
  break ;

case IDM_PAL_HALFTONE:
  hdc = GetDC (hwnd) ;

  if (hPalette = CreateHalftonePalette (hdc))
    fHalftonePalette = TRUE ;

  ReleaseDC (hwnd, hdc) ;
  break ;

case IDM_PAL_ALLPURPOSE:
  hPalette = DibPalAllPurpose () ;
  break ;

case IDM_PAL_GRAY2:hPalette = DibPalUniformGrays (2); break;
case IDM_PAL_GRAY3:hPalette = DibPalUniformGrays (3); break;
case IDM_PAL_GRAY4:hPalette = DibPalUniformGrays (4); break;
case IDM_PAL_GRAY8:hPalette = DibPalUniformGrays (8); break;
case IDM_PAL_GRAY16:hPalette = DibPalUniformGrays (16) ; break;
case IDM_PAL_GRAY32:hPalette = DibPalUniformGrays (32) ; break;
case IDM_PAL_GRAY64:hPalette = DibPalUniformGrays (64) ; break;
case IDM_PAL_GRAY128:hPalette = DibPalUniformGrays (128) ; break;
case IDM_PAL_GRAY256:hPalette = DibPalUniformGrays (256) ; break;
case IDM_PAL_RGB222:hPalette = DibPalUniformColors (2,2,2); break;
case IDM_PAL_RGB333:hPalette = DibPalUniformColors (3,3,3); break;
case IDM_PAL_RGB444:hPalette = DibPalUniformColors (4,4,4); break;
case IDM_PAL_RGB555:hPalette = DibPalUniformColors (5,5,5); break;
case IDM_PAL_RGB666:hPalette = DibPalUniformColors (6,6,6); break;
case IDM_PAL_RGB775:hPalette = DibPalUniformColors (7,7,5); break;
case IDM_PAL_RGB757:hPalette = DibPalUniformColors (7,5,7); break;
case IDM_PAL_RGB577:hPalette = DibPalUniformColors (5,7,7); break;
case IDM_PAL_RGB884:hPalette = DibPalUniformColors (8,8,4); break;
case IDM_PAL_RGB848:hPalette = DibPalUniformColors (8,4,8); break;
case IDM_PAL_RGB488:hPalette = DibPalUniformColors (4,8,8); break;
case IDM_PAL_OPT_POP4:hPalette = DibPalPopularity (hdib, 4) ; break ;
case IDM_PAL_OPT_POP5:hPalette = DibPalPopularity (hdib, 5) ; break ;
case IDM_PAL_OPT_POP6:hPalette = DibPalPopularity (hdib, 6) ; break ;
case IDM_PAL_OPT_MEDCUT:hPalette = DibPalMedianCut (hdib, 6) ; break ;
}

// After processing Palette items from the menu, the cursor
// is restored to an arrow, the menu item is checked, and
// the window is invalidated.
hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;

ShowCursor (FALSE) ;
SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

if (hPalette)
  PaletteMenu (hMenu, (LOWORD (wParam))) ;

InvalidateRect (hwnd, NULL, TRUE) ;
return 0 ;

// This programmer-defined message deletes an existing DIB
// in preparation for getting a new one. Invoked during
// File Open command, Edit Paste command, and others.
case WM_USER_DELETEDIB:
  if (hdib)
  {
    DibDelete (hdib) ;
    hdib = NULL ;
  }
  SendMessage (hwnd, WM_USER_DELETEPAL, 0, 0) ;
  return 0 ;

  // This programmer-defined message deletes an existing palette
  // in preparation for defining a new one.
```

```
case WM_USER_DELETEPAL:
  if (hPalette)
  {
    DeleteObject (hPalette) ;
    hPalette = NULL ;
    fHalftonePalette = FALSE ;
    PaletteMenu (hMenu, IDM_PAL_NONE) ;
  }
  if (hBitmap)
    DeleteObject (hBitmap) ;

  return 0 ;

  // Programmer-defined message to create a new palette based on
  // a new DIB. If wParam == TRUE, create a DDB as well.
case WM_USER_CREATEPAL:
  if (hdib)
  {
    hdc = GetDC (hwnd) ;

    if (!(RC_PALETTE & GetDeviceCaps (hdc, RASTERCAPS)))
    {
      PaletteMenu (hMenu, IDM_PAL_NONE) ;
    }
    else if (hPalette = DibPalDibTable (hdib))
    {
      PaletteMenu (hMenu, IDM_PAL_DIBTABLE) ;
    }
    else if (hPalette = CreateHalftonePalette (hdc))
    {
      fHalftonePalette = TRUE ;
      PaletteMenu (hMenu, IDM_PAL_HALFTONE) ;
    }
    ReleaseDC (hwnd, hdc) ;

    if ((BOOL) wParam)
      hBitmap = DibCopyToDdb (hdib, hwnd, hPalette) ;
  }
  return 0 ;

case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;

  if (hPalette)
  {
    SelectPalette (hdc, hPalette, FALSE) ;
    RealizePalette (hdc) ;
  }
  if (hBitmap)
  {
    DisplayDib ( hdc,
      fHalftonePalette ? DibBitmapHandle (hdib) : hBitmap,
      iHscroll, iVscroll,
      cxClient, cyClient,
      wShow, fHalftonePalette) ;
  }
  EndPaint (hwnd, &ps) ;
  return 0 ;

case WM_QUERYNEWPALETTE:
  if (!hPalette)
    return FALSE ;

  hdc = GetDC (hwnd) ;
  SelectPalette (hdc, hPalette, FALSE) ;
  RealizePalette (hdc) ;
  InvalidateRect (hwnd, NULL, TRUE) ;

  ReleaseDC (hwnd, hdc) ;
```

```
  return TRUE ;

 case WM_PALETTECHANGED:
  if (!hPalette || (HWND) wParam == hwnd)
    break ;

  hdc = GetDC (hwnd) ;
  SelectPalette (hdc, hPalette, FALSE) ;
  RealizePalette (hdc) ;
  UpdateColors (hdc) ;

  ReleaseDC (hwnd, hdc) ;
  break ;

 case WM_DESTROY:
  if (hdib)
    DibDelete (hdib) ;

  if (hBitmap)
    DeleteObject (hBitmap) ;

  if (hPalette)
    DeleteObject (hPalette) ;

  PostQuitMessage (0) ;
  return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

DIBBLE.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
DIBBLE MENU DISCARDABLE BEGIN POPUP "&File"
BEGIN
MENUITEM "&Open...\tCtrl+O", IDM_FILE_OPEN
MENUITEM "&Save...\tCtrl+S", IDM_FILE_SAVE
MENUITEM SEPARATOR
MENUITEM "&Print...\tCtrl+P", IDM_FILE_PRINT
MENUITEM SEPARATOR
MENUITEM "Propert&ies...", IDM_FILE_PROPERTIES
MENUITEM SEPARATOR
MENUITEM "E&xit",  IDM_APP_EXIT
END
POPUP "&Edit"
BEGIN
MENUITEM "Cu&t\tCtrl+X", IDM_EDIT_CUT
MENUITEM "&Copy\tCtrl+C", IDM_EDIT_COPY
MENUITEM "&Paste\tCtrl+V", IDM_EDIT_PASTE
MENUITEM "&Delete\tDelete", IDM_EDIT_DELETE
MENUITEM SEPARATOR
MENUITEM "&Flip",  IDM_EDIT_FLIP
MENUITEM "&Rotate", IDM_EDIT_ROTATE
END
POPUP "&Show"
BEGIN
MENUITEM "&Actual Size", IDM_SHOW_NORMAL, CHECKED
MENUITEM "&Center", IDM_SHOW_CENTER
MENUITEM "&Stretch to Window", IDM_SHOW_STRETCH
MENUITEM "Stretch &Isotropically",IDM_SHOW_ISOSTRETCH
END
POPUP "&Palette"
BEGIN
MENUITEM "&None",  IDM_PAL_NONE, CHECKED
```

```
MENUITEM "&Dib ColorTable", IDM_PAL_DIBTABLE
MENUITEM "&Halftone", IDM_PAL_HALFTONE
MENUITEM "&All-Purpose", IDM_PAL_ALLPURPOSE
POPUP "&Gray Shades"
BEGIN
MENUITEM "&1. 2 Grays", IDM_PAL_GRAY2
MENUITEM "&2. 3 Grays", IDM_PAL_GRAY3
MENUITEM "&3. 4 Grays", IDM_PAL_GRAY4
MENUITEM "&4. 8 Grays", IDM_PAL_GRAY8
MENUITEM "&5. 16 Grays", IDM_PAL_GRAY16
MENUITEM "&6. 32 Grays", IDM_PAL_GRAY32
MENUITEM "&7. 64 Grays", IDM_PAL_GRAY64
MENUITEM "&8. 128 Grays", IDM_PAL_GRAY128
MENUITEM "&9. 256 Grays", IDM_PAL_GRAY256
END
POPUP "&Uniform Colors"
BEGIN
MENUITEM "&1. 2R x 2G x 2B (8)",  IDM_PAL_RGB222
MENUITEM "&2. 3R x 3G x 3B (27)", IDM_PAL_RGB333
MENUITEM "&3. 4R x 4G x 4B (64)", IDM_PAL_RGB444
MENUITEM "&4. 5R x 5G x 5B (125)", IDM_PAL_RGB555
MENUITEM "&5. 6R x 6G x 6B (216)", IDM_PAL_RGB666
MENUITEM "&6. 7R x 7G x 5B (245)", IDM_PAL_RGB775
MENUITEM "&7. 7R x 5B x 7B (245)", IDM_PAL_RGB757
MENUITEM "&8. 5R x 7G x 7B (245)", IDM_PAL_RGB577
MENUITEM "&9. 8R x 8G x 4B (256)", IDM_PAL_RGB884
MENUITEM "&A. 8R x 4G x 8B (256)", IDM_PAL_RGB848
MENUITEM "&B. 4R x 8G x 8B (256)", IDM_PAL_RGB488
END
POPUP "&Optimized"
BEGIN
MENUITEM "&1. Popularity Algorithm (4 bits)"IDM_PAL_OPT_POP4
MENUITEM "&2. Popularity Algorithm (5 bits)"IDM_PAL_OPT_POP5
MENUITEM "&3. Popularity Algorithm (6 bits)"IDM_PAL_OPT_POP6
MENUITEM "&4. Median Cut Algorithm ", IDM_PAL_OPT_MEDCUT
END
END
POPUP "Con&vert"
BEGIN
MENUITEM "&1. to 1 bit per pixel", IDM_CONVERT_01
MENUITEM "&2. to 4 bits per pixel", IDM_CONVERT_04
MENUITEM "&3. to 8 bits per pixel", IDM_CONVERT_08
MENUITEM "&4. to 16 bits per pixel", IDM_CONVERT_16
MENUITEM "&5. to 24 bits per pixel", IDM_CONVERT_24
MENUITEM "&6. to 32 bits per pixel", IDM_CONVERT_32
END
POPUP "&Help"
BEGIN
MENUITEM "&About",
IDM_APP_ABOUT
END
END
////////////////////////////////////////////////////////////////////////////
// Accelerator
DIBBLE ACCELERATORS DISCARDABLE
BEGIN
"C", IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
"O", IDM_FILE_OPEN, VIRTKEY, CONTROL, NOINVERT
"P", IDM_FILE_PRINT, VIRTKEY, CONTROL, NOINVERT
"S", IDM_FILE_SAVE, VIRTKEY, CONTROL, NOINVERT
"V", IDM_EDIT_PASTE, VIRTKEY, CONTROL, NOINVERT
VK_DELETE, IDM_EDIT_DELETE, VIRTKEY, NOINVERT
"X", IDM_EDIT_CUT, VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
```

```
// Used by Dibble.rc
#define IDM_FILE_OPEN  40001
#define IDM_FILE_SAVE  40002
#define IDM_FILE_PRINT  40003
#define IDM_FILE_PROPERTIES  40004
#define IDM_APP_EXIT  40005
#define IDM_EDIT_CUT  40006
#define IDM_EDIT_COPY  40007
#define IDM_EDIT_PASTE  40008
#define IDM_EDIT_DELETE  40009
#define IDM_EDIT_FLIP  40010
#define IDM_EDIT_ROTATE  40011
#define IDM_SHOW_NORMAL  40012
#define IDM_SHOW_CENTER  40013
#define IDM_SHOW_STRETCH  40014
#define IDM_SHOW_ISOSTRETCH  40015
#define IDM_PAL_NONE  40016
#define IDM_PAL_DIBTABLE  40017
#define IDM_PAL_HALFTONE  40018
#define IDM_PAL_ALLPURPOSE  40019
#define IDM_PAL_GRAY2  40020
#define IDM_PAL_GRAY3  40021
#define IDM_PAL_GRAY4  40022
#define IDM_PAL_GRAY8  40023
#define IDM_PAL_GRAY16  40024
#define IDM_PAL_GRAY32  40025
#define IDM_PAL_GRAY64  40026
#define IDM_PAL_GRAY128  40027
#define IDM_PAL_GRAY256  40028
#define IDM_PAL_RGB222  40029
#define IDM_PAL_RGB333  40030
#define IDM_PAL_RGB444  40031
#define IDM_PAL_RGB555  40032
#define IDM_PAL_RGB666  40033
#define IDM_PAL_RGB775  40034
#define IDM_PAL_RGB757  40035
#define IDM_PAL_RGB577  40036
#define IDM_PAL_RGB884  40037
#define IDM_PAL_RGB848  40038
#define IDM_PAL_RGB488  40039
#define IDM_PAL_OPT_POP4  40040
#define IDM_PAL_OPT_POP5  40041
#define IDM_PAL_OPT_POP6  40042
#define IDM_PAL_OPT_MEDCUT  40043
#define IDM_CONVERT_01  40044
#define IDM_CONVERT_04  40045
#define IDM_CONVERT_08  40046
#define IDM_CONVERT_16  40047
#define IDM_CONVERT_24  40048
#define IDM_CONVERT_32  40049
#define IDM_APP_ABOUT  40050
```

DIBBLE使用了两个其它文件，我将简要地说明它们。DIBCONV文件（DIBCONV.C和DIBCONV.H）在两种不同格式之间转换－例如，从每像素24位转换成每像素8位。DIBPAL文件（DIBPAL.C和DIBPAL.H）建立调色盘。

DIBBLE维护WndProc中的三个重要的静态变量。这些是呼叫hdib的HDIB句柄、呼叫hPalette的HPALETTE句柄和呼叫hBitmap的HBITMAP句柄。HDIB来自DIBHELP中的不同函数；HPALETTE来自DIBPAL中的不同函数或CreateHalftonePalette函数；而HBITMAP句柄来自DIBHELP.C中的DibCopyToDdb函数并帮助加速屏幕显示，特别是在256色显示模式下。不过，无论在程序建立新的「DIB Section」（显而易见地）或在程序建立不同的调色盘（不很明显）时，这个句柄都必须重新建立。

让我们从功能上而非循序渐进地来介绍一下DIBBLE。

文件载入和储存

DIBBLE可以在响应IDM_FILE_LOAD和IDM_FILE_SAVE的WM_COMMAND消息处理过程中加载DIB文件并储存这些文件。在处理这些消息处理期间，DIBBLE通过分别呼叫GetOpenFileName和GetSaveFileName来启动公用文件对话框。

对于「File」、「Save」菜单命令，DIBBLE只需要呼叫DibFileSave。对于「File」、「Open」菜单命令，DIBBLE必须首先删除前面的HDIB、调色盘和位图对象。它透过发送一个WM_USER_DELETEDIB消息来完成这件事，此消息通过呼叫DibDelete和DeleteObject来处理。然后DIBBLE呼叫DIBHELP中的DibFileLoad函数，发送WM_USER_SETSCROLLS和WM_USER_CREATEPAL消息来重新设定滚动条并建立调色盘。WM_USER_CREATEPAL消息也位于程序从DIB区块建立的新的DDB位置。

显示、卷动和打印

DIBBLE的菜单允许它以实际尺寸在显示区域左上角显示DIB，或在显示区域中间显示DIB，或伸展到填充显示区域，或者在保持纵横比的情况下尽量填充显示区域。您可以在DIBBLE的「Show」菜单上来选择需要的选项。注意，这些与上一章的SHOWDIB2程序中四个选项相同。

在WM_PAINT消息处理期间 – 也是处理「File」、「Print」命令的过程中 – DIBBLE呼叫DisplayDib函数。注意，DisplayDib使用BitBlt和StretchBlt，而不是使用SetDIBitsToDevice和StretchDIBits。在WM_PAINT消息处理期间，传递给函数的位图句柄由DibCopyToDdb函数建立，并在WM_USER_CREATEPAL消息处理期间呼叫。其中DDB与视讯设备内容兼容。当处理「File」、「Print」命令时，DIBBLE呼叫DisplayDib，其中可用的DIB区块句柄来自DIBHELP.C中的DibBitmapHandle函数。

另外要注意，DIBBLE保留一个称作fHalftonePalette的静态BOOL变量，如果从CreateHalftonePalette函数中获得hPalette，则此变量设定为TRUE。这将迫使DisplayDib函数呼叫StretchBlt而不是呼叫BitBlt，即使DIB被指定按实际尺寸显示。fHalftonePalette变量也导致WM_PAINT处理程序将DIB区块句柄传递给DisplayDib函数，而不是由DibCopyToDdb函数建立的位图句柄。本章前面讨论过中间色调色盘的使用，并在SHOWDIB5程序中进行了展示。

第一次使用范例程序时，DIBBLE允许在显示区域中卷动DIB。只有按实际尺寸显示DIB时，才显示滚动条。在处理WM_PAINT时，WndProc简单地将滚动条的目前位置传递给DisplayDib函数。

剪贴簿

对于「Cut」和「Copy」菜单项，DIBBLE呼叫DIBHELP中的DibCopyToPackedDib函数。该函数将获得所有的DIB组件并将它们放入大的内存块中。

对于第一次使用本书中的某些范例程序来说，DIBBLE从剪贴簿中粘贴DIB。这包括呼叫DibCopyFromPackedDib函数，并替换窗口消息处理程序前面储存的HDIB、调色盘和位图。

翻转和旋转

DIBBLE中的「Edit」菜单中除了常见的「Cut」、「Copy」、「Paste」和「Delete」选项之外，还包括两个附加项 – 「Flip」和「Rotate」。「Flip」选项使位图绕水平轴翻转 – 即上下颠倒翻转。「Rotate」选项使位图顺时针旋转90度。这两个函数都需要透过将它们从一个DIB复制到另一个来存取所有的DIB像素（因为这两个函数不需要建立新的调色盘，所以不删除和重新建立调色盘）。

「Flip」菜单选项使用DibFlipHorizontal函数，此函数也位于DIBBLE.C文件。此函数呼叫DibCopy来获得DIB精确的副本。然后，进入将原DIB中的像素复制到新DIB的循环，但是复制这些像素是为了上下翻转图像。注意，此函数呼叫DibGetPixel和DibSetPixel。这些是DIBHELP.C中的

通用（但不像我们所希望的那么快）函数。

为了说明DibGetPixel和DibSetPixel函数与DIBHELP.H中执行更快的DibGetPixel和DibSetPixel宏之间的区别，DibRotateRight函数使用了宏。然而，首先要注意的是，该函数呼叫DibCopy时，第二个参数设定为TRUE。这导致DibCopy翻转原DIB的宽度和高度来建立新的DIB。另外，像素位不能由DibCopy函数复制。但是，DibRotateRight函数有六个不同的循环将像素位从原DIB复制到新的DIB – 每一个都对应不同的DIB像素宽度（1位、4位、8位、16位、24位和32位）。虽然包括了更多的程序代码，但是函数更快了。

尽管可以使用「Flip Horizontal」和「Rotate Right」选项来产生「Flip Vertical」、「Rotate Left」和「Rotate 180°」功能，但通常程序将直接执行所有选项。毕竟，DIBBLE只是个展示程序而已。

简单调色盘；最佳化调色盘

在DIBBLE中，您可以在256色视讯显示器上选择不同的调色盘来显示DIB。这些都在DIBBLE的「Palette」菜单中列出。除了中间色调色盘以外，其余的都直接由Windows函数呼叫建立，建立不同调色盘的所有函数都由程序16-24所示的DIBPAL文件提供。

程序16-24 DIBPAL文件

DIBPAL.H

```
/*----------------------------------------------------------------------------
  DIBPAL.H header file for DIBPAL.C
----------------------------------------------------------------------------*/
HPALETTE DibPalDibTable (HDIB hdib) ;
HPALETTE DibPalAllPurpose (void) ;
HPALETTE DibPalUniformGrays (int iNum) ;
HPALETTE DibPalUniformColors (int iNumR, int iNumG, int iNumB) ;
HPALETTE DibPalVga (void) ;
HPALETTE DibPalPopularity (HDIB hdib, int iRes) ;
HPALETTE DibPalMedianCut (HDIB hdib, int iRes) ;

   DIBPAL.C
/*----------------------------------------------------------------------------
DIBPAL.C -- Palette-Creation Functions
(c) Charles Petzold, 1998
----------------------------------------------------------------------------*/
#include <windows.h>
#include "dibhelp.h"
#include "dibpal.h"

/*----------------------------------------------------------------------------
DibPalDibTable: Creates a palette from the DIB color table
----------------------------------------------------------------------------*/

HPALETTE DibPalDibTable (HDIB hdib)
{
  HPALETTE hPalette ;
  int i, iNum ;
  LOGPALETTE * plp ;
  RGBQUAD rgb ;

  if (0 == (iNum = DibNumColors (hdib)))
    return NULL ;
  plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = iNum ;

  for (i = 0 ; i < iNum ; i++)
  {
    DibGetColor (hdib, i, &rgb) ;
    plp->palPalEntry[i].peRed = rgb.rgbRed ;
    plp->palPalEntry[i].peGreen = rgb.rgbGreen ;
```

```
    plp->palPalEntry[i].peBlue = rgb.rgbBlue ;
    plp->palPalEntry[i].peFlags = 0 ;
  }
  hPalette = CreatePalette (plp) ;
  free (plp) ;
  return hPalette ;
}
/*--------------------------------------------------------------------------
DibPalAllPurpose: Creates a palette suitable for a wide variety
of images; the palette has 247 entries, but 15 of them are
duplicates or match the standard 20 colors.
--------------------------------------------------------------------------*/

HPALETTE DibPalAllPurpose (void)
{
  HPALETTE hPalette ;
  int i, incr, R, G, B ;
  LOGPALETTE * plp ;

  plp = malloc (sizeof (LOGPALETTE) + 246 * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = 247 ;

  // The following loop calculates 31 gray shades, but 3 of them
  // will match the standard 20 colors

  for (i = 0, G = 0, incr = 8 ; G <= 0xFF ; i++, G += incr)
  {
    plp->palPalEntry[i].peRed = (BYTE) G ;
    plp->palPalEntry[i].peGreen = (BYTE) G ;
    plp->palPalEntry[i].peBlue = (BYTE) G ;
    plp->palPalEntry[i].peFlags = 0 ;

    incr = (incr == 9 ? 8 : 9) ;
  }

  // The following loop is responsible for 216 entries, but 8 of
  // them will match the standard 20 colors, and another
  // 4 of them will match the gray shades above.

  for (R = 0 ; R <= 0xFF ; R += 0x33)
    for (G = 0 ; G <= 0xFF ; G += 0x33)
      for (B = 0 ; B <= 0xFF ; B += 0x33)
      {
        plp->palPalEntry[i].peRed = (BYTE) R ;
        plp->palPalEntry[i].peGreen = (BYTE) G ;
        plp->palPalEntry[i].peBlue = (BYTE) B ;
        plp->palPalEntry[i].peFlags = 0 ;

        i++ ;
      }
     hPalette = CreatePalette (plp) ;
     free (plp) ;
     return hPalette ;
}

/*--------------------------------------------------------------------------
DibPalUniformGrays: Creates a palette of iNum grays, uniformly spaced
--------------------------------------------------------------------------*/

HPALETTE DibPalUniformGrays (int iNum)
{
  HPALETTE hPalette ;
  int i ;
  LOGPALETTE * plp ;

  plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = iNum ;
```

```
  for (i = 0 ; i < iNum ; i++)
  {
    plp->palPalEntry[i].peRed =
      plp->palPalEntry[i].peGreen =
      plp->palPalEntry[i].peBlue = (BYTE) (i * 255 / (iNum - 1)) ;
    plp->palPalEntry[i].peFlags = 0 ;
  }
  hPalette = CreatePalette (plp) ;
  free (plp) ;
  return hPalette ;
}

/*-------------------------------------------------------------------------
DibPalUniformColors: Creates a palette of iNumR x iNumG x iNumB colors
-------------------------------------------------------------------------*/

HPALETTE DibPalUniformColors (int iNumR, int iNumG, int iNumB)
{
  HPALETTE hPalette ;
  int i, iNum, R, G, B ;
  LOGPALETTE * plp ;

  iNum = iNumR * iNumG * iNumB ;
  plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = iNumR * iNumG * iNumB ;

  i = 0 ;
  for (R = 0 ; R < iNumR ; R++)
    for (G = 0 ; G < iNumG ; G++)
      for (B = 0 ; B < iNumB ; B++)
      {
        plp->palPalEntry[i].peRed = (BYTE) (R * 255 / (iNumR - 1)) ;
        plp->palPalEntry[i].peGreen = (BYTE) (G * 255 / (iNumG - 1)) ;
        plp->palPalEntry[i].peBlue = (BYTE) (B * 255 / (iNumB - 1)) ;
        plp->palPalEntry[i].peFlags = 0 ;

        i++ ;
      }
      hPalette = CreatePalette (plp) ;
      free (plp) ;
      return hPalette ;
}

/*-------------------------------------------------------------------------
DibPalVga: Creates a palette based on standard 16 VGA colors
-------------------------------------------------------------------------*/

HPALETTE DibPalVga (void)
{
  static RGBQUAD rgb [16] = { 0x00, 0x00, 0x00,0x00,
    0x00, 0x00, 0x80, 0x00,
    0x00, 0x80, 0x00, 0x00,
    0x00, 0x80, 0x80, 0x00,
    0x80, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x80, 0x00,
    0x80, 0x80, 0x00, 0x00,
    0x80, 0x80, 0x80, 0x00,
    0xC0, 0xC0, 0xC0, 0x00,
    0x00, 0x00, 0xFF, 0x00,
    0x00, 0xFF, 0x00, 0x00,
    0x00, 0xFF, 0xFF, 0x00,
    0xFF, 0x00, 0x00, 0x00,
    0xFF, 0x00, 0xFF, 0x00,
    0xFF, 0xFF, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0x00 } ;
  HPALETTE hPalette ;
  int i ;
```

```
  LOGPALETTE * plp ;

  plp = malloc (sizeof (LOGPALETTE) + 15 * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = 16 ;

  for (i = 0 ; i < 16 ; i++)
  {
    plp->palPalEntry[i].peRed = rgb[i].rgbRed ;
    plp->palPalEntry[i].peGreen = rgb[i].rgbGreen ;
    plp->palPalEntry[i].peBlue = rgb[i].rgbBlue ;
    plp->palPalEntry[i].peFlags = 0 ;
  }
  hPalette = CreatePalette (plp) ;
  free (plp) ;
  return hPalette ;
}

/*-------------------------------------------------------------------------
Macro used in palette optimization routines
-------------------------------------------------------------------------*/

#define PACK_RGB(R,G,B,iRes) ((int) (R) | ((int) (G) << (iRes)) | \
((int) (B) << ((iRes) + (iRes))))

/*-------------------------------------------------------------------------
AccumColorCounts: Fills up piCount (indexed by a packed RGB color)
with counts of pixels of that color.
-------------------------------------------------------------------------*/

static void AccumColorCounts (HDIB hdib, int * piCount, int iRes)
{
  int x, y, cx, cy ;
  RGBQUADrgb ;

  cx = DibWidth (hdib) ;
  cy = DibHeight (hdib) ;

  for (y = 0 ; y < cy ; y++)
    for (x = 0 ; x < cx ; x++)
    {
      DibGetPixelColor (hdib, x, y, &rgb) ;

      rgb.rgbRed >>= (8 - iRes) ;
      rgb.rgbGreen >>= (8 - iRes) ;
      rgb.rgbBlue >>= (8 - iRes) ;

      ++piCount [PACK_RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue, iRes)] ;
    }
}

/*-------------------------------------------------------------------------
DibPalPopularity: Popularity algorithm for optimized colors
-------------------------------------------------------------------------*/

HPALETTE DibPalPopularity (HDIB hdib, int iRes)
{
  HPALETTE hPalette ;
  int i, iArraySize, iEntry, iCount, iIndex, iMask, R, G, B ;
  int * piCount ;
  LOGPALETTE * plp ;

  // Validity checks

  if (DibBitCount (hdib) < 16)
    return NULL ;
  if (iRes < 3 || iRes > 8)
    return NULL ;
  // Allocate array for counting pixel colors
```

```
  iArraySize = 1 << (3 * iRes) ;
  iMask = (1 << iRes) - 1 ;

  if (NULL == (piCount = calloc (iArraySize, sizeof (int))))
    return NULL ;
  // Get the color counts
  AccumColorCounts (hdib, piCount, iRes) ;
  // Set up a palette
  plp = malloc (sizeof (LOGPALETTE) + 235 * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  for (iEntry = 0 ; iEntry < 236 ; iEntry++)
  {
    for (i = 0, iCount = 0 ; i < iArraySize ; i++)
      if (piCount[i] > iCount)

      {
        iCount = piCount[i] ;
        iIndex = i ;
      }
      if (iCount == 0)
        break ;
      R = (iMask & iIndex) << (8 - iRes) ;
      G = (iMask & (iIndex >> iRes )) << (8 - iRes) ;
      B = (iMask & (iIndex >> (iRes + iRes)))<< (8 - iRes) ;

      plp->palPalEntry[iEntry].peRed = (BYTE) R ;
      plp->palPalEntry[iEntry].peGreen = (BYTE) G ;
      plp->palPalEntry[iEntry].peBlue = (BYTE) B ;
      plp->palPalEntry[iEntry].peFlags = 0 ;

      piCount [iIndex] = 0 ;
  }
  // On exit from the loop iEntry will be the number of stored entries
  plp->palNumEntries = iEntry ;
  // Create the palette, clean up, and return the palette handle
  hPalette = CreatePalette (plp) ;
  free (piCount) ;
  free (plp) ;

  return hPalette ;
}

/*-------------------------------------------------------------------------
Structures used for implementing median cut algorithm
-------------------------------------------------------------------------*/

typedef struct // defines dimension of a box
{
  int Rmin, Rmax, Gmin, Gmax, Bmin, Bmax ;
}
MINMAX ;
typedef struct // for Compare routine for qsort
{
  int iBoxCount ;
  RGBQUAD rgbBoxAv ;
}

BOXES ;
/*-------------------------------------------------------------------------
FindAverageColor: In a box
-------------------------------------------------------------------------*/

static int FindAverageColor ( int * piCount, MINMAX mm,
                        int iRes, RGBQUAD * prgb)
{
  int R, G, B, iR, iG, iB, iTotal, iCount ;
  // Initialize some variables
  iTotal = iR = iG = iB = 0 ;
  // Loop through all colors in the box
```

```
  for (R = mm.Rmin ; R <= mm.Rmax ; R++)
    for (G = mm.Gmin ; G <= mm.Gmax ; G++)
      for (B = mm.Bmin ; B <= mm.Bmax ; B++)
      {
        // Get the number of pixels of that color
        iCount = piCount [PACK_RGB (R, G, B, iRes)] ;
        // Weight the pixel count by the color value
        iR += iCount * R ;
        iG += iCount * G ;
        iB += iCount * B ;

        iTotal += iCount ;
      }
      // Find the average color
      prgb->rgbRed = (BYTE) ((iR / iTotal) << (8 - iRes)) ;
      prgb->rgbGreen = (BYTE) ((iG / iTotal) << (8 - iRes)) ;
      prgb->rgbBlue = (BYTE) ((iB / iTotal) << (8 - iRes)) ;

      // Return the total number of pixels in the box

      return iTotal ;
}

/*-------------------------------------------------------------------------
CutBox: Divide a box in two
-------------------------------------------------------------------------*/
static void CutBox (int * piCount, int iBoxCount, MINMAX mm,
                    int iRes, int iLevel, BOXES * pboxes, int * piEntry)
{
  int iCount, R, G, B ;
  MINMAX mmNew ;

  // If the box is empty, return

  if (iBoxCount == 0)
    return ;

  // If the nesting level is 8, or the box is one pixel, we're ready
  // to find the average color in the box and save it along with
  // the number of pixels of that color

  if (iLevel == 8 || ( mm.Rmin == mm.Rmax &&
    mm.Gmin == mm.Gmax &&
    mm.Bmin == mm.Bmax))
  {
    pboxes[*piEntry].iBoxCount =
      FindAverageColor (piCount, mm, iRes, &pboxes[*piEntry].rgbBoxAv) ;
    (*piEntry) ++ ;
  }
  // Otherwise, if blue is the largest side, split it
  else if ((mm.Bmax - mm.Bmin > mm.Rmax - mm.Rmin) &&
    (mm.Bmax - mm.Bmin > mm.Gmax - mm.Gmin))
  {
    // Initialize a counter and loop through the blue side
    iCount = 0 ;
    for (B = mm.Bmin ; B < mm.Bmax ; B++)
    {
      // Accumulate all the pixels for each successive blue value
      for ( R = mm.Rmin ; R <= mm.Rmax ; R++)
        for ( G = mm.Gmin ; G <= mm.Gmax ; G++)
          iCount += piCount [PACK_RGB (R, G, B, iRes)] ;

      // If it's more than half the box count, we're there

      if (i Count >= iBoxCount / 2)
        break ;

      // If the next blue value will be the max, we're there
      if ( B == mm.Bmax - 1)
```

```
        break ;
    }
    // Cut the two split boxes.
    // The second argument to CutBox is the new box count.
    // The third argument is the new min and max values.

    mmNew = mm ;
    mmNew.Bmin = mm.Bmin ;
    mmNew.Bmax = B ;

    CutBox ( piCount, iCount, mmNew, iRes, iLevel + 1,
      pboxes, piEntry) ;

    mmNew.Bmin = B + 1 ;
    mmNew.Bmax = mm.Bmax ;

    CutBox ( piCount, iBoxCount - iCount, mmNew, iRes, iLevel + 1,
      pboxes, piEntry) ;
  }
  // Otherwise, if red is the largest side, split it (just like blue)
  else if (mm.Rmax - mm.Rmin > mm.Gmax - mm.Gmin)
  {
    iCount = 0 ;
    for (R = mm.Rmin ; R < mm.Rmax ; R++)
    {
      for (B = mm.Bmin ; B <= mm.Bmax ; B++)
        for (G = mm.Gmin ; G <= mm.Gmax ; G++)
          iCount += piCount [PACK_RGB (R, G, B, iRes)] ;
      if (iCount >= iBoxCount / 2)
        break ;
      if (R == mm.Rmax - 1)
        break ;
    }
    mmNew = mm ;
    mmNew.Rmin = mm.Rmin ;
    mmNew.Rmax = R ;

    CutBox ( piCount, iCount, mmNew, iRes, iLevel + 1,
      pboxes, piEntry) ;

    mmNew.Rmin = R + 1 ;
    mmNew.Rmax = mm.Rmax ;
    CutBox ( piCount, iBoxCount - iCount, mmNew, iRes, iLevel + 1,
      pboxes, piEntry) ;
  }
  // Otherwise, split along the green size
  else
  {
    iCount = 0 ;
    for (G = mm.Gmin ; G < mm.Gmax ; G++)
    {
      for ( B = mm.Bmin ; B <= mm.Bmax ; B++)
        for ( R = mm.Rmin ; R <= mm.Rmax ; R++)
          iCount += piCount [PACK_RGB (R, G, B, iRes)] ;

      if ( iCount >= iBoxCount / 2)
        break ;

      if ( G == mm.Gmax - 1)
        break ;
    }
    mmNew = mm ;
    mmNew.Gmin = mm.Gmin ;
    mmNew.Gmax = G ;

    CutBox ( piCount, iCount, mmNew, iRes, iLevel + 1,
      pboxes, piEntry) ;

    mmNew.Gmin = G + 1 ;
```

```
    mmNew.Gmax = mm.Gmax ;

    CutBox ( piCount, iBoxCount - iCount, mmNew, iRes, iLevel + 1,
      pboxes, piEntry) ;
  }
}

/*-------------------------------------------------------------------------
Compare routine for qsort
-------------------------------------------------------------------------*/

static int Compare (const BOXES * pbox1, const BOXES * pbox2)
{
  return pbox1->iBoxCount - pbox2->iBoxCount ;
}

/*-------------------------------------------------------------------------
DibPalMedianCut: Creates palette based on median cut algorithm
-------------------------------------------------------------------------*/
HPALETTE DibPalMedianCut (HDIB hdib, int iRes)
{
  BOXES boxes [256] ;
  HPALETTE hPalette ;
  int i, iArraySize, iCount, R, G, B, iTotCount, iDim, iEntry = 0 ;
  int * piCount ;
  LOGPALETTE * plp ;
  MINMAX mm ;

  // Validity checks

  if (DibBitCount (hdib) < 16)
    return NULL ;
  if (iRes < 3 || iRes > 8)
    return NULL ;
  // Accumulate counts of pixel colors
  iArraySize = 1 << (3 * iRes) ;
  if (NULL == (piCount = calloc (iArraySize, sizeof (int))))
    return NULL ;
  AccumColorCounts (hdib, piCount, iRes) ;
  // Find the dimensions of the total box
  iDim = 1 << iRes ;
  mm.Rmin = mm.Gmin = mm.Bmin = iDim - 1 ;
  mm.Rmax = mm.Gmax = mm.Bmax = 0 ;

  iTotCount = 0 ;
  for (R = 0 ; R < iDim ; R++)
    for (G = 0 ; G < iDim ; G++)
      for (B = 0 ; B < iDim ; B++)
        if ((iCount = piCount [PACK_RGB (R, G, B, iRes)]) > 0)
        {
          iTotCount += iCount ;
          if (R < mm.Rmin) mm.Rmin = R ;
          if (G < mm.Gmin) mm.Gmin = G ;
          if (B < mm.Bmin) mm.Bmin = B ;
          if (R > mm.Rmax) mm.Rmax = R ;
          if (G > mm.Gmax) mm.Gmax = G ;
          if (B > mm.Bmax) mm.Bmax = B ;
        }

        // Cut the first box (iterative function).
        // On return, the boxes structure will have up to 256 RGB values,
        // one for each of the boxes, and the number of pixels in
        // each box.
        // The iEntry value will indicate the number of non-empty boxes.

        CutBox (piCount, iTotCount, mm, iRes, 0, boxes, &iEntry) ;
        free (piCount) ;

        // Sort the RGB table by the number of pixels for each color
```

```
      qsort (boxes, iEntry, sizeof (BOXES), Compare) ;
      plp = malloc (sizeof (LOGPALETTE) +
               (iEntry - 1) * sizeof (PALETTEENTRY)) ;
      if (plp == NULL)
        return NULL ;
      plp->palVersion = 0x0300 ;
      plp->palNumEntries = iEntry ;

      for (i = 0 ; i < iEntry ; i++)
      {
        plp->palPalEntry[i].peRed = boxes[i].rgbBoxAv.rgbRed ;
        plp->palPalEntry[i].peGreen= boxes[i].rgbBoxAv.rgbGreen ;
        plp->palPalEntry[i].peBlue = boxes[i].rgbBoxAv.rgbBlue ;
        plp->palPalEntry[i].peFlags= 0 ;
      }

      hPalette = CreatePalette (plp) ;
      free (plp) ;
      return hPalette ;
}
```

第一个函数 – DibPalDibTable – 看起来应该很熟悉。它根据DIB的颜色表建立了调色盘。这与本章前面的SHOWDIB3中所用到的PACKEDIB.C里的PackedDibCreatePalette函数相似。在SHOWDIB3中，只有当DIB有颜色表时才执行此函数。在8位显示模式下试图显示16位、24位或32位DIB时，此函数就没用了。

预设情况下，执行在256色显示模式下时，DIBBLE将首先尝试呼叫DibPalDibTable来根据DIB颜色表建立调色盘。如果DIB没有颜色表，则DIBBLE将呼叫CreateHalftonePalette并将fHalftonePalette变量设定为TRUE。此逻辑发生在WM_USER_CREATEPAL消息处理期间。

DIBPAL.C也执行函数DibPalAllPurpose，因为此函数与SHOWDIB4中的CreateAllPurposePalette函数非常相似，所以它看起来也很熟悉。您也可以从DIBBLE菜单中选择此调色盘。

在256色模式下显示位图最有趣的是，您可以直接控制Windows用于显示图像的颜色。如果您选择并显现调色盘，则Winsows将使用此调色盘中的颜色，而不是其它调色盘中的颜色。

例如，您可以用DibPalUniformGrays函数来单独建立一种灰阶调色盘。使用两种灰阶的调色盘则只含有00-00-00（黑色）和FF-FF-FF（白色）。用此调色盘来输出图像将提供某些照片中常用的高对比「黑白」效果。使用3种灰阶将在黑色和白色中间添加中间灰色，使用4种灰阶将添加2种灰阶。

用8种灰阶，您就有可能看到明显的轮廓 – 相同灰阶的无规则斑点，虽然很明显地执行了最接近颜色算法，但是一般仍不带有任何审美判断。通常到16种灰阶就可以明显改善图像画质。使用32种灰阶差不多就可以消除全部轮廓了。而目前普遍认为64种灰阶是现在大多数显示设备的极限。在这点以上，再提升也没什么边际效益了。在6位颜色分辨率的设备上提供超过64种灰阶看不出有什么改进之处。

迄今为止，对于8位显示模式下显示16位、24位和32位彩色DIB，我们最多就是能够设计通用调色盘（这对灰阶图像很有效，但通常不适于彩色图像）或者使用中间色调色盘，它用混色显示与通用颜色调色盘合用。

还应注意，当您在8位颜色模式下为大张16位、24位或32位DIB选择通用调色盘时，为了要显示这些图像，程序将花费一些时间依据DIB的内容来建立GDI位图对象。如果不需要调色盘，则程序根据DIB来建立DDB的时间会更少（用8位彩色模式显示大24位DIB时，比较SHOWDIB1和SHOWDIB4的性能，您也能看出这点区别）。这是为什么呢?

它按最接近颜色搜寻。通常，用8位显示模式显示24位DIB时（或者将DIB转换为DDB），GDI必须将DIB中的每个像素都与静态20种颜色中的一种相贴近。完成此操作的唯一方法是决定哪种静态颜色与像素颜色最接近。这包括计算像素与三维RGB颜色中每种静态颜色的距离。这将花些时间，特别是在DIB图像中有上百万个像素时。

在建立232色调色盘时，例如DIBBLE和SHOWDIB4中的通用调色盘，您会很快将搜索最接近颜色的时间增加到超过11倍！GDI现在必须彻底检查232种颜色，而不是20种。那就是显示DIB的整个作业放慢的原因。

这里的教训是避免在8位显示模式下显示24位（或16位，或32位）DIB。您应该找出最接近DIB图像颜色范围的256色调色盘，来将它们转换成8位DIB。这经常称为「最佳调色盘」。当我研究这个问题的时候，Paul Heckbert编写的〈Color Image Quantization for Frame Buffer Displays〉（刊登在1982年7月出版的《Computer Graphics》）对此问题有所帮助。

均匀分布

建立256色调色盘最简单的方法是选择范围统一的RGB颜色值，它与DibPalAllPurpose中的方法相似。此方法的优点是您不必检查DIB中的实际像素。这个函数是DibPalCreateUniformColors，它依据范围统一的RGB三原色索引建立调色盘。

一个合理的分布包括8阶红色和绿色以及4阶蓝色（肉眼对蓝色较不敏感）。调色盘是RGB颜色值的集合，它是红色和绿色值0x00、0x24、0x49、0x6D、0x92、0xB6、0xDB和0xFF以及蓝色值0x00、0x55、0xAA和0xFF的所有可能的组合，共有256种颜色。另一种可能的统一分布调色盘使用6阶红色、绿色和蓝色。此调色盘是红色、绿色和蓝色值为0x00、0x33、0x66、0x99、0xCC和0xFF的所有可能的组合，调色盘中的颜色数是6的3次方，即216。

这两个选项和其它几个选项都由DIBBLE提供。

「Popularity」算法

「Popularity」算法是256色调色盘问题相当明显的解决方法。您要做的就是走遍位图中的所有像素，并找出256种最普通的RGB颜色值。这些就是您在调色盘中使用的值。DIBPAL的DibPalPopularity函数中实作了这种算法。

不过，如果每种颜色都使用整个24位，而且假设需要用整数来计算所有的颜色，那么数组将占据64MB内存。另外，您可以发现位图中实际上没有（或很少）重复的24位像素值，这样就没有所谓常见的颜色了。

要解决这个问题，您可以只使用每个红色、绿色和蓝色值中最重要的n位 – 例如，6位而不是8位。因为大多数的彩色扫描仪和视讯显示卡都只有6位的分辨率，所以这样规定更有意义。这将数组减少到大小更合理的256KB或1MB。只使用5位能将可用的颜色总数减少到32,768。通常，使用5位要比6位的性能更好。对此，您可以用DIBBLE和一些图像颜色来自己检验。

「Median Cut」算法

DIBPAL.C中的DibPalMedianCut函数执行Paul Heckbert的Median Cut算法。此算法在概念上相当简单，但在程序代码中实作要比Popularity算法更困难，它适合递归函数。

画出RGB颜色立方体。图像中的每个像素都是此立方体中的一个点。一些点可能代表图像中的多个像素。找出包括图像中所有像素的立体方块，找出此方块的最大尺寸，并将方块分成两个，每个方块都包括相同数量的像素。对于这两个方块，执行相同的操作。现在您就有4个方块，将这4个方块分成8个，然后再分成16个、32个、64个、128个和256个。

现在您有256个方块，每个方块都包括相同数量的像素。取每个方块中像素RGB颜色值的平均

值，并将结果用于调色盘。

实际上，这些方块通常包含像素的数量并不相同。例如，通常包括单个点的方块会有更多的像素。这发生在黑色和白色上。有时，一些方块里头根本没有像素。如果这样，您就可以省下更多的方块，但是我决定不这样做。

另一种最佳化调色盘的技术称为「octree quantization」，此技术由Jeff Prosise提出，并于1996年8月发表在《Microsoft Systems Journal》上（包含在MSDN的CD中）。

转换格式

DIBBLE还允许将DIB从一种格式转换到另一种格式。这用到了DIBCONV文件中的DibConvert函数，如程序16-25所示。

程序16-25 DIBCONV文件

DIBCONV.H

```
/*----------------------------------------------------------------------------
  DIBCONV.H header file for DIBCONV.C
----------------------------------------------------------------------------*/
HDIB DibConvert (HDIB hdibSrc, int iBitCountDst) ;

   DIBCONV.C
/*----------------------------------------------------------------------------
DIBCONV.C -- Converts DIBs from one format to another
(c) Charles Petzold, 1998
----------------------------------------------------------------------------*/
#include <windows.h>
#include "dibhelp.h"
#include "dibpal.h"
#include "dibconv.h"

HDIB DibConvert (HDIB hdibSrc, int iBitCountDst)
{
  HDIB hdibDst ;
  HPALETTE hPalette ;
  int i, x, y, cx, cy, iBitCountSrc, cColors ;
  PALETTEENTRY pe ;
  RGBQUAD rgb ;
  WORD wNumEntries ;

  cx = DibWidth (hdibSrc) ;
  cy = DibHeight (hdibSrc) ;
  iBitCountSrc = DibBitCount (hdibSrc) ;

  if (iBitCountSrc == iBitCountDst)
    return NULL ;
  // DIB with color table to DIB with larger color table:
  if ((iBitCountSrc < iBitCountDst) && (iBitCountDst <= 8))
  {
    cColors = DibNumColors (hdibSrc) ;
    hdibDst = DibCreate (cx, cy, iBitCountDst, cColors) ;

    for (i = 0 ; i < cColors ; i++)
    {
      DibGetColor (hdibSrc, i, &rgb) ;
      DibSetColor (hdibDst, i, &rgb) ;
    }

    for (x = 0 ; x < cx ; x++)
      for (y = 0 ; y < cy ; y++)
      {
        DibSetPixel (hdibDst, x, y, DibGetPixel (hdibSrc, x, y)) ;
      }
  }
  // Any DIB to DIB with no color table
```

```
else if (iBitCountDst >= 16)
{
  hdibDst = DibCreate (cx, cy, iBitCountDst, 0) ;
  for (x = 0 ; x < cx ; x++)
    for (y = 0 ; y < cy ; y++)
    {
      DibGetPixelColor (hdibSrc, x, y, &rgb) ;
      DibSetPixelColor (hdibDst, x, y, &rgb) ;
    }
}
// DIB with no color table to 8-bit DIB
else if (iBitCountSrc >= 16 && iBitCountDst == 8)
{
  hPalette = DibPalMedianCut (hdibSrc, 6) ;
  GetObject (hPalette, sizeof (WORD), &wNumEntries) ;

  hdibDst = DibCreate (cx, cy, 8, wNumEntries) ;
  for (i = 0 ; i < (int) wNumEntries ; i++)
  {
    GetPaletteEntries (hPalette, i, 1, &pe) ;
    rgb.rgbRed = pe.peRed ;
    rgb.rgbGreen = pe.peGreen ;
    rgb.rgbBlue = pe.peBlue ;
    rgb.rgbReserved = 0 ;

    DibSetColor (hdibDst, i, &rgb) ;
  }

  for (x = 0 ; x < cx ; x++)
    for (y = 0 ; y < cy ; y++)
    {
      DibGetPixelColor (hdibSrc, x, y, &rgb) ;

      DibSetPixel (hdibDst, x, y,
        GetNearestPaletteIndex (hPalette,
        RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue))) ;
    }
    DeleteObject (hPalette) ;
}
// Any DIB to monochrome DIB

else if (iBitCountDst == 1)
{
  hdibDst = DibCreate (cx, cy, 1, 0) ;
  hPalette = DibPalUniformGrays (2) ;

  for (i = 0 ; i < 2 ; i++)
  {
    GetPaletteEntries (hPalette, i, 1, &pe) ;

    rgb.rgbRed = pe.peRed ;
    rgb.rgbGreen = pe.peGreen ;
    rgb.rgbBlue = pe.peBlue ;
    rgb.rgbReserved = 0 ;

    DibSetColor (hdibDst, i, &rgb) ;
  }

  for (x = 0 ; x < cx ; x++)
    for (y = 0 ; y < cy ; y++)
    {
      DibGetPixelColor (hdibSrc, x, y, &rgb) ;

      DibSetPixel (hdibDst, x, y,
        GetNearestPaletteIndex (hPalette,
        RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue))) ;
    }
    DeleteObject (hPalette) ;
}
```

```
  // All non-monochrome DIBs to 4-bit DIB
  else if (iBitCountSrc >= 8 && iBitCountDst == 4)
  {
    hdibDst = DibCreate (cx, cy, 4, 0) ;
    hPalette = DibPalVga () ;

    for (i = 0 ; i < 16 ; i++)
    {
      GetPaletteEntries (hPalette, i, 1, &pe) ;
      rgb.rgbRed = pe.peRed ;
      rgb.rgbGreen = pe.peGreen ;
      rgb.rgbBlue = pe.peBlue ;
      rgb.rgbReserved = 0 ;

      DibSetColor (hdibDst, i, &rgb) ;
    }

    for (x = 0 ; x < cx ; x++)
      for (y = 0 ; y < cy ; y++)
      {
        DibGetPixelColor (hdibSrc, x, y, &rgb) ;

        DibSetPixel (hdibDst, x, y,
          GetNearestPaletteIndex (hPalette,
          RGB (rgb.rgbRed, rgb.rgbGreen, rgb.rgbBlue))) ;
      }
      DeleteObject (hPalette) ;
  }
  // Should not be necessary
  else
    hdibDst = NULL ;
  return hdibDst ;
}
```

将DIB从一种格式转换成另一种格式需要几种不同的方法。

要将带有颜色表的DIB转换成另一种也带有颜色表但有较大的像素宽度的DIB（亦即，将1位DIB转换成4位或8位DIB，或将4位DIB转换成8位DIB），所需要做的就是透过呼叫DibCreate来建立新的DIB，并在呼叫时带有希望的位数以及与原始DIB中的颜色数相等的颜色数。然后函数复制像素位和颜色表项目。

如果新的DIB没有颜色表（即位数是16、24或32），那么DIB只需要按新格式建立，而且通过呼叫DibGetPixelColor和DibSetPixelColor从现有的DIB中复制像素位。

下面的情况可能更普遍：现有的DIB没有颜色表（即位数是16、24或32），而新的DIB每像素占8位。这种情况下，DibConvert呼叫DibPalMedianCut来为图像建立最佳化的调色盘。新DIB的颜色表设定为调色盘中的RGB值。DibGetPixelColor函数从现有的DIB中获得像素颜色。透过呼叫GetNearestPaletteIndex来转换成8位DIB中的像素值，并透过呼叫DibSetPixel将像素值储存到DIB。

当DIB需要转换成单色DIB时，用包括两个项目 – 黑色和白色 – 的颜色表建立新的DIB。另外，GetNearestPaletteIndex有助于将现有DIB中的颜色转换成像素值0或1。类似地，当8个像素位或更多位的DIB要转换成4位DIB时，可从DibPalVga函数获得DIB颜色表，同时GetNearestPaletteIndex也有助于计算像素值。

尽管DIBBLE示范了如何开始写一个图像处理程序基础，但是程序最后还是没有全部完成，我们总是会想到还有些功能没有加进去里头。但是很可惜的是，我们现在得停止继续研究这些东西，而往下讨论别的东西了。

第十七章 文字和字体

显示文字是本书所要解决的首要问题，现在我们来研究Microsoft Windows中各种有效字体和字体大小的使用方法以及调整文字的方式。

Windows 3.1发表的TrueType使程序写作者和使用者以灵活的方式处理文字的能力大幅增强。TrueType是轮廓字体技术，由Apple Computer公司和Microsoft公司开发，并被许多字体制造商支持。由于TrueType字体能够连续缩放，并能应用于视讯显示器和打印机，现在能够在Windows下实作真的WYSIWYG（what you see is what you get：所见即所得）。TrueType也便于制作「奇妙」字体，例如旋转的字母、内部填充图案的字母或将它们用于剪裁区域，在本章我将展示它们。

简单的文字输出

让我们先来看看Windows为文字输出、影响文字的设备内容属性以及备用字体提供的各种函数。

文字输出函数

我已经在许多范例程序中使用过最常用的文字输出函数：

TextOut (hdc, xStart, yStart, pString, iCount) ;

参数xStart和yStart是逻辑坐标上字符串的起始点。通常，这是Windows开始绘制的第一个字母的左上角。TextOut需要指向字符串的指针和字符串的长度，这个函数不能识别以NULL终止的字符串。

TextOut函数的xStart和yStart参数的含义可由SetTextAlign函数改变。TA_LEFT、TA_RIGHT和TA_CENTER旗标影响使用xStart在水平方向上定位字符串的方式。默认值是TA_LEFT。如果在SetTextAlign函数中指定了TA_RIGHT，则后面的TextOut呼叫会将字符串的最后一个字符定位于xStart，如果指定了TA_CENTER，则字符串的中心位于xStart。

类似地，TA_TOP、TA_BOTTOM和TA_BASELINE旗标影响字符串的垂直位置。TA_TOP是默认值，它意味着字符串的字母顶端位于yStart，使用TA_BOTTOM意味着字符串位于yStart之上。可以使用TA_BASELINE定位字符串，使基准线位于yStart。基准线是如小写字母p、q、y等字母下部的线。

如果您使用TA_UPDATECP旗标呼叫SetTextAlign，Windows就会忽略TextOut的xStart和yStart参数，而使用由MoveToEx、LineTo或更改目前位置的另一个函数设定的位置。TA_UPDATECP旗标也使TextOut函数将目前位置更新为字符串的结尾（TA_LEFT）或字符串的开头（TA_RIGHT）。这在使用多个TextOut呼叫显示一行文字时非常有用。当水平位置是TA_CENTER时，在TextOut呼叫后，目前位置不变。

您应该还记得，第四章的一系列SYSMETS程序显示几列文字时，对每一列都需要呼叫一个TextOut，其替代函数是TabbedTextOut函数：

```
TabbedTextOut ( hdc, xStart, yStart, pString, iCount,
            iNumTabs, piTabStops, xTabOrigin) ;
```

如果文字字符串中含有嵌入的制表符（‘\t’或0x09），则TabbedTextOut会根据传递给它的整数数组将制表符扩展为空格。

TabbedTextOut的前五个参数与TextOut相同，第六个参数是跳位间隔数，第七个是以像素为单位的跳位间隔数组。例如，如果平均字符宽度是8个像素，而您希望每5个字符加一个跳位间隔，则这个数组将包含40、80、120，按递增顺序依此类推。

如果第六个和第七个参数是0或NULL，则跳位间隔按每八个平均字符宽度设定。如果第六个参数是1，则第七个参数指向一个整数，表示跳位间隔重复增大的倍数（例如，如果第六个参数是1，并且第七个参数指向值为30的变量，则跳位间隔设定在30、60、90…像素处）。最后一个参数给出了从跳位间隔开始测量的逻辑x坐标，它与字符串的起始位置可能相同也可能不同。

另一个进阶的文字输出函数是ExtTextOut（前缀Ext表示它是扩展的）：

```
ExtTextOut (hdc, xStart, yStart, iOptions, &rect,
          pString, iCount, pxDistance) ;
```

第五个参数是指向矩形结构的指针，在iOptions设定为ETO_CLIPPED时，该结构为剪裁矩形，在iOptions设定为ETO_OPAQUE时，该结构为用目前背景色填充的背景矩形。这两种选择您可以都采用，也可以都不采用。

最后一个参数是整数数组，它指定了字符串中连续字符的间隔。程序可以使用它使字符间距变窄或变宽，因为有时需要在较窄的列中调整单个文字。该参数可以设定为NULL来使用内定的字符间距。

用于写文字的高级函数是DrawText，我们第一次遇到它是在第三章讨论HELLOWIN程序时，它不指定坐标的起始位置，而是通过RECT结构型态定义希望显示文字的区域：

DrawText (hdc, pString, iCount, &rect, iFormat) ;

和其它文字输出函数一样，DrawText需要指向字符串的指针和字符串的长度。然而，如果在DrawText中使用以NULL结尾的字符串，就可以将iCount设定为-1，Windows会自动计算字符串的长度。

当iFormat设定为0时，Windows会将文字解释为一系列由carriage return字符（‘\r’或0x0D）或linefeed字符（‘\n’或0x0A）分隔的行。文字从矩形的左上角开始，carriage return字符或linefeed字符被解释为换行字符，因此Windows会结束目前行而开始新的一行。新的一行从矩形的左侧开始，在上一行的下面空开一个字符的高度（没有外部间隔）。包含字母的任何文字都应该显示在所剪裁矩形底部的右边或下边。

您可以使用iFormat参数更改DrawText的内定操作，iFormat由一个或多个旗标组成。DT_LEFT旗标（默认值）指定了左对齐的行，DT_RIGHT指定了向右对齐的行，而DT_CENTER指定了位于矩形左边和右边中间的行。因为DT_LEFT的值是0，所以如果只需要左对齐，就不需要包含标识符。

如果您不希望将carriage return字符或linefeed字符解释为换行字符，则可以包括标识符DT_SINGLELINE。然后，Windows会把carriage return字符和linefeed字符解释为可显示的字符，而不是控制字符。在使用DT_SINGLELINE时，还可以将行指定为位于矩形的顶端（DT_TOP）、底端(DT_BOTTOM)或者中间(DT_VCETER，V表示垂直)。

在显示多行文字时，Windows通常只在carriage return字符或linefeed字符处换行。然而，如果行的长度超出了矩形的宽度，则可以使用DT_WORDBREAK旗标，它使Windows在行内字的末尾换行。对于单行或多行文字的显示，Windows会把超出矩形的文字部分截去，可以使用DT_NOCLIP跳过这个操作，这个旗标还加快了函数的速度。当Windows确定多行文字的行距时，它通常使用不带外部间距的字符高度，如果您想在行距中加入外部间距，就可以使用旗标

DT_EXTERNALLEADING。

如果文字中包含制表符（‘\t’或0x09），则您需要包括旗标DT_EXPANDTABS。在内定情况下，跳位间隔设定于每八个字符的位置。通过使用旗标DT_TABSTOP，您可以指定不同的跳位间隔，在这种情况下，iFormat的高字节包含了每个新跳位间隔的字符位置数值。不过我建议您避免使用DT_TABSTOP，因为iFormat的高字节也用于其它旗标。

DT_TABSTOP旗标存在的问题，可以由新的函数DrawTextEx来解决，它含有一个额外的参数：

DrawTextEx (hdc, pString, iCount, &rect, iFormat, &drawtextparams) ;

最后一个参数是指向DRAWTEXTPARAMS结构的指针，它的定义如下：

```
typedef struct tagDRAWTEXTPARAMS
{
  UINT cbSize ; // size of structure
  int iTabLength ; // size of each tab stop
  int iLeftMargin ; // left margin
  int iRightMargin ; // right margin
  UINT uiLengthDrawn ; // receives number of characters processed
} DRAWTEXTPARAMS, * LPDRAWTEXTPARAMS ;
```

中间的三个字段是以平均字符的增量为单位的。

文字的设备内容属性

除了上面讨论的SerTextAlign外，其它几个设备内容属性也对文字产生了影响。在内定的设备内容下，文字颜色是黑色，但您可以用下面的叙述进行更改：

SetTextColor (hdc, rgbColor) ;

使用画笔的颜色和画刷的颜色，Windows把rgbColor的值转换为纯色，您可以通过呼叫GetTextColor取得目前文字的颜色。

Windows在矩形的背景区域中显示文字，它可能根据背景模式的设定进行着色，也可能不这样做。您可以使用

SetBkMode (hdc, iMode) ;

更改背景模式，其中iMode的值为OPAQUE或TRANSPARENT。内定的背景模式为OPAQUE，它表示Windows使用背景颜色来填充矩形的背景。您可以使用

SetBkColor (hdc, rgbColor) ;

来改变背景颜色。rgbColor的值是转换为纯色的值。内定背景色是白色。

如果两行文字靠得太近，其中一个的背景矩形就会遮盖另一个的文字。由于这种原因，我通常希望内定的背景模式是TRANSPARENT。在背景模式为TRANSPARENT的情况下，Windows会忽略背景色，也不对矩形背景区域着色。Windows也使用背景模式和背景色对点和虚线之间的空隙及阴影刷中阴影间的区域着色，就像第五章所讨论的那样。

许多Windows程序将WHITE_BRUSH指定为Windows用于擦出窗口背景的画刷，画刷在窗口类别结构中指定。然而，您可能希望您程序的窗口背景与使用者在「控制台」中设定的系统颜色保持一致，在这种情况下，可以在WNDCLASS结构中指定背景颜色的这种方式：

wndclass.hbrBackground = COLOR_WINDOW + 1 ;

当您想要在显示区域书写文字时，可以使用目前系统颜色设定文字色和背景色：

SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT)) ;

SetBkColor (hdc, GetSysColor (COLOR_WINDOW)) ;

完成这些以后，就可以使您的程序随系统颜色的更改而变化：

```
case WM_SYSCOLORCHANGE :
  InvalidateRect (hwnd, NULL, TRUE) ;
  break ;
```

另一个影响文字的设备内容属性是字符间距。它的默认值是0，表示Windows不在字符之间添加任何空间，但您可以使用以下函数插入空间：

SetTextCharacterExtra (hdc, iExtra) ;

参数iExtra是逻辑单位，Windows将其转换为最接近的像素，它可以是0。如果您将iExtra取为负值（希望将字符紧紧压在一起），Windows会接受这个数值的绝对值—也就是说，您不能使iExtra的值小于0。您可以通过呼叫GetTextCharacterExtra取得目前的字符间距，Windows在传回该值前会将像素间距转换为逻辑单位。

使用备用字体

当您呼叫TextOut、TabbedTextOut、ExtTextOut、DrawText或DrawTextEx书写文字时，Windows使用设备内容中目前选择的字体。字体定义了特定的字样和大小。以不同字体显示文字的最简单方法是使用Windows提供的备用字体，然而，它的范围是很有限的。

您可以呼叫下面的函数取得某种备用字体的句柄：

hFont = GetStockObject (iFont) ;

其中，iFont是几个标识符之一。然后，您就可以将该字体选入设备内容：

SelectObject (hdc, hFont) ;

这些您也可以只用一步完成：

SelectObject (hdc, GetStockObject (iFont)) ;

在内定的设备内容中选择的字体称为系统字体，能够由GetStockObject的SYSTEM_FONT参数识别。这是调和的ANSI字符集字体。在GetStockObject中指定SYSTEM_FIXED_FONT（我在本书的前面几个程序中应用过），可以获得等宽字体的句柄，这一字体与Windows 3.0以前的系统字体兼容。在您希望所有的字体都具有相同宽度时，这是很方便的。

备用字体OEM_FIXED_FONT也称为终端机字体，是Windows在MS-DOS命令提示窗口中使用的字体，它包括与原始IBM-PC扩展字符集兼容的字符集。Windows在窗口标题栏、菜单和对话框的文字中使用DEFULT_GUI_FONT。

当您将新字体选入设备内容时，必须使用GetTextMetrics计算字符的高度和平均宽度。如果选择了调和字体，那么一定要注意，字符的平均宽度只是个平均值，某些字符会比它宽或比它窄。在本章的后面，您会了解到确定由不同宽度字符所组成的字符串总宽度的方法。

尽管GetStockObject确实提供了存取不同字体的最简单方式，但是您还不能充分控件Windows所提供的字体。不久，您会看到指定字体字样和大小的方法。

字体的背景

本章剩余的部分致力于处理不同的字体。但是在您接触这些特定程序代码前，对Windows使用字体的基本知识有一个深入的了解是很有好处的。

字体型态

Windows支持两大类字体，即所谓的「GDI字体」和「设备字体」。GDI字体储存在硬盘的文件中，而设备字体是输出设备本来就有的。例如，通常打印机都具有内建的设备字体集。

GDI字体有三种样式：点阵字体，笔划字体和TrueType字体。

点阵字体的每个字符都以位图像素图案的形式储存，每种点阵字体都有特定的纵横比和字符大小。Windows通过简单地复制像素的行或列就可以由GDI点阵字体产生更大的字符。然而，只能以整数倍放大字体，并且不能超过一定的限度。由于这种原因，GDI点阵字体又称为「不可缩放的」字体。它们不能随意地放大或缩小。点阵字体的主要优点是显示性能（显示速度很快）和可读性（因为是手工设计的，所以尽可能清晰）。

字体是通过字体名称识别的，点阵字体的字体名称为：

System （用于SYSTEM_FONT）

FixedSys （用于SYSTEM_FIXED_FONT）

Terminal （用于OEM_FIXED_FONT）

Courier

MS Serif

MS Sans Serif（用于DEFAULT_GUI_FONT）

Small Fonts

每个点阵字体只有几种大小（不超过6种）。Courier字体是定宽字体，外形与用打字机打出的字体相似。「Serif」指字体字母笔划在结束时拐个小弯。「sans serif」字体不是serif类的字体。在Windows的早期版本中，MS（Microsoft）Serif和MS Sans Serif字体被称为Tms Rmn（指它与Times Roman相似）和Helv（与Helvetica相似）。Small Fonts是专为显示小字设计的。

在Windows3.1以前，除了GDI字体外，Windows所提供的字体只有笔划字体。笔划字体是以「连结点」的方式定义的一系列线段，笔划字体可以连续地缩放，这意味着同样的字体可以用于具有任何分辨率的图形输出设备，并且字体可以放大或缩小到任意尺寸。不过，它的性能不好，小字体的可读性也很糟，而大字体由于笔划是单根直线而显得很单薄。笔划字体有时也称为绘图机字体，因为它们特别适合于绘图机，但是不适合于别的场合。笔划字体的字样有：Modern、Roman和Script。

对于GDI点阵字体和GDI笔划字体，Windows都可以「合成」粗体、斜体、加底线和加删除线，而不需要为每种属性另外储存字体。例如，对于斜体，Windows只需要将字符的上部向右移动就可以了。

接下来是Truetype，我将在本章的剩部分主要讨论它。

TrueType 字体

TrueType字体的单个字符是通过填充的直线和曲线的轮廓来定义的。Windows可以通过改变定义轮廓的坐标对TrueType字体进行缩放。

当程序开始使用特定大小的TrueType字体时，Windows「点阵化」字体。这就是说Windows使用TrueType字体文件中包括的「提示」对每个字符的连结直线和曲线的坐标进行缩放。这些提示可以补偿误差，避免合成的字符变得很难看（例如，在某些字体中，大写H的两竖应该一样宽，但盲目地缩放字体可能会导致其中一竖的像素比另一竖宽。有了提示就可以避免这些现象发生）。然

后，每个字符的合成轮廓用于建立字符的位图，这些位图储存在内存以备将来使用。

最初，Windows使用了13种TrueType字体，它们的字体名称如下：

Courier New

Courier New Bold

Courier New Italic

Courier New Bold Italic

Times New Roman

Times New Roman Bold

Times New Roman Italic

Times New Roman Bold Italic

Arial

Arial Bold

Arial Italic

Arial Bold Italic

Symbol

在新的Windows版本中，这个列表更长了。在此特别指出，我将使用Lucida Sans Unicode字体，它包括了一些在世界其它地方使用的字母表。

三个主要字体系列与点阵字体相似，Courier New是定宽字体。它看起来就像是打字机输出的字体。Times New Roman是Times字体的复制品，该字体最初为《Times of London》设计，并用在许多印刷材料上，它具有很好的可读性。Arial是Helvetica字体的复制品，是一种sans serif字体。Symbol字体包含了手写符号集。

属性或样式

在上面的TrueType字体列表中，您会注意到，Courier、Times New Roman和Arial的粗体和斜体是带有自己字体名称的单独字体，这一命名与传统的板式一致。然而，计算机使用者认为粗体和斜体只是已有字体的特殊「属性」。Windows在定义点阵字体命名、列举和选择的方式时，采用了属性的方法。但对于TrueType字体，更倾向于使用传统的命名方式。

这种冲突在Windows中还没有完全解决，简而言之，您可以完全通过命名或特定属性来选择字体。然而在处理字体列举时，应用程序需要系统中的字体列表，正如您所预料，这种双重处理使问题复杂化了。

点值

在传统的版式中，您可以用字体名称和大小来指定字体，字体的大小以点的单位来表示。一点与1/72英寸很接近——它们非常接近，因此在电脑中它通常定义为1/72英寸。点值通常描述为字母顶端（不包括发音符号）到字母底端的高度，例如，字母「bq」的总高度。这是一个考虑字体大小的简单方式，但它通常不是很精确。

字体的点值实际上是排版设计的概念而不是度量概念。特定字体中字元的大小可能会大于或小于其点值所表示的大小。在传统的排版中，您使用点值来指定字体的大小，在电脑排版中，还有其他方法来确定字元的实际大小。

间隔和间距

在第四章我们曾提到，可以通过呼叫GetTextMetrics取得设备内容中目前选择的字体信息，我们也多次使用过这个函数。图4-3显示了FONTMETRIC结构中字体的垂直大小。

TEXTMETRIC结构的另一个字段是tmExternalLeading，词「间隔（leading）」来自排字工人在金属字块间插入的铅，它用于在两行文字之间产生空白。tmInternalLeading值与为发音符号保留的空间有关，tmExternalLeading表示字符的连续行之间所留的附加空间。程序写作者可以使用或忽略外部的间隔值。

当我们说一个字体是8点或12点时，指的是不带内部间隔的高度。某种大写字母上的发音符号占据了分隔行的间距。这样，TEXTMETRIC结构的tmHeight值实际指行间距而不是字体的点值。字体的点值可由tmHeight减tmInternalLeading得到。

逻辑英寸问题

正如我们在第五章〈设备的大小〉一节中所讨论的，Windows 98将系统字体定义为带有12点行距的10点字体。根据在「显示属性」对话方块中选择的是「小字体」还是「大字体」，该字体的tmHeight值为16或20像素，tmHeight减去tmInternalLeading的值为13或16像素。这样，字体的选择就暗指以每英寸的点数为单位的设备分辨率，选择「小字体」即为96dpi，选择「大字体」即为120dpi。

您可以用LOGPIXELSX或LOGPIXELSY参数呼叫GetDeviceCaps来取得该设备分辨率。因此，96或120像素在萤幕上占有的度量距离可以称为「逻辑英寸」。如果您用尺测量屏幕并计算像素，就可能发现逻辑英寸要比实际的英寸大一些，为什么会这样呢?

在纸张上，每英寸放设14个8点的字元很方便阅读。如果您在作文书处理或写作应用程式时，可能希望在显示器上显示清晰的8点字型，但如果使用视讯显示器的实际尺寸，就没有足够的像素清晰地显示字元。即使显示器具有足够的分辨率，在屏幕上阅读8点字体仍然会有问题。当人们阅读纸上的印刷物时，眼睛与文字的距离通常为一英尺，而使用视讯显示器时，这个距离通常为两英尺。

逻辑英寸有效地对屏幕进行了放大，能够显示小至8点的清晰字体。而且，每英寸96点使640像素的最小显示大小等于大约6.5英寸。这恰恰是在页边距为1英寸的8.5英寸宽的纸上列印的文字的宽度。因而，逻辑英寸也利用了屏幕宽度，尽可能大地显示文字。

您可能还记得在第五章，Windows NT的做法有些不同。在Windows NT中，从GetDeviceCaps中得到的LOGPIXELSX（每英寸的像素数）值不等于HORZRES值（像素数）除以HORZSIZE值（毫米数）再乘以25.4的值。以此类似，LOGPIXELSY、VERTRES和VERTSIZE也不一致。Windows在为不同映射方式计算视窗和偏移范围时，使用HORZRES、HORZSIZE、VERTRES和VERTSIZE值。然而，显示文字的程式最好不要使用根据LOGPIXELSX和LOGPIXELSY使用假定的显示分辨率，这一点与Windows 98更为一致。

所以，在Windows NT下，当程式以特定的点值显示文字时，它可能不使用Windows提供的映射方式，程式根据与Windows 98一样的每英寸的逻辑像素数来定义自己的映射方式。我将这种用於文字的映射方式称为「Logical Twips」映射方式。您可以设定如下：

```
SetMapMode (hdc, MM_ANISOTROPIC) ;
SetWindowExtEx (hdc, 1440, 1440, NULL) ;
SetViewportExt (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
                GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
```

使用这种映像方式设定，您能够以点值的20倍来指定字体大小，例如，为12点字取240。注意，与MM_TWIPS映像方式不同，y值在屏幕中向下增长，这在显示文字的连续行时很方便。

请记住，逻辑英吋与实际英吋间的差异仅对显示器存在。在打印设备上，GDI和尺是完全一致的。

逻辑字体

既然我们已经明确了逻辑英吋和逻辑单位的概念，那么现在我们就来讨论逻辑字体

逻辑字体是一个GDI对象，它的句柄储存在HFONT型态的变量中，逻辑字体是字体的描述。和逻辑画笔及逻辑画刷一样，它是抽象的对象，只有当应用程序呼叫SelectObject将它选入设备内容时，它才成为真实的对象。例如，对于逻辑画笔，您可以为画笔指定任意的颜色，但是在您将画笔选入设备内容时，Windows才将其转换为设备中有效的颜色。只有此时，Windows才知道设备的色彩能力。

逻辑字体的建立和选择

您可以透过呼叫CreateFont或CreateFontIndirect来建立逻辑字体。CreateFontIndirect函数接受一个指向LOGFONT结构的指针，该结构有14个字段。CreateFont函数接受14个参数，它们与LOGFONT结构的14个字段形式相同。它们是仅有的两个建立逻辑字体的函数（我提到这一点，是因为Windows中有许多用于其它字体操作的函数）。因为很难记住14个字段，所以很少使用CreateFont。因此，我主要讨论CreateFontIndirect。

有三种基本的方式用于定义LOGFONT结构中的字段，以便呼叫CreateFontIndirect：

您可以简单地将LOGFONT结构的字段设定为所需的字体特征。在这种情况下，在呼叫SelectObject时，Windows使用「字体映像」算法从设备上有效的字体中选择与这些特征最匹配的字体。由于这依赖于视讯显示器和打印机上的有效字体，所以其结果可能与您的要求有相当大的差别。

您可以列举设备上的所有字体并从中选择，甚至用对话框把它们显示给使用者。我将在本章后面讨论字体列举函数。不过，它们现在已经不常用了，因为第三种方法也可以进行列举。

您可以采用简单的方法并呼叫ChooseFont函数，我在第十一章曾讨论过这个函数，能够使用LOGFONT结构直接建立字体。

在本章，我使用第一种和第三种方法。

下面是建立、选择和删除逻辑字体的程序：

通过呼叫CreateFont或CreateFontIndirect建立逻辑字体，这些函数传回HFONT型态的逻辑字体句柄。

使用SelectObject将逻辑字体选入设备内容，Windows会选择与逻辑字体最匹配的真实字体。

使用GetTextMetrics（及可能用到的其它函数）确定真实字体的大小和特征。在该字体选入设备内容后，可以使用这些信息来适当地设定文字的间距。

在使用完逻辑字体后，呼叫DeleteObject删除逻辑字体，当字体选入有效的设备内容时，不要删除字体，也不要删除备用字体。

GetTextFace函数使程序能够确定目前选入设备内容的字体名称：

```
GetTextFace (hdc, sizeof (szFaceName) / sizeof (TCHAR), szFaceName) ;
```

详细的字体信息可以从GetTextMetrics中得到：

GetTextMetrics (hdc, &textmetric) ;

其中，textmetric是TEXTMETRIC型态的变量，它具有20个字段。

稍后我将详细讨论LOGFONT和TEXTMETRIC结构的字段，这两个结构有一些相似的字段，所以它们容易混淆。现在您只需记住，LOGFONT用于定义逻辑字体，而TEXTMETRIC用于取得目前选入设备内容中的字体信息。

PICKFONT程序

使用程序17-1所示的PICKFONT，可以定义LOGFONT结构的许多字段。这个程序建立逻辑字体，并在逻辑字体选入设备内容后显示真实字体的特征。这是个方便的程序，通过它我们可以了解逻辑字体映像为真实字体的方式。

程序17-1 PICKFONT

PICKFONT.C

```
/*---------------------------------------------------------------------------
PICKFONT.C -- Create Logical Font
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

// Structure shared between main window and dialog box
typedef struct
{
  int iDevice, iMapMode ;
  BOOL fMatchAspect ;
  BOOL fAdvGraphics ;
  LOGFONT lf ;
  TEXTMETRIC tm ;
  TCHAR szFaceName [LF_FULLFACESIZE] ;
}
DLGPARAMS ;
// Formatting for BCHAR fields of TEXTMETRIC structure
#ifdef UNICODE
#define BCHARFORM TEXT ("0x%04X")
#else
#define BCHARFORM TEXT ("0x%02X")
#endif

// Global variables
HWND hdlg ;
TCHAR szAppName[] = TEXT ("PickFont") ;

// Forward declarations of functions
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
void SetLogFontFromFields (HWND hdlg, DLGPARAMS * pdp) ;
void SetFieldsFromTextMetric (HWND hdlg, DLGPARAMS * pdp) ;
void MySetMapMode (HDC hdc, int iMapMode) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("PickFont: Create Logical Font"),
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (hdlg == 0 || !IsDialogMessage (hdlg, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static DLGPARAMS dp ;
  static TCHAR szText[] = TEXT ("\x41\x42\x43\x44\x45 ")
    TEXT ("\x61\x62\x63\x64\x65 ")

    TEXT ("\xC0\xC1\xC2\xC3\xC4\xC5 ")
    TEXT ("\xE0\xE1\xE2\xE3\xE4\xE5 ")
#ifdef UNICODE
    TEXT ("\x0390\x0391\x0392\x0393\x0394\x0395 ")
    TEXT ("\x03B0\x03B1\x03B2\x03B3\x03B4\x03B5 ")
    TEXT ("\x0410\x0411\x0412\x0413\x0414\x0415 ")
    TEXT ("\x0430\x0431\x0432\x0433\x0434\x0435 ")
    TEXT ("\x5000\x5001\x5002\x5003\x5004")
#endif
    ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    dp.iDevice = IDM_DEVICE_SCREEN ;
    hdlg = CreateDialogParam (((LPCREATESTRUCT) lParam)->hInstance,
      szAppName, hwnd, DlgProc, (LPARAM) &dp) ;
    return 0 ;
  case WM_SETFOCUS:
    SetFocus (hdlg) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_DEVICE_SCREEN:
    case IDM_DEVICE_PRINTER:
      CheckMenuItem (GetMenu (hwnd), dp.iDevice, MF_UNCHECKED) ;
```

```
     dp.iDevice = LOWORD (wParam) ;
     CheckMenuItem (GetMenu (hwnd), dp.iDevice, MF_CHECKED) ;
     SendMessage (hwnd, WM_COMMAND, IDOK, 0) ;
     return 0 ;
    }
    break ;

    case WM_PAINT:
     hdc = BeginPaint (hwnd, &ps) ;

     // Set graphics mode so escapement works in Windows NT
     SetGraphicsMode (hdc, dp.fAdvGraphics ? GM_ADVANCED : GM_COMPATIBLE) ;

     // Set the mapping mode and the mapper flag
     MySetMapMode (hdc, dp.iMapMode) ;
     SetMapperFlags (hdc, dp.fMatchAspect) ;

     // Find the point to begin drawing text
     GetClientRect (hdlg, &rect) ;
     rect.bottom += 1 ;
     DPtoLP (hdc, (PPOINT) &rect, 2) ;

     // Create and select the font; display the text
     SelectObject (hdc, CreateFontIndirect (&dp.lf)) ;
     TextOut (hdc, rect.left, rect.bottom, szText, lstrlen (szText)) ;

     DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
     EndPaint (hwnd, &ps) ;
     return 0 ;

    case WM_DESTROY:
     PostQuitMessage (0) ;
     return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK DlgProc ( HWND hdlg, UINT message, WPARAM wParam,LPARAM lParam)
{
  static DLGPARAMS * pdp ;
  static PRINTDLG pd = { sizeof (PRINTDLG) } ;
  HDC hdcDevice ;
  HFONT hFont ;

  switch (message)
  {
  case WM_INITDIALOG:
    // Save pointer to dialog-parameters structure in WndProc
    pdp = (DLGPARAMS *) lParam ;

    SendDlgItemMessage (hdlg, IDC_LF_FACENAME, EM_LIMITTEXT,
      LF_FACESIZE - 1, 0) ;
    CheckRadioButton (hdlg,IDC_OUT_DEFAULT, IDC_OUT_OUTLINE,
      IDC_OUT_DEFAULT) ;
    CheckRadioButton (hdlg,IDC_DEFAULT_QUALITY, IDC_PROOF_QUALITY,
      IDC_DEFAULT_QUALITY) ;
    CheckRadioButton (hdlg,IDC_DEFAULT_PITCH, IDC_VARIABLE_PITCH,
      IDC_DEFAULT_PITCH) ;
    CheckRadioButton (hdlg,IDC_FF_DONTCARE, IDC_FF_DECORATIVE,
      IDC_FF_DONTCARE) ;
    CheckRadioButton (hdlg,IDC_MM_TEXT, IDC_MM_LOGTWIPS,
      IDC_MM_TEXT) ;
    SendMessage (hdlg, WM_COMMAND, IDOK, 0) ;
    // fall through
  case WM_SETFOCUS:
    SetFocus (GetDlgItem (hdlg, IDC_LF_HEIGHT)) ;
    return FALSE ;

  case WM_COMMAND:
```

```
    switch (LOWORD (wParam))
    {
    case IDC_CHARSET_HELP:
      MessageBox ( hdlg,
        TEXT ("0 = Ansi\n")
        TEXT ("1 = Default\n")
        TEXT ("2 = Symbol\n")
        TEXT ("128 = Shift JIS (Japanese)\n")
        TEXT ("129 = Hangul (Korean)\n")
        TEXT ("130 = Johab (Korean)\n")
        TEXT ("134 = GB 2312 (Simplified Chinese)\n")
        TEXT ("136 = Chinese Big 5 (Traditional Chinese)\n")
        TEXT ("177 = Hebrew\n")
        TEXT ("178 = Arabic\n")
        TEXT ("161 = Greek\n")
        TEXT ("162 = Turkish\n")
        TEXT ("163 = Vietnamese\n")
        TEXT ("204 = Russian\n")
        TEXT ("222 = Thai\n")
        TEXT ("238 = East European\n")
        TEXT ("255 = OEM"),
        szAppName, MB_OK | MB_ICONINFORMATION) ;
      return TRUE ;

      // These radio buttons set the lfOutPrecision field
    case IDC_OUT_DEFAULT:
      pdp->lf.lfOutPrecision = OUT_DEFAULT_PRECIS ;
      return TRUE ;

    case IDC_OUT_STRING:
      pdp->lf.lfOutPrecision = OUT_STRING_PRECIS ;
      return TRUE ;

    case IDC_OUT_CHARACTER:
      pdp->lf.lfOutPrecision = OUT_CHARACTER_PRECIS ;
      return TRUE ;

    case IDC_OUT_STROKE:
      pdp->lf.lfOutPrecision = OUT_STROKE_PRECIS ;
      return TRUE ;

    case IDC_OUT_TT:
      pdp->lf.lfOutPrecision = OUT_TT_PRECIS ;
      return TRUE ;

    case IDC_OUT_DEVICE:
      pdp->lf.lfOutPrecision = OUT_DEVICE_PRECIS ;
      return TRUE ;

    case IDC_OUT_RASTER:
      pdp->lf.lfOutPrecision = OUT_RASTER_PRECIS ;
      return TRUE ;

    case IDC_OUT_TT_ONLY:
      pdp->lf.lfOutPrecision = OUT_TT_ONLY_PRECIS ;
      return TRUE ;

    case IDC_OUT_OUTLINE:
      pdp->lf.lfOutPrecision = OUT_OUTLINE_PRECIS ;
      return TRUE ;

      // These three radio buttons set the lfQuality field
    case IDC_DEFAULT_QUALITY:
      pdp->lf.lfQuality = DEFAULT_QUALITY ;
      return TRUE ;

    case IDC_DRAFT_QUALITY:
      pdp->lf.lfQuality = DRAFT_QUALITY ;
      return TRUE ;
```

```
  case IDC_PROOF_QUALITY:
    pdp->lf.lfQuality = PROOF_QUALITY ;
    return TRUE ;

    // These three radio buttons set the lower nibble
    // of the lfPitchAndFamily field
  case IDC_DEFAULT_PITCH:
    pdp->lf.lfPitchAndFamily =
      (0xF0 & pdp->lf.lfPitchAndFamily) | DEFAULT_PITCH ;
    return TRUE ;

  case IDC_FIXED_PITCH:
    pdp->lf.lfPitchAndFamily =
      (0xF0 & pdp->lf.lfPitchAndFamily) | FIXED_PITCH ;
    return TRUE ;

  case IDC_VARIABLE_PITCH:
    pdp->lf.lfPitchAndFamily =
      (0xF0 & pdp->lf.lfPitchAndFamily) | VARIABLE_PITCH ;
    return TRUE ;

    // These six radio buttons set the upper nibble
    // of the lfPitchAndFamily field
  case IDC_FF_DONTCARE:
    pdp->lf.lfPitchAndFamily =
      (0x0F & pdp->lf.lfPitchAndFamily) | FF_DONTCARE ;
    return TRUE ;

  case IDC_FF_ROMAN:
    pdp->lf.lfPitchAndFamily =
      (0x0F & pdp->lf.lfPitchAndFamily) | FF_ROMAN ;
    return TRUE ;

  case IDC_FF_SWISS:
    pdp->lf.lfPitchAndFamily =
      (0x0F & pdp->lf.lfPitchAndFamily) | FF_SWISS ;
    return TRUE ;

  case IDC_FF_MODERN:
    pdp->lf.lfPitchAndFamily =
      (0x0F & pdp->lf.lfPitchAndFamily) | FF_MODERN ;
    return TRUE ;

  case IDC_FF_SCRIPT:
    pdp->lf.lfPitchAndFamily =
      (0x0F & pdp->lf.lfPitchAndFamily) | FF_SCRIPT ;
    return TRUE ;

  case IDC_FF_DECORATIVE:
    pdp->lf.lfPitchAndFamily =
      (0x0F & pdp->lf.lfPitchAndFamily) | FF_DECORATIVE ;
    return TRUE ;

    // Mapping mode:
  case IDC_MM_TEXT:
  case IDC_MM_LOMETRIC:
  case IDC_MM_HIMETRIC:
  case IDC_MM_LOENGLISH:
  case IDC_MM_HIENGLISH:
  case IDC_MM_TWIPS:
  case IDC_MM_LOGTWIPS:
    pdp->iMapMode = LOWORD (wParam) ;
    return TRUE ;

    // OK button pressed
    // -----------------

  case IDOK:
```

```
     // Get LOGFONT structure
     SetLogFontFromFields (hdlg, pdp) ;

     // Set Match-Aspect and Advanced Graphics flags
     pdp->fMatchAspect = IsDlgButtonChecked (hdlg, IDC_MATCH_ASPECT) ;
     pdp->fAdvGraphics = IsDlgButtonChecked (hdlg, IDC_ADV_GRAPHICS) ;

     // Get Information Context
     if (pdp->iDevice == IDM_DEVICE_SCREEN)
     {
       hdcDevice = CreateIC (TEXT ("DISPLAY"), NULL, NULL, NULL) ;
     }
     else
     {
       pd.hwndOwner = hdlg ;
       pd.Flags = PD_RETURNDEFAULT | PD_RETURNIC ;
       pd.hDevNames = NULL ;
       pd.hDevMode = NULL ;

       PrintDlg (&pd) ;

       hdcDevice = pd.hDC ;
     }
     // Set the mapping mode and the mapper flag
     MySetMapMode (hdcDevice, pdp->iMapMode) ;
     SetMapperFlags (hdcDevice, pdp->fMatchAspect) ;

     // Create font and select it into IC
     hFont = CreateFontIndirect (&pdp->lf) ;
     SelectObject (hdcDevice, hFont) ;

     // Get the text metrics and face name
     GetTextMetrics (hdcDevice, &pdp->tm) ;
     GetTextFace (hdcDevice, LF_FULLFACESIZE, pdp->szFaceName) ;
     DeleteDC (hdcDevice) ;
     DeleteObject (hFont) ;

     // Update dialog fields and invalidate main window
     SetFieldsFromTextMetric (hdlg, pdp) ;
     InvalidateRect (GetParent (hdlg), NULL, TRUE) ;
     return TRUE ;
 }
 break ;
 }
 return FALSE ;
}
void SetLogFontFromFields (HWND hdlg, DLGPARAMS * pdp)
{
  pdp->lf.lfHeight = GetDlgItemInt (hdlg, IDC_LF_HEIGHT, NULL, TRUE) ;
  pdp->lf.lfWidth = GetDlgItemInt (hdlg, IDC_LF_WIDTH, NULL, TRUE) ;
  pdp->lf.lfEscapement=GetDlgItemInt (hdlg, IDC_LF_ESCAPE, NULL, TRUE) ;
  pdp->lf.lfOrientation=GetDlgItemInt (hdlg,IDC_LF_ORIENT, NULL, TRUE) ;
  pdp->lf.lfWeight =GetDlgItemInt (hdlg, IDC_LF_WEIGHT, NULL, TRUE) ;
  pdp->lf.lfCharSet =GetDlgItemInt (hdlg, IDC_LF_CHARSET, NULL, FALSE) ;
  pdp->lf.lfItalic =IsDlgButtonChecked(hdlg,IDC_LF_ITALIC) == BST_CHECKED ;
  pdp->lf.lfUnderline =IsDlgButtonChecked (hdlg, IDC_LF_UNDER) == BST_CHECKED ;
  pdp->lf.lfStrikeOut =IsDlgButtonChecked (hdlg, IDC_LF_STRIKE) == BST_CHECKED ;
  GetDlgItemText (hdlg, IDC_LF_FACENAME, pdp->lf.lfFaceName, LF_FACESIZE) ;
}

void SetFieldsFromTextMetric (HWND hdlg, DLGPARAMS * pdp)
{
  TCHAR szBuffer [10] ;
  TCHAR * szYes = TEXT ("Yes") ;
  TCHAR * szNo = TEXT ("No") ;
  TCHAR * szFamily [] = {TEXT ("Don't Know"),
    TEXT ("Roman"),
    TEXT ("Swiss"), TEXT ("Modern"),
    TEXT ("Script"), TEXT ("Decorative"),
```

```
    TEXT ("Undefined") } ;

  SetDlgItemInt (hdlg, IDC_TM_HEIGHT, pdp->tm.tmHeight, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_ASCENT, pdp->tm.tmAscent, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_DESCENT,pdp->tm.tmDescent, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_INTLEAD,pdp->tm.tmInternalLeading, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_EXTLEAD,pdp->tm.tmExternalLeading, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_AVECHAR,pdp->tm.tmAveCharWidth, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_MAXCHAR, pdp->tm.tmMaxCharWidth, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_WEIGHT, pdp->tm.tmWeight, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_OVERHANG,pdp->tm.tmOverhang, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_DIGASPX, pdp->tm.tmDigitizedAspectX, TRUE) ;
  SetDlgItemInt (hdlg, IDC_TM_DIGASPY, pdp->tm.tmDigitizedAspectY, TRUE) ;

  wsprintf (szBuffer, BCHARFORM, pdp->tm.tmFirstChar) ;
  SetDlgItemText (hdlg, IDC_TM_FIRSTCHAR, szBuffer) ;

  wsprintf (szBuffer, BCHARFORM, pdp->tm.tmLastChar) ;
  SetDlgItemText (hdlg, IDC_TM_LASTCHAR, szBuffer) ;

  wsprintf (szBuffer, BCHARFORM, pdp->tm.tmDefaultChar) ;
  SetDlgItemText (hdlg, IDC_TM_DEFCHAR, szBuffer) ;

  wsprintf (szBuffer, BCHARFORM, pdp->tm.tmBreakChar) ;
  SetDlgItemText (hdlg, IDC_TM_BREAKCHAR, szBuffer) ;

  SetDlgItemText (hdlg, IDC_TM_ITALIC, pdp->tm.tmItalic ? szYes : szNo) ;
  SetDlgItemText (hdlg, IDC_TM_UNDER, pdp->tm.tmUnderlined ? szYes : szNo) ;
  SetDlgItemText (hdlg, IDC_TM_STRUCK, pdp->tm.tmStruckOut ? szYes : szNo) ;

  SetDlgItem Text (hdlg, IDC_TM_VARIABLE,
    TMPF_FIXED_PITCH & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

  SetDlgItem Text (hdlg, IDC_TM_VECTOR,
    TMPF_VECTOR & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

  SetDlgItem Text (hdlg, IDC_TM_TRUETYPE,
    TMPF_TRUETYPE & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

  SetDlgItem Text (hdlg, IDC_TM_DEVICE,
    TMPF_DEVICE & pdp->tm.tmPitchAndFamily ? szYes : szNo) ;

  SetDlgItem Text (hdlg, IDC_TM_FAMILY,
    szFamily [min (6, pdp->tm.tmPitchAndFamily >> 4)]) ;

  SetDlgItemInt (hdlg, IDC_TM_CHARSET, pdp->tm.tmCharSet, FALSE) ;
  SetDlgItemText(hdlg, IDC_TM_FACENAME, pdp->szFaceName) ;
}

void MySetMapMode (HDC hdc, int iMapMode)
{
  switch (iMapMode)
  {
  case IDC_MM_TEXT: SetMapMode (hdc, MM_TEXT) ; break ;
  case IDC_MM_LOMETRIC: SetMapMode (hdc, MM_LOMETRIC) ; break ;
  case IDC_MM_HIMETRIC: SetMapMode (hdc, MM_HIMETRIC) ; break ;
  case IDC_MM_LOENGLISH: SetMapMode (hdc, MM_LOENGLISH) ; break ;
  case IDC_MM_HIENGLISH: SetMapMode (hdc, MM_HIENGLISH) ; break ;
  case IDC_MM_TWIPS: SetMapMode (hdc, MM_TWIPS) ; break ;
  case IDC_MM_LOGTWIPS:
    SetMapMode (hdc, MM_ANISOTROPIC) ;
    SetWindowExtEx (hdc, 1440, 1440, NULL) ;
    SetViewportExtEx (hdc,GetDeviceCaps (hdc, LOGPIXELSX),
      GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;
    break ;
  }
}
```

PICKFONT.RC

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Dialog
PICKFONT DIALOG DISCARDABLE 0, 0, 348, 308
STYLE WS_CHILD | WS_VISIBLE | WS_BORDER
FONT 8, "MS Sans Serif"
BEGIN
LTEXT  "&Height:",IDC_STATIC,8,10,44,8
EDITTEXT  IDC_LF_HEIGHT,64,8,24,12,ES_AUTOHSCROLL
LTEXT   "&Width",IDC_STATIC,8,26,44,8
EDITTEXT  IDC_LF_WIDTH,64,24,24,12,ES_AUTOHSCROLL
LTEXT   "Escapement:",IDC_STATIC,8,42,44,8
EDITTEXT  IDC_LF_ESCAPE,64,40,24,12,ES_AUTOHSCROLL
LTEXT   "Orientation:",IDC_STATIC,8,58,44,8
EDITTEXT  IDC_LF_ORIENT,64,56,24,12,ES_AUTOHSCROLL
LTEXT   "Weight:",IDC_STATIC,8,74,44,8
EDITTEXT  IDC_LF_WEIGHT,64,74,24,12,ES_AUTOHSCROLL
GROUPBOX  "Mapping Mode",IDC_STATIC,97,3,96,90,WS_GROUP
CONTROL  "Text",IDC_MM_TEXT,"Button",BS_AUTORADIOBUTTON,104,13,56,
8
CONTROL  "Low Metric",IDC_MM_LOMETRIC,"Button",BS_AUTORADIOBUTTON,
104,24,56,8
CONTROL  High Metric",IDC_MM_HIMETRIC,"Button",
BS_AUTORADIOBUTTON,104,35,56,8
CONTROL  "Low English",IDC_MM_LOENGLISH,"Button",
BS_AUTORADIOBUTTON,104,46,56,8
CONTROL " High English",IDC_MM_HIENGLISH,"Button",
BS_AUTORADIOBUTTON,104,57,56,8
CONTROL  "Twips",IDC_MM_TWIPS,"Button",BS_AUTORADIOBUTTON,104,68,
56,8
CONTROL  "Logical Twips",IDC_MM_LOGTWIPS,"Button",
BS_AUTORADIOBUTTON,104,79,64,8
CONTROL  "Italic",IDC_LF_ITALIC,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,8,90,48,12
CONTROL "Underline",IDC_LF_UNDER,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,8,104,48,12
CONTROL  "Strike Out",IDC_LF_STRIKE,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,8,118,48,12
CONTROL  "Match Aspect",IDC_MATCH_ASPECT,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,60,104,62,8
CONTROL  "Adv Grfx Mode",IDC_ADV_GRAPHICS,"Button",
BS_AUTOCHECKBOX | WS_TABSTOP,60,118,62,8
LTEXT   "Character Set:",IDC_STATIC,8,137,46,8
EDITTEXT  IDC_LF_CHARSET,58,135,24,12,ES_AUTOHSCROLL
PUSHBUTTON "?",IDC_CHARSET_HELP,90,135,14,14
GROUPBOX  "Quality",IDC_STATIC,132,98,62,48,WS_GROUP
CONTROL  "Default",IDC_DEFAULT_QUALITY,"Button",
BS_AUTORADIOBUTTON,136,110,40,8
CONTROL  "Draft",IDC_DRAFT_QUALITY,"Button",BS_AUTORADIOBUTTON,
136,122,40,8
CONTROL  "Proof",IDC_PROOF_QUALITY,"Button",BS_AUTORADIOBUTTON,
136,134,40,8
LTEXT   "Face Name:",IDC_STATIC,8,154,44,8
EDITTEXT  IDC_LF_FACENAME,58,152,136,12,ES_AUTOHSCROLL
GROUPBOX  "Output Precision",IDC_STATIC,8,166,118,133,WS_GROUP
CONTROL  "OUT_DEFAULT_PRECIS",IDC_OUT_DEFAULT,"Button",
BS_AUTORADIOBUTTON,12,178,112,8
CONTROL  "OUT_STRING_PRECIS",IDC_OUT_STRING,"Button",
BS_AUTORADIOBUTTON,12,191,112,8
CONTROL  "OUT_CHARACTER_PRECIS",IDC_OUT_CHARACTER,"Button",
BS_AUTORADIOBUTTON,12,204,112,8
CONTROL  "OUT_STROKE_PRECIS",IDC_OUT_STROKE,"Button",
BS_AUTORADIOBUTTON,12,217,112,8
CONTROL  "OUT_TT_PRECIS",IDC_OUT_TT,"Button",BS_AUTORADIOBUTTON,
12,230,112,8
```

```
CONTROL  "OUT_DEVICE_PRECIS",IDC_OUT_DEVICE,"Button",
BS_AUTORADIOBUTTON,12,243,112,8
CONTROL  "OUT_RASTER_PRECIS",IDC_OUT_RASTER,"Button",
BS_AUTORADIOBUTTON,12,256,112,8
CONTROL  "OUT_TT_ONLY_PRECIS",IDC_OUT_TT_ONLY,"Button",
BS_AUTORADIOBUTTON,12,269,112,8
CONTROL  "OUT_OUTLINE_PRECIS",IDC_OUT_OUTLINE,"Button",
BS_AUTORADIOBUTTON,12,282,112,8
GROUPBOX  "Pitch",IDC_STATIC,132,166,62,50,WS_GROUP
CONTROL  "Default",IDC_DEFAULT_PITCH,"Button",BS_AUTORADIOBUTTON,
137,176,52,8
CONTROL  "Fixed",IDC_FIXED_PITCH,"Button",BS_AUTORADIOBUTTON,137,
189,52,8
CONTROL  "Variable",IDC_VARIABLE_PITCH,"Button",
BS_AUTORADIOBUTTON,137,203,52,8
GROUPBOX  "Family",IDC_STATIC,132,218,62,82,WS_GROUP
CONTROL  "Don't Care",IDC_FF_DONTCARE,"Button",BS_AUTORADIOBUTTON,
137,229,52,8
CONTROL  "Roman",IDC_FF_ROMAN,"Button",BS_AUTORADIOBUTTON,137,241,
52,8
CONTROL  "Swiss",IDC_FF_SWISS,"Button",BS_AUTORADIOBUTTON,137,253,
52,8
CONTROL  "Modern",IDC_FF_MODERN,"Button",BS_AUTORADIOBUTTON,137,
265,52,8
CONTROL  "Script",IDC_FF_SCRIPT,"Button",BS_AUTORADIOBUTTON,137,
277,52,8
CONTROL   "Decorative",IDC_FF_DECORATIVE,"Button",
BS_AUTORADIOBUTTON,137,289,52,8
DEFPUSHBUTTON "OK",IDOK,247,286,50,14
GROUPBOX   "Text Metrics",IDC_STATIC,201,2,140,272,WS_GROUP
LTEXT   "Height:",IDC_STATIC,207,12,64,8
LTEXT   "0",IDC_TM_HEIGHT,281,12,44,8
LTEXT   "Ascent:",IDC_STATIC,207,22,64,8
LTEXT   "0",IDC_TM_ASCENT,281,22,44,8
LTEXT   "Descent:",IDC_STATIC,207,32,64,8
LTEXT   "0",IDC_TM_DESCENT,281,32,44,8
LTEXT   "Internal Leading:",IDC_STATIC,207,42,64,8
LTEXT   "0",IDC_TM_INTLEAD,281,42,44,8
LTEXT   "External Leading:",IDC_STATIC,207,52,64,8
LTEXT   "0",IDC_TM_EXTLEAD,281,52,44,8
LTEXT   "Ave Char Width:",IDC_STATIC,207,62,64,8
LTEXT   "0",IDC_TM_AVECHAR,281,62,44,8
LTEXT   "Max Char Width:",IDC_STATIC,207,72,64,8
LTEXT   "0",IDC_TM_MAXCHAR,281,72,44,8
LTEXT   "Weight:",IDC_STATIC,207,82,64,8
LTEXT   "0",IDC_TM_WEIGHT,281,82,44,8
LTEXT   "Overhang:",IDC_STATIC,207,92,64,8
LTEXT   "0",IDC_TM_OVERHANG,281,92,44,8
LTEXT   "Digitized Aspect X:",IDC_STATIC,207,102,64,8
LTEXT   "0",IDC_TM_DIGASPX,281,102,44,8
LTEXT   "Digitized Aspect Y:",IDC_STATIC,207,112,64,8
LTEXT   "0",IDC_TM_DIGASPY,281,112,44,8
LTEXT   "First Char:",IDC_STATIC,207,122,64,8
LTEXT   "0",IDC_TM_FIRSTCHAR,281,122,44,8
LTEXT   "Last Char:",IDC_STATIC,207,132,64,8
LTEXT   "0",IDC_TM_LASTCHAR,281,132,44,8
LTEXT   "Default Char:",IDC_STATIC,207,142,64,8
LTEXT   "0",IDC_TM_DEFCHAR,281,142,44,8
LTEXT   "Break Char:",IDC_STATIC,207,152,64,8
LTEXT   "0",IDC_TM_BREAKCHAR,281,152,44,8
LTEXT   "Italic?",IDC_STATIC,207,162,64,8
LTEXT   "0",IDC_TM_ITALIC,281,162,44,8
LTEXT   "Underlined?",IDC_STATIC,207,172,64,8
LTEXT   "0",IDC_TM_UNDER,281,172,44,8
LTEXT   "Struck Out?",IDC_STATIC,207,182,64,8
LTEXT   "0",IDC_TM_STRUCK,281,182,44,8
LTEXT   "Variable Pitch?",IDC_STATIC,207,192,64,8
LTEXT   "0",IDC_TM_VARIABLE,281,192,44,8
LTEXT   "Vector Font?",IDC_STATIC,207,202,64,8
```

```
LTEXT   "0",IDC_TM_VECTOR,281,202,44,8
LTEXT   "TrueType Font?",IDC_STATIC,207,212,64,8
LTEXT   "0",IDC_TM_TRUETYPE,281,212,44,8
LTEXT   "Device Font?",IDC_STATIC,207,222,64,8
LTEXT   "0",IDC_TM_DEVICE,281,222,44,8
LTEXT   "Family:",IDC_STATIC,207,232,64,8
LTEXT   "0",IDC_TM_FAMILY,281,232,44,8
LTEXT   "Character Set:",IDC_STATIC,207,242,64,8
LTEXT   "0",IDC_TM_CHARSET,281,242,44,8
LTEXT   "0",IDC_TM_FACENAME,207,262,128,8
END

/////////////////////////////////////////////////////////////////////////////
// Menu
PICKFONT MENU DISCARDABLE
BEGIN
POPUP "&Device"
BEGIN
MENUITEM "&Screen", IDM_DEVICE_SCREEN, CHECKED
MENUITEM "&Printer",  IDM_DEVICE_PRINTER
END
END
```

RESOURCE.H

```
// Microsoft Developer Studio generated include file.
// Used by PickFont.rc
#define  IDC_LF_HEIGHT  1000
#define  IDC_LF_WIDTH  1001
#define  IDC_LF_ESCAPE  1002
#define  IDC_LF_ORIENT  1003
#define  IDC_LF_WEIGHT  1004
#define  IDC_MM_TEXT  1005
#define  IDC_MM_LOMETRIC 1006
#define  IDC_MM_HIMETRIC 1007
#define  IDC_MM_LOENGLISH 1008
#define  IDC_MM_HIENGLISH 1009
#define  IDC_MM_TWIPS  1010
#define  IDC_MM_LOGTWIPS 1011
#define  IDC_LF_ITALIC  1012
#define  IDC_LF_UNDER  1013
#define  IDC_LF_STRIKE  1014
#define  IDC_MATCH_ASPECT 1015
#define  IDC_ADV_GRAPHICS 1016
#define  IDC_LF_CHARSET  1017
#define  IDC_CHARSET_HELP 1018
#define  IDC_DEFAULT_QUALITY  1019
#define  IDC_DRAFT_QUALITY  1020
#define  IDC_PROOF_QUALITY  1021
#define  IDC_LF_FACENAME   1022
#define  IDC_OUT_DEFAULT   1023
#define  IDC_OUT_STRING   1024
#define  IDC_OUT_CHARACTER  1025
#define  IDC_OUT_STROKE   1026
#define  IDC_OUT_TT    1027
#define  IDC_OUT_DEVICE   1028
#define  IDC_OUT_RASTER   1029
#define  IDC_OUT_TT_ONLY   1030
#define  IDC_OUT_OUTLINE   1031
#define  IDC_DEFAULT_PITCH  1032
#define  IDC_FIXED_PITCH   1033
#define  IDC_VARIABLE_PITCH  1034
#define  IDC_FF_DONTCARE   1035
#define  IDC_FF_ROMAN   1036
#define  IDC_FF_SWISS   1037
#define  IDC_FF_MODERN   1038
#define  IDC_FF_SCRIPT   1039
#define  IDC_FF_DECORATIVE  1040
```

```
#define   IDC_TM_HEIGHT   1041
#define   IDC_TM_ASCENT   1042
#define   IDC_TM_DESCENT   1043
#define   IDC_TM_INTLEAD   1044
#define   IDC_TM_EXTLEAD   1045
#define   IDC_TM_AVECHAR   1046
#define   IDC_TM_MAXCHAR   1047
#define   IDC_TM_WEIGHT   1048
#define   IDC_TM_OVERHANG   1049
#define   IDC_TM_DIGASPX   1050
#define   IDC_TM_DIGASPY   1051
#define   IDC_TM_FIRSTCHAR  1052
#define   IDC_TM_LASTCHAR   1053
#define   IDC_TM_DEFCHAR   1054
#define   IDC_TM_BREAKCHAR  1055
#define   IDC_TM_ITALIC   1056
#define   IDC_TM_UNDER   1057
#define   IDC_TM_STRUCK   1058
#define   IDC_TM_VARIABLE   1059
#define   IDC_TM_VECTOR   1060
#define   IDC_TM_TRUETYPE   1061
#define   IDC_TM_DEVICE   1062
#define   IDC_TM_FAMILY   1063
#define   IDC_TM_CHARSET   1064
#define   IDC_TM_FACENAME   1065
#define   IDM_DEVICE_SCREEN  40001
#define   IDM_DEVICE_PRINTER  40002
```

图17-1显示了典型的PICKFONT屏幕显示。PICKFONT左半部分显示了一个非模态对话框，透过它，您可以选择逻辑字体结构的大部分字段。对话框的右半部分显示了字体选入设备内容后GetTextMetrics的结果。对话框的下部，程序使用这种字体显示一个字符串。因为非模态对话框非常大，所以最好在1024×768或更大的显示大小下执行这个程序。

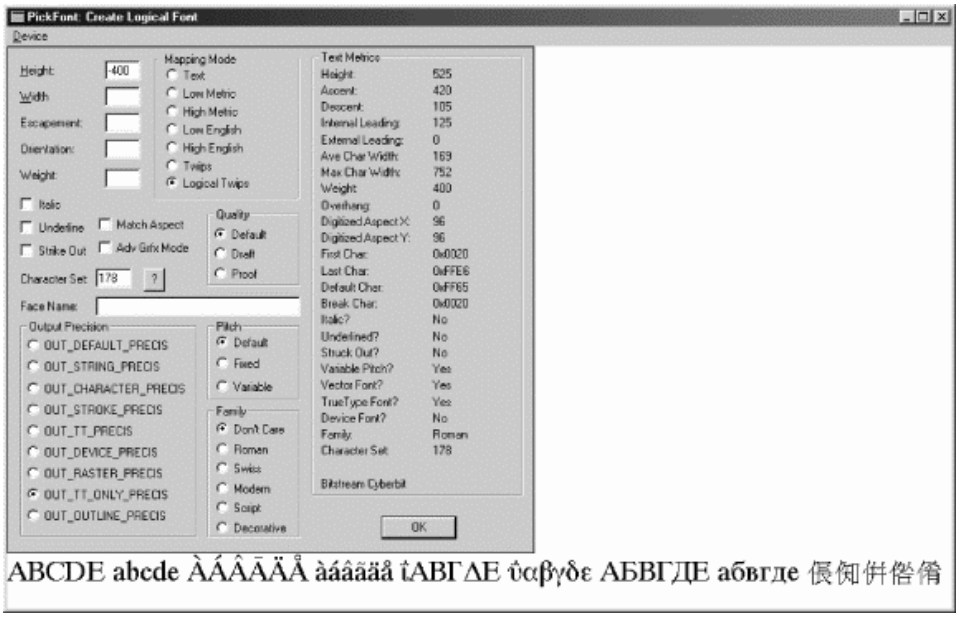


图17-1 典型的PICKFONT屏幕显示（Windows NT下的Unicode版本）

非模态对话框还包含一些非逻辑字体结构的选项，它们是包括「Logical Twips」方式的映像方式、「Match Aspect」选项（更改Windows将逻辑字体与真实字体匹配的方式）和「Adv Grtx Mode」（设定Windows NT中的高级图形模式）。稍后我将对这些作详细讨论。

从「Device」菜单中，可以选择内定打印机而不是视讯显示器。在这种情况下，PICKFONT将逻辑字体选入打印机设备内容中，并从打印机显示TEXTMETRIC结构。然后，程序将逻辑字体选入窗口设备内容中，以显示样本字符串。因此，程序显示的文字可能会使用与TEXTMETRIC字段所

描述的字体（打印机字体）不同的字体（屏幕字体）。

PICKFONT程序的大部分逻辑都在处理对话框的必要动作，因此我不会详细讨论该程序的工作方式，只解释建立和选择逻辑字体的原理。

逻辑字体结构

您可以呼叫CreateFont来建立逻辑字体，它是具有14个参数的函数。一般，定义一个LOGFONT型态的结构

LOGFONT lf ;

然后再定义该结构的字段会更容易一些。完成后，可以使用指向该结构的指针呼叫CreateFontIndirect：

hFont = CreatFontIndirect (&lf) ;

您不必设定LOGFONT结构的每个字段。如果逻辑字体结构定义为静态变量，那么所有的字段都会初始化为0，0一般是默认值。然后，可以不用更改而直接使用这个结构，CreateFontIndirect会传回字体的句柄。当您将该字体选入设备内容时，会得到一个合理的内定字体。您可以根据自己的需要，明确或模糊地填充LOGFONT结构，Windows会用一种真实字体与您的要求相匹配。

在我讨论LOGFONT结构中每个字段时，您可能想用PICKFONT程序来测试它们。当您希望程序使用您输入的任何字段时，别忘了按下Enter或「OK」按钮。

LOGFONT结构的前两个字段是逻辑单位，因此它们依赖于映像方式的目前设定：

lfHeight这是以逻辑单位表示的希望的字符高度。您可以将lfHeight设定0，以使用内定大小，或者根据字段代表的含义将其设定为正数或负数。如果将lfHeight设定为正数，就表示您希望该值表示含有内部间隔（不是外部间隔）的高度。实际上，所要求的字体行距为lfHeight。如果将lfHeight设定为负值，则Windows会将其绝对值作为与点值一致的字体高度。这是一个很重要的区别：如果想要特定点值的字体，可将点值转换为逻辑单位，并将lfHeight字段设定为该值的负数。如果lfHeight是正值，则TEXTMETRIC结构的tmHeight字段近似为该值（有时有微小的偏差，可能由于舍入误差所引起）。如果lfHeight是负值，则它粗略地与不包括tmInternalLeading字段的TEXTMETRIC结构的tmHeight字段相匹配。

lfWidth是逻辑单位的字符期望宽度。在多数情况下，可以将此值设定为0，让Windows仅根据高度选择字体。使用非零值对点阵字体并不会起太大作用，但对于TrueType字体，您能轻松地用它来获得比正常字符更宽或更窄的字体。这个字段对应于TEXTMETRIC结构的tmAveCharWidth字段。要正确使用lfWidth字段，首先把带有lfWidth字段的LOGFONT结构设定为0，建立逻辑字体，将它选入设备内容，然后呼叫GetTextMetrics。得到tmAveCharWidth字段，可按比例调节其值的大小，然后使用所调节的lfWidth的tmAveCharWidth值建立第二种字体。

下两个字段指定文字的「移位角度」和「方向」。理论上，lfEscapement使字符串能够以一定的角度书写（但每个字符的基准线仍与水平轴平行），而lfOrientation使单个字符倾斜。但是这两个字段并不是那么有效，即使现在它们只有在下面的情况下才能很好地起作用：使用TureType字体、执行Windows NT以及首先用CM_ADVANCED旗标设定呼叫SetGraphicsMode。通过选中「Adv Grfx Mode」复选框，您能够完成PICKFONT中的最终需要。

在验证PICKFONT中的这些字段时，要注意单位是十分之一度，逆时针方向旋转。它很容易输入一个值使范例字符串消失！因此，请使用0到-600或3000到3600之间的值。

lfEscapement这是从水平方向上逆时针测量的十分之几的角度。它指定在书写文字时字

符串的连续字符放置的方式。表17-1提供了几个例子：

表17-1

值	字符的放置
0	从左向右（内定）
900	向上
1800	从右向左
2700	向下

在Windows 98中，这个值设定了TrueType文字的移位角度和方向。在Windows NT中，这个值通常也是这样设定，除了用GM_ADVANCED参数呼叫SetGraphicsMode时，它按文件中说明的那样工作。

lfOrientation这是从水平方向逆时针测量的十分之几的角度，它影响单个字符的外观。表17-2提供了几个例子：

表17-2

值	字符外观
0	正常（内定）
900	向右倾斜90度
1800	颠倒
2700	向左倾斜90度

这个字段一般不起作用，除非在Windows NT下使用TrueType字体，并把图像模式设定为GM_ADVANCED，在这种情况下它按文件中说明的那样工作。

其余10个字段如下：

lfWeight这个字段使您能够指定粗体。WINGDI.H表头文件定义了可用于这个字段的一组值（参见表17-3）。

表17-3

值	标识符
0	FW_DONTCARE
100	FW_THIN
200	FW_EXTRALIGHT或FW_ULTRALIGHT
300	FW_LIGHT
400	FW_NORMAL或FW_REGULAR
500	FW_MEDIUM
600	FW_SEMIBOLD或FW_DEMIBOLD
700	FW_BOLD
800	FW_EXTRABOLD或FW_ULTRABOLD
900	FW_HEAVY或FW_BLACK

事实上，它比以前用过的任何一组值都完善。您可以对标准字使用0或400，对粗体使用700。

lfItalic在非零值时，它指定斜体。Windows能在GDI点阵字体上合成斜体。亦即，Windows仅仅移动若干行字符位图来模仿斜体。对于TrueType字体，Windows使用真正的斜体或字体的倾斜版本。

lfUnderline在非零值时，它指定底线，这项属性在GDI字体上都是用合成的。也就是说，Windows GDI只是在包括空格的每个字符底线。

lfStrikeOut在非零值时，它指定字体上应该有一条线穿过。这也是由GDI字体合成的。

lfCharSet这是指定字体字符集的一个字节的值。我会在下一节「字符集和Unicode」中更详细地讨论这个字段。在PICKFONT中，您可以按下带有问号的按钮来取得能够使用的字符集列表。

注意lfCharSet字段是唯一不用零表示默认值的字段。零值相当于ANSI_CHARSET，ANSI字符在美国和西欧使用。DEFAULT_CHARSET代码等于1，表示程序执行的机器上内定的字符集。

lfOutPrecision它指定了Windows用实际的字体匹配期望的字体大小和特征的方式。这是一个复杂的字段，一般很少使用。请查看关于LOGFONT结构的文件以得到更详细的信息。注意，可以使用OUT_TT_ONLY_PRECIS旗标来确保得到的是TrueType字体。

lfClipPrecision这个字段指定了当字符的一部分位于剪裁区以外时，剪裁字符的方式。这个字段不经常使用，PICKFONT程序也没有使用它。

lfQuality这是一个给Windows的指令，有关于期望字体与实际字体相匹配的指令。它实际只对点阵字体有意义，并不影响TrueType字体。DRAFT_QUALITY旗标指出需要GDI缩放点阵字体以得到想要的大小；PROOF_QUALITY旗标指出不需缩放。PROOF_QUALITY字体最漂亮，但它们可能比所希望的要小一些。这个字段中也可以使用DEFAULT_QUALITY（或0）。

lfPitchAndFamily这个字节由两部分组成。您可以使用位或运算符号结合用于此字段的两个标识符。最低的两位指定字体是定宽（即所有字符的宽度相等）还是变宽（参见表17-4）。

表17-4

值	标识符
0	DEFAULT_PITCH
1	FIXED_PITCH
2	VARIABLE_PITCH

字节的上半部分指定字体系列（参见表17-5）。

表17-5

值	标识符
0x00	FW_DONTCARE
0x10	FF_ROMAN（变宽，serifs）
0x20	FF_SWISS（变宽，非serifs）
0x30	FF_MODERN（定宽）
0x40	FF_SCRIPT（模仿手写）
0x50	FF_DECORATIVE

lfFaceName这是关于字样（如Courier、Arial或Times New Roman）的实际文字名称。这个字段是宽度为LF_FACESIZE（或32个字符）的字节数组。如果要得到TrueType的斜体或粗体字体，有两种方法。在lfFaceName字段中使用完整的字体名称（如Times New Roman Italic），或者可以使用基本名称（即Times New Roman），并设定lfItalic字段。

字体映像算法

在设定了逻辑字体结构后，呼叫CreateFontIndirect来得到逻辑字体句柄。当呼叫SelectObject把逻辑字体选入设备内容时，Windows寻找与所需字体最接近匹配的实际字体。它使用「字体映像算法」。结构的某些字段要比其它字段更重要一些。

了解字体映像的最好方式是花一些时间试验PICKFONT。以下是几条指南：

lfCharSet（字符集）字段是非常重要的。如果您指定了OEM_CHARSET(255)，会得到某种笔划字体或终端机字体，因为它们是唯一使用OEM字符集的字体。然而，随着TrueType「Big Fonts」的出现（在第六章〈TrueType和大字体〉一节讨论过），单一的TrueType字体能映像到包括OEM字符集等不同的字符集。您需要使用SYMBOL_CHARSET(2) 来得到Symbol字体或Wingdings字体。

lfPitchAndFamily字段的FIXED_PITCH间距值很重要，因为您实际上告诉Windows不想处理变宽字体。

lfFaceName字段很重要，因为您指定了所需字体的字样。如果让lfFaceName设定为NULL，并在lfPitchAndFamily字段中将组值设定为FF_DONTCARE以外的值，因为指定了字体系列，所以该字段也很重要。

对于点阵字体，Windows会试图配合lfHeight值，即使需要增加较小字体的大小。实际字体的高度总是小于或等于所需的字体，除非没有更小的字体满足您的要求。对于笔划或TrueType字体，Windows仅简单地将字体缩放到需要的高度。

可以通过将lfQuality设定为PROOF_QUALITY来防止Windows缩放点阵字体。这么做可以告诉Windows所需的字体高度没有字体外观重要。

如果指明了对于显示器的特定纵横比不协调的lfHeight和lfWeight值，Windows能映射到为显示器或其它不同纵横比的设备设计的点阵字体。这是得到细或粗字体的技巧（当然，对于TrueType字体是不必要的）。一般而言，您可能想避免为另一种设备挑配字体。您可以通过单击标有「Match Aspect」的复选框，在PICKFONT中完成。如果选中了复选框，PICKFONT会使用TRUE参数呼叫SetMapperFlags。

取得字体信息

在PICKFONT中非模态对话框的右侧是字体选入设备内容后从GetTextMetrics函数中获得的信息（注意，可以使用PICKFONT的「Device」菜单指出设备内容是屏幕还是内定打印机。因为在打印机上有效的字体可能不同，所以结果也可能不同）。在PICKFONT中列表的底部是从GetTextFace得到的有效字体名称。

除了数值化的纵横比以外，Windows复制到TEXTMETRIC结构的所有大小值都以逻辑单位表示。TEXTMETRIC结构的字段如下：

tmHeight逻辑单位的字符高度。它近似等于LOGFONT结构中指定的lfHeight字段的值，如果该值为正，它就代表行距，而非点值。如果LOGFONT结构的lfHeight字段为负，则tmHeight字段减tmInternalLeading字段应近似等于lfHeight字段的绝对值。

tmAscent逻辑单位的基准线以上的字符垂直大小。

tmDescent逻辑单位的基准线以下的字符垂直大小。

tmInternalLeading包含在tmHeight值内的垂直大小，通常被一些大写字母上注音符号占据。同样，可以用tmHeight值减tmInternalLeading值来计算字体的点值。

tmExternalLeading **tmHeight**以外的行距附加量，字体的设计者推荐用于隔开文字的连续行。

tmAveCharWidth字体中小写字母的平均宽度。

tmMaxCharWidth逻辑单位的字符最大宽度。对于定宽字体，这个值与tmAveCharWidth相同。

tmWeight字体重量，范围从0到999。实际上，这个字段为400时是标准字体，700时是粗体。

tmOverhangWindows在合成斜体或粗体时添加到点阵字体字符的额外宽度量（逻辑单

位)。当点阵字体斜体化时，tmAveCharWidth值保持不变，因为斜体化的字符串与相同的正常字符串的总宽度相等。要为字体加粗，Windows必须稍微增加每个字符的宽度。对于粗体，tmAveCharWidth值小于tmOverhang值，等于没有加粗的相同字体的tmAveCharWidth值。

tmDigitizedAspectX 和 tmDigitizedAspectY 字体合适的纵横比。它们与使用LOGPIXELSX和LOGPIXELSY标识符从GetDeviceCaps得到的值相同。

tmFirstChar字体中第一个字符的字符代码。

tmLastChar 字体中最后一个字符的字符代码。如果TEXTMETRIC结构通过呼叫GetTextMetricsW（函数的宽字符版本）获得，那么这个值可能大于255。

tmDefaultCharWindows用于显示不在字体中的字符的字符代码，通常是矩形。

tmBreakChar在调整文字时，Windows和您的程序用于确定单字断开的字符。如果您不用一些奇怪的东西（例如EBCDIC字体），它就是32 – 空格符。

tmItalic对于斜体字为非零值。

tmUnderlined对于底线字体为非零值。

tmStruckOut对于删除线字体为非零值。

tmPitchAndFamily低四位是表示字体某些特征的旗标，由在WINGDI.H中定义的标识符指出（参见表17-6）。

表17-6

值	标识符
0x01	TMPF_FIXED_PITCH
0x02	TMPF_VECTOR
0x04	TMPF_TRUETYPE
0x08	TMPF_DEVICE

不管TMPF_FIXED_PITCH旗标的名称是什么，如果字体字符是变宽的，则最低位为1。第二最低位（TMPF_VECTOR）对于TrueType字体和使用其它可缩放的轮廓技术的字体（如PostScript的字体）为1。TMPF_DEVICE旗标表示设备字体（即打印机内置的字体），而不是依据GDI的字体。

这个字段的第四高的位表示字体系列，并且与LOGFONT的lfPitchAndFamily字段中所用的值相同。

tmCharSet字符集标识符。

字符集和Unicode

我在第六章讨论了Windows字符集的概念，在那里我们必须处理涉及键盘的国际化问题。在LOGFONT和TEXTMETRIC结构中，所需字体（或实际字体）的字符集由0至255之间的单个字节的数值表示。定义在WINGDI.H中的字符集标识符如下所示：

```
#define  ANSI_CHARSET    0
#define  DEFAULT_CHARSET   1
#define  SYMBOL_CHARSET   2
#define  MAC_CHARSET    77
#define  SHIFTJIS_CHARSET   128
#define  HANGEUL_CHARSET   129
#define  HANGUL_CHARSET   129
#define  JOHAB_CHARSET   130
#define  GB2312_CHARSET   134
#define  CHINESEBIG5_CHARSET   136
#define  GREEK_CHARSET   161
#define  TURKISH_CHARSET   162
#define  VIETNAMESE_CHARSET   163
#define  HEBREW_CHARSET   177
```

```
#define ARABIC_CHARSET    178
#define BALTIC_CHARSET    186
#define RUSSIAN_CHARSET    204
#define THAI_CHARSET    222
#define EASTEUROPE_CHARSET    238
#define OEM_CHARSET    255
```

字符集与页码表的概念类似，但是字符集特定于Windows，且通常小于或等于255。

与本书的所有程序一样，您可以带有定义的UNICODE标识符编译PICKFONT，也可以不带UNICODE标识符编译它。和往常一样，本书内附光盘上的程序的两个版本分别位于DEBUG和RELEASE目录中。

注意，在程序的Unicode版本中PICKFONT在其窗口底部显示的字符串要更长一些。在两个版本中，字符串的字符代码由0x40到0x45、0x60到0x65。不管您选择了哪种字符集（除了SYMBOL_CHARSET),这些字符代码都显示拉丁字母表的前五个大写和小写字母(即A到E和a到e)。

当执行PICKFONT程序的非Unicode版本时，接下来的12个字符 – 字符代码0xC0到0xC5以及0xE0到0xE5 – 将依赖于所选择的字符集。对于ANSI_CHARSET，这个字符代码对应于大写和小写字母A的加重音版本。对于GREEK_CHARSET，这些代码对应于希腊字母表的字母。对于RUSSIAN_CHARSET，对应于斯拉夫字母表的字母。注意，当您选择一种字符集时，字体可能会改变，这是因为点阵字体可能没有这些字符，但TrueType字体可能有。您可能回忆起大多数TrueType字体是「Big fonts」并且包含几种不同字符集的字母。如果您执行Windows的远东版本，这些字符会被解释为双字节字符，并且会按方块字显示，而不是按字母显示。

在Windows NT下执行PICKFONT的Unicode版本时，代码0xC0到0xC5以及0xE0到0xE5通常是大写和小写字母A的加重音版本（除了SYMBOL_CHARSET)，因为Unicode中定义了这些代码。程序也显示0x0390到0x0395以及0x03B0到0x03B5的字符代码。由于它们在Unicode中有定义，这些代码总是对应于希腊字母表的字母。同样地，程序显示0x0410到0x0415以及0x0430到0x0435的字符代码，它们对应于斯拉夫字母表的字母。然而，这些字符不可能存在于内定字体中，您必须选择GREEK_CHARSET或RUSSIAN_CHARSET来得到它们。在这种情况下，LOGFONT结构中的字符集ID不更改实际的字符集；字符集总是Unicode。而字符集ID指出来自所需字符集的字符。

现在选择HEBREW_CHARSET（代码177)。希伯来字母表不包括在Windows通常的Big Fonts中，因此操作系统选择Lucida Sans Unicode，这一点您可以在非模态对话框的右下角中验证。

PICKFONT也显示0x5000到0x5004的字符代码，它们对应于汉语、日语和朝鲜语象形文字的一部分。如果您执行Windows的远东版本，或者下载了比Lucida Sans Unicode范围更广的免费Unicode字体，就可以看到这些。Bitstream CyberBit字体就是这样的一种字体，您可以从http://www.bitstream.com/products/world/cyberbits中找到。(Lucida Sans Unicode大约有300K，而Bitstream CyberBit大约有13M)。如果您安装了这种字体，当需要一种Lucida Sans Unicode不支持的字体时，Windows会选择它，这些字体如：SHIFTJIS_CHARSET（日语)、HANGUL_CHARSET（朝鲜语)、JOHAB_CHARSET（朝鲜语)、GB2312_CHARSET（简体中文）或CHINESEBIG5_CHARSET （繁体中文)。

本章的后面有一个程序可让您查看Unicode字体的所有字母。

EZFONT系统

TrueType字体系统（以传统的排版为基础）为Windows以不同的方式显示文字提供了牢固的基础。但是一些Windows的字体选择函数依据较旧技术，使得画面上的点阵字体必须趋近打印机设备字体的样子。下一节将讲到列举字体的做法，它能够使程序获得显示器或打印机上全部有效字体的列表。不过，「ChooseFont」对话框（稍后讨论）确实大幅度消除了程序行举字体的必要性。

因为标准TrueType字体可以在任何系统上使用，且这些字体可以用于显示器以及打印机，如此一来，程序在选择TrueType字体或在缺乏信息的情况下取得某种相似的字体时，就没有必要列举字体了。程序只需简单并明确地选择系统中存在的TrueType字体（当然，除非使用者故意删除它们）。这种方法与指定字体名称（可能是第十七章中〈TrueType字体〉一节中列出的13种字体中的一种）和字体大小一样简单。我把这种方法称做EZFONT（「简便字体」），程序17-2列出了它的两个文件。

程序17-2 EZFONT

EZFONT.H

```
/*------------------------------------------------------------------------
EZFONT.H header file
------------------------------------------------------------------------*/
HFONT EzCreateFont (HDC hdc, TCHAR * szFaceName, int iDeciPtHeight,
                    int iDeciPtWidth, int iAttributes, BOOL fLogRes) ;

#define  EZ_ATTR_BOLD      1
#define  EZ_ATTR_ITALIC    2
#define  EZ_ATTR_UNDERLINE     4
#define  EZ_ATTR_STRIKEOUT     8

   EZFONT.C
/*------------------------------------------------------------------------
EZFONT.C -- Easy Font Creation
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#include "ezfont.h"

HFONT EzCreateFont (HDC hdc, TCHAR * szFaceName, int iDeciPtHeight,
                    int iDeciPtWidth, int iAttributes, BOOL fLogRes)
{
  FLOAT cxDpi, cyDpi ;
  HFONT hFont ;
  LOGFONT lf ;
  POINT pt ;
  TEXTMETRIC tm ;

  SaveDC (hdc) ;
  SetGraphicsMode (hdc, GM_ADVANCED) ;
  ModifyWorldTransform (hdc, NULL, MWT_IDENTITY) ;
  SetViewportOrgEx (hdc, 0, 0, NULL) ;
  SetWindowOrgEx (hdc, 0, 0, NULL) ;

  if (fLogRes)
  {
    cxDpi = (FLOAT) GetDeviceCaps (hdc, LOGPIXELSX) ;
    cyDpi = (FLOAT) GetDeviceCaps (hdc, LOGPIXELSY) ;
  }
  else
  {
    cxDpi = (FLOAT) (25.4 * GetDeviceCaps (hdc, HORZRES) /
      GetDeviceCaps (hdc, HORZSIZE)) ;
    cyDpi = (FLOAT) (25.4 * GetDeviceCaps (hdc, VERTRES) /
      GetDeviceCaps (hdc, VERTSIZE)) ;
  }

  pt.x = (int) (iDeciPtWidth * cxDpi / 72) ;
  pt.y = (int) (iDeciPtHeight * cyDpi / 72) ;

  DPtoLP (hdc, &pt, 1) ;
  lf.lfHeight = - (int) (fabs (pt.y) / 10.0 + 0.5) ;
  lf.lfWidth = 0 ;
  lf.lfEscapement = 0 ;
  lf.lfOrientation = 0 ;
  lf.lfWeight = iAttributes & EZ_ATTR_BOLD ? 700:0 ;
```

```
  lf.lfItalic = iAttributes & EZ_ATTR_ITALIC ? 1 :0 ;
  lf.lfUnderline = iAttributes & EZ_ATTR_UNDERLINE ?1 :0 ;
  lf.lfStrikeOut = iAttributes & EZ_ATTR_STRIKEOUT ? 1 :0 ;
  lf.lfCharSet = DEFAULT_CHARSET ;
  lf.lfOutPrecision = 0 ;
  lf.lfClipPrecision = 0 ;
  lf.lfQuality = 0 ;
  lf.lfPitchAndFamily = 0 ;

  lstrcpy (lf.lfFaceName, szFaceName) ;

  hFont = CreateFontIndirect (&lf) ;

  if (iDeciPtWidth != 0)
  {
    hFont = (HFONT) SelectObject (hdc, hFont) ;
    GetTextMetrics (hdc, &tm) ;
    DeleteObject (SelectObject (hdc, hFont)) ;
    lf.lfWidth = (int) (tm.tmAveCharWidth *
      fabs (pt.x) / fabs (pt.y) + 0.5) ;
    hFont = CreateFontIndirect (&lf) ;
  }

  RestoreDC (hdc, -1) ;
  return hFont ;
}
```

EZFONT.C只有一个函数，称为EzCreateFont，如下所示：

```
hFont = EzCreateFont ( hdc, szFaceName, iDeciPtHeight, iDeciPtWidth,
                  iAttributes, fLogRes) ;
```

函数传回字体句柄。可通过呼叫SelectObject将该字体选入设备内容，然后呼叫GetTextMetrics或GetOutlineTextMetrics以确定字体尺寸在逻辑坐标中的实际大小。在程序终止前，应该呼叫DeleteObject删除任何建立的字体。

szFaceName参数可以是任何TrueType字体名称。您选择的字体越接近标准字体，则该字体在系统中存在的机率就越大。

第三个参数指出所需的点值，但是它的单位是十分之一点。因而，如果所需要的点值为十二又二分之一，则值应为125。

第四个参数通常应设定为零或与第三个参数相同。然而，通过将此字段设定为不同值可以建立更宽或更窄的TrueType字体。它以点为单位描述了字体的宽度，有时称之为字体的「全宽（em-width）」。不要将它与字体字符的平均宽度或其它类似的东西相混淆。在过去的排版技术中，大写字母M的宽度与高度是相等的。于是，「完全正方形（em-square）」的概念产生了，这是全宽测量的起源。当字体的全宽等于字体的全高（字体的点值）时，字符宽度是字体设计者设定的宽度。宽或窄的全宽值可以产生更细或更宽的字符。

您可以将iAttributes参数设定为以下定义在EZFONT.H中的值：

EZ_ATTR_BOLD

EZ_ATTR_ITALIC

EZ_ATTR_UNDERLINE

EZ_ATTR_STRIKEOUT

可以使用EZ_ATTR_BOLD或EZ_ATTR_ITALIC或者将样式作为完整TrueType字体名称的一部分。

最后，我们将参数fLogRes设定为逻辑值TRUE，以表示字体点值与设备的「逻辑分辨率」相吻合，其中「逻辑分辨率」是GetDeviceCaps函数使用LOGPIXELSX和LOGPIXELSY参数的传回值。

另外，依据分辨率的字体大小是从HORZRES、HORZSIZE、VERTRES和VERTSIZE计算出来的。这仅对于Windows NT下的视讯显示器才有所不同。

EzCreateFont函数开始只进行一些用于Windows NT的调整。即呼叫SetGraphicsMode和ModifyWorldTransform函数，它们在Windows 98下不起作用。因为Windows NT的全球转换应该有修改字体可视大小的作用，因此在计算字体大小之前，全球转换设定为默认值－无转换。

EzCreateFont基本上设定LOGFONT结构的字段并呼叫CreateFontIndirect，CreateFontIndirect传回字体的句柄。EzCreateFont函数的主要任务是将字体的点值转换为LOGFONT结构的lfHeight字段所要求的逻辑单位。其实是首先将点值转换为设备单位（像素），然后再转换为逻辑单位。为完成第一步，函数使用GetDeviceCaps。从像素到逻辑单位的转换似乎只需简单地呼叫DPtoLP（「从设备点到逻辑点」）函数。但是为了使DPtoLP转换正常工作，在以后使用建立的字体显示文字时，相同的映像方式必须有效。这就意味着应该在呼叫EzCreateFont函数前设定映像方式。在大多数情况下，只使用一种映像方式在窗口的特定区域绘制，因此这种要求不是什么问题。

程序17-3所示的EZTEST程序不很严格地考验了EZFONT文件。此程序使用上面的EZTEST文件，还包括了本书后面程序要使用的FONTDEMO文件。

程序17-3 EZTEST

EZTEST.C

```
/*-----------------------------------------------------------------------------
EZTEST.C -- Test of EZFONT
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include "ezfont.h"

TCHAR szAppName [] = TEXT ("EZTest") ;
TCHAR szTitle [] = TEXT ("EZTest: Test of EZFONT") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  HFONT hFont ;
  int y, iPointSize ;
  LOGFONT lf ;
  TCHAR szBuffer [100] ;
  TEXTMETRIC tm ;

  // Set Logical Twips mapping mode

  SetMapMode (hdc, MM_ANISOTROPIC) ;
  SetWindowExtEx (hdc, 1440, 1440, NULL) ;
  SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
    GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;

  // Try some fonts

  y = 0 ;
  for (iPointSize = 80 ; iPointSize <= 120 ; iPointSize++)
  {
    hFont = EzCreateFont (hdc, TEXT ("Times New Roman"),
      iPointSize, 0, 0, TRUE) ;

    GetObject (hFont, sizeof (LOGFONT), &lf) ;

    SelectObject (hdc, hFont) ;
    GetTextMetrics (hdc, &tm) ;
    TextOut (hdc, 0, y, szBuffer,
      wsprintf ( szBuffer,
      TEXT ("Times New Roman font of %i.%i points, ")
```

```
     TEXT ("lf.lfHeight = %i, tm.tmHeight = %i"),
     iPointSize / 10, iPointSize % 10,
     lf.lfHeight, tm.tmHeight)) ;

    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    y += tm.tmHeight ;
  }
}
```

FONTDEMO.C

```
/*-----------------------------------------------------------------------------
FONTDEMO.C -- Font Demonstration Shell Program
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\EZTest\\EzFont.h"
#include "..\\EZTest\\resource.h"

extern void PaintRoutine (HWND, HDC, int, int) ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

HINSTANCE hInst ;

extern TCHAR szAppName [] ;
extern TCHAR szTitle [] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  TCHAR szResource [] = TEXT ("FontDemo") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szResource ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
```

```
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("Font Demo: Printing") } ;
  static int cxClient, cyClient ;
  static PRINTDLG pd = { sizeof (PRINTDLG) } ;
  BOOL fSuccess ;
  HDC hdc, hdcPrn ;
  int cxPage, cyPage ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_COMMAND:
    switch (wParam)
    {
    case IDM_PRINT:

      // Get printer DC
      pd.hwndOwner = hwnd ;
      pd.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

      if (! PrintDlg (&pd))
        return 0 ;

      if (NULL == (hdcPrn = pd.hDC))
      {
        MessageBox( hwnd, TEXT ("Cannot obtain Printer DC"),
          szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;
      }
      // Get size of printable area of page
      cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
      cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

      fSuccess = FALSE ;
      // Do the printer page
      SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
      ShowCursor (TRUE) ;

      if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
      {
        PaintRoutine (hwnd, hdcPrn, cxPage, cyPage) ;

        if (EndPage (hdcPrn) > 0)
        {
          fSuccess = TRUE ;
          EndDoc (hdcPrn) ;
        }
      }
      DeleteDC (hdcPrn) ;

      ShowCursor (FALSE) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

      if (!fSuccess)
        MessageBox (hwnd,
        TEXT ("Error encountered during printing"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;
    case IDM_ABOUT:
      MessageBox ( hwnd, TEXT ("Font Demonstration Program\n")
        TEXT ("(c) Charles Petzold, 1998"),
        szAppName, MB_ICONINFORMATION | MB_OK) ;
      return 0 ;
    }
    break ;
```

```
   case WM_SIZE:
     cxClient = LOWORD (lParam) ;
     cyClient = HIWORD (lParam) ;
     return 0 ;
   case WM_PAINT:
     hdc = BeginPaint (hwnd, &ps) ;
     PaintRoutine (hwnd, hdc, cxClient, cyClient) ;

     EndPaint (hwnd, &ps) ;
     return 0 ;
   case WM_DESTROY :
     PostQuitMessage (0) ;
     return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

FONTDEMO.RC

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
FONTDEMO MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Print...",   IDM_PRINT
END
POPUP "&Help"
BEGIN
MENUITEM "&About...",   IDM_ABOUT
END
END
```

RESOURCE.H

```
// Microsoft Developer Studio generated include file.
// Used by FontDemo.rc
#define IDM_PRINT  40001
#define IDM_ABOUT  40002
```

EZTEST.C中的PaintRoutine函数将映像方式设定为Logical Twips，然后建立字体范围从8点到12点（间隔为0.1点）的Times New Roman字体。第一次执行此程序时，它的输出可能会使您困惑。许多行文字使用大小明显相同的字体，并且TEXTMETRIC函数也报告这些字体具有相同的高度。这一切都是点阵处理的结果。显示器上的像素是不连续的，它不能显示每一个可能的字体大小。但是，FONTDEMO外壳程序使打印输出的字体是不同的。这里您会发现字体大小区分得更加精确。

字体的旋转

您在PICKFONT中可能已经实验过了，LOGFONT结构的lfOrientation和lfEscapement字段可以旋转TrueType文字。如果仔细考虑一下，这对GDI不会造成多大困难，因为围绕原点旋转坐标点的公式是公开的。

虽然EzCreateFont不能指定字体的旋转角度，但是如FONTROT（「字体旋转」）程序展示的那样，在呼叫函数后，进行调整是非常容易的。程序17-4显示了FONTROT.C文件，该程序也需要上面显示的EZFONT文件和FONTDEMO文件。

程序17-4 FONTROT

FONTROT.C

```
/*------------------------------------------------------------------------
FONTROT.C -- Rotated Fonts
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\eztest\\ezfont.h"
TCHAR szAppName [] = TEXT ("FontRot") ;
TCHAR szTitle [] = TEXT ("FontRot: Rotated Fonts") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  static TCHAR szString [] = TEXT (" Rotation") ;
  HFONT hFont ;
  int i ;
  LOGFONT lf ;
  hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 540, 0, 0, TRUE) ;
  GetObject (hFont, sizeof (LOGFONT), &lf) ;
  DeleteObject (hFont) ;

  SetBkMode (hdc, TRANSPARENT) ;
  SetTextAlign (hdc, TA_BASELINE) ;
  SetViewportOrgEx (hdc, cxArea / 2, cyArea / 2, NULL) ;

  for (i = 0 ; i < 12 ; i ++)
  {
    lf.lfEscapement = lf.lfOrientation = i * 300 ;
    SelectObject (hdc, CreateFontIndirect (&lf)) ;

    TextOut (hdc, 0, 0, szString, lstrlen (szString)) ;
    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
  }
}
```

FONTROT呼叫EzCreateFont只是为了获得与54点Times New Roman字体相关的LOGFONT结构。然后，程序删除该字体。在for循环中，对于每隔30度的角度，建立新字体并显示文字。结果如图17-2所示。

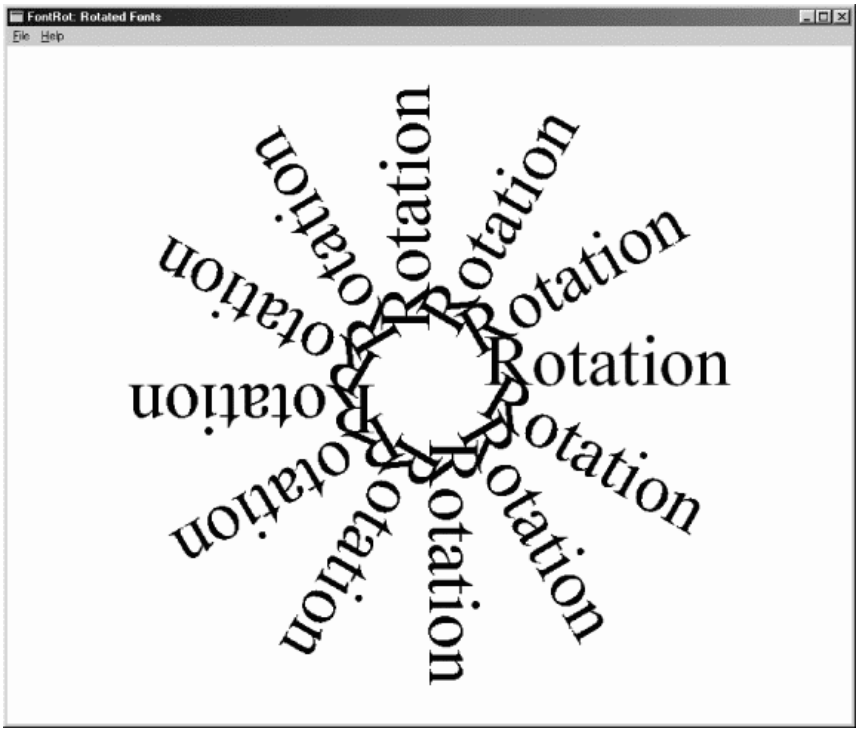


图17-2 FONTROT的屏幕显示

如果您对图形旋转和其它线性转换的更专业方法感兴趣，并且知道您的程序在Windows NT下执行将受到限制，您可以使用XFORM矩阵和坐标转换函数数。

字体列举

字体列举是从GDI中取得设备的全部有效字体列表的程序。程序可以选择其中一种字体，或将它们显示在对话框中供使用者选择。我先简单地介绍一下列举函数，然后显示使用ChooseFont函数的方法，ChooseFont降低了应用程序中进行字体列举的必要性。

列举函数

在Windows的早期，字体列举需要使用EnumFonts函数：

EnumFonts (hdc, szTypeFace, EnumProc, pData) ;

程序可以列举所有的字体（将第二个参数设定为NULL）或只列出特定的字样。第三个参数是列举callback函数；第四个参数是传递给该函数的可选数据。GDI为系统中的每种字体呼叫callback函数，将定义字体的LOGFONT和TEXTMETRIC结构以及一些表示字体型态的旗标传递给它。

EnumFontFamilies函数是Windows 3.1下列举TrueType字体的函数：

EnumFontFamilies (hdc, szFaceName, EnumProc, pData) ;

通常第一次呼叫EnumFontFamilies时，第二个参数设定为NULL。为每个字体系列（例如Times New Roman）呼叫一次EnumProccallback函数。然后，应用程序使用该字体名称和不同的callback函数再次呼叫EnumFontFamilies。GDI为字体系列中的每种字体（例如Times New Roman Italic）呼叫第二个callback函数。对于非TrueType字体，向callback函数传递ENUMLOGFONT结构（它是由LOGFONT结构加上「全名」字段和「型态」字段构成，「型态」字段如文字名称「Italic」或「Bold」）和TEXTMETRIC结构，对于TrueType字体传递NEWTEXTMETRIC结构。NEWTEXTMETRIC结构相对于TEXTMETRIC结构中的信息添加了四个字段。

EnumFontFamiliesEx函数被推荐在Windows的32位的版本下使用：

EnumFontFamiliesEx (hdc, &logfont, EnumProc, pData, dwFlags) ;

第二个参数是指向LOGFONT结构的指针，其中lfCharSet和lfFaceName字段指出了所要列举的字体信息。Callback函数在ENUMLOGFONTEX和NEWTEXTMETRICEX结构中得到每种字体的信息。

「ChooseFont」对话框

在第十一章稍微介绍了ChooseFont的通用对话框。现在，我们讨论字体列举，需要详细了解一下ChooseFont函数的内部工作原理。ChooseFont函数得到指向CHOOSEFONT结构的指针以此作为它的唯一参数，并显示列出所有字体的对话框。利用从ChooseFont中的传回值，LOGFONT结构（CHOOSEFONT结构的一部分）能够建立逻辑字体。

程序17-5所示的CHOSFONT程序展示了使用ChooseFont函数的方法，并显示了函数定义的LOGFONT结构的字段。程序也显示了在PICKFONT中显示的相同字符串。

程序17-5 CHOSFONT

CHOSFONT.C

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

```
CHOSFONT.C -- ChooseFont Demo
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("ChosFont") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("ChooseFont"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static CHOOSEFONT cf ;
  static int cyChar ;
  static LOGFONT if ;
  static TCHAR szText[] = TEXT ("\x41\x42\x43\x44\x45 ")
    TEXT ("\x61\x62\x63\x64\x65 ")
    TEXT ("\xC0\xC1\xC2\xC3\xC4\xC5 ")
    TEXT ("\xE0\xE1\xE2\xE3\xE4\xE5 ")
#ifdef UNICODE
    TEXT ("\x0390\x0391\x0392\x0393\x0394\x0395 ")
    TEXT ("\x03B0\x03B1\x03B2\x03B3\x03B4\x03B5 ")
    TEXT ("\x0410\x0411\x0412\x0413\x0414\x0415 ")
    TEXT ("\x0430\x0431\x0432\x0433\x0434\x0435 ")
    TEXT ("\x5000\x5001\x5002\x5003\x5004")
#endif
    ;
  HDC hdc ;
  int y ;
  PAINTSTRUCT ps ;
```

```
TCHAR szBuffer [64] ;
TEXTMETRIC tm ;

switch (message)
{
case WM_CREATE:
  // Get text height
  cyChar = HIWORD (GetDialogBaseUnits ()) ;
  // Initialize the LOGFONT structure
  GetObject (GetStockObject (SYSTEM_FONT), sizeof (lf), &lf) ;
  // Initialize the CHOOSEFONT structure
  cf.lStructSize = sizeof (CHOOSEFONT) ;
  cf.hwndOwner = hwnd ;
  cf.hDC = NULL ;
  cf.lpLogFont = &lf ;
  cf.iPointSize = 0 ;
  cf.Flags = CF_INITTOLOGFONTSTRUCT |
    CF_SCREENFONTS | CF_EFFECTS ;
  cf.rgbColors = 0 ;
  cf.lCustData = 0 ;
  cf.lpfnHook = NULL ;
  cf.lpTemplateName = NULL ;
  cf.hInstance = NULL ;
  cf.lpszStyle = NULL ;
  cf.nFontType = 0 ;
  cf.nSizeMin = 0 ;
  cf.nSizeMax = 0 ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDM_FONT:
    if (ChooseFont (&cf))
      InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;
  }
  return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    // Display sample text using selected font

    SelectObject (hdc, CreateFontIndirect (&lf)) ;
    GetTextMetrics (hdc, &tm) ;
    SetTextColor (hdc, cf.rgbColors) ;
    TextOut (hdc, 0, y = tm.tmExternalLeading, szText, lstrlen (szText)) ;

    // Display LOGFONT structure fields using system font

    DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
    SetTextColor (hdc, 0) ;

    TextOut (hdc, 0, y += tm.tmHeight, szBuffer,
      wsprintf (szBuffer, TEXT ("lfHeight = %i"), lf.lfHeight)) ;

    TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("lfWidth = %i"), lf.lfWidth)) ;

    TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf ( szBuffer, TEXT ("lfEscapement = %i"),
      lf.lfEscapement)) ;

    TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf ( szBuffer, TEXT ("lfOrientation = %i"),
      lf.lfOrientation)) ;

    TextOut (hdc, 0, y += cyChar, szBuffer,
```

```
      wsprintf (szBuffer, TEXT ("lfWeight = %i"),lf.lfWeight)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("lfItalic = %i"),lf.lfItalic)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("lfUnderline = %i"),lf.lfUnderline)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("lfStrikeOut = %i"),lf.lfStrikeOut)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("lfCharSet = %i"),lf.lfCharSet)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf ( szBuffer, TEXT ("lfOutPrecision = %i"),
      lf.lfOutPrecision)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("lfClipPrecision = %i"),
      lf.lfClipPrecision)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf ( szBuffer, TEXT ("lfQuality = %i"),lf.lfQuality)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf ( szBuffer, TEXT ("lfPitchAndFamily = 0x%02X"),
      lf.lfPitchAndFamily)) ;

     TextOut (hdc, 0, y += cyChar, szBuffer,
      wsprintf ( szBuffer, TEXT ("lfFaceName = %s"),lf.lfFaceName)) ;

     EndPaint (hwnd, &ps) ;
     return 0 ;
   case WM_DESTROY:
     PostQuitMessage (0) ;
     return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

CHOSFONT.RC

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Menu
CHOSFONT MENU DISCARDABLE
BEGIN
MENUITEM "&Font!",    IDM_FONT
END
```

RESOURCE.H

```
// Microsoft Developer Studio generated include file.
// Used by ChosFont.rc
#define IDM_FONT    40001
```

与一般的对话框一样，CHOOSEFONT结构的Flags字段列出了许多选项。CHOSFONT指定的CF_INITLOGFONTSTRUCT旗标使Windows根据传递给ChooseFont结构的LOGFONT结构对对话框的选择进行初始化。您可以使用旗标来指定只要列出TrueType字体（CF_TTONLY）或只要列出定宽字体（CF_FIXEDPITCHONLY）或无符号字体（CF_SCRIPTSONLY）。也可以显示屏幕字体（CF_SCREENFONTS）、打印字体（CF_PRINTERFONTS）或者两者都显示（CF_BOTH）。在后两

种情况下，CHOOSEFONT结构的hDC字段必须是打印机设备内容。CHOSFONT程序使用CF_SCREENFONTS旗标。

CF_EFFECTS旗标（CHOSFONT程序使用的第三个旗标）强迫对话框包括用于底线和删除线的复选框并且允许选择文字的颜色。在程序代码中变换文字颜色不难，您可以试一试。

注意「Font」对话框中由ChooseFont显示的「Script」字段。它让使用者选择用于特殊字体的字符集，适当的字符集ID在LOGFONT结构中传回。

ChooseFont函数使用逻辑英…即拥阗抵屑扑鈷fHeight字段。例如，假定您从「显示属性」对话框中安装了「小字体」。这意味着带有视讯显示装置内容的GetDeviceCaps和参数LOGPIXELSY传回96。如果使用ChooseFont选择72点的Times Roman字体，实际上是想要1英寸高的字体。当ChooseFont传回后，LOGFONT结构的lfHeight字段等于-96（注意负号），这是指字体的点值等于96像素，或者1逻辑英寸。

以上大概是我们想要知道的。但请记住以下几点:

如果在Windows NT下设定了度量映像方式，则逻辑坐标与字体的实际大小不一致。例如，如果在依据度量映像方式的文字旁画一把尺，会发现它与字体不搭调。应该使用上面描述的Logical Twips映射方式来绘制图形，才能与字体大小一致。

如果要使用任何非MM_TEXT映像方式，请确保在把字体选入设备内容和显示文字时，没有设定映像方式。否则， GDI会认为LOGFONT结构的lfHeight字段是逻辑坐标。

由ChooseFont设定的LOGFONT结构的lfHeight字段总是像素值，并且它只适用于视频显示器。当您为打印机设备内容建立字体时，必须调整lfHeight值。ChooseFont函数使用CHOOSEFONT结构的hDC字段只为获得列在对话框中的打印机字体。此设备内容句柄不影响lfHeight值。

幸运的是，CHOOSEFONT结构包括一个iPointSize字段，它提供以十分之一点为单位的所选字体的大小。无论是什么设备内容和映像方式，都能把这个字段转化为逻辑大小并用于lfHeight字段。在EZFONT.C中能找到合适的程序代码，您可以根据需要简化它。

另一个使用ChooseFont的程序是UNICHARS，这个程序让您查看一种字体的所有字符，对于研究Lucida Sans Unicode字体（内定的显示字体）或Bitstream CyberBit字体尤其有用。UNICHARS总是使用TextOutW函数来显示字体的字符，因此可以在Windows NT或Windows 98下执行它。

程序17-6 UNICHARS

UNICHARS.C

```
/*------------------------------------------------------------------------
UNICHARS.C -- Displays 16-bit character codes
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("UniChars") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requies Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Unicode Characters"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static CHOOSEFONT cf ;
  static int iPage ;
  static LOGFONT lf ;
  HDC hdc ;
  int cxChar, cyChar, x, y, i, cxLabels ;
  PAINTSTRUCT ps ;
  SIZE size ;
  TCHAR szBuffer [8] ;
  TEXTMETRIC tm ;
  WCHAR ch ;

  switch (message)
  {
  case WM_CREATE:
    hdc = GetDC (hwnd) ;
    lf.lfHeight = - GetDeviceCaps (hdc, LOGPIXELSY) / 6 ; // 12 points
    lstrcpy (lf.lfFaceName, TEXT ("Lucida Sans Unicode")) ;
    ReleaseDC (hwnd, hdc) ;

    cf.lStructSize = sizeof (CHOOSEFONT) ;
    cf.hwndOwner = hwnd ;
    cf.lpLogFont = &lf ;
    cf.Flags = CF_INITTOLOGFONTSTRUCT | CF_SCREENFONTS ;

    SetScrollRange (hwnd, SB_VERT, 0, 255, FALSE) ;
    SetScrollPos (hwnd, SB_VERT, iPage, TRUE ) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FONT:
      if ( ChooseFont (&cf))
        InvalidateRect (hwnd, NULL, TRUE) ;
```

```
      return 0 ;
    }
    return 0 ;
    case WM_VSCROLL:
      switch (LOWORD (wParam))
      {
      case SB_LINEUP: iPage -= 1 ; break ;
      case SB_LINEDOWN: iPage += 1 ; break ;
      case SB_PAGEUP: iPage -= 16 ; break ;
      case SB_PAGEDOWN: iPage += 16 ; break ;
      case SB_THUMBPOSITION:iPage= HIWORD (wParam); break ;

      default:
        return 0 ;
      }

      iPage = max (0, min (iPage, 255)) ;

      SetScrollPos (hwnd, SB_VERT, iPage, TRUE) ;
      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;

      case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;

        SelectObject (hdc, CreateFontIndirect (&lf)) ;

        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmMaxCharWidth ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;

        cxLabels = 0 ;

        for (i = 0 ; i < 16 ; i++)
        {
          wsprintf (szBuffer, TEXT (" 000%1X: "), i) ;
          GetTextExtentPoint (hdc, szBuffer, 7, &size) ;

          cxLabels = max (cxLabels, size.cx) ;
        }

        for (y = 0 ; y < 16 ; y++)
        {
          wsprintf (szBuffer, TEXT (" %03X_: "), 16 * iPage + y) ;
          TextOut (hdc, 0, y * cyChar, szBuffer, 7) ;

          for (x = 0 ; x < 16 ; x++)
          {
            ch = (WCHAR) (256 * iPage + 16 * y + x) ;
            TextOutW (hdc, x * cxChar + cxLabels,
              y * cyChar, &ch, 1) ;
          }
        }

        DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
        EndPaint (hwnd, &ps) ;
        return 0 ;

      case WM_DESTROY:
        PostQuitMessage (0) ;
        return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

UNICHARS.RC

```
//Microsoft Developer Studio generated resource script.
```

```
#include "resource.h"
#include "afxres.h"
/////////////////////////////////////////////////////////////////////////////
// Menu
UNICHARS MENU DISCARDABLE
BEGIN
MENUITEM "&Font!",       IDM_FONT
END
```

RESOURCE.H

```
// Microsoft Developer Studio generated include file.
// Used by Unichars.rc
#define IDM_FONT          40001
```

段落格式

具有选择并建立逻辑字体的能力后，就可以处理文字格式了。这个程序包括以四种方式之一来把文字的每一行放在页边距内：左对齐、向右对齐、居中或分散对齐 – 即从页边距的一端到另一端，文字间距相等。对于前三种方式，可以使用带有DT_WORDBREAK参数的DrawText函数，但这种方法有局限性。例如，您无法确定DrawText会把文字的哪个部分恰好放在矩形内。DrawText对于一些简单任务是很方便的，但对更复杂的格式化任务，则可能要用到TextOut。

简单文字格式

对文字的最有用的一个函数是GetTextExtentPoint32（这个函数的名称显示了Windows早期版本的一些变化）。该函数根据设备内容中选入的目前字体得出字符串的宽度和高度：

GetTextExtentPoint32 (hdc, pString, iCount, &size) ;

逻辑单位的文字宽度和高度在SIZE结构的cx和cy字段中传回。我使用一行文字的例子，假定您把一种字体选入设备内容，现在要写入文字：

TCHAR * szText [] = TEXT ("Hello, how are you?") ;

您希望文字从垂直坐标yStart开始，页边距由坐标xLeft和xRight设定。您的任务就是计算文字开始处的水平坐标的xStart值。

如果文字以定宽字体显示，那么这项任务就相当容易，但通常不是这样的。首先您得到字符串的文字宽度：

GetTextExtentPoint32 (hdc, szText, lstrlen (szText), &size) ;

如果size.cx比（xRight – xLeft）大，这一行就太长了，不能放在页边距内。我们假定它能放进去。

要向左对齐文字，只要把xStart设定为与xLeft相等，然后写入文字：

TextOut (hdc, xStart, yStart, szText, lstrlen (szText)) ;

这很容易。现在可以把size.cy加到yStart中写下一行文字了。

要向右对齐文字，用以下公式计算xStart：

xStart = xRight - size.cx ;

居中文字用以下公式：

xStart = (xLeft + xRight - size.cx) / 2 ;

现在开始艰巨的任务 – 在左右页边距内分散对齐文字。页边距之间的距离是（xRight – xLeft）。如不调整，文字宽度就是size.cx。两者之差

xRight - xLeft - size.cx

必须在字符串的三个空格字符处平均配置。这听起来很讨厌，但还不是太糟。可以呼叫

SetTextJustification (hdc, xRight - xLeft - size.cx, 3)

来完成。第二个参数是字符串内空格字符中需要分配的空间量。第三个参数是空格字符的数量，这里为3。现在把xStart设定与xLeft相等，用TextOut写入文字：

TextOut (hdc, xStart, yStart, szText, lstrlen (szText)) ;

文字会在xLeft和xRight页边距之间分散对齐。

无论何时呼叫SetTextJustification，如果空间量不能在空格字符中平均分配，它就会累积一个错误值。这将影响后面的GetTextExtentPoint32呼叫。每次开始新的一行，都必须通过呼叫

SetTextJustification (hdc, 0, 0) ;

来清除错误值。

使用段落

如果您处理整个段落，就必须从头开始并扫描字符串来寻找空格字符。每当碰到一个空格（或其它能用于断开一行的字符），需呼叫GetTextExtentPoint32来确定文字是否能放入左右页边距之间。当文字超出允许的空间时，就要退回上一个空白。现在，您已经能够确定一行的字符串了。如果想要分散对齐该行，呼叫SetTextJustification和TextOut，清除错误值，并继续下一行。

显示在程序17-7中的JUSTIFY1对Mark Twain的《The Adventures of Huckleberry Finn》中的第一段做了这样的处理。您可以从对话框中选择想要的字体，也可以使用菜单选项来更改对齐方式（左对齐、向右对齐、居中或分散对齐）。图17-3是典型的JUSTIFY1屏幕显示。

程序17-7 JUSTIFY1

JUSTIFY1.C

```
/*------------------------------------------------------------------------
JUSTIFY1.C -- Justified Type Program #1
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Justify1") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Justified Type #1"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void DrawRuler (HDC hdc, RECT * prc)
{
  static int iRuleSize [16] = {360, 72,144, 72,216, 72,144,72,
    288, 72,144, 72,216, 72,144,72 } ;
  int i, j ;
  POINT ptClient ;

  SaveDC (hdc) ;
  // Set Logical Twips mapping mode
  SetMapMode (hdc, MM_ANISOTROPIC) ;
  SetWindowExtEx (hdc, 1440, 1440, NULL) ;
  SetViewportExtEx (hdc,GetDeviceCaps (hdc, LOGPIXELSX),
    GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;

  // Move the origin to a half inch from upper left
  SetWindowOrgEx (hdc, -720, -720, NULL) ;
  // Find the right margin (quarter inch from right)
  ptClient.x = prc->right ;
  ptClient.y = prc->bottom ;
  DPtoLP (hdc, &ptClient, 1) ;
  ptClient.x -= 360 ;

  // Draw the rulers
  MoveToEx (hdc, 0, -360, NULL) ;
  LineTo (hdc, ptClient.x, -360) ;
  MoveToEx (hdc, -360, 0, NULL) ;
  LineTo (hdc, -360, ptClient.y) ;

  for (i = 0, j = 0 ; i <= ptClient.x ; i += 1440 / 16, j++)
  {
    MoveToEx (hdc, i, -360, NULL) ;
    LineTo (hdc, i, -360 - iRuleSize [j % 16]) ;
  }

  for (i = 0, j = 0 ; i <= ptClient.y ; i += 1440 / 16, j++)
  {
    MoveToEx (hdc, -360, i, NULL) ;
    LineTo (hdc, -360 - iRuleSize [j % 16], i) ;
  }
```

```
  RestoreDC (hdc, -1) ;
}
void Justify (HDC hdc, PTSTR pText, RECT * prc, int iAlign)
{
  int xStart, yStart, cSpaceChars ;
  PTSTR pBegin, pEnd ;
  SIZE size ;

  yStart = prc->top ;
  do // for each text line
  {
    cSpaceChars = 0 ; // initialize number of spaces in line

    while (* pText == '') // skip over leading spaces
      pText++ ;

    pBegin = pText ; // set pointer to char at beginning of line

    do // until the line is known
    {
      pEnd =pText ;// set pointer to char at end of line

      // skip to next space
      while (*pText != '\0' && *pText++ != ' ') ;

      if (*pText == '\0')
        break ;

      // after each space encountered, calculate extents
      cSpaceChars++ ;
      GetTextExtentPoint32(hdc, pBegin, pText - pBegin - 1, &size) ;
    }
    while (size.cx < (prc->right - prc->left)) ;
    cSpaceChars-- ; // discount last space at end of line

    while (*(pEnd - 1) == ' ') // eliminate trailing spaces
    {
      pEnd-- ;
      cSpaceChars-- ;
    }

    // if end of text and no space characters, set pEnd to end
    if (* pText == '\0' || cSpaceChars <= 0)
      pEnd = pText ;

    GetTextExtentPoint32 (hdc, pBegin, pEnd - pBegin, &size) ;

    switch (iAlign) // use alignment for xStart
    {
    case IDM_ALIGN_LEFT:
      xStart = prc->left ;
      break ;

    case IDM_ALIGN_RIGHT:
      xStart = prc->right - size.cx ;
      break ;

    case IDM_ALIGN_CENTER:
      xStart = (prc->right + prc->left - size.cx) / 2 ;
      break ;

    case IDM_ALIGN_JUSTIFIED:
      if (* pText != '\0' && cSpaceChars > 0)
        SetTextJustification (hdc, prc->right-prc->left - size.cx, cSpaceChars) ;
      xStart = prc->left ;
      break ;
    }
    // display the text
```

```
    TextOut (hdc, xStart, yStart, pBegin, pEnd - pBegin) ;

    // prepare for next line
    SetTextJustification (hdc, 0, 0) ;
    yStart += size.cy ;
    pText = pEnd ;
  }
  while (*pText && yStart < prc->bottom - size.cy) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static CHOOSEFONT cf ;
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("Justify1: Printing") } ;
  static int iAlign = IDM_ALIGN_LEFT ;
  static LOGFONT lf ;
  static PRINTDLG pd ;
  static TCHAR szText[] = {
    TEXT ("You don't know about me, without you ")
      TEXT ("have read a book by the name of \"The ")
      TEXT ("Adventures of Tom Sawyer,\" but that ")
      TEXT ("ain't no matter. That book was made by ")
      TEXT ("Mr. Mark Twain, and he told the truth, ")
      TEXT ("mainly. There was things which he ")
      TEXT ("stretched, but mainly he told the truth. ")
      TEXT ("That is nothing. I never seen anybody ")
      TEXT ("but lied, one time or another, without ")
      TEXT ("it was Aunt Polly, or the widow, or ")
      TEXT ("maybe Mary. Aunt Polly -- Tom's Aunt ")
      TEXT ("Polly, she is -- and Mary, and the Widow ")
      TEXT ("Douglas, is all told about in that book ")
      TEXT ("-- which is mostly a true book; with ")
      TEXT ("some stretchers, as I said before.") } ;
    BOOL fSuccess ;
    HDC hdc, hdcPrn ;
    HMENU hMenu ;
    int iSavePointSize ;
    PAINTSTRUCT ps ;
    RECT rect ;

    switch (message)
    {
    case WM_CREATE:
      // Initialize the CHOOSEFONT structure
      GetObject (GetStockObject (SYSTEM_FONT), sizeof (lf), &lf) ;

      cf.lStructSize = sizeof (CHOOSEFONT) ;
      cf.hwndOwner = hwnd ;
      cf.hDC = NULL ;
      cf.lpLogFont = &lf ;
      cf.iPointSize = 0 ;
      cf.Flags = CF_INITTOLOGFONTSTRUCT | CF_SCREENFONTS |
        CF_EFFECTS ;
      cf.rgbColors = 0 ;
      cf.lCustData = 0 ;
      cf.lpfnHook = NULL ;
      cf.lpTemplateName = NULL ;
      cf.hInstance = NULL ;
      cf.lpszStyle = NULL ;
      cf.nFontType = 0 ;
      cf.nSizeMin = 0 ;
      cf.nSizeMax = 0 ;

      return 0 ;

    case WM_COMMAND:
      hMenu = GetMenu (hwnd) ;

      switch (LOWORD (wParam))
```

```
    {
    case IDM_FILE_PRINT:
      // Get printer DC
      pd.lStructSize = sizeof (PRINTDLG) ;
      pd.hwndOwner = hwnd ;
      pd.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

      if (!PrintDlg (&pd))
        return 0 ;

      if (NULL == (hdcPrn = pd.hDC))
      {
        MessageBox ( hwnd, TEXT ("Cannot obtain Printer DC"),
          szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;
      }
      // Set margins of 1 inch
      rect.left = GetDeviceCaps (hdcPrn, LOGPIXELSX) -
        GetDeviceCaps (hdcPrn, PHYSICALOFFSETX) ;

      rect.top = GetDeviceCaps (hdcPrn, LOGPIXELSY)
        GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;
      rect.right = GetDeviceCaps (hdcPrn, PHYSICALWIDTH) -
        GetDeviceCaps (hdcPrn, LOGPIXELSX) -
        GetDeviceCaps (hdcPrn, PHYSICALOFFSETX) ;
      rect.bottom= GetDeviceCaps (hdcPrn, PHYSICALHEIGHT) -
        GetDeviceCaps (hdcPrn, LOGPIXELSY) -
        GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;

      // Display text on printer
      SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
      ShowCursor (TRUE) ;

      fSuccess = FALSE ;
      if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
      {
        // Select font using adjusted lfHeight
        iSavePointSize = lf.lfHeight ;
        lf.lfHeight = -(GetDeviceCaps (hdcPrn, LOGPIXELSY) *
          cf.iPointSize) / 720 ;

        SelectObject (hdcPrn, CreateFontIndirect (&lf)) ;
        lf.lfHeight = iSavePointSize ;

        // Set text color
        SetTextColor (hdcPrn, cf.rgbColors) ;

        // Display text
        Justify (hdcPrn, szText, &rect, iAlign) ;

        if (EndPage (hdcPrn) > 0)
        {
          fSuccess = TRUE ;
          EndDoc (hdcPrn) ;
        }
      }
      ShowCursor (FALSE) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

      DeleteDC (hdcPrn) ;

      if (!fSuccess)
        MessageBox (hwnd, TEXT ("Could not print text"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;

    case IDM_FONT:
      if (ChooseFont (&cf))
        InvalidateRect (hwnd, NULL, TRUE) ;
```

```
      return 0 ;

    case IDM_ALIGN_LEFT:
    case IDM_ALIGN_RIGHT:
    case IDM_ALIGN_CENTER:
    case IDM_ALIGN_JUSTIFIED:
      CheckMenuItem (hMenu, iAlign, MF_UNCHECKED) ;
      iAlign = LOWORD (wParam) ;
      CheckMenuItem (hMenu, iAlign, MF_CHECKED) ;
      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
    }
    return 0 ;

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      GetClientRect (hwnd, &rect) ;
      DrawRuler (hdc, &rect) ;

      rect.left += GetDeviceCaps (hdc, LOGPIXELSX) / 2 ;
      rect.top += GetDeviceCaps (hdc, LOGPIXELSY) / 2 ;
      rect.right -= GetDeviceCaps (hdc, LOGPIXELSX) / 4 ;

      SelectObject (hdc, CreateFontIndirect (&lf)) ;
      SetTextColor (hdc, cf.rgbColors) ;

      Justify (hdc, szText, &rect, iAlign) ;

      DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;

    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

JUSTIFY1.RC

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
JUSTIFY1 MENU DISCARDABLE BEGIN POPUP "&File"
BEGIN
MENUITEM "&Print",     IDM_FILE_PRINT
END
POPUP "&Font"
BEGIN
MENUITEM "&Font...", IDM_FONT
END
POPUP "&Align"
BEGIN
MENUITEM "&Left",    IDM_ALIGN_LEFT, CHECKED
MENUITEM "&Right",    IDM_ALIGN_RIGHT
MENUITEM "&Centered",  IDM_ALIGN_CENTER
MENUITEM "&Justified",  IDM_ALIGN_JUSTIFIED
END
END
```

RESOURCE.H

```
// Microsoft Developer Studio generated include file.
// Used by Justify1.rc
#define   IDM_FILE_PRINT 40001
#define   IDM_FONT   40002
#define   IDM_ALIGN_LEFT 40003
#define   IDM_ALIGN_RIGHT 40004
#define   IDM_ALIGN_CENTER 40005
#define   IDM_ALIGN_JUSTIFIED 40006
```

JUSTIFY1在显示区域的上部和左侧显示了尺规（当然单位是逻辑英寸）。尺规由DrawRuler函数画出。一个矩形结构定义了分散对齐文字的区域。

涉及对文字进行格式处理的大量工作由Justify函数实作。函数搜寻文字开始的空白，并使用GetTextExtentPoint32测量每一行。当行的长度超过显示区域的宽度，JUSTIFY1传回先前的空格并使该行到达linefeed处。根据iAlign常数的值，行的对齐方式有：同左对齐、向右对齐、居中或分散对齐。

JUSTIFY1并不完美。例如，它没有处理连字符的问题。此外，当每行少于两个字时，分散对齐的做法会失效。即使我们解决了这个不是特别难的问题，当一个单字太长在左右边距间放不下时，程序仍不能正常运作。当然，当我们在程序中对同一行使用多种字体（如同Windows文书处理程序轻松做出的那样）时，情况会更复杂。还没有人声称这种处理容易，它只是比我们亲自做所有的工作容易一些。

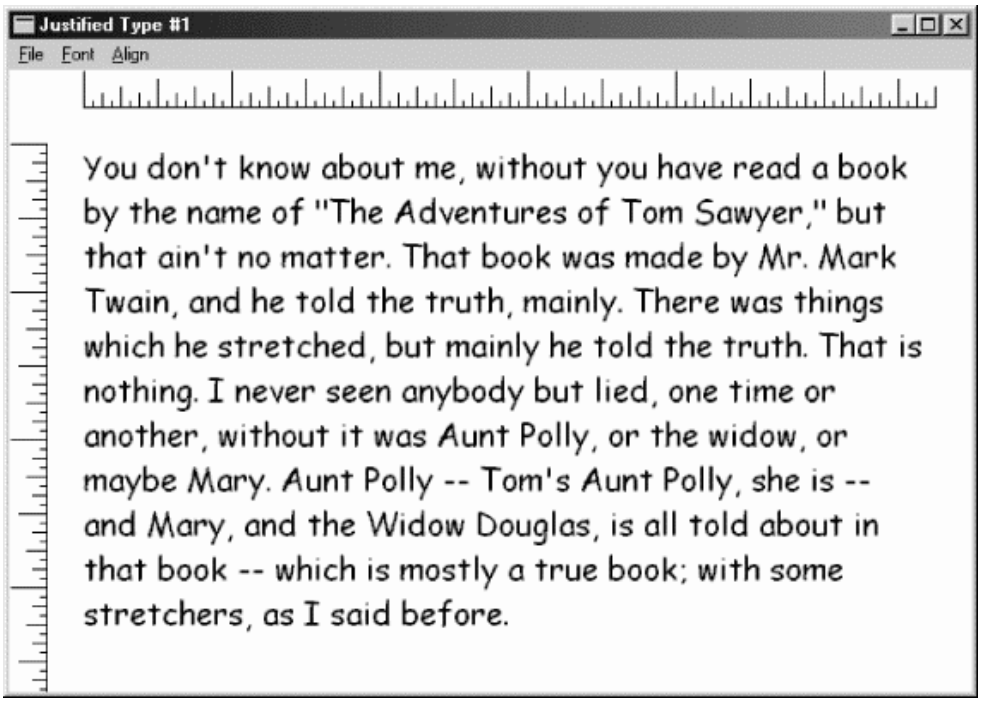


图17-3 典型的JUSTIFY1屏幕显示

打印输出预览

有些字体不是为了在屏幕上查看用的，这些字体是用于打印的。通常在这种情况下，文字的屏幕预览必须与打印输出的格式精确配合。显示同样的字体、大小和字符格式是不够的。使用TrueType是个快捷方式。另外还需要将段落中的每一行在同样位置断开。这是WYSIWYG中的难点。

JUSTIFY1包含一个「Print」选项，但该选项仅在页面的上、左和右边设定1英寸的边距。这样，格式化完全与显示器无关。这里有一个有趣的练习：在JUSTIFY1中更改几行程式码，使屏幕和打印机逻辑依据一个6英寸的格式化矩形。方法就是在WM_PAINT和「Print」命令处理程序中更改

rect.right的定义。在WM_PAINT处理程序中，表述如下：:

rect.right = rect.left + 6 * GetDeviceCaps (hdc, LOGPIXELSX) ;

在「Print」命令处理程序中，相对应叙述为：

rect.right = rect.left + 6 * GetDeviceCaps (hdcPrn, LOGPIXELSX) ;

如果选择了一种TrueType字体，屏幕上的linefeed情况应与打印机的输出相同。

但实际情况并不是这样。即使两种设备使用同样点值的相同字体，并将文字显示在同样的格式化矩形中，不同的显示分辨率及凑整误差也会使linefeed出现在不同地方。显然，需要一种更高明的方法进行屏幕上的打印输出预览。

程序17-8所示的JUSTIFY2示范了这种方法的一个尝试。JUSTIFY2中的程序代码是依据Microsoft的David Weise所写的TTJUST（「TrueType Justify」）程序，而该程序又是依据本书前面的一个版本中的JUSTIFY1程序。为表现出这一程序中所增加的复杂性，用Herman Melville的《Moby-Dick》中的第一章代替了Mark Twain小说的摘录。

程序17-8 JUSTIFY2

JUSTIFY2.C

```
/*-----------------------------------------------------------------------------
JUSTIFY2.C -- Justified Type Program #2
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define OUTWIDTH 6 // Width of formatted output in inches
#define LASTCHAR 127 // Last character code used in text

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("Justify2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Justified Type #2"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void DrawRuler (HDC hdc, RECT * prc)
{
  static int iRuleSize [16] = {360,72,144, 72,216,72,144,72,288,72,144,
    72,216,72,144, 72 } ;
  int i, j ;
  POINT ptClient ;

  SaveDC (hdc) ;
  // Set Logical Twips mapping mode
  SetMapMode (hdc, MM_ANISOTROPIC) ;
  SetWindowExtEx (hdc, 1440, 1440, NULL) ;
  SetViewportExtEx (hdc, GetDeviceCaps (hdc, LOGPIXELSX),
    GetDeviceCaps (hdc, LOGPIXELSY), NULL) ;

  // Move the origin to a half inch from upper left
  SetWindowOrgEx (hdc, -720, -720, NULL) ;
  // Find the right margin (quarter inch from right)
  ptClient.x = prc->right ;
  ptClient.y = prc->bottom ;
  DPtoLP (hdc, &ptClient, 1) ;
  ptClient.x -= 360 ;

  // Draw the rulers
  MoveToEx (hdc, 0, -360, NULL) ;
  LineTo (hdc, OUTWIDTH * 1440,-36 0) ;
  MoveToEx (hdc, -360, 0, NULL) ;
  LineTo (hdc, -360, ptClient.y) ;

  for (i = 0, j = 0 ; i <= ptClient.x && i <= OUTWIDTH * 1440 ;
  i += 1440 / 16, j++)
  {
    MoveToEx (hdc, i, -360, NULL) ;
    LineTo (hdc, i, -360 - iRuleSize [j % 16]) ;
  }

  for (i = 0, j = 0 ; i <= ptClient.y ; i += 1440 / 16, j++)
  {
    MoveToEx (hdc, -360, i, NULL) ;
    LineTo (hdc, -360 - iRuleSize [j % 16], i) ;
  }

  RestoreDC (hdc, -1) ;
}

/*-------------------------------------------------------------------------
GetCharDesignWidths: Gets character widths for font as large as the
original design size
-------------------------------------------------------------------------*/

UINT GetCharDesignWidths (HDC hdc, UINT uFirst, UINT uLast, int * piWidths)
{
  HFONT hFont, hFontDesign ;
  LOGFONT lf ;
  OUTLINETEXTMETRIC otm ;

  hFont = GetCurrentObject (hdc, OBJ_FONT) ;
  GetObject (hFont, sizeof (LOGFONT), &lf) ;

  // Get outline text metrics (we'll only be using a field that is
  // independent of the DC the font is selected into)
```

```
  otm.otmSize = sizeof (OUTLINETEXTMETRIC) ;
  GetOutlineTextMetrics (hdc, sizeof (OUTLINETEXTMETRIC), &otm) ;

  // Create a new font based on the design size
  lf.lfHeight = - (int) otm.otmEMSquare ;
  lf.lfWidth = 0 ;
  hFontDesign = CreateFontIndirect (&lf) ;

  // Select the font into the DC and get the character widths
  SaveDC (hdc) ;
  SetMapMode (hdc, MM_TEXT) ;
  SelectObject (hdc, hFontDesign) ;

  GetCharWidth (hdc, uFirst, uLast, piWidths) ;
  SelectObject (hdc, hFont) ;
  RestoreDC (hdc, -1) ;

  // Clean up
  DeleteObject (hFontDesign) ;
  return otm.otmEMSquare ;
}

/*-------------------------------------------------------------------------
GetScaledWidths: Gets floating point character widths for selected
font size
-------------------------------------------------------------------------*/

void GetScaledWidths (HDC hdc, double * pdWidths)
{
  double dScale ;
  HFONT hFont ;
  int aiDesignWidths [LASTCHAR + 1] ;
  int i ;
  LOGFONT lf ;
  UINT uEMSquare ;

  // Call function above
  uEMSquare = GetCharDesignWidths (hdc, 0, LASTCHAR, aiDesignWidths) ;
  // Get LOGFONT for current font in device context
  hFont = GetCurrentObject (hdc, OBJ_FONT) ;
  GetObject (hFont, sizeof (LOGFONT), &lf) ;
  // Scale the widths and store as floating point values
  dScale = (double) -lf.lfHeight / (double) uEMSquare ;
  for ( i = 0 ; i <= LASTCHAR ; i++)
    pdWidths[i] = dScale * aiDesignWidths[i] ;
}

/*-------------------------------------------------------------------------
GetTextExtentFloat: Calculates text width in floating point
-------------------------------------------------------------------------*/

double GetTextExtentFloat (double * pdWidths, PTSTR psText, int iCount)
{
  double dWidth = 0 ;
  int i ;

  for ( i = 0 ; i < iCount ; i++)
    dWidth += pdWidths [psText[i]] ;

  return dWidth ;
}

/*-------------------------------------------------------------------------
Justify: Based on design units for screen/printer compatibility
-------------------------------------------------------------------------*/

void Justify (HDC hdc, PTSTR pText, RECT * prc, int iAlign)
{
  double dWidth, adWidths[LASTCHAR + 1] ;
```

```
int xStart, yStart, cSpaceChars ;
PTSTR pBegin, pEnd ;
SIZE size ;

// Fill the adWidths array with floating point character widths
GetScaledWidths (hdc, adWidths) ;
yStart = prc->top ;
do // for each text line
{
  cSpaceChars = 0 ; // initialize number of spaces in line

  while (*pText == ' ')// skip over leading spaces
    pText++ ;

  pBegin = pText ; // set pointer to char at beginning of line

  do // until the line is known
  {
    pEnd = pText ; // set pointer to char at end of line

    // skip to next space
    while (*pText != '\0' && *pText++ != ' ') ;

    if (*pText == '\0')
      break ;

    // after each space encountered, calculate extents
    cSpaceChars++ ;
    dWidth = GetTextExtentFloat (adWidths, pBegin,
      pText - pBegin - 1) ;
  }
  while (dWidth < (double) (prc->right - prc->left)) ;

  cSpaceChars-- ; // discount last space at end of line

  while (*(pEnd - 1) == ' ') // eliminate trailing spaces
  {
    pEnd-- ;
    cSpaceChars-- ;
  }

  // if end of text and no space characters, set pEnd to end
  if (*pText == '\0' || cSpaceChars <= 0)
    pEnd = pText ;

  // Now get integer extents
  GetTextExtentPoint32(hdc, pBegin, pEnd - pBegin, &size) ;

  switch (iAlign) // use alignment for xStart
  {
  case IDM_ALIGN_LEFT:
    xStart = prc->left ;
    break ;

  case IDM_ALIGN_RIGHT:
    xStart = prc->right - size.cx ;
    break ;
  case IDM_ALIGN_CENTER:
    xStart = (prc->right + prc->left - size.cx) / 2 ;
    break ;

  case IDM_ALIGN_JUSTIFIED:
    if (*pText != '\0' && cSpaceChars > 0)
      SetTextJustification (hdc,
      prc->right - prc->left - size.cx,
      cSpaceChars) ;
    xStart = prc->left ;
    break ;
  }
```

```
    // display the text
    TextOut (hdc, xStart, yStart, pBegin, pEnd - pBegin) ;

    // prepare for next line
    SetTextJustification (hdc, 0, 0) ;
    yStart += size.cy ;
    pText = pEnd ;
  }
  while (*pText && yStart < prc->bottom - size.cy) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static CHOOSEFONT cf ;
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("Justify2: Printing") } ;
  static int iAlign = IDM_ALIGN_LEFT ;
  static LOGFONT lf ;
  static PRINTDLG pd ;
  static TCHAR szText[] = {
    TEXT ("Call me Ishmael. Some years ago -- never ")
      TEXT ("mind how long precisely -- having little ")
      TEXT ("or no money in my purse, and nothing ")
      TEXT ("particular to interest me on shore, I ")
      TEXT ("thought I would sail about a little and ")
      TEXT ("see the watery part of the world. It is ")
      TEXT ("a way I have of driving off the spleen, ")
      TEXT ("and regulating the circulation. Whenever ")
      TEXT ("I find myself growing grim about the ")
      TEXT ("mouth; whenever it is a damp, drizzly ")
      TEXT ("November in my soul; whenever I find ")
      TEXT ("myself involuntarily pausing before ")
      TEXT ("coffin warehouses, and bringing up the ")
      TEXT ("rear of every funeral I meet; and ")
      TEXT ("especially whenever my hypos get such an ")
      TEXT ("upper hand of me, that it requires a ")
      TEXT ("strong moral principle to prevent me ")
      TEXT ("from deliberately stepping into the ")
      TEXT ("street, and methodically knocking ")
      TEXT ("people's hats off -- then, I account it ")
      TEXT ("high time to get to sea as soon as I ")
      TEXT ("can. This is my substitute for pistol ")
      TEXT ("and ball. With a philosophical flourish ")
      TEXT ("Cato throws himself upon his sword; I ")
      TEXT ("quietly take to the ship. There is ")
      TEXT ("nothing surprising in this. If they but ")
      TEXT ("knew it, almost all men in their degree, ")
      TEXT ("some time or other, cherish very nearly ")
      TEXT ("the same feelings towards the ocean with ")
      TEXT ("me.") } ;
    BOOL fSuccess ;
    HDC hdc, hdcPrn ;
    HMENU hMenu ;
    int iSavePointSize ;
    PAINTSTRUCT ps ;
    RECT rect ;

    switch (message)
    {
    case WM_CREATE:
      // Initialize the CHOOSEFONT structure
      hdc = GetDC (hwnd) ;
      lf.lfHeight = - GetDeviceCaps (hdc, LOGPIXELSY) / 6 ;
      lf.lfOutPrecision = OUT_TT_ONLY_PRECIS ;
      lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;
      ReleaseDC (hwnd, hdc) ;

      cf.lStructSize = sizeof (CHOOSEFONT) ;
      cf.hwndOwner = hwnd ;
      cf.hDC = NULL ;
```

```
    cf.lpLogFont = &lf ;
    cf.iPointSize = 120 ;

    // Set flags for TrueType only!
    cf.Flags = CF_INITTOLOGFONTSTRUCT | CF_SCREENFONTS |
      CF_TTONLY | CF_EFFECTS ;
    cf.rgbColors = 0 ;
    cf.lCustData = 0 ;
    cf.lpfnHook = NULL ;
    cf.lpTemplateName = NULL ;
    cf.hInstance = NULL ;
    cf.lpszStyle = NULL ;
    cf.nFontType = 0 ;
    cf.nSizeMin = 0 ;
    cf.nSizeMax = 0 ;

    return 0 ;

  case WM_COMMAND:
    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
    case IDM_FILE_PRINT:
      // Get printer DC
      pd.lStructSize = sizeof (PRINTDLG) ;
      pd.hwndOwner = hwnd ;
      pd.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

      if (!PrintDlg (&pd))
        return 0 ;

      if (NULL == (hdcPrn = pd.hDC))
      {
        MessageBox(hwnd, TEXT ("Cannot obtain Printer DC"),
          szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;
      }
      // Set margins for OUTWIDTH inches wide
      rect.left = (GetDeviceCaps (hdcPrn, PHYSICALWIDTH) -
        GetDeviceCaps (hdcPrn, LOGPIXELSX)*OUTWIDTH)/2
        - GetDeviceCaps (hdcPrn, PHYSICALOFFSETX) ;

      rect.right = rect.left +
        GetDeviceCaps (hdcPrn, LOGPIXELSX) * OUTWIDTH ;

      // Set margins of 1 inch at top and bottom
      rect.top = GetDeviceCaps (hdcPrn, LOGPIXELSY) -
        GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;

      rect.bottom =GetDeviceCaps (hdcPrn, PHYSICALHEIGHT) -
        GetDeviceCaps (hdcPrn, LOGPIXELSY) -
        GetDeviceCaps (hdcPrn, PHYSICALOFFSETY) ;

      // Display text on printer

      SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
      ShowCursor (TRUE) ;

      fSuccess = FALSE ;

      if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
      {
        // Select font using adjusted lfHeight
        iSavePointSize = lf.lfHeight ;
        lf.lfHeight = -(GetDeviceCaps (hdcPrn, LOGPIXELSY) *
          cf.iPointSize) / 720 ;

        SelectObject (hdcPrn, CreateFontIndirect (&lf)) ;
```

```
        lf.lfHeight = iSavePointSize ;

        // Set text color
        SetTextColor (hdcPrn, cf.rgbColors) ;

        // Display text
        Justify (hdcPrn, szText, &rect, iAlign) ;

        if (EndPage (hdcPrn) > 0)
        {
          fSuccess = TRUE ;
          EndDoc (hdcPrn) ;
        }
      }
      ShowCursor (FALSE) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

      DeleteDC (hdcPrn) ;

      if (!fSuccess)
        MessageBox (hwnd, TEXT ("Could not print text"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;
    case IDM_FONT:
      if (ChooseFont (&cf))
        InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;

    case IDM_ALIGN_LEFT:
    case IDM_ALIGN_RIGHT:
    case IDM_ALIGN_CENTER:
    case IDM_ALIGN_JUSTIFIED:
      CheckMenuItem (hMenu, iAlign, MF_UNCHECKED) ;
      iAlign = LOWORD (wParam) ;
      CheckMenuItem (hMenu, iAlign, MF_CHECKED) ;
      InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
    }
    return 0 ;
    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;
      GetClientRect (hwnd, &rect) ;
      DrawRuler (hdc, &rect) ;

      rect.left += GetDeviceCaps (hdc, LOGPIXELSX) / 2 ;
      rect.top += GetDeviceCaps (hdc, LOGPIXELSY) / 2 ;
      rect.right = rect.left + OUTWIDTH * GetDeviceCaps (hdc, LOGPIXELSX) ;

      SelectObject (hdc, CreateFontIndirect (&lf)) ;
      SetTextColor (hdc, cf.rgbColors) ;

      Justify (hdc, szText, &rect, iAlign) ;
      DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
      EndPaint (hwnd, &ps) ;
      return 0 ;
    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

JUSTIFY2.RC

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
```

```
// Menu
JUSTIFY2 MENU DISCARDABLE BEGIN POPUP "&File"
BEGIN
MENUITEM "&Print",   IDM_FILE_PRINT
END
POPUP "&Font"
BEGIN
MENUITEM "&Font...", IDM_FONT
END
POPUP "&Align"
BEGIN
MENUITEM "&Left",     IDM_ALIGN_LEFT, CHECKED
MENUITEM "&Right", IDM_ALIGN_RIGHT
MENUITEM "&Centered",   IDM_ALIGN_CENTER
MENUITEM "&Justified",  IDM_ALIGN_JUSTIFIED
END
END
```

RESOURCE.H

```
// Microsoft Developer Studio generated include file.
// Used by Justify2.rc
#define   IDM_FILE_PRINT  40001
#define   IDM_FONT   40002
#define   IDM_ALIGN_LEFT  40003
#define   IDM_ALIGN_RIGHT  40004
#define   IDM_ALIGN_CENTER 40005
#define   IDM_ALIGN_JUSTIFIED 40006
```

JUSTIFY2仅使用TrueType字体。在它的GetCharDesignWidths函数中，程序使用GetOutlineTextMetrics函数取得一个表面上似乎不重要的信息，即OUTLINETEXTMETRIC的otmEMSquare字段。

TrueType字体在全方（em-square）的网格上设计（如我说过「em」是指一种方块型态的宽度，M在宽度上等于字体点值的大小）。任何特定TrueType字体的所有字符都是在同样的网格上设计的，虽然这些字符通常有不同的宽度。OUTLINETEXTMETRIC结构的otmEMSquare字段给出了任意特定字体的这种全方形式的大小。您会发现：对于大多数TrueType字体，otmEMSquare字段等于2048，这意味着字体是在2048×2048的网格上设计的。

关键在于：可以为想要使用的特定TrueType字体名称设定一个LOGFONT结构，其lfHeight字段等于otmEMSquare值的负数。在建立字体并将其选入设备内容后，可呼叫GetCharWidth。该函数以逻辑单位提供字体中单个字符的宽度。通常，因为这些字符被缩放为不同的字体大小，所以字符宽度并不准确。但使用依据otmEMSquare大小的字体，这些宽度总是与任何设备内容无关的精确整数。

GetCharDesignWidths函数以这种方式获得原始的字符设计宽度，并将它们储存在整数数组中。JUSTIFY2程序在自己的文字中仅使用ASCII字符，因此，这个数组不需要很大。GetScaledWidths函数将这些整数型态宽度转变为依据设备逻辑坐标中字体的实际点值的浮点宽度。GetTextExtentFloat函数使用这些浮点宽度计算整个字符串的宽度。这是新的Justify函数用以计算文字行宽度的操作。

有趣的东西

根据外形轮廓表示字体字符提供了将字体与其它图形技术相结合的可能性。前面我们讨论了旋转字体的方式。这里讲述一些其它技巧。继续之前，先了解两个重要的预备知识：绘图路径和扩展

画笔。

GDI绘图路径

绘图路径是储存在GDI内的直线和曲线的集合。绘图路径是在Windows的32位版本中发表的。绘图路径看上去类似于区域，我们确实可以将绘图路径转换为区域，并使用绘图路径进行剪裁。但随后我们会发现两者的不同。

要定义绘图路径，可先简单呼叫

BeginPath (hdc) ;

进行该呼叫之后，所画的任何线（例如，直线、弧及贝塞尔曲线）将作为绘图路径储存在GDI内部，不被显示到设备内容上。绘图路径经常由连结起来的线组成。要制作连结线，应使用LineTo、PolylineTo和BezierTo函数，这些函数都以目前位置为起点划线。如果使用MoveToEx改变了目前位置，或呼叫其它的画线函数，或者呼叫了会导致目前位置改变的窗口/视端口函数，您就在整个绘图路径中建立了一个新的子绘图路径。因此，绘图路径包含一或多个子绘图路径，每一个子绘图路径是一系列连结的线段。

绘图路径中的每个子绘图路径可以是敞开的或封闭的。封闭子绘图路径之第一条连结线的第一个点与最后一条连结线的最后一点相同，并且子绘图路径通过呼叫CloseFigure结束。如果必要的话，CloseFigure将用一条直线封闭子绘图路径。随后的画线函数将开始一个新的子绘图路径。最后，通过下面的呼叫结束绘图路径定义：

EndPath (hdc) ;

这时，接着呼叫下列五个函数之一：

StrokePath (hdc) ;

FillPath (hdc) ;

StrokeAndFillPath (hdc) ;

hRgn = PathToRegion (hdc) ;

SelectClipPath (hdc, iCombine) ;

这些函数中的每一个都会在绘图路径定义完成后，将其清除。

StrokePath使用目前画笔绘制绘图路径。您可能会好奇：绘图路径上的点有哪些？为什么不能跳过这些绘图路径片段正常地画线？稍后我会告诉您原因。

另外四个函数用直线关闭任何敞开的绘图路径。FillPath依照目前的多边填充模式使用目前画刷填充绘图路径。StrokeAndFillPath一次完成这两项工作。也可将绘图路径转换为区域，或者将绘图路径用于某个剪裁区域。iCombine参数是CombineRgn函数使用的RGN_ 系列常数之一，它指出了绘图路径与目前剪裁区域的结合方式。

用于填充或剪取时，绘图路径比绘图区域更灵活，这是因为绘图区域仅能由矩形、椭圆及多边形的组合定义；绘图路径可由贝塞尔曲线定义，至少在Windows NT中还可由弧线组成。在GDI中，绘图路径和区域的储存也完全不同。绘图路径是直线及曲线定义的集合；而绘图区域（通常意义上）是扫描线的集合。

扩展画笔

在呼叫StrokePath时，使用目前画笔绘制绘图路径。在第四章讨论了用以建立画笔对象的CreatePen函数。伴随绘图路径的发表，Windows也支持一个称为ExtCreatePen的扩展画笔函数

呼叫。该函数揭示了其建立绘图路径以及使用绘图路径要比不使用绘图路径画线有用。ExtCreatePen函数如下所示：

hPen = ExtCreatePen (iStyle, iWidth, &lBrush, 0, NULL) ;

您可以使用该函数正常地绘制线段，但在这种情况下Windows 98不支持一些功能。甚至用以显示绘图路径时，Windows 98仍不支持一些功能，这就是上面函数的最后两个参数被设定为0及NULL的原因。

对于ExtCreatePen的第一个参数，可使用第四章中所讨论的用在CreatePen上的所有样式。您可使用PS_GEOMETRIC另外组合这些样式（其中iWidth参数以逻辑单位表示线宽并能够转换），或者使用PS_COSMETIC（其中iWidth参数必须是1）。Windows 98中，虚线或点画线样式的画笔必须是PS_COSMETIC，在Windows NT中取消了这个限制。

CreatePen的一个参数表示颜色；ExtCreatePen的相应参数不只表示颜色，它还使用画刷给PS_GEOMETRIC画笔内部着色。该画刷甚至能透过位图定义。

在绘制宽线段时，我们可能要关注线段端点的外观。在连结直线或曲线时，可能还要关注线段间连结点的外观。画笔由CreatePen建立时，这些端点及连结点通常是圆形的；使用ExtCreatePen建立画笔时我们可以选择。（实际上 ，在Windows 98中，只有在使用画笔实作绘图路径时我们可以选择；在Windows NT中要更加灵活）。宽线段的端点可以使用ExtCreatePen中的下列画笔样式定义：

PS_ENDCAP_ROUND

PS_ENDCAP_SQUARE

PS_ENDCAP_FLAT

「square」样式与「flat」样式的不同点是：前者将线伸展到一半宽。与端点类似，绘图路径中线段间的连结点可通过如下样式设定：

PS_JOIN_ROUND

PS_JOIN_BEVEL

PS_JOIN_MITER

「bevel」样式将连结点切断；「miter」样式将连结点变为箭头。程序17-9所示的ENDJOIN是对此的一个较好的说明。

程序17-9 ENDJOIN

ENDJOIN.C

```
/*---------------------------------------------------------------------------
ENDJOIN.C -- Ends and Joins Demo
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EndJoin") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Ends and Joins Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
LRESULT CALLBACK WndProc ( HWND hwnd, UINT iMsg, WPARAM wParam,LPARAM lParam)
{
  static int iEnd[] = {PS_ENDCAP_ROUND,PS_ENDCAP_SQUARE,PS_ENDCAP_FLAT } ;
  static int iJoin[]= {PS_JOIN_ROUND,PS_JOIN_BEVEL,PS_JOIN_MITER } ;
  static int cxClient, cyClient ;
  HDC hdc ;
  int i ;
  LOGBRUSH ib ;
  PAINTSTRUCT ps ;

  switch (iMsg)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SetMapMode (hdc, MM_ANISOTROPIC) ;
    SetWindowExtEx (hdc, 100, 100, NULL) ;
    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

    lb.lbStyle = BS_SOLID ;
    lb.lbColor = RGB (128, 128, 128) ;
    lb.lbHatch = 0 ;

    for (i = 0 ; i < 3 ; i++)
    {
      SelectObject (hdc, ExtCreatePen (PS_SOLID | PS_GEOMETRIC |
        iEnd [i] | iJoin [i], 10, &lb, 0, NULL)) ;
      BeginPath (hdc) ;
      MoveToEx (hdc, 10 + 30 * i, 25, NULL) ;
      LineTo (hdc, 20 + 30 * i, 75) ;
      LineTo (hdc, 30 + 30 * i, 25) ;
```

```
    EndPath (hdc) ;
    StrokePath (hdc) ;

    DeleteObject (
      SelectObject (hdc,GetStockObject (BLACK_PEN))) ;

    MoveToEx (hdc, 10 + 30 * i, 25, NULL) ;
    LineTo (hdc, 20 + 30 * i, 75) ;
    LineTo (hdc, 30 + 30 * i, 25) ;
   }
   EndPaint (hwnd, &ps) ;
   return 0 ;
  case WM_DESTROY:
   PostQuitMessage (0) ;
   return 0 ;
  }
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```

程序使用上述端点和连结点样式画了三条V形的宽线段。程序也使用备用黑色画笔画了三条同样的线。这样就将宽线与通常的细线做了比较。结果如图17-4所示。

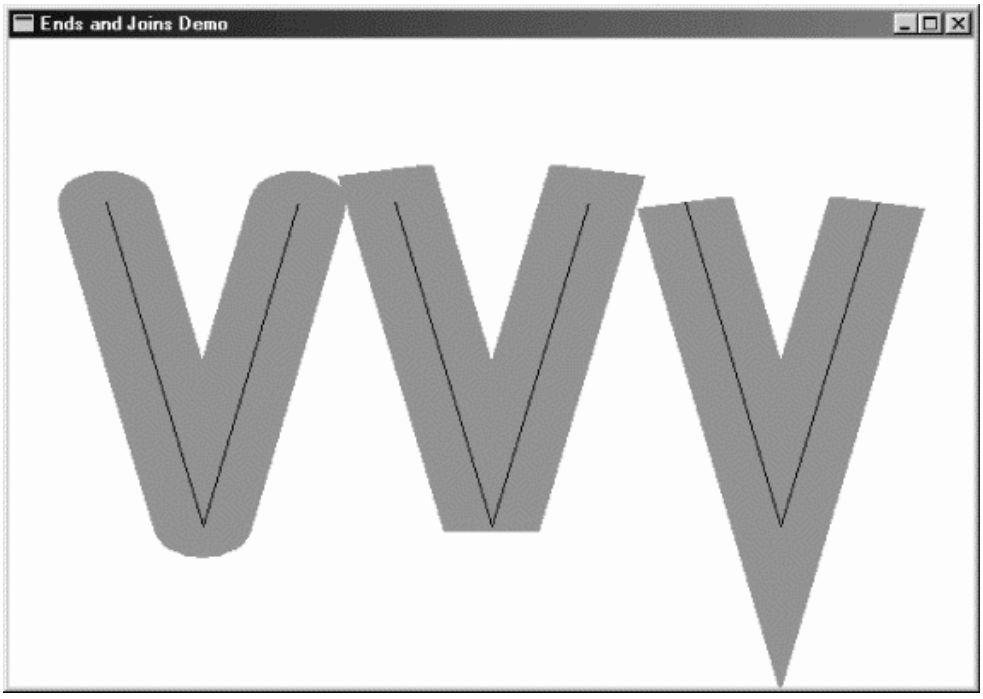


图17-4 ENDJOIN的屏幕显示

现在大家该明白为什么Windows支持StrokePath函数了：如果分别画两条直线，GDI不得不在每一条在线使用端点。只有在绘图路径定义中，GDI知道线段是连结的并使用线段的连结点。

四个范例程序

这究竟有什么好处呢？仔细考虑一下：轮廓字体的字符由一系列坐标值定义，这些坐标定义了直线和转折线。因而，直线及曲线能成为绘图路径定义的一部分。

确实可以！程序17-10所示的FONTOUT1程序对此做了展示。

程序17-10 FONTOUT1

FONTOUT1.C

```
/*------------------------------------------------------------------------
FONTOUT1.C -- Using Path to Outline Font
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\eztest\\ezfont.h"

TCHAR szAppName [] = TEXT ("FontOut1") ;
TCHAR szTitle [] = TEXT ("FontOut1: Using Path to Outline Font") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  static TCHAR szString [] = TEXT ("Outline") ;
  HFONT hFont ;
  SIZE  size ;

  hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1440, 0, 0, TRUE) ;
  SelectObject (hdc, hFont) ;
  GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
  BeginPath (hdc) ;
  TextOut (hdc, (cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
    szString, lstrlen (szString)) ;
  EndPath (hdc) ;
  StrokePath (hdc) ;
  SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
  DeleteObject (hFont) ;
}
```

此程序和本章后面的程序都使用了前面所示的EZFONT和FONTDEMO文件。

程序建立了144点的TrueType字体并呼叫GetTextExtentPoint32函数取得文字方块的大小。然后，呼叫绘图路径定义中的TextOut函数使文字在显示区域窗口中处于中心的位置。因为对TextOut函数的呼叫是被绘图路径设定命令所包围的（即BeginPath和EndPath呼叫之间）程序中进行的，GDI不立即显示文字。相反，程序将字符轮廓储存在绘图路径定义中。

在绘图路径定义结束后，FONTOUT1呼叫StrokePath。因为设备内容中未选入指定的画笔，所以GDI仅仅使用内定画笔绘制字符轮廓，如图17-5所示。

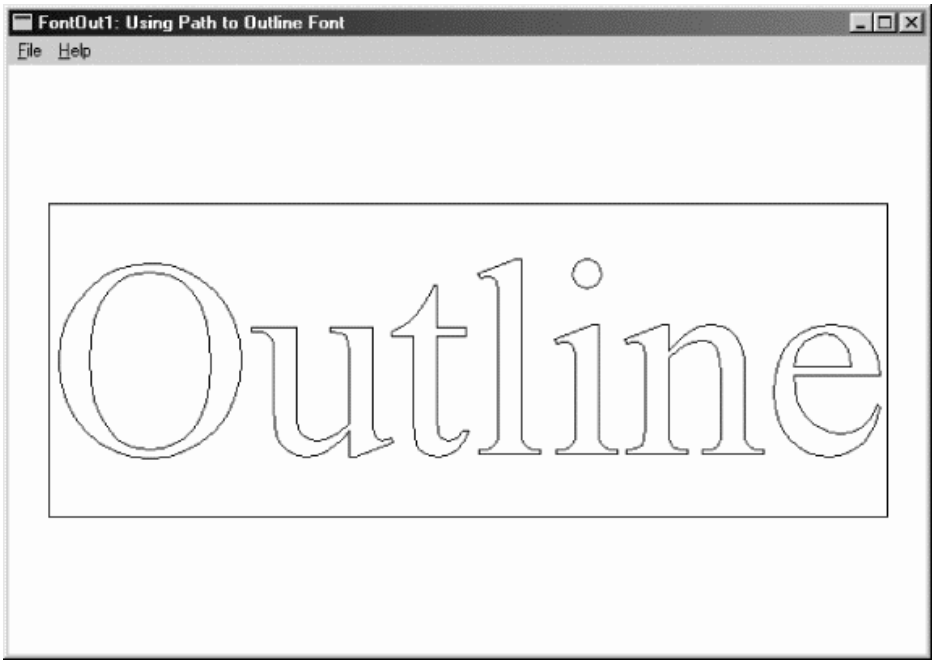


图17-5 FONTOUT1的屏幕显示

现在我们都得到什么呢？我们已经获得了所期望的轮廓字符，但是字符串外面为什么会围绕着矩形呢？

回想一下，文字背景模式使用内定的OPAQUE，而不是TRANSPARENT。该矩形就是文字方块的轮廓。这清晰地展示了在内定的OPAQUE模式下GDI绘制文字时所使用的两个步骤：首先绘制一个填充的矩形，接着绘制字符。文字方块矩形的轮廓也因此成为绘图路径的一部分。

使用ExtCreatePen函数就能够使用内定画笔以外的东西绘制字体字符的轮廓。程序17-11所示的FONTOUT2对此做了展示。

程序17-11 FONTOUT2

FONTOUT2.C

```
/*-------------------------------------------------------------------------
FONTOUT2.C -- Using Path to Outline Font
(c) Charles Petzold, 1998
-------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\eztest\\ezfont.h"

TCHAR szAppName [] = TEXT ("FontOut2") ;
TCHAR szTitle [] = TEXT ("FontOut2: Using Path to Outline Font") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  static TCHAR szString [] = TEXT ("Outline") ;
  HFONT hFont ;
  LOGBRUSH lb ;
  SIZE size ;

  hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1440, 0, 0, TRUE) ;
  SelectObject (hdc, hFont) ;
  SetBkMode (hdc, TRANSPARENT) ;

  GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
  BeginPath (hdc) ;
  TextOut (hdc, (cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
    szString, lstrlen (szString)) ;
  EndPath (hdc) ;
  lb.lbStyle = BS_SOLID ;
  lb.lbColor = RGB (255, 0, 0) ;
  lb.lbHatch = 0 ;

  SelectObject (hdc, ExtCreatePen (PS_GEOMETRIC | PS_DOT,
    GetDeviceCaps (hdc, LOGPIXELSX) / 24, &lb, 0, NULL)) ;
  StrokePath (hdc) ;
  DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
  SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
  DeleteObject (hFont) ;
}
```

此程序呼叫StrokePath之前建立（并选入设备内容）一个3点（1/24英寸）宽的红色点线笔。程序在Windows NT下执行时，结果如图17-6所示。Windows 98不支持超过1像素宽的非实心笔，因此Windows 98将以实心的红色笔绘制。

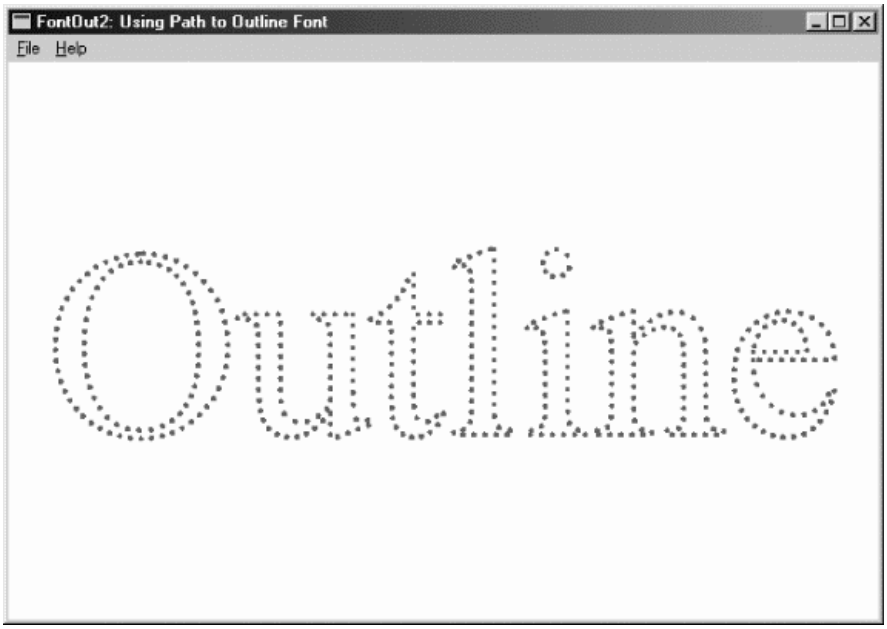


图17-6 FONTOUT2的屏幕显示

您也可以使用绘图路径定义填充区域。请用前面两个程序所示的方法建立绘图路径，选择一种填充图案，然后呼叫FillPath。能呼叫的另一个函数是StrokeAndFillPath，它绘制绘图路径的轮廓并用一个函数呼叫将其填充。

StrokeAndFillPath函数如程序17-12 FONTFILL所展示。

程序17-12 FONTFILL

FONTFILL.C

```
/*---------------------------------------------------------------------------
FONTFILL.C -- Using Path to Fill Font
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\eztest\\ezfont.h"

TCHAR szAppName [] = TEXT ("FontFill") ;
TCHAR szTitle [] = TEXT ("FontFill: Using Path to Fill Font") ;
void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  static TCHAR szString [] = TEXT ("Filling") ;
  HFONT hFont ;
  SIZE size ;

  hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1440, 0, 0, TRUE) ;
  SelectObject (hdc, hFont) ;
  SetBkMode (hdc, TRANSPARENT) ;

  GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
  BeginPath (hdc) ;
  TextOut (hdc, (cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
    szString, lstrlen (szString)) ;
  EndPath (hdc) ;
  SelectObject (hdc, CreateHatchBrush (HS_DIAGCROSS, RGB (255, 0, 0))) ;
  SetBkColor (hdc, RGB (0, 0, 255)) ;
  SetBkMode (hdc, OPAQUE) ;
  StrokeAndFillPath (hdc) ;
  DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
  SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
  DeleteObject (hFont) ;
}
```

FONTFILL使用内定画笔绘制绘图路径的轮廓，但使用HS_DIAGCROSS样式建立红色的阴影画刷。注意程序在建立绘图路径时将背景模式设定为TRANSPARENT，在填充绘图路径时又将其重设为OPAQUE，这样它能够为区域图案使用蓝色的背景颜色。结果如图17-7所示。

您可能想在本程序中尝试几个变更，观察变更的影响。首先，如果您将第一个SetBkMode呼叫变为注解，将得到由图案而不是字符本身所覆盖的文字方块背景。这通常不是我们实际所需要的，但确实可这样做。

此外，填充字符及将它们用做剪裁时，您可能想有效地放弃内定的ALTERNATE多边填充模式。我的经验表示：如果使用WINDING填充模式，则构建TrueType字体以避免出现奇怪的现象（例如「O」的内部被填充），但使用ALTERNATE模式更安全。

图17-7 FONTFILL的屏幕显示

最后，可使用一个绘图路径，因此也是一个TrueType字体，来定义剪裁区域。如程序17-13 FONTCLIP所示。

程序17-13 FONTCLIP

FONTCLIP.C

```
/*------------------------------------------------------------------------
FONTCLIP.C -- Using Path for Clipping on Font
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\eztest\\ezfont.h"

TCHAR szAppName [] = TEXT ("FontClip") ;
TCHAR szTitle [] = TEXT ("FontClip: Using Path for Clipping on Font") ;

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  static TCHAR szString [] = TEXT ("Clipping") ;
  HFONT hFont ;
  int  y, iOffset ;
  POINT pt [4] ;
  SIZE size ;

  hFont = EzCreateFont (hdc, TEXT ("Times New Roman"), 1200, 0, 0, TRUE) ;
  SelectObject (hdc, hFont) ;
```

```
  GetTextExtentPoint32 (hdc, szString, lstrlen (szString), &size) ;
  BeginPath (hdc) ;
  TextOut (hdc, (cxArea - size.cx) / 2, (cyArea - size.cy) / 2,
    szString, lstrlen (szString)) ;
  EndPath (hdc) ;
  // Set clipping area
  SelectClipPath (hdc, RGN_COPY) ;
  // Draw Bezier splines
  iOffset = (cxArea + cyArea) / 4 ;
  for (y = -iOffset ; y < cyArea + iOffset ; y++)
  {
    pt[0].x = 0 ;
    pt[0].y = y ;
    pt[1].x = cxArea / 3 ;
    pt[1].y = y + iOffset ;
    pt[2].x = 2 * cxArea / 3 ;
    pt[2].y = y - iOffset ;
    pt[3].x = cxArea ;
    pt[3].y = y ;
    SelectObject (hdc, CreatePen (PS_SOLID, 1,
      RGB (rand () % 256, rand () % 256, rand () % 256))) ;
    PolyBezier (hdc, pt, 4) ;
    DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
  }
  DeleteObject (SelectObject (hdc, GetStockObject (WHITE_BRUSH))) ;
  SelectObject (hdc, GetStockObject (SYSTEM_FONT)) ;
  DeleteObject (hFont) ;
}
```

程序中故意不使用SetBkMode呼叫以实作不同的效果。程序在绘图路径支架中绘制一些文字，然后呼叫SelectClipPath。接着使用随机颜色绘制一系列贝塞尔曲线。

如果FONTCLIP程序使用TRANSPARENT选项呼叫SetBkMode，贝塞尔曲线将被限制在字符轮廓的内部。在内定OPAQUE选项的背景模式下，剪裁区域被限制在文字方块内部而不是文字内部。如图17-8所示。

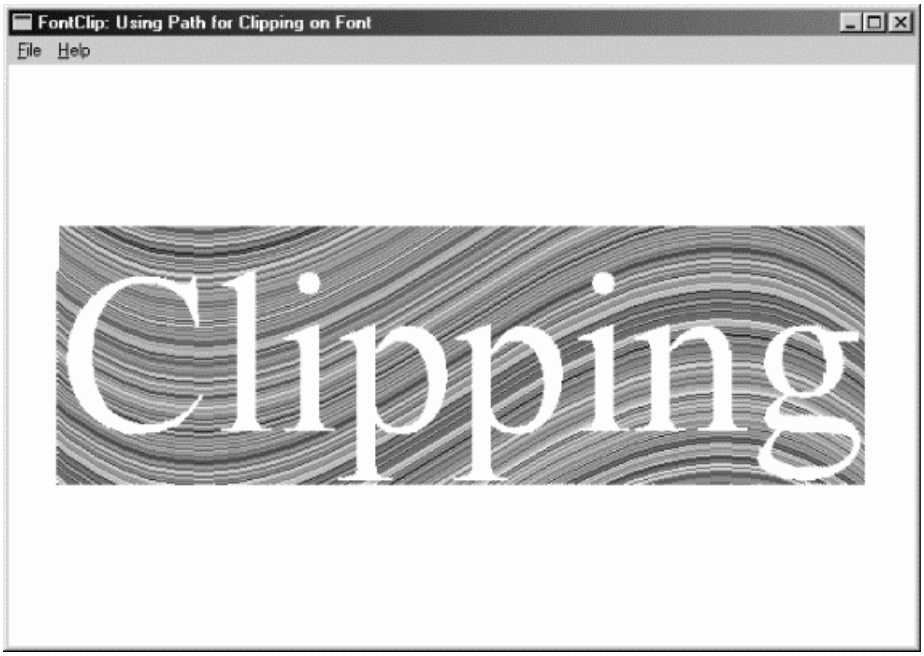


图17-8 FONTCLIP得屏幕显示

您或许会想在FONTCLIP中插入SetBkMode呼叫来观察TRANSPARENT选项的变化。

FONTDEMO外壳程序允许您打印并显示这些效果，甚至允许您尝试自己的一些特殊效果。

第十八章 MetaFile

MetaFile和向量图形的关系，就像位图和位映像图形的关系一样。位图通常来自实际的图像，而MetaFile则大多是通过计算机程序人为建立的。MetaFile由一系列与图形函数呼叫相同的二进制记录组成，这些记录一般用于绘制直线、曲线、填入的区域和文字等。

「画图（paint)」程序建立位图，而「绘图(draw)」程序建立MetaFile。在优秀的绘图程序中，能轻易地「抓住」某个独立的图形对象（例如一条直线）并将它移动到其它位置。这是因为组成图形的每个成员都是以单独的记录储存的。在画图程序中，这是不可能的 – 您通常都会局限于删除或插入位图矩形块。

由于MetaFile以图形绘制命令描述图像，因此可以对图像进行缩放而不会失真。位图则不然，如果以二倍大小来显示位图，您却无法得到二倍的分辨率，而只是在水平和垂直方向上重复位图的位。

MetaFile可以转换为位图，但是会丢失一些信息：组成MetaFile的图形对象将不再是独立的，而是被合并进大的图像。将位图转换为MetaFile要艰难得多，一般仅限于非常简单的图像，而且它需要大量处理来分析边界和轮廓。而MetaFile可以包含绘制位图的命令。

虽然MetaFile可以作为图片剪辑储存在磁盘上，但是它们大多用于程序通过剪贴簿共享图片的情况。由于MetaFile将图片描述为图像函数呼叫的集合，因而它们既比位图占用更少的空间，又比位图更与设备无关。

Microsoft Windows支持两种MetaFile格式和支持这些格式的两组函数。我首先讨论从Windows 1.0到目前的32位Windows版本都支持的MetaFile函数，然后讨论为32位Windows系统开发的「增强型MetaFile」。增强型MetaFile在原有MetaFile的基础上有了一些改进，应该尽可能地加以利用。

旧的 MetaFile 格式

MetaFile既能够暂时储存在内存中，也能够以文件的形式储存在磁盘上。对应用程序来说，两者区别不大，尤其是由Windows来处理磁盘上储存和加载MetaFile资料的文件I/O时，更是如此。

内存MetaFile的简单利用

如果呼叫CreateMetaFile函数来建立MetaFile设备内容，Windows就会以早期的格式建立一个MetaFile，然后您可以使用大部分GDI绘图函数在该MetaFile设备内容上进行绘图。这些GDI呼叫并不在任何具体的设备上绘图，相反地，它们被储存在MetaFile中。当关闭MetaFile设备内容时，会得到MetaFile的句柄。这时就可以在某个具体的设备内容上「播放」这个MetaFile，这与直接执行MetaFile中GDI函数的效果等同。

CreateMetaFile只有一个参数，它可以是NULL或文件名称。如果是NULL，则MetaFile储存在内存中。如果是文件名称（以.WMF作为「Windows MetaFile」的扩展名），则MetaFile储存在磁盘文件中。

程序18-1中的MetaFile显示了在WM_CREATE消息处理期间建立内存MetaFile的方法，并在

WM_PAINT消息处理期间将图像显示100遍。

程序18-1 MetaFile

MetaFile.C

```
/*---------------------------------------------------------------------------
MetaFile.C -- MetaFile Demonstration Program
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("MetaFile") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("MetaFile Demonstration"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HMETAFILE hmf ;
  static int cxClient, cyClient ;
  HBRUSH hBrush ;
  HDC hdc, hdcMeta ;
  int x, y ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    hdcMeta= CreateMetaFile (NULL) ;
    hBrush= CreateSolidBrush (RGB (0, 0, 255)) ;
```

```
    Rectangle (hdcMeta, 0, 0, 100, 100) ;

    MoveToEx (hdcMeta, 0, 0, NULL) ;
    LineTo (hdcMeta, 100, 100) ;
    MoveToEx (hdcMeta, 0, 100, NULL) ;
    LineTo (hdcMeta, 100, 0) ;

    SelectObject (hdcMeta, hBrush) ;
    Ellipse (hdcMeta, 20, 20, 80, 80) ;

    hmf = CloseMetaFile (hdcMeta) ;

    DeleteObject (hBrush) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    SetMapMode (hdc, MM_ANISOTROPIC) ;
    SetWindowExtEx (hdc, 1000, 1000, NULL) ;
    SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;

    for (x = 0 ; x < 10 ; x++)
      for (y = 0 ; y < 10 ; y++)
      {
        SetWindowOrgEx (hdc, -100 * x, -100 * y, NULL) ;
        PlayMetaFile (hdc, hmf) ;
      }
      EndPaint (hwnd, &ps) ;
      return 0 ;

  case WM_DESTROY:
    DeleteMetaFile (hmf) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这个程序展示了在使用内存MetaFile时所涉及的4个MetaFile函数的用法。第一个是CreateMetaFile。在WM_CREATE消息处理期间用NULL参数呼叫该函数，并传回MetaFile设备内容的句柄。然后，MetaFile利用这个MetaFileDC来绘制两条直线和一个蓝色椭圆。这些函数呼叫以二进制形式储存在MetaFile中。CloseMetaFile函数传回MetaFile的句柄。因为以后还要用到该MetaFile句柄，所以把它储存在静态变量。

该MetaFile包含GDI函数呼叫的二进制表示码，它们是两个MoveToEx呼叫、两个LineTo呼叫、一个SelectObject呼叫（指定蓝色画刷）和一个Ellipse呼叫。坐标没有指定任何映像方式或转换，它们只是作为数值数据被储存在MetaFile中。

在WM_PAINT消息处理期间，MetaFile设定一种映像方式并呼叫PlayMetaFile在窗口中绘制对象100次。MetaFile中函数呼叫的坐标按照目的设备内容的目前变换方式加以解释。在呼叫PlayMetaFile时，事实上是在重复地呼叫最初在WM_CREATE消息处理期间建立MetaFile时，在CreateMetaFile和CloseMetaFile之间所做的所有呼叫。

和任何GDI对象一样，MetaFile对象也应该在程序终止前被删除。这是在WM_DESTROY消息处理期间用DeleteMetaFile函数处理的工作。

MetaFile程序的结果如图18-1所示。

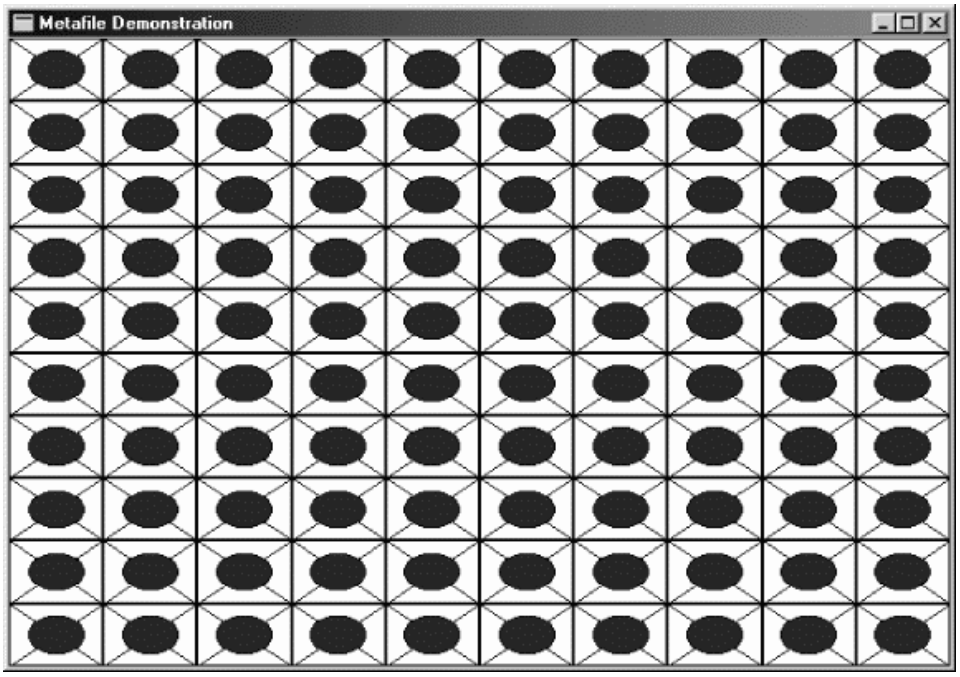


图18-1 MetaFile程序执行结果显示

将MetaFile储存在磁盘上

在上面的例子中，CreateMetaFile的NULL参数表示要建立储存在内存中的MetaFile。我们也可以建立作为文件储存在磁盘上的MetaFile，这种方法对于大的MetaFile比较合适，因为可以节省内存空间。而另一方面，每次使用磁盘上的MetaFile时，就需要存取磁盘。

要把MetaFile转换为使用MetaFile磁盘文件的程序，必须把CreateMetaFile的NULL参数替换为文件名称。在WM_CREATE处理结束时，可以用MetaFile句柄来呼叫DeleteMetaFile，这样句柄被删除，但是磁盘文件仍然被储存着。

在处理WM_PAINT消息处理期间，可以通过呼叫GetMetaFile来取得此磁盘文件的MetaFile句柄：

```
hmf = GetMetaFile (szFileName) ;
```

现在就可以像前面那样显示这个MetaFile。在WM_PAINT消息处理结束时，可以用下面的叙述删除该MetaFile句柄：

```
DeleteMetaFile (hmf) ;
```

在开始处理WM_DESTROY消息时，不必删除MetaFile，因为它已经在WM_CREATE消息和每个WM_PAINT消息结束时被删除了，但是仍然需要删除磁盘文件：

```
DeleteFile (szFileName) ;
```

当然，除非您想储存该文件。

正如在第十章讨论过的，MetaFile也可以作为使用者自订资源。您可以简单地把它当作数据块加载。如果您有一块包含MetaFile内容的资料，那么您可以使用

```
hmf = SetMetaFileBitsEx (iSize, pData) ;
```

来建立MetaFile。SetMetaFileBitsEx有一个对应的函数 – GetMetaFileBitsEx，此函数将MetaFile的内容复制到内存块中。

老式MetaFile与剪贴簿

老式MetaFile有个讨厌的缺陷。如果您具有老式MetaFile的句柄，那么，当您在显示MetaFile时如何确定它的大小呢？除非您深入分析MetaFile的内部结构，否则无法得知。

此外，当程序从剪贴簿取得老式MetaFile时，如果MetaFile被定义为在MM_ISOTROPIC或MM_ANISOTROPIC映像方式下显示，则此程序在使用该MetaFile时具有最大程度的灵活性。程序收到该MetaFile后，就可以在显示它之前简单地通过设定视埠的范围来缩放图像。然而，如果MetaFile内的映像方式被设定为MM_ISOTROPIC或MM_ANISOTROPIC，则收到该MetaFile的程序将无法继续执行。程序仅能在显示MetaFile之前或之后进行GDI呼叫，不允许在显示MetaFile当中进行GDI呼叫。

为了解决这些问题，老式MetaFile句柄不直接放入剪贴簿供其它程序取得，而是作为「MetaFile图片」（MetaFilePICT结构型态）的一部分。此结构使得从剪贴簿上取得MetaFile图片的程序能够在显示MetaFile之前设定映像方式和视埠范围。

MetaFilePICT结构的长度为16个字节，定义如下：

```
typedef struct tagMetaFilePICT
{
  LONG mm ; // mapping mode
  LONG xExt ; // width of the MetaFile image
  LONG yExt ; // height of the MetaFile image
  LONG hMF ; // handle to the MetaFile
}
MetaFilePICT ;
```

对于MM_ISOTROPIC和MM_ANISOTROPIC以外的所有映像方式，图像大小用xExt和yExt值表示，其单位是由mm给出的映像方式的单位。利用这些信息，从剪贴簿复制MetaFile图片结构的程序就能够确定在显示MetaFile时所需的显示空间。建立该MetaFile的程序可以将这些值设定为输入MetaFile的GDI绘制函数中所使用的最大的x坐标和y坐标值。

在MM_ISOTROPIC和MM_ANISOTROPIC映射方式下，xExt和yExt字段有不同的功能。我们在第五章中曾介绍过一个程序，该程序为了在GDI函数中使用与图像实际尺寸无关的逻辑单位而采用MM_ISOTROPIC或MM_ANISOTROPIC映射方式。当程序只想保持纵横比而可以忽略图形显示平面的大小时，采用MM_ISOTROPIC模式；反之，当不需要考虑纵横比时采用MM_ANISOTROPIC模式。您也许还记得，第五章中在程序将映像方式设定为MM_ISOTROPIC或MM_ANISOTROPIC后，通常会呼叫SetWindowExtEx和SetViewportExtEx。SetWindowExtEx呼叫使用逻辑单位来指定程序在绘制时使用的单位，而SetViewportExtEx呼叫使用的设备单位大小则取决于图形显示平面（例如，窗口显示区域的大小）。

如果程序为剪贴簿建立了MM_ISOTROPIC或MM_ANISOTROPIC方式的MetaFile，则该MetaFile本身不应包含对SetViewportExtEx的呼叫，因为该呼叫中的设备单位应该依据建立MetaFile的程序的显示平面，而不是依据从剪贴簿读取并显示MetaFile的程序的显示平面。从剪贴簿取得MetaFile的程序可以利用xExt和yExt值来设定合适的视埠范围以便显示MetaFile。但是当映像方式是MM_ISOTROPIC或MM_ANISOTROPIC时，MetaFile本身包含设定窗口范围的呼叫。MetaFile内的GDI绘图函数的坐标依据这些窗口的范围。

建立MetaFile和MetaFile图片遵循以下规则：

设定MetaFilePICT结构的mm字段来指定映像方式。

对于MM_ISOTROPIC和MM_ANISOTROPIC以外的映像方式，xExt与yExt字段设定为图像的宽和高，单位与mm字段相对应。对于在MM_ISOTROPIC或MM_ANISOTROPIC方式下显示的MetaFile，工作要复杂一些。在MM_ANISOTROPIC模式下，当程序既不对图片大小跟纵横

比给出任何建议信息时，xExt和yExt的值均为零。在这两种模式下，如果xExt和yExt的值为正数，它们就是以0.01mm单位（MM_HIMETRIC单位）表示该图像的宽度和高度。在MM_ISOTROPIC方式下，如果xExt和yExt为负值，它们就指出了图像的纵横比而不是大小。

在MM_ISOTROPIC和MM_ANISOTROPIC映像方式下，MetaFile本身含有对SetWindowExtEx的呼叫，也可能有对SetWindowOrgEx的呼叫。亦即，建立MetaFile的程序在MetaFile设备内容中呼叫这些函数。MetaFile一般不会包含对SetMapMode、SetViewportExtEx或SetViewportOrgEx的呼叫。

MetaFile应该是内存MetaFile，而不是MetaFile文件。

这里有一段范例程序代码，它建立MetaFile并将其复制到剪贴簿。如果MetaFile使用MM_ISOTROPIC或MM_ANISOTROPIC映像方式，则该MetaFile的第一个呼叫应该设定窗口范围（在其它模式中，窗口的大小是固定的）。无论在哪种模式下，窗口的位置应如下设定：

hdcMeta = CreateMetaFile (NULL) ;

SetWindowExtEx (hdcMeta, ...) ;

SetWindowOrgEx (hdcMeta, ...) ;

MetaFile绘图函数中的坐标决定于这些窗口范围和窗口原点。当程序使用GDI呼叫在MetaFile设备内容中绘制完成后，关闭MetaFile以得到MetaFile句柄：

hmf = CloseMetaFile (hdcMeta) ;

该程序还需要定义指向MetaFilePICT型态结构的指针，并为此结构配置一块整体内存：

```
GLOBALHANDLE  hGlobal ;
LPMETAFILEPICT  pMFP ;
//其它行程序
hGlobal= GlobalAlloc (GHND | GMEM_SHARE, sizeof (METAFILEPICT)) ;
pMFP = (LPMETAFILEPICT) GlobalLock (hGlobal) ;
//接着，程序设定该结构的4个字段：
pMFP->mm = MM_... ;
pMFP->xExt = ... ;
pMFP->yExt = ... ;
pMFP->hMF = hmf ;
GlobalUnlock (hGlobal) ;
```

然后，程序将包含有MetaFile图片的整体内存块传送给剪贴簿：

```
OpenClipboard (hwnd) ;
EmptyClipboard () ;
SetClipboardData (CF_MetaFilePICT, hGlobal) ;
CloseClipboard () ;
```

完成这些呼叫后，hGlobal句柄（包含MetaFile图片结构的内存块）和hmf句柄（MetaFile本身）就对建立它们的程序失效了。

现在来看一看难的部分。当程序从剪贴簿取得MetaFile并显示它时，必须完成下列步骤：

程序利用MetaFile图片结构的mm字段设定映像方式。

对于MM_ISOTROPIC或MM_ANISOTROPIC以外的映像方式，程序用xExt和yExt值设定剪贴矩形或简单地设定图像大小。而在MM_ISOTROPIC和MM_ANISOTROPIC映像方式，程序使用xExt和yExt来设定视埠范围。

然后，程序显示MetaFile。

下面程序代码，首先打开剪贴簿，得到MetaFile图片结构句柄并将其锁定：

OpenClipboard (hwnd) ;

hGlobal = GetClipboardData (CF_MetaFilePICT) ;

pMFP = (LPMetaFilePICT) GlobalLock (hGlobal) ;

现在可以储存目前设备内容的属性，并将映像方式设定为结构中的mm值：

SaveDC (hdc) ;

SetMappingMode (pMFP->mm) ;

如果映像方式不是MM_ISOTROPIC或MM_ANISOTROPIC，则可以用xExt和yExt的值设定剪贴矩形。由于这两个值是逻辑单位，必须用LPtoDP将其转换为用于剪贴矩形的设备单位的坐标。也可以简单地储存这些值以掌握图像的大小。

对于MM_ISOTROPIC或MM_ANISOTROPIC映像方式，xExt和yExt用来设定视埠范围。下面有一个用来完成此项任务的函数，如果xExt和yExt没有建议的大小，则该函数假定cxClient和cyClient分别表示MetaFile显示区域的像素高度和宽度。

```
void PrepareMetaFile ( HDC hdc, LPMetaFilePICT pmfp,
                  int cxClient, int cyClient)
{
  int xScale, yScale, iScale ;
  SetMapMode (hdc, pmfp->mm) ;
  if (pmfp->mm == MM_ISOTROPIC || pmfp->mm == MM_ANISOTROPIC)
  {
    if (pmfp->xExt == 0)
      SetViewportExtEx (hdc, cxClient, cyClient, NULL) ;
    else if (pmfp->xExt > 0)
      SetViewportExtEx (hdc,
      pmfp->xExt * GetDeviceCaps (hdc, HORZRES) /
      GetDeviceCaps (hdc, HORZSIZE) / 100),
      pmfp->yExt * GetDeviceCaps (hdc, VERTRES) /
      GetDeviceCaps (hdc, VERTSIZE) / 100), NULL) ;
    else if (pmfp->xExt < 0)
    {
      xScale = 100 *cxClient * GetDeviceCaps (hdc, HORZSIZE) /
        GetDeviceCaps (hdc, HORZRES) / -pmfp->xExt ;
      lScale = 100 *cyClient * GetDeviceCaps (hdc, VERTSIZE) /
        GetDeviceCaps (hdc, VERTRES) / -pmfp->yExt ;
      iScale = min (xScale, yScale) ;
      SetViewportExtEx (hdc, -pmfp->xExt * iScale * GetDeviceCaps (hdc, HORZRES) /
        GetDeviceCaps (hdc, HORZSIZE) / 100, -pmfp->yExt * iScale
        * GetDeviceCaps (hdc, VERTRES) / GetDeviceCaps (hdc, VERTSIZE) / 100,
        NULL) ;
    }
  }
}
```

上面的程序代码假设xExt和yExt同时都为零、大于零或小于零，这三种状态之一。如果范围为零，表示没有建议大小或纵横比，视埠范围设定为显示MetaFile的区域。如果大于零，则xExt和yExt的值代表图像的建议大小，单位是0.01mm。GetDeviceCaps函数用来确定每0.01mm中包含的像素数，并且该值与MetaFile图片结构的范围值相乘。如果小于零，则xExt和yExt的值表示建议的纵横比而不是建议的大小。iScale的值首先根据对应cxClient和cyClient的毫米表示的纵横比计算出来，该缩放因子用于设定像素单位的视端口范围。

完成了上述工作后，可以设定视埠原点，显示MetaFile，并恢复设备内容：

PlayMetaFile (pMFP->hMF) ;

RestoreDC (hdc, -1) ;

然后，对内存块解锁并关闭剪贴簿：

GlobalUnlock (hGlobal) ;

CloseClipboard () ;

如果程序使用增强型MetaFile就可以省去这项工作。当某个应用程序将这些格式放入剪贴簿而另一个程序却要求从剪贴簿中获得其它格式时，Windows剪贴簿会自动在老式MetaFile和增强型MetaFile之间进行格式转换。

增强型 MetaFile

「增强型MetaFile」格式是在32位Windows版本中发表的。它包含一组新的函数呼叫、一对新的数据结构、新的剪贴簿格式和新的文件扩展名.EMF。

这种新的MetaFile格式最重要的改进是加入可通过函数呼叫取得的更丰富的表头信息，这种表头信息可用来帮助应用程序显示MetaFile图像。

有些增强型MetaFile函数使您能够在增强型MetaFile(EMF)格式和老式MetaFile格式（也称作Windows MetaFile(WMF)格式）之间来回转换。当然，这种转换很可能遇到麻烦，因为老式MetaFile格式并不支持某些，例如GDI绘图路径等，新的32位图形功能。

基本程序

程序18-2所示的EMF1建立并显示增强型MetaFile。

程序18-2 EMF1

EMF1.C

```
/*------------------------------------------------------------------------
EMF1.C -- Enhanced MetaFile Demo #1
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #1"),
    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 ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static HENHMETAFILE hemf ;
  HDC  hdc, hdcEMF ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    hdcEMF = CreateEnhMetaFile (NULL, NULL, NULL, NULL) ;

    Rectangle (hdcEMF, 100, 100, 200, 200) ;

    MoveToEx (hdcEMF, 100, 100, NULL) ;
    LineTo (hdcEMF, 200, 200) ;

    MoveToEx (hdcEMF, 200, 100, NULL) ;
    LineTo (hdcEMF, 100, 200) ;

    hemf = CloseEnhMetaFile (hdcEMF) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    rect.left = rect.right / 4 ;
    rect.right = 3 * rect.right / 4 ;
    rect.top = rect.bottom / 4 ;
    rect.bottom = 3 * rect.bottom / 4 ;

    PlayEnhMetaFile (hdc, hemf, &rect) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    DeleteEnhMetaFile (hemf) ;

    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在EMF1的窗口消息处理程序处理WM_CREATE消息处理期间，程序首先通过呼叫CreateEnhMetaFile来建立增强型MetaFile。该函数有4个参数，但可以把它们都设为NULL。稍候我将说明这4个参数在非NULL情况下的使用方法。

和CreateMetaFile一样，CreateEnhMetaFile函数传回特定的设备内容句柄。该程序利用这个句柄绘制一个矩形和该矩形的两条对角线。这些函数呼叫及其参数被转换为二进制元的形式并储存在MetaFile中。

最后通过对CloseEnhMetaFile函数的呼叫结束了增强型MetaFile的建立并传回指向它的句柄。该文件句柄储存在HENHMetaFile型态的静态变量中。

在WM_PAINT消息处理期间，EMF1以RECT结构取得程序的显示区域窗口大小。通过调整结构中的4个字段，使该矩形的长和宽为显示区域窗口长和宽的一半并位于窗口的中央。然后EMF1呼叫PlayEnhMetaFile，该函数的第一个参数是窗口的设备内容句柄，第二个参数是该增强型MetaFile的句柄，第三个参数是指向RECT结构的指针。

在MetaFile的建立程序中，GDI得出整个MetaFile图像的尺寸。在本例中，图像的长和宽均为100个单位。在MetaFile的显示程序中，GDI将图像拉伸以适应PlayEnhMetaFile函数指定的矩形大小。EMF1在Windows下执行的三个执行实体如图18-2所示。

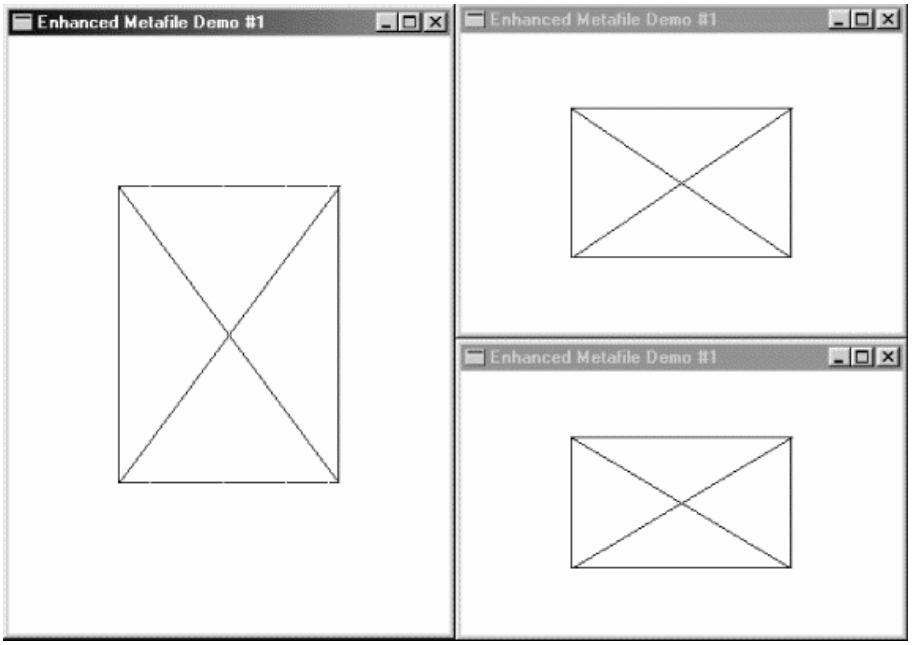


图18-2 EMF1得屏幕显示

最后，在WM_DESTROY消息处理期间，EMF1呼叫DeleteEnhMetaFile删除MetaFile。

让我们总结一下从EMF1程序学到的一些东西。

首先，该程序在建立增强型MetaFile时，画矩形和直线的函数所使用的坐标并不是实际意义上的坐标。您可以将它们同时加倍或都减去某个常数，而其结果不会改变。这些坐标只是在定义图像时说明彼此间的对应关系。

其次，为了适于在传递给PlayEnhMetaFile函数的矩形中显示，图像大小会被缩放。因此，如图18-2所示，图像可能会变形。尽管MetaFile坐标指出该图像是正方形的，但一般情况下我们却得不到这样的图像。而在某些时候，这又正是我们想要得到的图像。例如，将图像嵌入一段文书处理格式的文字中时，可能会要求使用者为图像指定矩形，并且确保整个图像恰好位于矩形中而不浪费空间。这样，使用者可通过适当调整矩形的大小来得到正确的纵横比。

然而有时候，您也许希望保留图像最初的纵横比，因为这一点对于表现视觉信息尤为重要。例如，警察的嫌疑犯草图既不能比原型胖也不能比原型瘦。或者您希望保留原来图像的度量尺寸，图像必须是两英寸高，否则就不能正常显示。在这种情况下，保留图像的原来尺寸就非常重要了。

同时也要注意MetaFile中画出的那些对角线似乎没有与矩形顶点相交。这是由于Windows在MetaFile中储存矩形坐标的方式造成的。稍后，会说明解决这个问题的方法。

揭开内幕

如果看一看MetaFile的内容会对MetaFile工作的方式有一个更好的理解。如果您有一个MetaFile文件，这将很容易做到，程序18-3中的EMF2程序建立了一个MetaFile。

程序18-3 EMF2

EMF2.C

```
/*---------------------------------------------------------------------------
EMF2.C -- Enhanced MetaFile Demo #2
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #2"),
    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 ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  HDC hdc, hdcEMF ;
  HENHMETAFILE hemf ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf2.emf"), NULL,
```

```
      TEXT ("EMF2\0EMF Demo #2\0")) ;

    if (!hdcEMF)
      return 0 ;

    Rectangle (hdcEMF, 100, 100, 200, 200) ;

    MoveToEx (hdcEMF, 100, 100, NULL) ;
    LineTo (hdcEMF, 200, 200) ;

    MoveToEx (hdcEMF, 200, 100, NULL) ;
    LineTo (hdcEMF, 100, 200) ;

    hemf = CloseEnhMetaFile (hdcEMF) ;

    DeleteEnhMetaFile (hemf) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    rect.left = rect.right / 4 ;
    rect.right = 3 * rect.right / 4 ;
    rect.top = rect.bottom / 4 ;
    rect.bottom = 3 * rect.bottom / 4 ;

    if (hemf = GetEnhMetaFile (TEXT ("emf2.emf")))
    {
      PlayEnhMetaFile (hdc, hemf, &rect) ;
      DeleteEnhMetaFile (hemf) ;
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在EMF1程序中，CreateEnhMetaFile函数的所有参数均被设定为NULL。在EMF2中，第一个参数仍旧设定为NULL，该参数还可以是设备内容句柄。GDI使用该参数在MetaFile表头中插入度量信息，很快我会讨论它。如果该参数为NULL，则GDI认为度量信息是由视频设备内容决定的。

CreateEnhMetaFile函数的第二个参数是文件名称。如果该参数为NULL（在EMF1中为NULL，但在EMF2中不为NULL），则该函数建立内存MetaFile。EMF2建立名为EMF2.EMF的MetaFile文件。

函数的第三个参数是RECT结构的地址，它指出了以0.01mm为单位的MetaFile的总大小。这是MetaFile表头数据中极其重要的信息（这是早期的Windows MetaFile格式的缺陷之一）。如果该参数为NULL，GDI会计算出尺寸。我比较喜欢让操作系统替我做这些事，所以将该参数设定为NULL。当应用程序对性能要求比较严格时，就需要使用该参数以避免让GDI处理太多东西。

最后的参数是描述该MetaFile的字符串。该字符串分为两部分：第一部分是以NULL字符结尾的应用程序名称（不一定是程序的文件名称），第二部分是描述视觉图像内容的说明，以两个NULL字符结尾。例如用C中的符号「\0」作为NULL字符，则该描述字符串可以是「LoonyCad V6.4\0Flying Frogs\0\0」。由于在C中通常会在使用的字符串末尾放入一个NULL字符，所以如EMF2所示，在末尾仅需一个「\0」。

建立完MetaFile后，与EMF1一样，EMF2也透过利用由CreateEnhMetaFile函数传回的设备内容句柄进行一些GDI函数呼叫。然后程序呼叫CloseEnhMetaFile删除设备内容句柄并取得完成

的MetaFile的句柄。

然后，在WM_CREATE消息还没处理完毕时，EMF2做了一些EMF1没有做的事情：在获得MetaFile句柄之后，程序呼叫DeleteEnhMetaFile。该操作释放了用于储存MetaFile的所有内存资源。然而，MetaFile文件仍然保留在磁盘驱动器中（如果愿意，您可以使用如DeleteFile的文件删除函数来删除该文件）。注意MetaFile句柄并不像EMF1中那样储存在静态变量中，这意味着在消息之间不需要储存它。

现在，为了使用该MetaFile，EMF2需要存取磁盘文件。这是在WM_PAINT消息处理期间透过呼叫GetEnhMetaFile进行的。MetaFile的文件名称是该函数的唯一参数，该函数传回MetaFile句柄。和EMF1一样，EMF2将这个文件句柄传递给PlayEnhMetaFile函数。该MetaFile图像在PlayEnhMetaFile函数的最后一个参数所指定的矩形中显示。与EMF1不同的是，EMF2在WM_PAINT消息结束之前就删除该MetaFile。此后每次处理WM_PAINT消息时，EMF2都会再次读取MetaFile，显示并删除它。

要记住，对MetaFile的删除操作仅是释放了用以储存MetaFile的内存资源而已，磁盘MetaFile甚至在程序执行结束后还保留在磁盘上。

由于EMF2留下了MetaFile文件，您可以看一看它的内容。图18-3显示了该程序建立的EMF2.EMF文件的一堆十六进制代码。

```
0000      01 00 00 00 88 00 00 00 64 00 00 00 64 00 00 00     ........d...d...
0010      C8 00 00 00 C8 00 00 00 35 0C 00 00 35 0C 00 00     ........5...5...
0020      6A 18 00 00 6A 18 00 00 20 45 4D 46 00 00 01 00     j...j... EMF....
0030      F4 00 00 00 07 00 00 00 01 00 00 00 12 00 00 00     ................
0040      64 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00     d...............
0050      40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00     @...............
0060      00 00 00 00 45 00 4D 00 46 00 32 00 00 00 45 00     ....E.M.F.2...E.
0070      4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00     M.F..D.e.m.o..
0080      23 00 32 00 00 00 00 00 2B 00 00 00 18 00 00 00     #.2.....+.......
0090      63 00 00 00 63 00 00 00 C6 00 00 00 C6 00 00 00     c...c...........
00A0      1B 00 00 00 10 00 00 00 64 00 00 00 64 00 00 00     ........d...d...
00B0      36 00 00 00 10 00 00 00 C8 00 00 00 C8 00 00 00     6...............
00C0      1B 00 00 00 10 00 00 00 C8 00 00 00 64 00 00 00     ............d...
00D0      36 00 00 00 10 00 00 00 64 00 00 00 C8 00 00 00     6.......d.......
00E0      0E 00 00 00 14 00 00 00 00 00 00 00 10 00 00 00     ................
00F0       14 00 00 00                                        ....
```

图18-3 EMF2.EMF的十六进制代码

图18-3所示的MetaFile是EMF2在Microsoft Windows NT 4下，视频显示器的分辨率为1024×768时建立的。同一程序在Windows 98下建立的MetaFile会比前者少12个字节，这一点将在稍后讨论。同样地，视频显示器的分辨率也影响MetaFile表头的某些信息。

增强型MetaFile格式使我们对MetaFile的工作方式有更深刻的理解。增强型MetaFile由可变长度的记录组成，这些记录的一般格式由ENHMETARECORD结构说明，它在WINGDI.H表头文件中定义如下：

```
typedef struct tagENHMETARECORD
{
  DWORD iType ; // record type
  DWORD nSize ; // record size
  DWORD dParm [1] ; // parameters
}
ENHMETARECORD ;
```

当然，那个只有一个元素的数组指出了数组元素的变量。参数的数量取决于记录型态。iType字段可以是定义在WINGDI.H文件中以前缀EMR_开始的近百个常数之一。nSize字段是总记录的大

小，包括iType和nSize字段以及一个或多个dParm字段。

有了这些知识后，让我们看一下图18-3。第一个字段型态为0x00000001，大小为0x00000088，所以它占据文件的前136个字节。记录型态为1表示常数EMR_HEADER。我们不妨把对表头纪录的讨论往后搁，先跳到位于第一个记录末尾的偏移量0x0088处。

后面的5个记录与EMF2建立MetaFile之后的5个GDI函数呼叫有关。该记录在偏移量0x0088处有一个值为0x0000002B的型态代码，这代表EMR_RECTANGLE，很明显是用于Rectangle呼叫的MetaFile记录。它的长度为0x00000018 （十进制24）字节，用以容纳4个32位参数。实际上Rectangle函数有5个参数，但是第一个参数，也就是设备内容句柄并未储存在MetaFile中，因为它没有实际意义。尽管在EMF2的函数呼叫中指定了矩形的顶点坐标分别是(100,100)和(200,200)，但4个参数中的2个是0x00000063 (99)，另外2个是0x000000C6 (198)。EMF2程序在Windows 98下建立的MetaFile显示出前两个参数是0x00000064 (100)，后2个参数是0x000000C7(199)。显然，在Rectangle参数储存到MetaFile之前，Windows对它们作了调整，但没有保持一致。这就是对角线端点与矩形顶点不能重合的原因。

其次，有4个16位记录与2个MoveToEx (0x0000001B或EMR_MOVETOEX)和LineTo (0x00000036或EMR_LINETO)呼叫有关。位于MetaFile中的参数与传递给函数的参数相同。

MetaFile以20个字节长的型态代码为0x0000000E或EMR_EOF（「end of file」）的记录结尾。

增强型MetaFile总是以表头纪录开始。它对应于ENHMETAHEADER型态的结构，定义如下：

```
typedef struct tagENHMETAHEADER
{
  DWORD iType ; // EMR_HEADER = 1
  DWORD nSize ; // structure size
  RECTL rclBounds ; // bounding rectangle in pixels
  RECTL rclFrame ; // size of image in 0.01 millimeters
  DWORD dSignature ; // ENHMETA_SIGNATURE = " EMF"
  DWORD nVersion ; // 0x00010000
  DWORD nBytes ; // file size in bytes
  DWORD nRecords ; // total number of records
  WORD nHandles ; // number of handles in handle table
  WORD sReserved ;
  DWORD nDescription ; // character length of description string
  DWORD offDescription ; // offset of description string in file
  DWORD nPalEntries ; // number of entries in palette
  SIZEL szlDevice ; // device resolution in pixels
  SIZEL szlMillimeters ; // device resolution in millimeters
  DWORD cbPixelFormat ; // size of pixel format
  DWORD offPixelFormat ; // offset of pixel format
  DWORD bOpenGL ; // FALSE if no OpenGL records
}
ENHMETAHEADER ;
```

这种表头纪录的存在可能是增强型MetaFile格式对早期Windows MetaFile所做的最为重要的改进。不需要对MetaFile文件使用文件I/O函数来取得这些表头信息。如果具有MetaFile句柄，就可以使用GetEnhMetaFileHeader函数：

GetEnhMetaFileHeader (hemf, cbSize, &emh) ;

第一个参数是MetaFile句柄。最后一个参数是指向ENHMETAHEADER结构的指针。第二个参数是该结构的大小。可以使用类似的GetEnh-MetaFileDescription函数取得描述字符串。

如上面所定义的，ENHMETAHEADER结构有100字节长，但在MF2.EMFMetaFile中，记录的大小包括描述字符串，所以大小为0x88，即136字节。而Windows 98MetaFile的表头纪录不包含ENHMETAHEADER结构的最后3个字段，这一点解释了12个字节的差别。

rclBounds字段是指出图像大小的RECT结构，单位是像素。将其从十六进制转换过来，我们看

到该图像正如我们希望的那样，其左上角位于(100,100)，右下角位于(200,200)。

rclFrame字段是提供相同信息的另一个矩形结构，但它是以0.01毫米为单位。在这种情况下，该文件显示两对角顶点分别位于(0x0C35,0x0C35)和(0x186A,0x186A)，用十进制表示为(3125,3125)和(6250,6250)的矩形。这些数字是怎么来的？我们很快就会明白。

dSignature字段始终为值ENHMETA_SIGNATURE或0x464D4520。这看上去是一个奇怪的数字，但如果将字节的排列顺序倒过来（就像Intel处理器在内存中储存多字节数那样）并转换成ASCII码，就变成字符串" EMF"。dVersion字段的值始终是0x00010000。

其后是nBytes字段，该字段在本例中是0x000000F4，这是该MetaFile的总字节数。nRecords字段（在本例中是0x00000007）指出了记录数 – 包括表头纪录、5个GDI函数呼叫和文件结束记录。

下面是两个十六位的字段。nHandles字段为0x0001。该字段一般指出MetaFile所使用的图形对象（如画笔、画刷和字体）的非内定句柄的数量。由于没有使用这些图形对象，您可能会认为该字段为零，但实际上GDI自己保留了第一个字段。我们将很快见到句柄储存在MetaFile中的方式。

下两个字段指出描述字符串的字符个数，以及描述字符串在文件中的偏移量，这里它们分别为0x00000012（十进制数18）和0x00000064。如果MetaFile没有描述字符串，则这两个字段均为零。

nPalEntries字段指出在MetaFile的调色盘表中条目的个数，本例中没有这种情况。

接着表头纪录包括两个SIZEL结构，它们包含两个32位字段，cx和cy。szlDevice字段（在MetaFile中的偏移量为0x0040）指出了以像素为单位的输出设备大小，szlMillimeters字段（偏移量为0x0050）指出了以毫米为单位的输出设备大小。在增强型MetaFile文件中，这个输出设备被称作「参考设备（reference device）」。它是依据作为第一个参数传递给CreateEnhMetaFile呼叫的句柄所指出的设备内容。如果该参数设为NULL，则GDI使用视频显示器。当EMF2建立上面所示的MetaFile时，正巧是在Windows NT上以1024×768显示模式工作，因此这就是GDI使用的参考设备。

GDI通过呼叫GetDeviceCaps取得此信息。EMF2.EMF中的szlDevice字段是0x0400 × 0x0300（即1024×768），它是以HORZRES和VERTRES作为参数呼叫GetDeviceCaps得到的。szlMillimeters字段是0x140 × 0xF0，或320 × 240，是以HORZSIZE和VERTSIZE作为参数呼叫GetDeviceCaps得到的。

通过简单的除法就可以得出像素为0.3125mm高和0.3125mm宽，这就是前面描述的GDI计算rclFrame矩形尺寸的方法。

在MetaFile中，ENHMETAHEADER结构后跟一个描述字符串，该字符串是CreateEnhMetaFile函数的最后一个参数。在本例中，该字符串由后跟一个NULL字符的「EMF2」字符串和后跟两个NULL字符的「EMF Demo #2」字符串组成。总共18个字符，要是以Unicode方式储存则为36个字符。无论建立MetaFile的程序执行在Windows NT还是Windows 98下，该字符串始终以Unicode方式储存。

MetaFile与GDI物件

我们已经知道了GDI绘图命令储存在MetaFile中方式，现在看一下GDI对象的储存方式。程序18-4 EMF3除了建立用于绘制矩形和直线的非内定画笔和画刷以外，与前面介绍的EMF2程序很相似。该程序也对Rectangle坐标的问题提出了一点修改。EMF3程序使用GetVersion来确定执行环境是Windows 98还是Windows NT，并适当地调整参数。

程序18-4 EMF3

EMF3.C

```
/*--------------------------------------------------------------------------
EMF3.C -- Enhanced MetaFile Demo #3
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF3") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #3"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  LOGBRUSH lb ;
  HDC hdc, hdcEMF ;
  HENHMETAFILE hemf ;
  PAINTSTRUCT ps ;
  RECT rect ;
  switch (message)
  {
  case WM_CREATE:
    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf3.emf"), NULL,
      TEXT ("EMF3\0EMF Demo #3\0")) ;

    SelectObject (hdcEMF, CreateSolidBrush (RGB (0, 0, 255))) ;

    lb.lbStyle = BS_SOLID ;
    lb.lbColor = RGB (255, 0, 0) ;
    lb.lbHatch = 0 ;

    SelectObject (hdcEMF,
```

```
    ExtCreatePen (PS_SOLID | PS_GEOMETRIC, 5, &lb, 0, NULL)) ;

   if (GetVersion () & 0x80000000) // Windows 98
     Rectangle (hdcEMF, 100, 100, 201, 201) ;
   else
     // Windows NT
     Rectangle (hdcEMF, 101, 101, 202, 202) ;

   MoveToEx (hdcEMF, 100, 100, NULL) ;
   LineTo (hdcEMF, 200, 200) ;

   MoveToEx (hdcEMF, 200, 100, NULL) ;
   LineTo (hdcEMF, 100, 200) ;

   DeleteObject (SelectObject (hdcEMF, GetStockObject (BLACK_PEN))) ;
   DeleteObject (SelectObject (hdcEMF, GetStockObject (WHITE_BRUSH))) ;

   hemf = CloseEnhMetaFile (hdcEMF) ;

   DeleteEnhMetaFile (hemf) ;
   return 0 ;

 case WM_PAINT:
   hdc = BeginPaint (hwnd, &ps) ;

   GetClientRect (hwnd, &rect) ;

   rect.left = rect.right / 4 ;
   rect.right = 3 * rect.right / 4 ;
   rect.top = rect.bottom / 4 ;
   rect.bottom = 3 * rect.bottom / 4 ;

   hemf = GetEnhMetaFile (TEXT ("emf3.emf")) ;

   PlayEnhMetaFile (hdc, hemf, &rect) ;
   DeleteEnhMetaFile (hemf) ;
   EndPaint (hwnd, &ps) ;
   return 0 ;

 case WM_DESTROY:
   PostQuitMessage (0) ;
   return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

如我们所看到的，当利用CreateEnhMetaFile传回的设备内容句柄来呼叫GDI函数时，这些函数呼叫被储存在MetaFile中而不是直接输出到屏幕或打印机上。然而，一些GDI函数根本不涉及特定的设备内容。其中有关建立画笔和画刷等图形对象的GDI函数十分重要。虽然逻辑画笔和画刷的定义储存在由GDI保留的内存中，但是在建立这些对象时，这些抽象的定义并未与任何特定的设备内容相关。

EMF3呼叫CreateSolidBrush和ExtCreatePen函数。因为这些函数不需要设备内容句柄，`所以GDI不会把这些呼叫储存在MetaFile里。当呼叫它们时，GDI函数只是简单地建立图形绘制对象而不会影响MetaFile。

然而，当程序呼叫SelectObject函数将GDI对象选入MetaFile设备内容时，GDI既为对象建立函数编码（源自用于储存对象的内部GDI数据）也为MetaFile中的SelectObject呼叫进行编码。为了解其工作方式，我们来看一下EMF3.EMF文件的十六进制代码，如图18-4所示：

```
0000      01 00 00 00 88 00 00 00 60 00 00 00 60 00 00 00     ........`...`...
0010      CC 00 00 00 CC 00 00 00 B8 0B 00 00 B8 0B 00 00     ................
0020      E7 18 00 00 E7 18 00 00 20 45 4D 46 00 00 01 00     ........EMF.....
0030      88 01 00 00 0F 00 00 00 03 00 00 00 12 00 00 00     ................
0040      64 00 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00  d..............
```

```
0050      40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00     @...............
0060      00 00 00 00 45 00 4D 00 46 00 33 00 00 00 45 00     ....E.M.F.3...E.
0070      4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00     M.F....D.e.m.o..
0080      23 00 33 00 00 00 00 00 27 00 00 00 18 00 00 00     #.3.....'.......
0090      01 00 00 00 00 00 00 00 00 00 FF 00 00 00 00 00     ................
00A0      25 00 00 00 0C 00 00 00 01 00 00 00 5F 00 00 00     %..........._...
00B0      34 00 00 00 02 00 00 00 34 00 00 00 00 00 00 00     4.......4.......
00C0      34 00 00 00 00 00 00 00 00 00 01 00 05 00 00 00     4...............
00D0      00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00     ................
00E0      25 00 00 00 0C 00 00 00 02 00 00 00 2B 00 00 00     %...........+...
00F0      18 00 00 00 63 00 00 00 63 00 00 00 C6 00 00 00     ....c...c.......
0100      C6 00 00 00 1B 00 00 00 10 00 00 00 64 00 00 00     ............d...
0110      64 00 00 00 36 00 00 00 10 00 00 00 C8 00 00 00     d...6...........
0120      C8 00 00 00 1B 00 00 00 10 00 00 00 C8 00 00 00     ................
0130      64 00 00 00 36 00 00 00 10 00 00 00 64 00 00 00     d...6.......d...
0140      C8 00 00 00 25 00 00 00 0C 00 00 00 07 00 00 80     ....%...........
0150      28 00 00 00 0C 00 00 00 02 00 00 00 25 00 00 00     (...........%...
0160      0C 00 00 00 00 00 00 80 28 00 00 00 0C 00 00 00     ........(.......
0170      01 00 00 00 0E 00 00 00 14 00 00 00 00 00 00 00     ................
0180      10 00 00 00 14 00 00 00                             ........
```

图18-4 EMF3.EMF的十六进制代码

如果把这个MetaFile跟前面的EMF2.EMF文件进行比较，第一个不同点就是EMF3.EMF表头部分中的rclBounds字段。在EMF2.EMF中，它指出图像限定在坐标(0x64,0x64)和(0xC8,0xC8)区域内。而在EMF3.EMF中，坐标是(0x60,0x60)和(0xCC,0xCC)。这表示使用了较粗的笔。rclFrame字段（以0.01mm为单位指出图像大小）也受到影响。

EMF2.EMF中的nBytes字段（偏移量为0x0030）显示该MetaFile长度为0xFA字节，EMF3.EMF中长度为0x0188字节。EMF2.EMFMetaFile包含7个记录（一个表头纪录，5个GDI函数呼叫和一个文件结束记录），但是EMF3.EMF文件包含15个记录。多出的8个记录是两个对象建立函数、4个对SelectObject函数的呼叫和两个对DeleteObject函数的呼叫。

nHandles字段（在文件中偏移量为0x0038）指出GDI对象的句柄个数。该字段的值总是比MetaFile使用的非内定对象数多一。(Platform SDK文件解释这个多出来的一是「此表中保留的零索引」)。该字段在EMF2.EMF的值为1，而在EMF3.EMF中的值为3，多出的数指出了画笔和画刷。

让我们跳到文件中偏移量为0x0088的地方，即第二个记录（表头纪录之后的第一个记录）。记录型态为0x27，对应常数为EMR_CREATE-BRUSHINDIRECT。该MetaFile记录用于CreateBrushIndirect函数，此函数需要指向LOGBRUSH结构的指针作为参数。该记录的长度为0x18（或24）字节。

每个被选入MetaFile设备内容的非备用GDI对象得到一个号码，该号码从1开始编号。这在此记录的下4个字节中指出，在MetaFile中的偏移量是0x0090。此记录下面的3个4字节字段分别对应LOGBRUSH结构的3个字段：0x00000000（BS_SOLID的lbStyle字段）、0x00FF0000（lbColor字段）和0x00000000（lbHatch字段）。

下一个记录在EMF3.EMF中的偏移量为0x00A0，记录型态为0x25，或EMR_SELECTOBJECT，是用于SelectObject呼叫的MetaFile记录。该记录的长度为0x0C（或12）字节，下一个字段是数值0x01，指出它是选中的第一个GDI对象，这就是逻辑画刷。

EMF3.EMF中的偏移量0x00AC是下一个记录，它的记录型态为0x5F或EMR_EXTCREATEPEN。该记录有0x34（或52）个字节。下一个4字节字段是0x02，它表示这是在MetaFile内使用的第二个非备用GDI对象。

EMR_EXTCREATEPEN记录的下4个字段重复记录大小两次，之间用0字段隔开：0x34、0x00、0x34和0x00。下一个字段是0x00010000，它是PS_SOLID (0x00000000)与PS_GEOMETRIC

(0x00010000)组合的画笔样式。接下来是5个单元的宽度，紧接着是ExtCreatePen中使用的逻辑画刷结构的3个字段，后接0字段。

如果建立了自订的扩展画笔样式，EMR_EXTCREATEPEN记录会超过52个字节，这样会影响记录的第二字段及两个重复的大小字段。在描述LOGBRUSH结构的3个字段后面不会是0（像在EMF3.EMF中那样），而是指出了虚线和空格的数量。这后面接着用于虚线和空格长度的许多字段。

EMF3.EMF的下一个12字节的字段是指出第二个对象（画笔）的另一个SelectObject呼叫。接下来的5个记录与EMF2.EMF中的一样 – 一个0x2B(EMR_RECTANGLE)的记录型态和两组0x1B (EMR_MOVETOEX)和0x36 (EMR_LINETO)记录。

这些绘图函数后面跟着两组0x25(EMR_SELECTOBJECT)和0x28 (EMR_DELETEOBJECT)的12字节记录。选择对象记录具有0x80000007和0x80000000的参数。在设定高位时，它指出一个备用对象，在此例中是0x07（对应BLACK_PEN）和0x00 (WHITE_BRUSH)。

DeleteObject呼叫有2和1两个参数，用于在MetaFile中使用的两个非内定对象。虽然DeleteObject函数并不需要设备内容句柄作为它的第一个参数，但GDI显然保留了MetaFile中使用的被程序删除的对象。

最后，MetaFile以0x0E（EMF_EOF）记录结束。

总结一下，每当非内定的GDI对象首次被选入MetaFile设备内容时，GDI都会为该对象建立函数的记录编码（此例中，为EMR_CREATEBRUSHINDIRECT和EMR_EXTCREATEPEN）。每个对象有一个依序从1开始的唯一数值，此数值由记录的第三个字段表示。跟在此记录后的是引用该数值的EMR_SELECTOBJECT记录。以后，将对象选入MetaFile设备内容时（在中间时期没有被删除），就只需要EMR_SELECTOBJECT记录了。

MetaFile和位图

现在，让我们做点稍微复杂的事，在MetaFile设备内容中绘制一幅位图，如程序18-5 EMF4所示。

程序18-5 EMF4

EMF4.C

```
/*------------------------------------------------------------------------
EMF4.C -- Enhanced MetaFile Demo #4
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#define OEMRESOURCE
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF4") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
```

```
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #4"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  BITMAP bm ;
  HBITMAP hbm ;
  HDC hdc, hdcEMF, hdcMem ;
  HENHMETAFILE hemf ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf4.emf"), NULL,
      TEXT ("EMF4\0EMF Demo #4\0")) ;

    hbm = LoadBitmap (NULL, MAKEINTRESOURCE (OBM_CLOSE)) ;

    GetObject (hbm, sizeof (BITMAP), &bm) ;

    hdcMem = CreateCompatibleDC (hdcEMF) ;

    SelectObject (hdcMem, hbm) ;

    StretchBlt (hdcEMF,100,100,100,100,
      hdcMem,0,0,bm.bmWidth, bm.bmHeight, SRCCOPY) ;

    DeleteDC (hdcMem) ;
    DeleteObject (hbm) ;

    hemf = CloseEnhMetaFile (hdcEMF) ;

    DeleteEnhMetaFile (hemf) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;
    rect.left = rect.right / 4 ;
    rect.right = 3 * rect.right / 4 ;
    rect.top = rect.bottom / 4 ;
    rect.bottom = 3 * rect.bottom / 4 ;

    hemf = GetEnhMetaFile (TEXT ("emf4.emf")) ;
```

```
    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

为了方便，EMF4加载由常数OEM_CLOSE指出的系统位图。在设备内容中显示位图的惯用方法是通过使用CreateCompatibleDC建立与目的设备内容（此例为MetaFile设备内容）兼容的内存设备内容。然后，通过使用SelectObject将位图选入该内存设备内容并且从该内存设备内容呼叫BitBlt或StretchBlt把位图画到目的设备内容。结束后，删除内存设备内容和位图。

您会注意到EMF4也呼叫GetObject来确定位图的大小。这对SelectObject呼叫是很必要的。

首先，这份程序代码储存MetaFile的空间对GDI来说就是个挑战。在StretchBlt呼叫前根本没有别的GDI函数去处理MetaFile的设备内容。因此，让我们来看一看EMF4.EMF里头是如何做的，图18-5只显示了一部分。

```
0000      01 00 00 00 88 00 00 00 64 00 00 00 64 00 00 00          ........d...d...
0010      C7 00 00 00 C7 00 00 00 35 0C 00 00 35 0C 00 00          ........5...5...
0020      4B 18 00 00 4B 18 00 00 20 45 4D 46 00 00 01 00          K...K.... EMF....
0030      F0 0E 00 00 03 00 00 00 01 00 00 00 12 00 00 00          ................
0040      64 00 00 00 00 00 00 00 00 04 00 00 00 03 00 00          d...............
0050      40 01 00 00 F0 00 00 00 00 00 00 00 00 00 00 00          @...............
0060      00 00 00 00 45 00 4D 00 46 00 34 00 00 00 45 00          ....E.M.F.4...E.
0070      4D 00 46 00 20 00 44 00 65 00 6D 00 6F 00 20 00          M.F... D.e.m.o...
0080      23 00 34 00 00 00 00 00 4D 00 00 00 54 0E 00 00          #.4..... M...T...
0090      64 00 00 00 64 00 00 00 C7 00 00 00 C7 00 00 00          d...d...........
00A0      64 00 00 00 64 00 00 00 64 00 00 00 64 00 00 00          d...d...d...d...
00B0      20 00 CC 00 00 00 00 00 00 00 00 00 00 00 80 3F          ..............?
00C0      00 00 00 00 00 00 00 00 00 00 80 3F 00 00 00 00          ...........?....
00D0      00 00 00 00 FF FF FF 00 00 00 00 00 6C 00 00 00          ............l...
00E0      28 00 00 00 94 00 00 00 C0 0D 00 00 28 00 00 00          (...........(...
00F0      16 00 00 00 28 00 00 00 28 00 00 00 16 00 00 00          ....(...(.......
0100      01 00 20 00 00 00 00 00 C0 0D 00 00 00 00 00 00          .. .............
0110      00 00 00 00 00 00 00 00 00 00 00 00 C0 C0 C0 00          ................
0120      C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00          ................
. . . .
0ED0      C0 C0 C0 00 C0 C0 C0 00 C0 C0 C0 00 0E 00 00 00          ................
0EE0      14 00 00 00 00 00 00 00 10 00 00 00 14 00 00 00          ................
```

图18-5 EMF4.EMF的部分十六进制代码

此MetaFile只包含3个记录 – 表头纪录、0x0E54位组长度的0x4D（或EMR_STRETCHBLT）和文件结束记录。

我不解释该记录每个字段的含义，但我会指出关键部分，以便理解GDI把EMF4.C中的一系列函数呼叫转化为单个MetaFile记录的方法。

GDI已经把原始的与设备相关的位图转化为与设备无关的位图（DIB）。整个DIB储存在记录着自身大小的记录中。我想，在显示MetaFile和位图时，GDI实际上使用StretchDIBits函数而不是StretchBlt。或者，GDI使用CreateDIBitmap把DIB转变回与设备相关的位图，然后使用内存设备内容及StretchBlt来显示位图。

EMR_STRETCHBLT记录开始于MetaFile的偏移量0x0088处。DIB储存在MetaFile中，以偏移量0x00F4开始，到0x0EDC处的记录结尾结束。DIB以BITMAPINFOHEADER型态的40字节的结构开始。

在偏移量0x011C处接有22个像素行，每行40个像素。这是每像素32位的DIB，所以每个像素需要4个字节。

列举MetaFile内容

当您希望存取MetaFile内的个别记录时，可以使用称作MetaFile列举的程序。如程序18-6 EMF5所示。此程序使用MetaFile来显示与EMF3相同的图像，但它是通过MetaFile列举来进行的。

程序18-6 EMF5

EMF5.C

```
/*------------------------------------------------------------------------
EMF5.C -- Enhanced MetaFile Demo #5
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF5") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #5"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}
int CALLBACK EnhMetaFileProc ( HDC hdc, HANDLETABLE * pHandleTable,
                        CONST ENHMETARECORD * pEmfRecord,
                        int iHandles, LPARAM pData)
{
  PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfRecord, iHandles) ;
  return TRUE ;
}
```

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  HDC hdc ;
  HENHMETAFILE hemf ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    rect.left = rect.right / 4 ;
    rect.right = 3 * rect.right / 4 ;
    rect.top = rect.bottom / 4 ;
    rect.bottom = 3 * rect.bottom / 4 ;

    hemf = GetEnhMetaFile (TEXT ("..\\emf3\\emf3.emf")) ;

    EnumEnhMetaFile (hdc, hemf, EnhMetaFileProc, NULL, &rect) ;
    DeleteEnhMetaFile (hemf) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

此程序使用EMF3程序建立的EMF3.EMF文件，所以确定在执行此程序前先执行EMF3程序。同时，需要在Visual C++环境中执行两个程序，以确保路径的正确。在处理WM_PAINT时，两个程序的主要区别是EMF3呼叫PlayEnhMetaFile，而EMF5呼叫EnumEnhMetaFile。PlayEnhMetaFile函数有下面的语法：

PlayEnhMetaFile (hdc, hemf, &rect) ;

第一个参数是要显示的MetaFile的设备内容句柄。第二个参数是增强型MetaFile句柄。第三个参数是指向描述设备内容平面上矩形的RECT结构的指针。MetaFile图像大小被缩放过，以便刚好能够显示在不超过该矩形的区域内。

EnumEnhMetaFile有5个参数，其中3个与PlayEnhMetaFile一样（虽然RECT结构的指针已经移到参数表的末尾）。

EnumEnhMetaFile的第三个参数是列举函数的名称，它用于呼叫EnhMetaFileProc。第四个参数是希望传递给列举函数的任意数据的指针，这里将该参数简单地设定为NULL。

现在看一看列举函数。当呼叫EnumEnhMetaFile时，对于MetaFile中的每一个记录，GDI都将呼叫EnhMetaFileProc一次，包括表头纪录和文件结束记录。通常列举函数传回TRUE，但它可能传回FALSE以略过剩下的列举程序。

该列举函数有5个参数，稍后会描述它们。在这个程序中，我仅把前4个参数传递给PlayEnhMetaFileRecord，它使GDI执行由该记录代表的函数呼叫，好像您明确地呼叫它一样。

EMF5使用EnumEnhMetaFile和PlayEnhMetaFileRecord得到的结果与EMF3呼叫PlayEnhMetaFile得到的结果一样。区别在于EMF5现在直接介入了MetaFile的显示程序，并能够存取各个MetaFile记录。这是很有用的。

列举函数的第一个参数是设备内容句柄。GDI从EnumEnhMetaFile的第一个参数中简单地取

得此句柄。列举函数把该句柄传递给PlayEnhMetaFileRecord来标识图像显示的目的设备内容。

我们先跳到列举函数的第三个参数，它是指向ENHMETARECORD型态结构的指针，前面已经提到过。这个结构描述实际的MetaFile记录，就像它亲自在MetaFile中编码一样。

您可以写一些程序代码来检查这些记录。您也许不想把某些记录传送到PlayEnhMetaFileRecord函数。例如，在EMF5.C中，把下行插入到PlayEnhMetaFileRecord呼叫的前面：

if (pEmfRecord->iType != EMR_LINETO)

重新编译程序，执行它，将只看到矩形，而没有两条线。或使用下面的叙述：

if (pEmfRecord->iType != EMR_SELECTOBJECT)

这个小改变会让GDI用内定对象显示图像，而不是用MetaFile所建立的画笔和画刷。

程序中不应该修改MetaFile记录，不过先不要担心这一点。先来看一看程序18-7 EMF6。

程序18-7 EMF6

EMF6.C

```
/*------------------------------------------------------------------------
EMF6.C -- Enhanced MetaFile Demo #6
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR lpszCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF6") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #6"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
```

```
  }
  return msg.wParam ;
}

int CALLBACK EnhMetaFileProc ( HDC hdc, HANDLETABLE * pHandleTable,
                        CONST ENHMETARECORD * pEmfRecord,
                        int iHandles, LPARAM pData)
{
  ENHMETARECORD * pEmfr ;
  pEmfr = (ENHMETARECORD *) malloc (pEmfRecord->nSize) ;
  CopyMemory (pEmfr, pEmfRecord, pEmfRecord->nSize) ;
  if (pEmfr->iType == EMR_RECTANGLE)
    pEmfr->iType = EMR_ELLIPSE ;
  PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfr, iHandles) ;
  free (pEmfr) ;
  return TRUE ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  HDC hdc ;
  HENHMETAFILE hemf ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    rect.left = rect.right / 4 ;
    rect.right = 3 * rect.right / 4 ;
    rect.top = rect.bottom / 4 ;
    rect.bottom = 3 * rect.bottom / 4 ;

    hemf = GetEnhMetaFile (TEXT ("..\\emf3\\emf3.emf")) ;
    EnumEnhMetaFile ( hdc, hemf, EnhMetaFileProc, NULL, &rect) ;
    DeleteEnhMetaFile (hemf) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

与EMF5一样，EMF6使用EMF3程序建立的EMF3.EMFMetaFile，因此要在Visual C++中执行这个程序之前先执行过EMF3程序。

EMF6展示了如果在显示MetaFile之前要修改它们，解决方法是非常简单的：做个被修改过的副本出来就好了。您可以看到，列举程序一开始使用malloc配置一块MetaFile记录大小的内存，它是由传递给该函数的pEmfRecord结构的nSize字段表示的。这个内存块的指针储存在变量pEmfr中，pEmfr本身是指向ENHMETARECORD结构的指针。

程序使用CopyMemory把pEmfRecord指向的结构内容复制到pEmfr指向的结构中。现在我们就可以做些修改了。程序检查记录是否为EMR_RECTANGLE型态，如果是，则用EMR_ELLIPSE取代iType字段。PEmfr指标被传递到PlayEnhMetaFileRecord然后被释放。结果是程序画出一个椭圆而不是矩形。其它的内容的修改方式都是相同的。

当然，我们的小改变很容易起作用，因为Rectangle和Ellipse函数有同样的参数，这些参数都定义同一件事 – 图画的边界框。要进行范围更广的修改需要一些不同MetaFile记录格式的相关知

识。

另一个可能性是插入一、两个额外的记录。例如，用下面的叙述代替EMF6.C中的if叙述：

```
if (pEmfr->iType == EMR_RECTANGLE)
{
  PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfr, nObjects) ;
  pEmfr->iType = EMR_ELLIPSE ;
}
```

无论何时出现Rectangle记录，程序都会处理此记录并把它更改为Ellipse，然后再显示。现在程序将画出矩形和椭圆。

现在讨论一下在列举MetaFile时GDI对象处理的方式。

在MetaFile表头中，ENHMETAHEADER结构的nHandles字段是比在MetaFile中建立的GDI对象数还要大的值。因此，对于EMF5和EMF6中的MetaFile，此字段是3，表示画笔、画刷和其它东西。「其它东西」的具体内容，稍后我会说明。

您会注意到EMF5和EMF6中列举函数的倒数第二个参数，也称作nHandles，它是同一个数，3。

列举函数的第二个参数是指向HANDLETABLE结构的指针，在WINGDI.H中定义如下：

```
typedef struct tagHANDLETABLE
{
  HGDIOBJ objectHandle [1] ;
}
HANDLETABLE ;
```

HGDIOBJ数据型态是GDI对象的句柄，被定义为32位的指针，类似于所有其它GDI对象。这是那些带有一个元素的数组字段的结构之一。这意味着此字段具有可变的长度。objectHandle数组中的元素数等于nHandles，在此程序中是3。

在列举函数中，可以使用以下表达式取得这些GDI对象句柄：

pHandleTable->objectHandle[i]

对于3个句柄，i是0、1和2。

每次呼叫列举函数时，数组的第一个元素都将包含所列举的MetaFile句柄。这就是前面提到的「其它东西」。

在第一次呼叫列举函数时，表的第二、第三个元素将是0。它们是画笔和画刷句柄的保留位置。

以下是列举函数运作的方式：MetaFile中的第一个对象建立函数具有EMR_CREATEBRUSHINDIRECT的记录型态，此记录指出了对象编号1。当把该记录传递给PlayEnhMetaFileRecord时，GDI建立画刷并取得它的句柄。此句柄储存在objectHandle数组的元素1（第二个元素）中。当把第一个EMR_SELECTOBJECT记录传递给PlayEnhMetaFileRecord时，GDI发现此对象编号为1，并能够从表中找到该对象实际的句柄，而把它用来呼叫SelectObject。当MetaFile最后删除画刷时，GDI将objectHandle数组的元素1设定回0。

通过存取objectHandle数组，可以使用例如GetObjectType和GetObject等呼叫取得在MetaFile中使用的对象信息。

嵌入图像

列举MetaFile的最重要应用也许是在现有的MetaFile中嵌入其它图像（甚至是整个MetaFile）。事实上，现有的MetaFile保持不变；真正进行的是建立包含现有MetaFile和新嵌入图像的新MetaFile。基本的技巧是把MetaFile设备内容句柄传递给EnumEnhMetaFile，作为它的第一个参数。这使您能够在MetaFile设备内容上显示MetaFile记录和GDI函数呼叫。

在MetaFile命令序列的开头或结尾嵌入新图像是极简单的 – 就在EMR_HEADER记录之后或在EMF_EOF记录之前。然而，如果您熟悉现有的MetaFile结构，就可以把新的绘图命令嵌入所需的任何地方。如程序18-8 EMF7所示。

程序18-8 EMF7

EMF7.C

```
/*------------------------------------------------------------------------
EMF7.C -- Enhanced MetaFile Demo #7
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR lpszCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("EMF7") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Demo #7"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

int CALLBACK EnhMetaFileProc ( HDC hdc, HANDLETABLE * pHandleTable,
                          CONST ENHMETARECORD * pEmfRecord,
                          int iHandles, LPARAM pData)
{
  HBRUSH hBrush ;
  HPEN hPen ;
  LOGBRUSH lb ;

  if (pEmfRecord->iType != EMR_HEADER && pEmfRecord->iType != EMR_EOF)
    PlayEnhMetaFileRecord (hdc, pHandleTable, pEmfRecord, iHandles) ;
  if (pEmfRecord->iType == EMR_RECTANGLE)
```

```
  {
    hBrush = (HBRUSH)SelectObject (hdc, GetStockObject (NULL_BRUSH)) ;
    lb.lbStyle = BS_SOLID ;
    lb.lbColor = RGB (0, 255, 0) ;
    lb.lbHatch = 0 ;

    hPen = (HPEN)SelectObject (hdc,
      ExtCreatePen (PS_SOLID | PS_GEOMETRIC, 5, &lb, 0, NULL)) ;
    Ellipse (hdc, 100, 100, 200, 200) ;
    DeleteObject (SelectObject (hdc, hPen)) ;
    SelectObject (hdc, hBrush) ;
  }
  return TRUE ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  ENHMETAHEADER emh ;
  HDC hdc, hdcEMF ;
  HENHMETAFILE hemfOld, hemf ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:

    // Retrieve existing MetaFile and header
    hemfOld = GetEnhMetaFile (TEXT ("..\\emf3\\emf3.emf")) ;

    GetEnhMetaFileHeader (hemfOld, sizeof (ENHMETAHEADER), &emh) ;

    // Create a new MetaFile DC
    hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf7.emf"), NULL,
      TEXT ("EMF7\0EMF Demo #7\0")) ;

    // Enumerate the existing MetaFile
    EnumEnhMetaFile (hdcEMF, hemfOld, EnhMetaFileProc, NULL,
      (RECT *) & emh.rclBounds) ;

    // Clean up
    hemf = CloseEnhMetaFile (hdcEMF) ;

    DeleteEnhMetaFile (hemfOld) ;
    DeleteEnhMetaFile (hemf) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;
    rect.left = rect.right / 4 ;
    rect.right = 3 * rect.right / 4 ;
    rect.top = rect.bottom / 4 ;
    rect.bottom = 3 * rect.bottom / 4 ;

    hemf = GetEnhMetaFile (TEXT ("emf7.emf")) ;

    PlayEnhMetaFile (hdc, hemf, &rect) ;
    DeleteEnhMetaFile (hemf) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

EMF7使用EMF3程序建立的EMF3.EMF，所以在执行EMF7之前要执行EMF3程序建立MetaFile。

EMF7 中的 WM_PAINT 处理使用 PlayEnhMetaFile 而不是 EnumEnhMetaFile，而且WM_CREATE处理有很大的差别。

首先，程序通过呼叫 GetEnhMetaFile 取得 EMF3.EMF 文件的 MetaFile 句柄，还呼叫GetEnhMetaFileHeader得到增强型MetaFile表头记录，目的是在后面的EnumEnhMetaFile呼叫中使用rclBounds字段。

接下来，程序建立新的MetaFile文件，名为EMF7.EMF。CreateEnhMetaFile函数为MetaFile传回设备内容句柄。然后，使用EMF7.EMF的MetaFile设备内容句柄和EMF3.EMF的MetaFile句柄呼叫EnumEnhMetaFile。

现在来看一看EnhMetaFileProc。如果被列举的记录不是表头纪录或文件结束记录，函数就呼叫PlayEnhMetaFileRecord把记录转换为新的MetaFile设备内容（并不一定排除表头纪录或文件结束记录，但它们会使MetaFile变大）。

如果刚转换的记录是Rectangle呼叫，则函数建立画笔用绿色的轮廓线和透明的内部来绘制椭圆。注意程序中经由储存先前的画笔和画刷句柄来恢复设备内容状态的方法。在此期间，所有这些函数都被插入到MetaFile中（记住，也可以使用PlayEnhMetaFile在现有的MetaFile中插入整个MetaFile）。

回到WM_CREATE处理，程序呼叫CloseEnhMetaFile取得新MetaFile的句柄。然后，它删除两个MetaFile句柄，将EMF3.EMF和EMF7.EMF文件留在磁盘上。

从程序显示输出中可以很明显地看到，椭圆是在矩形之后两条交叉线之前绘制的。

增强型MetaFile阅览器和打印机

使用剪贴簿转换增强型MetaFile非常简单，剪贴簿型态是CF_ENHMetaFile。GetClipboardData函数传回增强型MetaFile句柄，SetClipboardData也使用该MetaFile句柄。复制MetaFile时可以使用CopyEnhMetaFile函数。如果把增强型MetaFile放在剪贴簿中，Windows会让需要旧格式的那些程序也可以使用它。如果在剪贴簿中放置旧格式的MetaFile，Windows将也会自动视需要把内容转换为增强型MetaFile的格式。

程序18-9 EMFVIEW所示为在剪贴簿中传送MetaFile的程序代码，它也允许载入、储存和打印MetaFile。

程序18-9 EMFVIEW

EMFVIEW.C

```
/*-----------------------------------------------------------------------------
EMFVIEW.C -- View Enhanced MetaFiles
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("EmfView") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Enhanced MetaFile Viewer"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

HPALETTE CreatePaletteFromMetaFile (HENHMETAFILE hemf)
{
  HPALETTE hPalette ;
  int iNum ;
  LOGPALETTE * plp ;

  if (!hemf)
    return NULL ;
  if (0 == (iNum = GetEnhMetaFilePaletteEntries (hemf, 0, NULL)))
    return NULL ;
  plp = malloc (sizeof (LOGPALETTE) + (iNum - 1) * sizeof (PALETTEENTRY)) ;
  plp->palVersion = 0x0300 ;
  plp->palNumEntries = iNum ;

  GetEnhMetaFilePaletteEntries (hemf, iNum, plp->palPalEntry) ;
  hPalette = CreatePalette (plp) ;
  free (plp) ;
  return hPalette ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static DOCINFO di = { sizeof (DOCINFO), TEXT ("EmfView: Printing") } ;
  static HENHMETAFILE hemf ;
  static OPENFILENAME ofn ;
  static PRINTDLG printdlg = { sizeof (PRINTDLG) } ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  static TCHAR szFilter[] =
    TEXT ("Enhanced MetaFiles (*.EMF)\0*.emf\0")
    TEXT ("All Files (*.*)\0*.*\0\0") ;
  BOOL bSuccess ;
```

```
  ENHMETAHEADER header ;
  HDC hdc, hdcPrn ;
  HENHMETAFILE hemfCopy ;
  HMENU hMenu ;
  HPALETTE hPalette ;
  int i, iLength, iEnable ;
  PAINTSTRUCT ps ;
  RECT rect ;
  PTSTR pBuffer ;

  switch (message)
  {
  case WM_CREATE:
    // Initialize OPENFILENAME structure
    ofn.lStructSize = sizeof (OPENFILENAME) ;
    ofn.hwndOwner = hwnd ;
    ofn.hInstance = NULL ;
    ofn.lpstrFilter = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter = 0 ;
    ofn.nFilterIndex = 0 ;
    ofn.lpstrFile = szFileName ;
    ofn.nMaxFile = MAX_PATH ;
    ofn.lpstrFileTitle = szTitleName ;
    ofn.nMaxFileTitle = MAX_PATH ;
    ofn.lpstrInitialDir = NULL ;
    ofn.lpstrTitle = NULL ;
    ofn.Flags = 0 ;
    ofn.nFileOffset = 0 ;
    ofn.nFileExtension = 0 ;
    ofn.lpstrDefExt = TEXT ("emf") ;
    ofn.lCustData = 0 ;
    ofn.lpfnHook = NULL ;
    ofn.lpTemplateName = NULL ;
    return 0 ;

  case WM_INITMENUPOPUP:
    hMenu = GetMenu (hwnd) ;

    iEnable = hemf ? MF_ENABLED : MF_GRAYED ;

    EnableMenuItem (hMenu, IDM_FILE_SAVE_AS, iEnable) ;
    EnableMenuItem (hMenu, IDM_FILE_PRINT, iEnable) ;
    EnableMenuItem (hMenu, IDM_FILE_PROPERTIES, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_CUT, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_COPY, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_DELETE, iEnable) ;
    EnableMenuItem (hMenu, IDM_EDIT_PASTE,
      IsClipboardFormatAvailable (CF_ENHMetaFile) ?
MF_ENABLED : MF_GRAYED) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_OPEN:
      // Show the File Open dialog box
      ofn.Flags = 0 ;

      if (!GetOpenFileName (&ofn))
        return 0 ;

      // If there's an existing EMF, get rid of it.
      if (hemf)
      {
        DeleteEnhMetaFile (hemf) ;
        hemf = NULL ;
      }
      // Load the EMF into memory
```

```
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    hemf = GetEnhMetaFile (szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    // Invalidate the client area for later update
    InvalidateRect (hwnd, NULL, TRUE) ;

    if (hemf == NULL)
    {
      MessageBox ( hwnd, TEXT ("Cannot load MetaFile"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    }
    return 0 ;

  case IDM_FILE_SAVE_AS:
    if (!hemf)
      return 0 ;

    // Show the File Save dialog box
    ofn.Flags = OFN_OVERWRITEPROMPT ;

    if (!GetSaveFileName (&ofn))
      return 0 ;

    // Save the EMF to disk file
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    hemfCopy = CopyEnhMetaFile (hemf, szFileName) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
    if (hemfCopy)
    {
      DeleteEnhMetaFile (hemf) ;
      hemf = hemfCopy ;
    }
    else
      MessageBox ( hwnd, TEXT ("Cannot save MetaFile"),
      szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

  case IDM_FILE_PRINT:
    // Show the Print dialog box and get printer DC

    printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;

    if (!PrintDlg (&printdlg))
      return 0 ;

    if (NULL == (hdcPrn = printdlg.hDC))
    {
      MessageBox ( hwnd, TEXT ("Cannot obtain printer DC"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return 0 ;
    }
    // Get size of printable area of page
    rect.left = 0 ;
    rect.right = GetDeviceCaps (hdcPrn, HORZRES) ;
    rect.top = 0 ;
    rect.bottom = GetDeviceCaps (hdcPrn, VERTRES) ;

    bSuccess = FALSE ;

    // Play the EMF to the printer
```

```
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    if ((StartDoc (hdcPrn, &di) > 0) && (StartPage (hdcPrn) > 0))
    {
      PlayEnhMetaFile (hdcPrn, hemf, &rect) ;

      if (EndPage (hdcPrn) > 0)
      {
        bSuccess = TRUE ;
        EndDoc (hdcPrn) ;
      }
    }
    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    DeleteDC (hdcPrn) ;

    if (!bSuccess)
      MessageBox ( hwnd, TEXT ("Could not print MetaFile"),
      szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return 0 ;

  case IDM_FILE_PROPERTIES:
    if (!hemf)
      return 0 ;

    iLength = GetEnhMetaFileDescription (hemf, 0, NULL) ;
    pBuffer = malloc ((iLength + 256) * sizeof (TCHAR)) ;

    GetEnhMetaFileHeader (hemf, sizeof (ENHMETAHEADER), &header) ;

    // Format header file information
    i = wsprintf (pBuffer,
      TEXT ("Bounds = (%i, %i) to (%i, %i) pixels\n"),
      header.rclBounds.left, header.rclBounds.top,
      header.rclBounds.right, header.rclBounds.bottom) ;

    i += wsprintf (pBuffer + i,
      TEXT ("Frame = (%i, %i) to (%i, %i) mms\n"),
      header.rclFrame.left, header.rclFrame.top,
      header.rclFrame.right, header.rclFrame.bottom) ;

    i += wsprintf (pBuffer + i,
      TEXT ("Resolution = (%i, %i) pixels")
      TEXT (" = (%i, %i) mms\n"),
      header.szlDevice.cx, header.szlDevice.cy,
      header.szlMillimeters.cx,
      header.szlMillimeters.cy) ;

    i += wsprintf (pBuffer + i,
      TEXT ("Size = %i, Records = %i, ")
      TEXT ("Handles = %i, Palette entries = %i\n"),
      header.nBytes, header.nRecords,
      header.nHandles, header.nPalEntries) ;
    // Include the MetaFile description, if present
    if (iLength)
    {
      i += wsprintf (pBuffer + i, TEXT ("Description = ")) ;
      GetEnhMetaFileDescription (hemf, iLength, pBuffer + i) ;
      pBuffer [lstrlen (pBuffer)] = '\t' ;
    }

    MessageBox (hwnd, pBuffer, TEXT ("MetaFile Properties"), MB_OK) ;
    free (pBuffer) ;
    return 0 ;

  case IDM_EDIT_COPY:
  case IDM_EDIT_CUT:
```

```
    if (!hemf)
      return 0 ;

    // Transfer MetaFile copy to the clipboard
    hemfCopy = CopyEnhMetaFile (hemf, NULL) ;

    OpenClipboard (hwnd) ;
    EmptyClipboard () ;
    SetClipboardData (CF_ENHMetaFile, hemfCopy) ;
    CloseClipboard () ;

    if (LOWORD (wParam) == IDM_EDIT_COPY)
      return 0 ;
    // fall through if IDM_EDIT_CUT
  case IDM_EDIT_DELETE:
    if (hemf)
    {
      DeleteEnhMetaFile (hemf) ;
      hemf = NULL ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;

  case IDM_EDIT_PASTE:
    OpenClipboard (hwnd) ;
    hemfCopy = GetClipboardData (CF_ENHMetaFile) ;

    CloseClipboard () ;
    if (hemfCopy && hemf)
    {
      DeleteEnhMetaFile (hemf) ;
      hemf = NULL ;
    }

    hemf = CopyEnhMetaFile (hemfCopy, NULL) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case IDM_APP_ABOUT:
    MessageBox ( hwnd, TEXT ("Enhanced MetaFile Viewer\n")
      TEXT ("(c) Charles Petzold, 1998"),
      szAppName, MB_OK) ;
    return 0 ;

  case IDM_APP_EXIT:
    SendMessage (hwnd, WM_CLOSE, 0, 0L) ;
    return 0 ;
}
break ;

case WM_PAINT:
  hdc = BeginPaint (hwnd, &ps) ;

  if (hemf)
  {
    if ( hPalette = CreatePaletteFromMetaFile (hemf))
    {
      SelectPalette (hdc, hPalette, FALSE) ;
      RealizePalette (hdc) ;
    }
    GetClientRect (hwnd, &rect) ;
    PlayEnhMetaFile (hdc, hemf, &rect) ;

    if (hPalette)
      DeleteObject (hPalette) ;
  }
  EndPaint (hwnd, &ps) ;
  return 0 ;
```

```
 case WM_QUERYNEWPALETTE:
   if (!hemf || !(hPalette = CreatePaletteFromMetaFile (hemf)))
     return FALSE ;

   hdc = GetDC (hwnd) ;
   SelectPalette (hdc, hPalette, FALSE) ;
   RealizePalette (hdc) ;
   InvalidateRect (hwnd, NULL, FALSE) ;

   DeleteObject (hPalette) ;
   ReleaseDC (hwnd, hdc) ;
   return TRUE ;

 case WM_PALETTECHANGED:
   if ((HWND) wParam == hwnd)
     break ;

   if (!hemf || !(hPalette = CreatePaletteFromMetaFile (hemf)))
     break ;

   hdc = GetDC (hwnd) ;
   SelectPalette (hdc, hPalette, FALSE) ;
   RealizePalette (hdc) ;
   UpdateColors (hdc) ;

   DeleteObject (hPalette) ;
   ReleaseDC (hwnd, hdc) ;
   break ;

 case WM_DESTROY:
   if (hemf)
     DeleteEnhMetaFile (hemf) ;

   PostQuitMessage (0) ;
   return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

EMFVIEW.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
EMFVIEW MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Open\tCtrl+O", IDM_FILE_OPEN
MENUITEM "Save &As...", IDM_FILE_SAVE_AS
MENUITEM SEPARATOR
MENUITEM "&Print...\tCtrl+P",IDM_FILE_PRINT
MENUITEM SEPARATOR
MENUITEM "&Properties", IDM_FILE_PROPERTIES
MENUITEM SEPARATOR
MENUITEM "E&xit",  IDM_APP_EXIT
END
POPUP "&Edit"
BEGIN
MENUITEM "Cu&t\tCtrl+X",    IDM_EDIT_CUT
MENUITEM "&Copy\tCtrl+C",   IDM_EDIT_COPY
MENUITEM "&Paste\tCtrl+V",  IDM_EDIT_PASTE
MENUITEM "&Delete\tDel",   IDM_EDIT_DELETE
END
POPUP "Help"
BEGIN
```

```
MENUITEM "&About EmfView...",  IDM_APP_ABOUT
END
END

//////////////////////////////////////////////////////////////////////////
// Accelerator
EMFVIEW ACCELERATORS DISCARDABLE
BEGIN
"C",IDM_EDIT_COPY, VIRTKEY, CONTROL, NOINVERT
"O",IDM_FILE_OPEN, VIRTKEY, CONTROL, NOINVERT
"P",IDM_FILE_PRINT,VIRTKEY, CONTROL, NOINVERT
"V",IDM_EDIT_PASTE,VIRTKEY, CONTROL, NOINVERT
VK_DELETE,IDM_EDIT_DELETE,VIRTKEY, NOINVERT
"X",IDM_EDIT_CUT,VIRTKEY, CONTROL, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by EmfView.rc
#define   IDM_FILE_OPEN    40001
#define   IDM_FILE_SAVE_AS   40002
#define   IDM_FILE_PRINT   40003
#define   IDM_FILE_PROPERTIES  40004
#define   IDM_APP_EXIT    40005
#define   IDM_EDIT_CUT    40006
#define   IDM_EDIT_COPY    40007
#define   IDM_EDIT_PASTE   40008
#define   IDM_EDIT_DELETE   40009
#define   IDM_APP_ABOUT    40010
```

EMFVIEW也支持完整的调色盘处理，以便支持有调色盘编码信息的MetaFile。（透过呼叫Selectpalette来进行）。该程序在CreatePaletteFromMetaFile函数中处理调色盘，在处理WM_PAINT显示MetaFile以及处理WM_QUERYNEWPALETTE和WM_PALETTECHANGED消息时，呼叫这个函数。

在响应菜单中的「Print」命令时，EMFVIEW显示普通的打印机对话框，然后取得页面中可打印区域的大小。MetaFile被缩放成适当尺寸以填入整个区域。EMFVIEW在窗口中以类似方式显示MetaFile。

「File」菜单中的「Properties」项使EMFVIEW显示包含MetaFile表头信息的消息框。

如果打印本章前面建立的EMF2.EMFMetaFile图像，您将会发现用高分辨率的打印机打印出的线条非常细，几乎看不清楚线条的锯齿。打印向量图像时应该使用较宽的画笔（例如，一点宽）。本章后面所示的直尺图像就是这样做的。

显示精确的MetaFile图像

MetaFile图像的好处在于它能够以任意大小缩放并且仍能保持一定的逼真度。这是因为MetaFile通常由一系列向量图形的基本图形组成，基本图形是指线条、填入的区域以及轮廓字体等等。扩大或缩小图像只是简单地缩放定义这些基本图形的所有坐标点。另一方面，对位图来说，压缩图像会遗漏整行列的像素，因而失去重要的显示信息。

当然，MetaFile的压缩并不是完美无缺的。我们所使用的图形输出设备的像素大小是有限的。当MetaFile图像压缩到一定大小时，组成MetaFile的大量线条会变成模糊的斑点，同时区域填入图案和混色看起来也很奇怪。如果MetaFile中包含嵌入的位图或旧的点阵字体，同样会引起类似的问题。

尽管如此，大多数情况下MetaFile可以任意地缩放。这在把MetaFile放入文书处理或桌上印刷

文件内时非常有用。一般来说，在上述的应用程序中选择MetaFile图像时，会出现围绕图像的矩形，您可以用鼠标拖动该矩形，将它缩放为任意大小。图像送到打印机时，它也具有同样对应的大小。

然而，有时任意缩放MetaFile并不是个好主意。例如：假设您有一个储存着存款客户签名样本的银行系统，这些签名以一系列折线的方式储存在MetaFile中。将MetaFile变宽或变高会使签名变形，因此应该保持图像的纵横比一致。

在前面的范例程序中，是以显示区域的大小来确定PlayEnhMetaFile呼叫使用的围绕矩形范围。所以，如果改变程序窗体的大小，也就改变了图像的大小。这与在文书处理文件中改变MetaFile图像大小的概念相似。

正确地显示MetaFile图像（以特定的度量单位或用适当的纵横比），需要使用MetaFile表头中的大小信息并根据此信息设定矩形结构。

在本章剩下的范例程序中将使用名为EMF.C的程序架构，它包括打印处理的程序代码、资源描述文件EMF.RC和表头文件RESOURCE.H。程序18-10显示了这些文件以及EMF8.C程序，该程序使用这些文件显示一把6英寸的直尺。

程序18-10 EMF8

EMF8.C

```
/*------------------------------------------------------------------------
EMF8.C -- Enhanced MetaFile Demo #8
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
TCHAR szClass [] = TEXT ("EMF8") ;
TCHAR szTitle [] = TEXT ("EMF8: Enhanced MetaFile Demo #8") ;

void DrawRuler (HDC hdc, int cx, int cy)
{
  int iAdj, i, iHeight ;
  LOGFONT lf ;
  TCHAR ch ;

  iAdj = GetVersion () & 0x80000000 ? 0 : 1 ;
  // Black pen with 1-point width
  SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
  // Rectangle surrounding entire pen (with adjustment)
  Rectangle (hdc, iAdj, iAdj, cx + iAdj + 1, cy + iAdj + 1) ;
  // Tick marks
  for (i = 1 ; i < 96 ; i++)
  {
    if (i % 16 == 0) iHeight = cy / 2 ; // inches
    else if (i % 8 == 0) iHeight = cy / 3 ; // half inches
    else if (i % 4 == 0) iHeight = cy / 5 ; // quarter inches
    else if (i % 2 == 0) iHeight = cy / 8 ; // eighths
    else iHeight = cy / 12 ; // sixteenths

    MoveToEx (hdc, i * cx / 96, cy, NULL) ;
    LineTo (hdc, i * cx / 96, cy - iHeight) ;
  }
  // Create logical font
  FillMemory (&lf, sizeof (lf), 0) ;
  lf.lfHeight = cy / 2 ;
  lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

  SelectObject (hdc, CreateFontIndirect (&lf)) ;
  SetTextAlign (hdc, TA_BOTTOM | TA_CENTER) ;
  SetBkMode (hdc, TRANSPARENT) ;

  // Display numbers
```

```
  for (i = 1 ; i <= 5 ; i++)
  {
    ch = (TCHAR) (i + '0') ;
    TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
  }
  // Clean up
  DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
  DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
  HDC hdcEMF ;
  HENHMETAFILE hemf ;
  int cxMms, cyMms, cxPix, cyPix, xDpi, yDpi ;

  hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf8.emf"), NULL,
    TEXT ("EMF8\0EMF Demo #8\0")) ;
  if (hdcEMF == NULL)
    return ;
  cxMms = GetDeviceCaps (hdcEMF, HORZSIZE) ;
  cyMms = GetDeviceCaps (hdcEMF, VERTSIZE) ;
  cxPix = GetDeviceCaps (hdcEMF, HORZRES) ;
  cyPix = GetDeviceCaps (hdcEMF, VERTRES) ;

  xDpi = cxPix * 254 / cxMms / 10 ;
  yDpi = cyPix * 254 / cyMms / 10 ;

  DrawRuler (hdcEMF, 6 * xDpi, yDpi) ;
  hemf = CloseEnhMetaFile (hdcEMF) ;
  DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  ENHMETAHEADER emh ;
  HENHMETAFILE hemf ;
  int cxImage, cyImage ;
  RECT rect ;

  hemf = GetEnhMetaFile (TEXT ("emf8.emf")) ;
  GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
  cxImage = emh.rclBounds.right - emh.rclBounds.left ;
  cyImage = emh.rclBounds.bottom - emh.rclBounds.top ;

  rect.left = (cxArea - cxImage) / 2 ;
  rect.right = (cxArea + cxImage) / 2 ;
  rect.top = (cyArea - cyImage) / 2 ;
  rect.bottom = (cyArea + cyImage) / 2 ;

  PlayEnhMetaFile (hdc, hemf, &rect) ;
  DeleteEnhMetaFile (hemf) ;
}
```

EMF.C

```
/*-------------------------------------------------------------------------
EMF.C -- Enhanced MetaFile Demonstration Shell Program
(c) Charles Petzold, 1998
-------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "..\\emf8\\resource.h"

extern void CreateRoutine (HWND) ;
extern void PaintRoutine (HWND, HDC, int, int) ;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
```

```
HANDLE hInst ;
extern TCHAR szClass [] ;
extern TCHAR szTitle [] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  TCHAR szResource [] = TEXT ("EMF") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  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 = szResource ;
  wndclass.lpszClassName = szClass ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szClass, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szClass, szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

BOOL PrintRoutine (HWND hwnd)
{
  static DOCINFO di ;
  static PRINTDLG printdlg = { sizeof (PRINTDLG) } ;
  static TCHAR szMessage [32] ;
  BOOL bSuccess = FALSE ;
  HDC hdcPrn ;
  int cxPage, cyPage ;

  printdlg.Flags = PD_RETURNDC | PD_NOPAGENUMS | PD_NOSELECTION ;
  if (!PrintDlg (&printdlg))
    return TRUE ;
  if (NULL == (hdcPrn = printdlg.hDC))
    return FALSE ;
  cxPage = GetDeviceCaps (hdcPrn, HORZRES) ;
  cyPage = GetDeviceCaps (hdcPrn, VERTRES) ;

  lstrcpy (szMessage, szClass) ;
  lstrcat (szMessage, TEXT (": Printing")) ;

  di.cbSize = sizeof (DOCINFO) ;
```

```
  di.lpszDocName = szMessage ;

  if (StartDoc (hdcPrn, &di) > 0)
  {
    if (StartPage (hdcPrn) > 0)
    {
      PaintRoutine (hwnd, hdcPrn, cxPage, cyPage) ;
      if (EndPage (hdcPrn) > 0)
      {
        EndDoc (hdcPrn) ;
        bSuccess = TRUE ;
      }
    }
  }
  DeleteDC (hdcPrn) ;
  return bSuccess ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  BOOL bSuccess ;
  static int cxClient, cyClient ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    CreateRoutine (hwnd) ;
    return 0 ;

  case WM_COMMAND:
    switch (wParam)
    {
    case IDM_PRINT:
      SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
      ShowCursor (TRUE) ;

      bSuccess = PrintRoutine (hwnd) ;

      ShowCursor (FALSE) ;
      SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

      if (!bSuccess)
        MessageBox (hwnd, TEXT ("Error encountered during printing"),
        szClass, MB_ICONASTERISK | MB_OK) ;
      return 0 ;

    case IDM_EXIT:
      SendMessage (hwnd, WM_CLOSE, 0, 0) ;
      return 0 ;

    case IDM_ABOUT:
      MessageBox (hwnd, TEXT ("Enhanced MetaFile Demo Program\n")
        TEXT ("Copyright (c) Charles Petzold, 1998"),
        szClass, MB_ICONINFORMATION | MB_OK) ;
      return 0 ;
    }
    break ;

    case WM_SIZE:
      cxClient = LOWORD (lParam) ;
      cyClient = HIWORD (lParam) ;
      return 0 ;

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      PaintRoutine (hwnd, hdc, cxClient, cyClient) ;
```

```
     EndPaint (hwnd, &ps) ;
     return 0 ;

   case WM_DESTROY :
     PostQuitMessage (0) ;
     return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

EMF.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
EMF MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&Print...",    IDM_PRINT
MENUITEM SEPARATOR
MENUITEM "E&xit",     IDM_EXIT
END
POPUP "&Help"
BEGIN
MENUITEM "&About...",     IDM_ABOUT
END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Emf.rc
//
#define IDM_PRINT       40001
#define IDM_EXIT        40002
#define IDM_ABOUT       40003
```

在处理WM_CREATE消息处理期间，EMF.C呼叫名为CreateRoutine的外部函数，该函数建立MetaFile。EMF.C在两个地方呼叫PaintRoutine函数：一处在WM_PAINT消息处理期间，另一处在函数PrintRoutine中以响应菜单命令打印图像。

因为现代的打印机通常比视频显示器有更高的分辨率，打印MetaFile的能力是测试以特定大小处理图像能力的重要工具。当EMF8建立的MetaFile图像以特定大小显示时，最有意义。该图像是一把6英寸长1英寸宽的直尺，每英寸分为十六格，数字从1到5为TrueType字体。

要绘制一把6英寸的直尺，需要知道一些设备分辨率的知识。EMF8.C中的CreateRoutine函数首先建立MetaFile，然后使用从CreateEnhMetaFile传回的设备内容句柄呼叫GetDeviceCaps四次。这些呼叫取得单位分别为毫米和像素的显示平面的高度与宽度。

这听起来有点怪。MetaFile设备内容通常是作为GDI绘制命令的储存媒介，它不是像视频显示器或打印机的真正设备，那么它的宽度和高度从何而来?

您可能已经想起来了，CreateEnhMetaFile的第一个参数被称作「参考设备内容」。GDI用这为MetaFile建立设备特征。如果参数设定为NULL（如EMF8中），GDI就把显示器作为参考设备内容。因而，当EMF8使用设备内容呼叫GetDeviceCaps时，它实际上取得有关显示器的信息。

EMF8.C以像素大小除以毫米大小并乘以25.4（1英寸为25.4毫米）计算以每英寸的点数为单位的分辨率。

即使我们非常认真地以MetaFile直尺的正确大小绘制它，但是这样子作还是不够的。PlayEnhMetaFile函数在显示图像时，使用作为最后一个参数传递给它的矩形来缩放图像大小，因此该矩形必须设定为直尺的大小。

由于此原因，EMF8中的PaintRoutine函数呼叫GetEnhMetaFileHeader函数来取得MetaFile的表头信息。ENHMETAHEADER结构的rclBounds字段指出以像素为单位的MetaFile图像的围绕矩形。程序使用此信息使直尺位于显示区域中央，如图18-6所示。

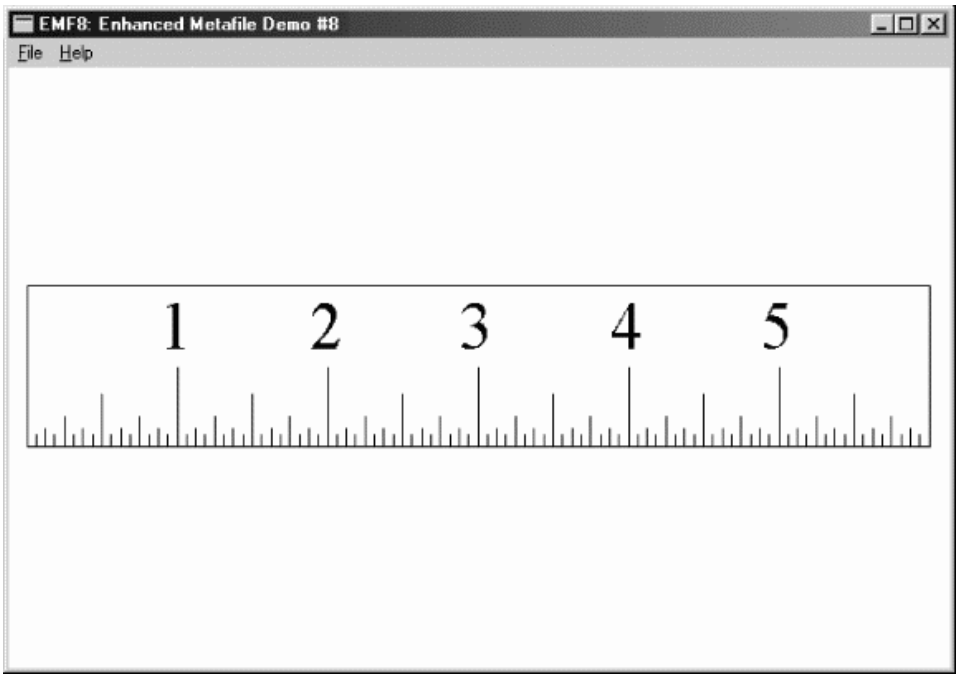


图18-6 EMF8得屏幕显示

记住，如果您拿直尺与屏幕中的直尺比较时，两者并不一定非常吻合。如同第五章中所论述的，显示器只能近似地实际显示尺寸。

既然这样做好像有用了，现在就来试着打印图像。哇！如果您有一台300dpi的激光打印机，那么打印出的直尺的宽将会是11/3英寸。这是由于我们依据视频显示器的像素尺寸来打印。虽然您可能认为这把小尺很可爱，但它不是我们所需要的。让我们再试一试。

ENHMETAHEADER结构包括两个描述图像大小的矩形结构。第一个是rclBounds，EMF8使用这个，它以像素为单位给出图像的大小。第二为rclFrame，它以0.01毫米为单位给出图像的大小。这两个字段之间的关系是由最初建立MetaFile时使用的参考设备内容决定的，在此情况下为显示器（MetaFile表头也包括两个名为szlDevice和szlMillimeters的字段，它们是SIZEL结构，分别以像素单位和毫米单位指出了参考设备的大小，这与从GetDeviceCaps得到的信息一样）。

EMF9使用图像的毫米大小信息，如程序18-11所示。

程序18-11 EMF9

EMF9.C

```
/*------------------------------------------------------------------------
EMF9.C -- Enhanced MetaFile Demo #9
(c) Charles Petzold, 1998
```

```
-------------------------------------------------------------------------*/
#include <windows.h>
#include <string.h>

TCHAR szClass [] = TEXT ("EMF9") ;
TCHAR szTitle [] = TEXT ("EMF9: Enhanced MetaFile Demo #9") ;

void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  ENHMETAHEADER emh ;
  HENHMETAFILE hemf ;
  int cxMms, cyMms, cxPix, cyPix, cxImage, cyImage ;
  RECT rect ;

  cxMms = GetDeviceCaps (hdc, HORZSIZE) ;
  cyMms = GetDeviceCaps (hdc, VERTSIZE) ;
  cxPix = GetDeviceCaps (hdc, HORZRES) ;
  cyPix = GetDeviceCaps (hdc, VERTRES) ;

  hemf = GetEnhMetaFile (TEXT ("..\\emf8\\emf8.emf")) ;
  GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
  cxImage = emh.rclFrame.right - emh.rclFrame.left ;
  cyImage = emh.rclFrame.bottom - emh.rclFrame.top ;

  cxImage = cxImage * cxPix / cxMms / 100 ;
  cyImage = cyImage * cyPix / cyMms / 100 ;

  rect.left = (cxArea - cxImage) / 2 ;
  rect.right = (cxArea + cxImage) / 2 ;
  rect.top = (cyArea - cyImage) / 2 ;
  rect.bottom = (cyArea + cyImage) / 2 ;

  PlayEnhMetaFile (hdc, hemf, &rect) ;
  DeleteEnhMetaFile (hemf) ;
}
```

EMF9使用EMF8建立的MetaFile，因此确定执行EMF8。

EMF9中的PaintRoutine函数首先使用目的设备内容呼叫GetDeviceCaps四次。像在EMF8中的CreateRoutine函数一样，这些呼叫提供有关设备分辨率的信息。在得到MetaFile句柄之后，它取得表头结构并使用rclFrame字段来计算以0.01毫米为单位的MetaFile图像大小。这是第一步。

然后，函数通过乘以输出设备的像素大小、除以毫米大小再除以100（因为度量尺寸以0.01毫米为单位）将此大小转换为像素大小。现在，PaintRoutine函数具有以像素为单位的直尺大小 – 与显示器无关。这是适合目的设备的像素大小，而且很容易使图像居中对齐。

就显示器而言，EMF9的显示与EMF8显示的一样。但是如果从EMF9打印直尺，您会看到更正常的直尺 – 6英寸长、1英寸宽。

缩放比例和纵横比

您也可能想要使用EMF8建立的直尺MetaFile，而不必显示6英寸的图像。保持图像正确的6比1的纵横比是重要的。如前所述，在文书处理程序或别的应用程序中使用围绕方框来改变MetaFile的大小是很方便的，但是这样会导致某种程度的失真。在这种应用程序中，应该给使用者一个选项来保持原先的纵横比，而不用管围绕方框的大小如何变化。这就是说，传递给PlayEnhMetaFile的矩形结构不能直接由使用者选择的围绕方框定义。传递给该函数的矩形结构只是围绕方框的一部分。

让我们看一看程序18-12 EMF10是如何做的。

程序18-12 EMF10

EMF10.C

```
/*------------------------------------------------------------------------
EMF10.C -- Enhanced MetaFile Demo #10
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
TCHAR szClass [] = TEXT ("EMF10") ;
TCHAR szTitle [] = TEXT ("EMF10: Enhanced MetaFile Demo #10") ;
void CreateRoutine (HWND hwnd)
{
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  ENHMETAHEADER emh ;
  float fScale ;
  HENHMETAFILE hemf ;
  int cxMms, cyMms, cxPix, cyPix, cxImage, cyImage ;
  RECT rect ;

  cxMms = GetDeviceCaps (hdc, HORZSIZE) ;
  cyMms = GetDeviceCaps (hdc, VERTSIZE) ;
  cxPix = GetDeviceCaps (hdc, HORZRES) ;
  cyPix = GetDeviceCaps (hdc, VERTRES) ;

  hemf = GetEnhMetaFile (TEXT ("..\\emf8\\emf8.emf")) ;

  GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;

  cxImage = emh.rclFrame.right - emh.rclFrame.left ;
  cyImage = emh.rclFrame.bottom - emh.rclFrame.top ;

  cxImage = cxImage * cxPix / cxMms / 100 ;
  cyImage = cyImage * cyPix / cyMms / 100 ;

  fScale = min ((float) cxArea / cxImage, (float) cyArea / cyImage) ;

  cxImage = (int) (fScale * cxImage) ;
  cyImage = (int) (fScale * cyImage) ;

  rect.left = (cxArea - cxImage) / 2 ;
  rect.right = (cxArea + cxImage) / 2 ;
  rect.top = (cyArea - cyImage) / 2 ;
  rect.bottom = (cyArea + cyImage) / 2 ;
  PlayEnhMetaFile (hdc, hemf, &rect) ;
  DeleteEnhMetaFile (hemf) ;
}
```

EMF10伸展直尺图像以适应显示区域（或打印页面的可打印部分），但不会失真。通常直尺会伸展到显示区域的整个宽度，但是会上下居中对齐。如果您把窗口拉得太小，则直尺会与显示区域一般高，但是会水平居中对齐。

可能有许多种方法来计算合适的显示矩形，但是我们只根据EMF9的方式完成该项工作。EMF10.C中的PaintRoutine函数开始部分与EMF9.C相同，为目的设备内容计算6英…汲さ闹背咄枷祫实钡耐妓卮笮　?/p>

然后，程序计算名为fScale的浮点值，它是显示区域宽度与图像宽度的比值以及显示区域高度与图像高度比值两者的最小值。这个因子在计算围绕矩形前用于增加图像的像素大小。

MetaFile中的映像方式

前面绘制的直尺单位有英…迹　灿泻撩住U庵止ぶ魇褂宵DI提供的各种映射方式似乎非常适合。但是我坚持使用像素，并「手工」完成所有必要的计算。为什么呢?

答案很简单，就是将映射方式与MetaFile一起使用会十分混乱。我们不妨实验一下。

当使用MetaFile设备内容呼叫SetMapMode时，该函数在MetaFile中像其它GDI函数一样被编码。如程序18-13 EMF11显示的那样。

程序18-13 EMF11

EMF11.C

```
/*---------------------------------------------------------------------------
EMF11.C -- Enhanced MetaFile Demo #11
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
TCHAR szClass [] = TEXT ("EMF11") ;
TCHAR szTitle [] = TEXT ("EMF11: Enhanced MetaFile Demo #11") ;

void DrawRuler (HDC hdc, int cx, int cy)
{
  int i, iHeight ;
  LOGFONT lf ;
  TCHAR ch ;

  // Black pen with 1-point width

  SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
  // Rectangle surrounding entire pen (with adjustment)
  if (GetVersion () & 0x80000000)  // Windows 98
    Rectangle (hdc, 0, -2, cx + 2, cy) ;
  else
    // Windows NT
    Rectangle (hdc, 0, -1, cx + 1, cy) ;

  // Tick marks

  for (i = 1 ; i < 96 ; i++)
  {
    if(i %16== 0) iHeight = cy /2 ; // inches
    else if(i % 8 == 0)iHeight = cy / 3 ; // half inches
    else if(i % 4 == 0)iHeight = cy / 5 ; // quarter inches
    else if(i % 2 == 0)iHeight = cy / 8 ; // eighths
    else iHeight = cy /12 ; // sixteenths

    MoveToEx (hdc, i * cx / 96, 0, NULL) ;
    LineTo (hdc, i * cx / 96, iHeight) ;
  }
  // Create logical font
  FillMemory (&lf, sizeof (lf), 0) ;
  lf.lfHeight = cy / 2 ;
  lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

  SelectObject (hdc, CreateFontIndirect (&lf)) ;
  SetTextAlign (hdc, TA_BOTTOM | TA_CENTER) ;
  SetBkMode (hdc, TRANSPARENT) ;

  // Display numbers

  for (i = 1 ; i <= 5 ; i++)
  {
    ch = (TCHAR) (i + '0') ;
    TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
  }
  // Clean up
  DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
  DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
  HDC hdcEMF ;
  HENHMETAFILE hemf ;
```

```
  hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf11.emf"), NULL,
    TEXT ("EMF11\0EMF Demo #11\0")) ;
  SetMapMode (hdcEMF, MM_LOENGLISH) ;
  DrawRuler (hdcEMF, 600, 100) ;
  hemf = CloseEnhMetaFile (hdcEMF) ;
  DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  ENHMETAHEADER emh ;
  HENHMETAFILE hemf ;
  int cxMms, cyMms, cxPix, cyPix, cxImage, cyImage ;
  RECT rect ;

  cxMms = GetDeviceCaps (hdc, HORZSIZE) ;
  cyMms = GetDeviceCaps (hdc, VERTSIZE) ;
  cxPix = GetDeviceCaps (hdc, HORZRES) ;
  cyPix = GetDeviceCaps (hdc, VERTRES) ;

  hemf = GetEnhMetaFile (TEXT ("emf11.emf")) ;
  GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
  cxImage = emh.rclFrame.right - emh.rclFrame.left ;
  cyImage = emh.rclFrame.bottom - emh.rclFrame.top ;

  cxImage = cxImage * cxPix / cxMms / 100 ;
  cyImage = cyImage * cyPix / cyMms / 100 ;

  rect.left = (cxArea - cxImage) / 2 ;
  rect.top = (cyArea - cyImage) / 2 ;
  rect.right = (cxArea + cxImage) / 2 ;
  rect.bottom = (cyArea + cyImage) / 2 ;

  PlayEnhMetaFile (hdc, hemf, &rect) ;
  DeleteEnhMetaFile (hemf) ;
}
```

EMF11中的CreateRoutine函数比EMF8（最初的直尺metafile程序）中的那个简单，因为它不需要呼叫GetDeviceCaps来确定以每英寸点数为单位的显示器分辨率。相反，EMF11呼叫SetMapMode将映射方式设定为MM_LOENGLISH，其逻辑单位等於0.01英寸。因而，直尺的大小为600×100个单位，并将这些数值传递给DrawRuler。

除了MoveToEx和LineTo呼叫绘制直尺的刻度外，EMF11中的DrawRuler函数与EMF9中的一样。当以像素单位绘制时（内定的MM_TEXT映像方式），垂直轴上的单位沿着屏幕向下增长。对于MM_LOENGLISH映像方式（以及其它度量映像方式），则向上增长。这就需要修改程序代码。同时，也需要更改Rectangle函数中的调节因子。

EMF11中的PaintRoutine函数基本上与EMF9中的相同，那个版本的程序能在显示器和打印机上以正确尺寸显示直尺。唯一不同之处在于EMF11使用EMF11.EMF文件，而EMF9使用EMF8建立的EMF8.EMF文件。

EMF11显示的图像基本上与EMF9所显示的相同。因此，在这里可以看到将SetMapMode呼叫嵌入MetaFile能够简化MetaFile的建立，而且不影响以其正确大小显示MetaFile的机制。

映射与显示

在EMF11中计算目的矩形包括对GetDeviceCaps的几个呼叫。我们的第二个目的是使用映像方式代替这些呼叫。GDI将目的矩形的坐标视为逻辑坐标。为这些坐标使用MM_HIMETRIC似乎是个好方案，因为它使用0.01毫米作为逻辑单位，与增强型MetaFile表头中用于围绕矩形的单位相同。

程序18-14中所示的EMF12程序，保留了EMF8中使用的DrawRuler处理方式，但是使用

MM_HIMETRIC映像方式显示MetaFile。

程序18-14 EMF12

EMF12.C

```
/*------------------------------------------------------------------------
EMF12.C -- Enhanced MetaFile Demo #12
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
TCHAR szClass [] = TEXT ("EMF12") ;
TCHAR szTitle [] = TEXT ("EMF12: Enhanced MetaFile Demo #12") ;
void DrawRuler (HDC hdc, int cx, int cy)
{
  int iAdj, i, iHeight ;
  LOGFONT lf ;
  TCHAR ch ;

  iAdj = GetVersion () & 0x80000000 ? 0 : 1 ;
  // Black pen with 1-point width
  SelectObject (hdc, CreatePen (PS_SOLID, cx / 72 / 6, 0)) ;
  // Rectangle surrounding entire pen (with adjustment)
  Rectangle (hdc, iAdj, iAdj, cx + iAdj + 1, cy + iAdj + 1) ;
  // Tick marks
  for (i = 1 ; i < 96 ; i++)
  {
    if (i % 16 == 0) iHeight = cy / 2 ; // inches
    else if (i % 8 == 0) iHeight = cy / 3 ; // half inches
    else if (i % 4 == 0) iHeight = cy / 5 ; // quarter inches
    else if (i % 2 == 0) iHeight = cy / 8 ; // eighths
    else iHeight = cy / 12 ; // sixteenths

    MoveToEx (hdc, i * cx / 96, cy, NULL) ;
    LineTo (hdc, i * cx / 96, cy - iHeight) ;
  }
  // Create logical font
  FillMemory (&lf, sizeof (lf), 0) ;
  lf.lfHeight = cy / 2 ;
  lstrcpy (lf.lfFaceName, TEXT ("Times New Roman")) ;

  SelectObject (hdc, CreateFontIndirect (&lf)) ;
  SetTextAlign (hdc, TA_BOTTOM | TA_CENTER) ;
  SetBkMode (hdc, TRANSPARENT) ;

  // Display numbers

  for (i = 1 ; i <= 5 ; i++)
  {
    ch = (TCHAR) (i + '0') ;
    TextOut (hdc, i * cx / 6, cy / 2, &ch, 1) ;
  }
  // Clean up
  DeleteObject (SelectObject (hdc, GetStockObject (SYSTEM_FONT))) ;
  DeleteObject (SelectObject (hdc, GetStockObject (BLACK_PEN))) ;
}

void CreateRoutine (HWND hwnd)
{
  HDC hdcEMF ;
  HENHMETAFILE hemf ;
  int cxMms, cyMms, cxPix, cyPix, xDpi, yDpi ;

  hdcEMF = CreateEnhMetaFile (NULL, TEXT ("emf12.emf"), NULL,
    TEXT ("EMF13\0EMF Demo #12\0")) ;

  cxMms = GetDeviceCaps (hdcEMF, HORZSIZE) ;
  cyMms = GetDeviceCaps (hdcEMF, VERTSIZE) ;
  cxPix = GetDeviceCaps (hdcEMF, HORZRES) ;
```

```
  cyPix = GetDeviceCaps (hdcEMF, VERTRES) ;

  xDpi = cxPix * 254 / cxMms / 10 ;
  yDpi = cyPix * 254 / cyMms / 10 ;

  DrawRuler (hdcEMF, 6 * xDpi, yDpi) ;
  hemf = CloseEnhMetaFile (hdcEMF) ;
  DeleteEnhMetaFile (hemf) ;
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  ENHMETAHEADER emh ;
  HENHMETAFILE hemf ;
  POINT pt ;
  int cxImage, cyImage ;
  RECT rect ;

  SetMapMode (hdc, MM_HIMETRIC) ;
  SetViewportOrgEx (hdc, 0, cyArea, NULL) ;
  pt.x = cxArea ;
  pt.y = 0 ;

  DPtoLP (hdc, &pt, 1) ;
  hemf = GetEnhMetaFile (TEXT ("emf12.emf")) ;
  GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;
  cxImage = emh.rclFrame.right - emh.rclFrame.left ;
  cyImage = emh.rclFrame.bottom - emh.rclFrame.top ;

  rect.left = (pt.x - cxImage) / 2 ;
  rect.top = (pt.y + cyImage) / 2 ;
  rect.right = (pt.x + cxImage) / 2 ;
  rect.bottom = (pt.y - cyImage) / 2 ;

  PlayEnhMetaFile (hdc, hemf, &rect) ;
  DeleteEnhMetaFile (hemf) ;
}
```

EMF12中的PaintRoutine函数首先将映像方式设定为MM_HIMETRIC。像其它度量映像方式一样，y值沿着屏幕向上增长。然而，原点坐标仍在屏幕的左上角，这就意味显示区域内的y坐标值是负数。为了纠正这个问题，程序呼叫SetViewportOrgEx将原点坐标设定在左下角。

设备坐标(cxArea,0)位于屏幕的右上角。把该坐标点传递给DPtoLP（「设备坐标点到逻辑坐标点」）函数，得到以0.01毫米为单位的显示区域大小。

然后，程序加载MetaFile，取得文件表头，并找到以0.01毫米为单位的MetaFile大小。这样计算目的矩形在显示区域居中对齐的位置就变得十分简单。

现在我们看到了在建立MetaFile时能够使用映射方式，显示它时也能使用映射方式。我们能一起完成它们吗?

如程序18-15 EMF13展示的那样，这是可以的。

程序18-15 EMF13

EMF13.C

```
/*------------------------------------------------------------------------
EMF13.C -- Enhanced MetaFile Demo #13
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
TCHAR szClass [] = TEXT ("EMF13") ;
TCHAR szTitle [] = TEXT ("EMF13: Enhanced MetaFile Demo #13") ;

void CreateRoutine (HWND hwnd)
{
```

```
}

void PaintRoutine (HWND hwnd, HDC hdc, int cxArea, int cyArea)
{
  ENHMETAHEADER emh ;
  HENHMETAFILE hemf ;
  POINT pt ;
  int cxImage, cyImage ;
  RECT rect ;

  SetMapMode (hdc, MM_HIMETRIC) ;
  SetViewportOrgEx (hdc, 0, cyArea, NULL) ;
  pt.x = cxArea ;
  pt.y = 0 ;

  DPtoLP (hdc, &pt, 1) ;

  hemf = GetEnhMetaFile (TEXT ("..\\emf11\\emf11.emf")) ;

  GetEnhMetaFileHeader (hemf, sizeof (emh), &emh) ;

  cxImage = emh.rclFrame.right - emh.rclFrame.left ;
  cyImage = emh.rclFrame.bottom - emh.rclFrame.top ;

  rect.left = (pt.x - cxImage) / 2 ;
  rect.top = (pt.y + cyImage) / 2 ;
  rect.right = (pt.x + cxImage) / 2 ;
  rect.bottom = (pt.y - cyImage) / 2 ;

  PlayEnhMetaFile (hdc, hemf, &rect) ;
  DeleteEnhMetaFile (hemf) ;
}
```

在EMF13中，由于直尺 MetaFile 已由EMF11建立，所以它没有使用映射方式建立 MetaFile。EMF13只是简单地加载 MetaFile，然后像EMF11一样使用映射方式计算目的矩形。

现在，我们可以建立一些规则。在建立 MetaFile 时，GDI使用对映射方式的任意嵌入修改，来计算以像素和毫米为单位的MetaFile图像的大小。图像的大小储存在 MetaFile 表头内。在显示 MetaFile 时，GDI在呼叫 PlayEnhMetaFile 时根据有效的映像方式建立目的矩形的实际位置，而本来的 MetaFile 中并没有任何记录去更改这个位置。

进阶篇

第十九章 多重文件界面

多重文件接口（MDI）是Microsoft Windows文件处理应用程序的一种规范，该规范描述了窗口结构和允许使用者在单个应用程序中使用多个文件的使用者接口（如文书处理程序中的文字文件和电子表格程序中的电子表格）。简单地说，就像Windows在一个屏幕上维护多个应用程序窗口一样，MDI应用程序在一个显示区域内维护多个文件窗口。Windows中的第一个MDI应用程序是Windows下的Microsoft Excel的第一个版本。紧接着又出现了许多其它的应用程序。

MDI 概念

尽管MDI规范随着Windows 2.0的推出已经很普及，但在那时，MDI应用程序写起来很困难，并且需要一些非常复杂的程序设计工作。从Windows 3.0起，其中许多工作就都由Windows为您做好了。Windows 95中增强的支持也已经被添加进Windows 98和Microsoft Windows NT中。

MDI 的组成

MDI程序的主应用程序窗口是很普通的：它有一个标题栏、一个菜单、一个缩放边框、一个系统菜单图标和最大化/最小化/关闭按钮。显示区域经常被称为「工作空间」，它不直接用于显示程序输出。这个工作空间包括零个或多个子窗口，每个窗口都显示一个文件。

这些子窗口看起来与通常的应用程序窗口以及MDI程序的主窗口很相似。它们有一个标题栏、一个缩放边框、一个系统菜单图标和最大化/最小化/关闭按钮，可能还包括滚动条。但是文件窗口没有菜单，主应用程序窗口上的菜单适用于文件窗口。

在任何时候都只能有一个文件窗口是活动的（加亮标题栏来表示），它出现在其它所有文件窗口之前。所有文件窗口都由工作空间区域加以剪裁，而不会出现在应用程序窗口之外。

初看起来，对Windows程序写作者来说，MDI似乎是相当简单。需要程序写作者做的工作好像就是为每个文件建立一个WS_CHILD窗口，并使程序的主应用程序窗口成为文件窗口的父窗口。但对现有的MDI应用程序稍加研究，就会发现一些导致程序写作困难的复杂问题。例如：

MDI文件窗口可以最小化。它的图标出现在工作空间的底部。一般来说，MDI应用程序可以将不同的图标分别用于主应用程序窗口和每一类文件应用。

MDI文件窗口可以最大化。在这种情况下，文件窗口的标题栏（一般用来显示窗口中文件的文件名称）消失，文件名称出现在应用程序窗口标题栏的应用程序名称之后，文件窗口的系统菜单图标成为应用程序窗口的顶层菜单中的第一项。关闭文件窗口按钮变成顶层菜单中的最后一项，且出现在最右边。

用以关闭文件窗口的系统键盘快捷键与关闭主窗口的系统键盘快捷键一样，只是Ctrl键代替了Alt键。这也就是说，Alt+F4用于关闭应用程序窗口，而Ctrl+F4用于关闭文件窗口。此外，Ctrl+F6可以在活动MDI应用程序的子文件窗口之间切换。与平时一样，Alt+空格键启动主窗口的系统菜单，Alt+-（减号）启动活动子文件窗口的系统菜单。

当使用光标键在菜单项间移动时，控件权通常从系统菜单转到菜单列中的第一项。在MDI

应用程序中，控件权是从应用程序系统菜单转到活动文件系统菜单，然后再转到菜单列中的第一项。

如果应用程序能够支持若干种型态的子窗口（如Microsoft Excel中的工作表和图表文件），那么菜单应能反映出与这种型态的文件有关的操作。这就要求当不同的文字窗口变成活动窗口时，程序能更换菜单。此外，当没有文件窗口存在时，菜单应该被缩减到只剩下与打开新文件有关的操作。

顶层菜单上有一个叫做「窗口（Window）」的菜单项。按照习惯，这是顶层菜单上「Help」之前的那一项，即倒数第二项。「窗口」子菜单上通常包含在工作空间内安排文件窗口的选项。文件窗口可以从左上方开始「平铺」或「层迭」。在前一种方式下，可以完整地看到每一个文件窗口。这个子菜单同时也包含所有文件窗口的列表。从中选择一个文件窗口，就可以把此文件窗口移到前景。

Windows 98支持MDI的所有这些方面。当然，需要您做一些工作（如下面的范例程序所示），但是，这远不是要您程序写作来直接支持所有这些功能。

MDI支援

探讨Windows的MDI支持时需要发表一些新术语。主应用程序窗口称为「框架窗口」，就像传统的Windows程序一样，它是WS_OVERLAPPEDWINDOW样式的窗口。

MDI应用程序还根据预先定义的窗口类别MDICLIENT建立「客户窗口」，这一客户窗口是用这种窗口类别和WS_CHILD样式呼叫CreateWindow来建立的。这一呼叫的最后一个参数是指向一个CLIENTCREATESTRUCT型态的结构的指针。这个客户窗口覆盖框架窗口的显示区域，并提供许多MDI支持。此客户窗口的颜色是系统颜色COLOR_APPWORKSPACE。

文件窗口被称为「子窗口」。通过初始化一个MDICREATESTRUCT型态的结构，以一个指向此结构的指针为参数将消息WM_MDICREATE发送给客户窗口，就可以建立这些文件窗口。

文件窗口是客户窗口的子窗口，而客户窗口又是框架窗口的子窗口。父-子窗口分层结构如图19-1所示。

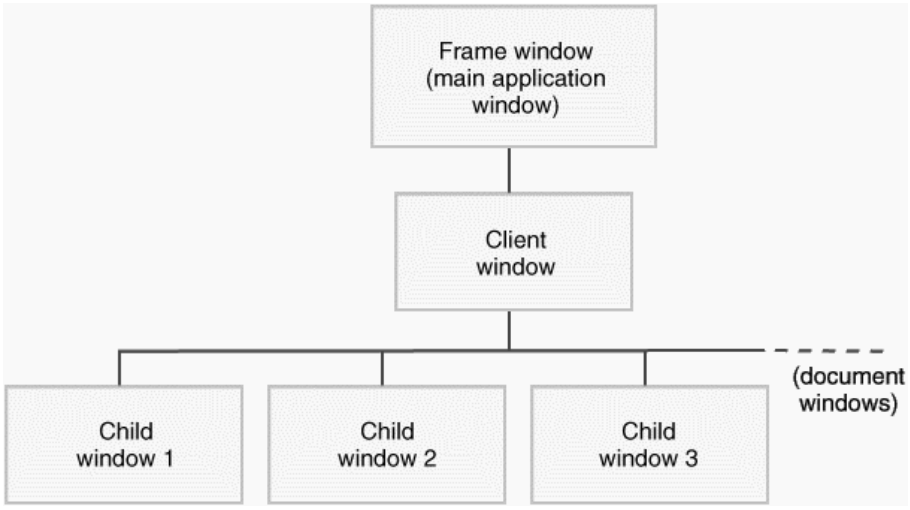


图19-1 Windows MDI应用程序的父-子层次图

您需要框架窗口的窗口类别（及窗口消息处理程序）和一个由应用程序支持的每类子窗口的窗口类别（及窗口消息处理程序）。由于已经预先注册了窗口类别，所以不需要客户窗口的窗口消息处理程序。

Windows 98的MDI支持包括一个窗口类别、五个函数、两个数据结构和12个消息。前面已经提到了MDI窗口类别，即MDICLIENT，以及数据结构CLIENTCREATESTRUCT和MDICREATESTRUCT。在MDI应用程序中，这五个函数中的两个用于取代DefWindowProc：不再将DefWindowProc呼叫用于所有未处理的消息，而是由框架窗口过程调用DefFrameProc，子窗口过程调用DefMDIChildProc。另一个MDI特有的函数TranslateMDISysAccel与第十章中讨论的TranslateAccelerator的使用方式相同。MDI支持也包括ArrangeIconicWindows函数，但有一条专用的MDI消息使得此函数对MDI程序来说不再必要。

第五个MDI函数是CreateMDIWindow，它使得子窗口可以在单独的线程中被建立。这个函数不需要在单线程的程序中，我会展示这一点。

在下面的程序中，我将展示12条MDI消息中的9条（其它三个消息一般不用），这些消息的前缀是WM_MDI。框架窗口向客户窗口发送其中某个消息，以便在子窗口上完成一项操作或者取得关于子窗口的信息（例如，框架窗口发送一个WM_MDICREATE消息给客户窗口，以建立子窗口）。消息WM_MDIACTIVATE消息有点特别：框架窗口可以发送这个消息给客户窗口来启动一个子窗口，而客户窗口也把这个消息发送给将被启动或者失去活动的子窗口，以便通知它们这一变化。

MDI 的范例程序

程序19-1 MDIDEMO程序说明了编写MDI应用程序的基本方法。

程序19-1 MDIDEMO

MDIDEMO.C

```
/*------------------------------------------------------------------------
MDIDEMO.C -- Multiple-Document Interface Demonstration
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define INIT_MENU_POS 0
#define HELLO_MENU_POS 2
#define RECT_MENU_POS 1

#define IDM_FIRSTCHILD 50000

LRESULT CALLBACK FrameWndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK CloseEnumProc (HWND, LPARAM) ;
LRESULT CALLBACK HelloWndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK RectWndProc (HWND, UINT, WPARAM, LPARAM) ;

// structure for storing data unique to each Hello child window

typedef struct tagHELLODATA
{
  UINT iColor ;
  COLORREF clrText ;
}
HELLODATA, * PHELLODATA ;
// structure for storing data unique to each Rect child window
typedef struct tagRECTDATA
{
  short cxClient ;
  short cyClient ;
}
RECTDATA, * PRECTDATA ;
// global variables
TCHAR szAppName[] = TEXT ("MDIDemo") ;
```

```
TCHAR szFrameClass[] = TEXT ("MdiFrame") ;
TCHAR szHelloClass[] = TEXT ("MdiHelloChild") ;
TCHAR szRectClass[] = TEXT ("MdiRectChild") ;
HINSTANCE hInst ;
HMENU hMenuInit, hMenuHello, hMenuRect ;
HMENU hMenuInitWindow, hMenuHelloWindow, hMenuRectWindow ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HACCEL hAccel ;
  HWND hwndFrame, hwndClient ;
  MSG msg ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  // Register the frame window class
  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = FrameWndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) (COLOR_APPWORKSPACE + 1) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szFrameClass ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  // Register the Hello child window class
  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = HelloWndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = sizeof (HANDLE) ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szHelloClass ;

  RegisterClass (&wndclass) ;
  // Register the Rect child window class
  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = RectWndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = sizeof (HANDLE) ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szRectClass ;

  RegisterClass (&wndclass) ;
  // Obtain handles to three possible menus & submenus
  hMenuInit = LoadMenu (hInstance, TEXT ("MdiMenuInit")) ;
  hMenuHello = LoadMenu (hInstance, TEXT ("MdiMenuHello")) ;
  hMenuRect = LoadMenu (hInstance, TEXT ("MdiMenuRect")) ;

  hMenuInitWindow = GetSubMenu (hMenuInit, INIT_MENU_POS) ;
  hMenuHelloWindow = GetSubMenu (hMenuHello, HELLO_MENU_POS) ;
  hMenuRectWindow = GetSubMenu (hMenuRect, RECT_MENU_POS) ;
```

```
  // Load accelerator table

  hAccel = LoadAccelerators (hInstance, szAppName) ;
  // Create the frame window
  hwndFrame = CreateWindow (szFrameClass, TEXT ("MDI Demonstration"),
    WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, hMenuInit, hInstance, NULL) ;
  hwndClient = GetWindow (hwndFrame, GW_CHILD) ;
  ShowWindow (hwndFrame, iCmdShow) ;
  UpdateWindow (hwndFrame) ;

  // Enter the modified message loop
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if ( !TranslateMDISysAccel (hwndClient, &msg) &&
      !TranslateAccelerator (hwndFrame, hAccel, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  // Clean up by deleting unattached menus
  DestroyMenu (hMenuHello) ;
  DestroyMenu (hMenuRect) ;

  return msg.wParam ;
}

LRESULT CALLBACK FrameWndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndClient ;
  CLIENTCREATESTRUCT clientcreate ;
  HWND hwndChild ;
  MDICREATESTRUCT mdicreate ;

  switch (message)
  {
  case WM_CREATE: // Create the client window

    clientcreate.hWindowMenu = hMenuInitWindow ;
    clientcreate.idFirstChild = IDM_FIRSTCHILD ;

    hwndClient = CreateWindow ( TEXT ("MDICLIENT"), NULL,
      WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
      0, 0, 0, 0, hwnd, (HMENU) 1, hInst,
      (PSTR) &clientcreate) ;
    return 0 ;
  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_NEWHELLO: // Create a Hello child window
      mdicreate.szClass = szHelloClass ;
      mdicreate.szTitle = TEXT ("Hello") ;
      mdicreate.hOwner = hInst ;
      mdicreate.x = CW_USEDEFAULT ;
      mdicreate.y = CW_USEDEFAULT ;
      mdicreate.cx = CW_USEDEFAULT ;
      mdicreate.cy = CW_USEDEFAULT ;
      mdicreate.style = 0 ;
      mdicreate.lParam = 0 ;

      hwndChild = (HWND) SendMessage (hwndClient,
        WM_MDICREATE, 0, (LPARAM) (LPMDICREATESTRUCT) &mdicreate) ;
      return 0 ;

    case IDM_FILE_NEWRECT: // Create a Rect child window
```

```
    mdicreate.szClass = szRectClass ;
    mdicreate.szTitle = TEXT ("Rectangles") ;
    mdicreate.hOwner = hInst ;
    mdicreate.x = CW_USEDEFAULT ;
    mdicreate.y = CW_USEDEFAULT ;
    mdicreate.cx = CW_USEDEFAULT ;
    mdicreate.cy = CW_USEDEFAULT ;
    mdicreate.style = 0 ;
    mdicreate.lParam = 0 ;

    hwndChild = (HWND) SendMessage (hwndClient,
      WM_MDICREATE, 0,
      (LPARAM) (LPMDICREATESTRUCT) &mdicreate) ;
    return 0 ;

  case IDM_FILE_CLOSE: // Close the active window

    hwndChild = (HWND) SendMessage (hwndClient,
      WM_MDIGETACTIVE, 0, 0) ;

    if (SendMessage (hwndChild, WM_QUERYENDSESSION, 0, 0))
      SendMessage (hwndClient, WM_MDIDESTROY,
      (WPARAM) hwndChild, 0) ;
    return 0 ;

  case IDM_APP_EXIT:// Exit the program

    SendMessage (hwnd, WM_CLOSE, 0, 0) ;
    return 0 ;

    // messages for arranging windows

  case IDM_WINDOW_TILE:
    SendMessage (hwndClient, WM_MDITILE, 0, 0) ;
    return 0 ;

  case IDM_WINDOW_CASCADE:
    SendMessage (hwndClient, WM_MDICASCADE, 0, 0) ;
    return 0 ;

  case IDM_WINDOW_ARRANGE:
    SendMessage (hwndClient, WM_MDIICONARRANGE, 0, 0) ;
    return 0 ;

  case IDM_WINDOW_CLOSEALL: // Attempt to close all children

    EnumChildWindows (hwndClient, CloseEnumProc, 0) ;
    return 0 ;

  default: // Pass to active child...

    hwndChild = (HWND) SendMessage (hwndClient,
      WM_MDIGETACTIVE, 0, 0) ;
    if (IsWindow (hwndChild))
      SendMessage (hwndChild, WM_COMMAND, wParam, lParam) ;

    break ; // ...and then to DefFrameProc
  }
  break ;

  case WM_QUERYENDSESSION:
  case WM_CLOSE: // Attempt to close all children

    SendMessage (hwnd, WM_COMMAND, IDM_WINDOW_CLOSEALL, 0) ;

    if (NULL != GetWindow (hwndClient, GW_CHILD))
      return 0 ;
```

```
     break ; // i.e., call DefFrameProc
    case WM_DESTROY:
     PostQuitMessage (0) ;
     return 0 ;
 }
 // Pass unprocessed messages to DefFrameProc (not DefWindowProc)

 return DefFrameProc (hwnd, hwndClient, message, wParam, lParam) ;
}

BOOL CALLBACK CloseEnumProc (HWND hwnd, LPARAM lParam)
{
  if (GetWindow (hwnd, GW_OWNER)) // Check for icon title
    return TRUE ;

  SendMessage (GetParent (hwnd), WM_MDIRESTORE, (WPARAM) hwnd, 0) ;
  if (!SendMessage (hwnd, WM_QUERYENDSESSION, 0, 0))
    return TRUE ;
  SendMessage (GetParent (hwnd), WM_MDIDESTROY, (WPARAM) hwnd, 0) ;
  return TRUE ;
}

LRESULT CALLBACK HelloWndProc (HWND hwnd, UINT message,
                           WPARAM wParam, LPARAM lParam)
{
  static COLORREF clrTextArray[] = { RGB (0, 0, 0), RGB (255, 0, 0),
    RGB (0, 255, 0), RGB ( 0, 0, 255),
    RGB (255, 255, 255) } ;
  static HWND hwndClient, hwndFrame ;
  HDC hdc ;
  HMENU hMenu ;
  PHELLODATA pHelloData ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_CREATE:
    // Allocate memory for window private data

    pHelloData = (PHELLODATA) HeapAlloc (GetProcessHeap (),
      HEAP_ZERO_MEMORY, sizeof (HELLODATA)) ;
    pHelloData->iColor = IDM_COLOR_BLACK ;
    pHelloData->clrText = RGB (0, 0, 0) ;
    SetWindowLong (hwnd, 0, (long) pHelloData) ;

    // Save some window handles

    hwndClient = GetParent (hwnd) ;
    hwndFrame = GetParent (hwndClient) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_COLOR_BLACK:
    case IDM_COLOR_RED:
    case IDM_COLOR_GREEN:
    case IDM_COLOR_BLUE:
    case IDM_COLOR_WHITE:
      // Change the text color

      pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;

      hMenu = GetMenu (hwndFrame) ;

      CheckMenuItem (hMenu, pHelloData->iColor, MF_UNCHECKED) ;
      pHelloData->iColor = wParam ;
      CheckMenuItem (hMenu, pHelloData->iColor, MF_CHECKED) ;
```

```
    pHelloData->clrText = clrTextArray[wParam - IDM_COLOR_BLACK] ;

    InvalidateRect (hwnd, NULL, FALSE) ;
   }
   return 0 ;

   case WM_PAINT:
    // Paint the window

    hdc = BeginPaint (hwnd, &ps) ;

    pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
    SetTextColor (hdc, pHelloData->clrText) ;

    GetClientRect (hwnd, &rect) ;

    DrawText (hdc, TEXT ("Hello, World!"), -1, &rect,
      DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
    EndPaint (hwnd, &ps) ;
    return 0 ;

   case WM_MDIACTIVATE:
    // Set the Hello menu if gaining focus

    if (lParam == (LPARAM) hwnd)
      SendMessage (hwndClient, WM_MDISETMENU,(WPARAM)
      hMenuHello, (LPARAM) hMenuHelloWindow) ;

    // Check or uncheck menu item

    pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
    CheckMenuItem (hMenuHello, pHelloData->iColor,
      (lParam == (LPARAM) hwnd) ? MF_CHECKED : MF_UNCHECKED) ;

    // Set the Init menu if losing focus

    if (lParam != (LPARAM) hwnd)
      SendMessage (hwndClient, WM_MDISETMENU, (WPARAM)
      hMenuInit,(LPARAM) hMenuInitWindow) ;

    DrawMenuBar (hwndFrame) ;
    return 0 ;

   case WM_QUERYENDSESSION:
   case WM_CLOSE:
    if (IDOK != MessageBox (hwnd, TEXT ("OK to close window?"),
      TEXT ("Hello"),
      MB_ICONQUESTION | MB_OKCANCEL))
      return 0 ;

    break ; // i.e., call DefMDIChildProc

   case WM_DESTROY:
    pHelloData = (PHELLODATA) GetWindowLong (hwnd, 0) ;
    HeapFree (GetProcessHeap (), 0, pHelloData) ;
    return 0 ;
  }
  // Pass unprocessed message to DefMDIChildProc

  return DefMDIChildProc (hwnd, message, wParam, lParam) ;
}
LRESULT CALLBACK RectWndProc ( HWND hwnd, UINT message,WPARAM wParam, LPARAM lParam)
{
  static HWND hwndClient, hwndFrame ;
  HBRUSH hBrush ;
  HDC hdc ;
  PRECTDATA pRectData ;
  PAINTSTRUCT ps ;
```

```
int xLeft, xRight, yTop, yBottom ;
short nRed, nGreen, nBlue ;

switch (message)
{
case WM_CREATE:
  // Allocate memory for window private data

  pRectData = (PRECTDATA) HeapAlloc (GetProcessHeap (),
    HEAP_ZERO_MEMORY, sizeof (RECTDATA)) ;

  SetWindowLong (hwnd, 0, (long) pRectData) ;

  // Start the timer going

  SetTimer (hwnd, 1, 250, NULL) ;

  // Save some window handles
  hwndClient = GetParent (hwnd) ;
  hwndFrame = GetParent (hwndClient) ;
  return 0 ;

case WM_SIZE: // If not minimized, save the window size

  if (wParam != SIZE_MINIMIZED)
  {
    pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;

    pRectData->cxClient = LOWORD (lParam) ;
    pRectData->cyClient = HIWORD (lParam) ;
  }

  break ; // WM_SIZE must be processed by DefMDIChildProc

case WM_TIMER: // Display a random rectangle

  pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;
  xLeft = rand () % pRectData->cxClient ;
  xRight = rand () % pRectData->cxClient ;
  yTop = rand () % pRectData->cyClient ;
  yBottom = rand () % pRectData->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) ;
  return 0 ;

case WM_PAINT: // Clear the window

  InvalidateRect (hwnd, NULL, TRUE) ;
  hdc = BeginPaint (hwnd, &ps) ;
  EndPaint (hwnd, &ps) ;
  return 0 ;

case WM_MDIACTIVATE: // Set the appropriate menu
  if (lParam == (LPARAM) hwnd)
    SendMessage (hwndClient, WM_MDISETMENU, (WPARAM) hMenuRect, (LPARAM) hMenuRectWindow) ;
  else
    SendMessage (hwndClient, WM_MDISETMENU, (WPARAM) hMenuInit, (LPARAM) hMenuInitWindow) ;
```

```
    DrawMenuBar (hwndFrame) ;
    return 0 ;

  case WM_DESTROY:
    pRectData = (PRECTDATA) GetWindowLong (hwnd, 0) ;
    HeapFree (GetProcessHeap (), 0, pRectData) ;
    KillTimer (hwnd, 1) ;
    return 0 ;
  }
  // Pass unprocessed message to DefMDIChildProc
  return DefMDIChildProc (hwnd, message, wParam, lParam) ;
}
```

MDIDEMO.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
MDIMENUINIT MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "New &Hello", IDM_FILE_NEWHELLO
MENUITEM "New &Rectangle", IDM_FILE_NEWRECT
MENUITEM SEPARATOR
MENUITEM "E&xit", IDM_APP_EXIT
END
END
MDIMENUHELLO MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "New &Hello", IDM_FILE_NEWHELLO
MENUITEM "New &Rectangle", IDM_FILE_NEWRECT
MENUITEM "&Close", IDM_FILE_CLOSE
MENUITEM SEPARATOR
MENUITEM "E&xit", IDM_APP_EXIT
END
POPUP "&Color"
BEGIN
MENUITEM "&Black", IDM_COLOR_BLACK
MENUITEM "&Red", IDM_COLOR_RED
MENUITEM "&Green", IDM_COLOR_GREEN
MENUITEM "B&lue", IDM_COLOR_BLUE
MENUITEM "&White", IDM_COLOR_WHITE
END
POPUP "&Window"
BEGIN
MENUITEM "&Cascade\tShift+F5", IDM_WINDOW_CASCADE
MENUITEM "&Tile\tShift+F4", IDM_WINDOW_TILE
MENUITEM "Arrange &Icons", IDM_WINDOW_ARRANGE
MENUITEM "Close &All", IDM_WINDOW_CLOSEALL
END
END
MDIMENURECT MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "New &Hello", IDM_FILE_NEWHELLO
MENUITEM "New &Rectangle", IDM_FILE_NEWRECT
MENUITEM "&Close", IDM_FILE_CLOSE
MENUITEM SEPARATOR
MENUITEM "E&xit",  IDM_APP_EXIT
END
POPUP "&Window"
BEGIN
```

```
MENUITEM "&Cascade\tShift+F5", IDM_WINDOW_CASCADE
MENUITEM "&Tile\tShift+F4", IDM_WINDOW_TILE
MENUITEM "Arrange &Icons", IDM_WINDOW_ARRANGE
MENUITEM "Close &All", IDM_WINDOW_CLOSEALL
END
END

//////////////////////////////////////////////////////////////////////////
// Accelerator
MDIDEMO ACCELERATORS DISCARDABLE
BEGIN
VK_F4, IDM_WINDOW_TILE, VIRTKEY, SHIFT, NOINVERT
VK_F5, IDM_WINDOW_CASCADE, VIRTKEY, SHIFT, NOINVERT
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by MDIDemo.rc
#define IDM_FILE_NEWHELLO    40001
#define IDM_FILE_NEWRECT    40002
#define IDM_APP_EXIT    40003
#define IDM_FILE_CLOSE    40004
#define IDM_COLOR_BLACK    40005
#define IDM_COLOR_RED    40006
#define IDM_COLOR_GREEN    40007
#define IDM_COLOR_BLUE    40008
#define IDM_COLOR_WHITE    40009
#define IDM_WINDOW_CASCADE    40010
#define IDM_WINDOW_TILE    40011
#define IDM_WINDOW_ARRANGE    40012
#define IDM_WINDOW_CLOSEALL    40013
```

MDIDEMO支持两种型态的非常简单的文件窗口：第一种窗口在它的显示区域中央显示"Hello, World!"，另一种窗口显示一系列随机矩形（在原始码列表和标识符名中，它们分别叫做「Hello」文件和「Rect」文件）。这两类文件窗口的菜单不同，显示"Hello, World!"的文件窗口有一个允许使用者修改文字颜色的菜单。

三个菜单

现在让我们先看看MDIDEMO.RC资源描述文件，它定义了程序所使用的三个菜单模板。

当文件窗口不存在时，程序显示MdiMenuInit菜单，这个菜单只允许使用者建立新文件或退出程序。

MdiMenuHello菜单与显示「Hello, World!」的文件窗口相关联。「File」子菜单允许使用者打开任何一类新文件、关闭活动文件或退出程序。「Color」子菜单允许使用者设定文字颜色。Window子菜单包括以平铺或者重迭的方式安排文件窗口、安排文件图标或关闭所有窗口等选项，这个子菜单也列出了它们建立的所有文件窗口。

MdiMenuRect菜单与随机矩形文件相关联。除了不包含「Color」子菜单外，它与MdiMenuHello菜单一样。

RESOURCE.H表头文件定义所有的菜单标识符。另外，以下三个常数定义在MDIDEMO.C中：

#define INIT_MENU_POS 0

#define HELLO_MENU_POS 2

#define RECT_MENU_POS 1

这些标识符说明每个菜单模板中Windows子菜单的位置。程序需要这些信息来通知客户窗口文

件列表应出现在哪里。当然，MdiMenuInit菜单没有Windows子菜单，所以如前所述，文件列表应附加在第一个子菜单中（位置0）。不过，实际上永远不会在此看到文件列表（在后面讨论此程序时，您可以发现这样做的原因）。

定义在MDIDEMO.C中的IDM_FIRSTCHILD标识符不对应于菜单项，它与出现在Windows子菜单上的文件列表中的第一个文件窗口相关联。这个标识符的值应当大于所有其它菜单ID的值。

程序初始化

在MDIDEMO.C中，WinMain是从注册框架窗口和两个子窗口的窗口类别开始的。窗口消息处理程序是FrameWndProc、HelloWndProc和RectWndProc。一般来说，这些窗口类别应该与不同的图标相关联。为了简单起见，我们将标准IDI_APPLICATION图标用于框架窗口和子窗口。

注意，我们已经定义了框架窗口类别的WNDCLASS结构的hbrBackground字段为COLOR_APPWORKSPACE系统颜色。由于框架窗口的显示区域被客户窗口所覆盖并且客户窗口具有这种颜色，所以上面的定义不是绝对必要的。但是，在最初显示框架窗口时，使用这种颜色似乎要好一些。

这三种窗口类别中的lpszMenuName字段都设定为NULL。对「Hello」和「Rect」子窗口类别来说，这是很自然的。对于框架窗口类别，我在建立框架窗口时在CreateWindow函数中给出菜单句柄。

「Hello」和「Rect」子窗口的窗口类别将WNDCLASS结构中的cbWndExtra字段设为非零值来为每个窗口配置额外空间，这个空间将用于储存指向一个内存块的指针（HELLODATA和RECTDATA结构的大小定义在MDIDEMO.C的开始处），这个内存块被用于储存每个文件窗口特有的信息。

下一步，WinMain用LoadMenu载入三个菜单，并把它们的句柄储存到整体变量中。呼叫三次GetSubMenu函数可获得Windows子菜单（文件列表将加在它上面）的句柄，同样也把它们储存到整体变量中。LoadAccelerators函数加载加速键表。

在WinMain中呼叫CreateWindow建立框架窗口。在FrameWndProc中WM_CREATE消息处理期间，框架窗口建立客户窗口。这项操作涉及到再一次呼叫函数CreateWindow。窗口类别被设定为MDICLIENT，它是预先注册的MDI显示区域窗口类别。在Windows中许多对MDI的支持被放入了MDICLIENT窗口类别中。显示区域窗口消息处理程序作为框架窗口和不同文件窗口的中间层。当呼叫CreateWindow建立显示区域窗口时，最后一个参数必须被设定为指向CLIENTCREATESTRUCT型态结构的指针。这个结构有两个字段：

hWindowMenu是要加入文件列表的子菜单的句柄。在MDIDEMO中，它是hMenuInitWindow，是在WinMain期间获得的。后面将看到如何修改此菜单。

idFirstChild是与文件列表中的第一个文件窗口相关联的菜单ID。它就是IDM_FIRSTCHILD.

再让我们回过头来看看WinMain。MDIDEMO显示新建立的框架窗口并进入消息循环。消息循环与正常的循环稍有不同：在呼叫GetMessage从消息队列中获得消息之后，MDI程序把这个消息传送给了TranslateMDISysAccel（以及TranslateAccelerator，如果像MDIDEMO程序一样，程序本身也有菜单快捷键的话）。

TranslateMDISysAccel函数把可能对应特定MDI快捷键（例如Ctrl-F6）的按键转换成WM_SYSCOMMAND消息。如果TranslateMDISysAccel或TranslateAccelerator都传回TRUE（表示某个消息已被这些函数之一转换），就不能呼叫TranslateMessage和DispatchMessage。

注意传递到TranslateMDISysAccel和TranslateAccelerator的两个窗口句柄：hwndClient

和hwndFrame。WinMain函数通过用GW_CHILD参数呼叫GetWindow获得hwndClient窗口句柄。

建立子窗口

FrameWndProc的大部分工作是用于处理通知菜单选择的WM_COMMAND消息。与平时一样，FrameWndProc中wParam参数的低字组包含着菜单ID。

在菜单ID的值为IDM_FILE_NEWHELLO和IDM_FILE_NEWRECT的情况下，FrameWndProc必须建立一个新的文件窗口。这涉及到初始化MDICREATESTRUCT结构中的字段（大多数字段对应于CreateWindow的参数），并将消息WM_MDICREATE发送给客户窗口，消息的lParam参数设定为指向这个结构的指针。然后由客户窗口建立子文件窗口。（也可以使用CreateMDIWindow函数。）

MDICREATESTRUCT结构中的szTitle字段一般是对应于文件的文件名称。样式字段设定为窗口样式WS_HSCROLL、WS_VSCROLL或这两者的组合，以便在文件窗口中包括滚动条。样式字段也可以包括WS_MINIMIZE或WS_MAXIMIZE，以便在最初时以最小化或最大化状态显示文件窗口。

MDICREATESTRUCT结构的lParam字段为框架窗口和子窗口共享某些变量提供了一种方法。这个字段可以设定为含有一个结构的内存块的内存句柄。在子文件窗口的WM_CREATE消息处理期间，lParam是一个指向CREATESTRUCT结构的指针，这个结构的lpCreateParams字段是一个指向用于建立窗口的MDICREATESTRUCT结构的指针。

客户窗口一旦接收到WM_MDICREATE消息就建立一个子文件窗口，并把窗口标题加到用于建立客户窗口的MDICLIENTSTRUCT结构中所指定的子菜单的底部。当MDIDEMO程序建立它的第一个文件窗口时，这个子菜单就是「MdiMenuInit」菜单中的「File」子菜单。后面将看到这个文件列表将如何移到「MdiMenuHello」和「MdiMenuRect」菜单的「Windows」子菜单中。

菜单上可以列出9个文件，每个文件的前面是带有底线的数字1至9。如果建立的文件窗口多于9个，则这个清单后跟有「More Windows」菜单项。该项启动带有清单方块的对话框，清单方块列出了所有文件。这种文件列表的维护是Windows MDI支持的最好特性之一。

关于框架窗口的消息处理

在把注意力转移到子文件窗口之前，我们先继续讨论FrameWndProc的消息处理。

当从「File」菜单中选择「Close」时，MDIDEMO关闭活动子窗口。它通过把WM_MDIGETACTIVE消息发送给客户窗口，而获得活动子窗口的句柄。如果子窗口以WM_QUERYENDSESSION消息来响应，那么MDIDEMO将WM_MDIDESTROY消息发送给客户窗口，从而关闭子窗口。

处理「File」菜单中的「Exit」选项只需要框架窗口消息处理程序给自己发送一个WM_CLOSE消息。

处理Window子菜单的「Tile」、「Cascade」和「Arrange」选项是极容易的，只需把消息WM_MDITILE、WM_MDICASCADE和WM_MDIICONARRANGE发送给客户窗口。

处理「Close All」选项要稍微复杂一些。FrameWndProc呼叫EnumChildWindows，传送一个引用CloseEnumProc函数的指标。此函数把WM_MDIRESTORE消息发送给每个子窗口，紧跟着发出WM_QUERYENDSESSION和WM_MDIDESTROY。对图标平铺窗口来说并不就此结束，用GW_OWNER参数呼叫GetWindow时，传回的非NULL值可以显示出这一点。

FrameWndProc没有处理任何由「Color」菜单中对颜色的选择所导致的WM_COMMAND消息，这些消息应该由文件窗口负责处理。因此，FrameWndProc把所有未经处理的WM_COMMAND消息发送到活动子窗口，以便子窗口可以处理那些与它们有关的消息。

框架窗口消息处理程序不予处理的所有消息都要送到DefFrameProc，它在框架窗口消息处理程序中取代了DefWindowProc。即使框架窗口消息处理程序拦截了WM_MENUCHAR、WM_SETFOCUS或WM_SIZE消息，这些消息也要被送到DefFrameProc中。

所有未经处理的WM_COMMAND消息也必须送给DefFrameProc。具体地说，FrameWndProc并不处理任何WM_COMMAND消息，即使这些消息是使用者在Windows子菜单的文件列表中选择文件时产生的（这些选项的wParam值是以IDM_FIRSTCHILD开始的）。这些消息要传送到DefFrameProc，并在那里进行处理。

注意框架窗口并不需要维护它所建立的所有文件窗口的窗口句柄清单。如果需要这些窗口句柄（如处理菜单上的「Close All」选项时），可以使用EnumChildWindows得到它们。

子文件窗口

现在看一下HelloWndProc，它是用于显示「Hello, World!」的子文件窗口的窗口消息处理程序。

与用于多个窗口的窗口类别一样，所有在窗口消息处理程序（或从该窗口消息处理程序中呼叫的任何函数）中定义的静态变量由依据该窗口类别建立的所有窗口共享。

只有对于每个唯一于窗口的数据才必须采用非静态变量的方法来储存。这样的技术要用到窗口属性。另一种方法（我使用的方法）是使用预留的内存空间；可以在注册窗口类别时将WNDCLASS结构的cbWndExtra字段设定为非零值以便预留这部分内存空间。

MDIDEMO程序使用这个内存空间来储存一个指标，这个指标指向一块与HELLODATA结构大小相同的内存块。在处理WM_CREATE消息时，HelloWndProc配置这块内存，初始化它的两个字段（它们用于指定目前选中的菜单项和文字颜色），并用SetWindowLong将内存指针储存到预留的空间中。

当处理改变文字颜色的WM_COMMAND消息（回忆一下，这些消息来自框架窗口消息处理程序）时，HelloWndProc使用GetWindowLong获得包含HELLODATA结构的内存块的指针。利用这个结构，HelloWndProc清除原来对菜单项的选择，设定所选菜单项为选中状态，并储存新的颜色。

当窗口变成活动窗口或不活动的时候，文件窗口消息处理程序都会收到WM_MDIACTIVATE消息（lParam的值是否为这个窗口的句柄表示了该窗口是活动的还是不活动的）。您也许还能记起MDIDEMO程序中有三个不同的菜单：当无文件时为MdiMenuInit；当「Hello」文件窗口是活动窗口时为MdiMenuHello；当「Rect」文件窗口为活动窗口时为MdiMenuRect。

WM_MDIACTIVATE消息为文件窗口提供了一个修改菜单的机会。如果lParam中含有本窗口的句柄（意味着本窗口将变成活动的），那么HelloWndProc就将菜单改为MdiMenuHello。如果lParam中包含另一个窗口的句柄，那么HelloWndProc将菜单改为MdiMenuInit。

HelloWndProc经由把WM_MDISETMENU消息发送给客户窗口来修改菜单，客户窗口透过从目前菜单上删除文件列表并把它添加到一个新的菜单上来处理这个消息。这就是文件列表从MdiMenuInit菜单（它在建立第一个文件时有效）传送到MdiMenuHello菜单中的方法。在MDI应用程序中不要用SetMenu函数改变菜单。

另一项工作涉及到「Color」子菜单上的选中旗标。像这样的程序选项对每个文件来说都是不同的，例如，可以在一个窗口中设定黑色文字，在另一个窗口中设定红色文字。菜单选中旗标应能反映出活动窗口中选择的选项。由于这种原因，HelloWndProc在窗口变成非活动窗口时清除选中菜单项的选中旗标，而当窗口变成活动窗口时设定适当菜单项的选中旗标。

WM_MDIACTIVATE的wParam和lParam值分别是失去活动和被启动窗口的句柄。窗口消息处理程序得到的第一个WM_MDIACTIVATE消息的lParam参数被设定为目前窗口的句柄。而当窗口被消除时，窗口消息处理程序得到的最后一个消息的lParam参数被设定为另一个值。当使用者从一个文件切换到另一个文件时，前一个文件窗口收到一个WM_MDIACTIVATE消息，其lParam参数为第一个窗口的句柄（此时，窗口消息处理程序将菜单设定为MdiMenuInit）；后一个文件窗口收到一个WM_MDIACTIVATE消息，其lParam参数是第二个窗口的句柄（此时，窗口消息处理程序将菜单设定为MdiMenuHello或MdiMenuRect中适当的那个）。如果所有的窗口都关闭了，剩下的菜单就是MdiMenuInit。

当使用者从菜单中选择「Close」或「Close All」时，FrameWndProc给子窗口发送一个WM_QUERYENDSESSION消息。HelloWndProc将显示一个消息框并询问使用者是否要关闭窗口，以此来处理WM_QUERYENDSESSION和WM_CLOSE消息（在真实的应用程序中，消息框会询问是否需要储存文件）。如果使用者表示不能关闭窗口，那么窗口消息处理程序传回0。

在WM_DESTROY消息处理期间，HelloWndProc释放在WM_CREATE期间配置的内存块。

所有未经处理的消息必须传送到用于内定处理的DefMDIChildProc（不是DefWindowProc）。不论子窗口消息处理程序是否使用了这些消息，有几个消息必须被传送给DefMDIChildProc。这些消息是：WM_CHILDACTIVATE、WM_GETMINMAXINFO、WM_MENUCHAR、WM_MOVE、WM_SETFOCUS、WM_SIZE和WM_SYSCOMMAND。

RectWndProc与HelloWndProc非常相似，但是它比HelloWndProc要简单一些（不含菜单选项并且无需使用者确认是否关闭窗口），所以这里不对它进行讨论了。但应该注意到，在处理WM_SIZE之后RectWndProc使用了「break」叙述，所以WM_SIZE消息被传给DefMDIChildProc。

结束处理

在WinMain中，MDIDEMO使用LoadMenu加载资源描述文件中定义的三个菜单。一般说来，当菜单所在的窗口被清除时，Windows也要清除与之关联的菜单。对于Init菜单，应该清除那些没有联系到窗口的菜单。由于这个原因，MDIDEMO在WinMain的末尾呼叫了两次DestroyMenu来清除「Hello」和「Rect」菜单。

第二十章 多任务和多线程

多任务是一个操作系统可以同时执行多个程序的能力。基本上，操作系统使用一个硬件时钟为同时执行的每个程序配置「时间片段」。如果时间片段够小，并且机器也没有由于太多的程序而超出负荷时，那么在使用者看来，所有的这些程序似乎在同时执行着。

多任务并不是什么新的东西。在大型计算机上，多任务是必然的。这些大型主机通常有几十甚至几百个终端机和它连结，而每个终端机使用者都应该感觉到他或者她独占了整个计算机。另外，大型主机的操作系统通常允许使用者「提交工作到背景」，这些背景作业可以在使用者进行其它工作时，由机器执行完成。

个人计算机上的多任务花了更长的时间才普及化。但是现在PC多任务也被认为是很正常的了。我马上就会讨论到，Microsoft Windows的16位版本支持有限度的多任务，Windows的32位版本支持真正的多任务，而且，还多了一种额外的优点，多线程。

多线程是在一个程序内部实作多任务的能力。程序可以把它自己分隔为各自独立的「线程」，这些线程似乎也同时在执行着。这一概念初看起来似乎没有什么用处，但是它可以让程序使用多执行绪在背景执行冗长作业，从而让使用者不必长时间地无法使用其计算机进行其它工作（有时这也许不是人们所希望的，不过这种时候去冲冲凉或者到冰箱去看看总是很不错的）！但是，即使在计算机繁忙的时候，使用者也应该能够使用它。

多任务的各种模式

在PC的早期，有人曾经提倡未来应该朝多任务的方向前进，但是大多数的人还是很迷惑：在一个单使用者的个人计算机上，多任务有什么用呢？好了，最后事实表示即使是不知道这一概念的使用者也都需要多任务的。

DOS下的多任务

在最初PC上的Intel 8088微处理器并不是为多任务而设计的。部分原因（我在上一章中讨论过）是内存管理不够强。当启动和结束多个程序时，多任务的操作系统通常需要移动内存块以收集空闲内存。在8088上是不可能透明于应用系统来做到这一点的。

DOS本身对多任务没有太大的帮助，它的设计目的是尽可能小巧，并且与独立于应用程序之外，因此，除了加载程序以及对程序提供文件系统的存取功能，它几乎没有提供任何支持。

不过，有创意的程序写作者仍然在DOS的早期就找到了一种克服这些缺陷的方法，大多数是使用常驻（TSR：terminate-and-stay-resident）程序。有些TSR，比如背景打印队列程序等，透过拦截硬件时钟中断来执行真正的背景处理。其它的TSR，诸如SideKick等弹出式工具，可以执行某种型态的工作切换 – 暂停目前的应用程序，执行弹出式工具。DOS也逐渐有所增强以便提供对TSR的支持。

一些软件厂商试图在DOS之上架构出工作切换或者多任务的外壳程序（shell）（诸如Quarterdeck的DesqView），但是在这些环境中，仅有其中一个占据了大部分市场，当然，这就是Windows。

非优先权式的多任务

当Microsoft在1985年发表Windows 1.0时，它是最成熟的解决方案，目的是突破DOS的局限。Windows在实际模式下执行。但是即使这样，它已可以在物理内存中移动内存块。这是多任务的前提，虽然移动的方法尚未完全透明于应用程序，但是几乎可以忍受了。

在图形窗口环境中，多任务比在一种命令列单使用者操作系统中显得更有意义。例如，在传统的命令列UNIX中，可以在命令列之外执行程序，让它们在背景执行。然而，程序的所有显示输出必须被重新转向到一个文件中，否则输出将和使用者正在做的事情混在一起。

窗口环境允许多个程序在相同屏幕上一起执行，前后切换非常容易，并且还可以快速地将数据从一个程序移动到另一个程序中。例如，将绘图程序中建立的图片嵌入由文书处理程序编辑的文本文件中。在Windows中，以多种方式支持数据转移，首先是使用剪贴簿，后来又使用动态数据交换（DDE），而现在则是透过对象连结和嵌入（OLE）。

不过，早期Windows的多任务实作还不是多使用者操作系统中传统的优先权式的分时多任务。这些操作系统使用系统时钟周期性地中断一个工作并开始另一个工作。Windows的这些16位版本支持一种被称为「非优先权式的多任务」，由于Windows消息驱动的架构而使这种型态的多任务成为可能。通常情况下，一个Windows程序将在内存中睡眠，直到它收到一个消息为止。这些消息通常是使用者的键盘或鼠标输入的直接或间接结果。当处理完消息之后，程序将控制权返回给Windows。

Windows的16位版本不会绝对地依据一个timer tick将控制权从一个Windows程序切换到另一个，任何的工作切换都发生在当程序完成对消息的处理后将控制权返回给Windows时。这种非优先权式的多任务也被称为「合作式的多任务」，因为它要求来自应用程序方面的一些合作。一个Windows程序可以占用整个系统，如果它要花很长一段时间来处理消息的话。

虽然非优先权式的多任务是16位Windows的一般规则，但仍然出现了某些形式的优先权式多任务。Windows使用优先权式多任务来执行DOS程序，而且，为了实作多媒体，还允许动态链接库接收硬件时钟中断。

16位Windows包括几个功能特性来帮助程序写作者解决（或者，至少可以说是对付）非优先权式多任务中的局限，最显著的当然是时钟式鼠标光标。当然，这并非一种解决方案，而仅仅是让使用者知道一个程序正在忙于处理一件冗长作业，因而让使用者在一段时间内无法使用系统。另一种解决方案是Windows定时器，它允许程序周期性地接收消息并完成一些工作。定时器通常用于时钟应用和动画。

针对非优先权式多任务的另一种解决方案是PeekMessage函数呼叫，我们曾在第五章中的RANDRECT程序里看到过。一个程序通常使用GetMessage呼叫从它的消息队列中找寻下一个消息，不过，如果在消息队列中没有消息，那么GetMessage不会传回，一直到出现一个消息为止。而另一方面，PeekMessage将控制权传回程序，即使没有等待的消息。这样，一个程序可以执行一个冗长作业，并在程序代码中混入PeekMessage呼叫。只要没有这个程序或其它任何程序的消息要处理，那么这个冗长作业将继续执行。

Presentation Manager和序列化的消息队列

Microsoft在一种半DOS/半Windows的环境下实作多任务的第一个尝试（和IBM合作）是OS/2和Presentation Manager（缩写成PM）。虽然OS/2明确地支持优先权式多任务，但是这种多任务方式似乎并未在Presentation Manager中得以落实。问题在于PM序列化来自键盘和鼠标的使用者输入消息。这意味着，在前一个使用者输入消息被完全处理以前，PM不会将一个键盘或者鼠标消息传送给程序。

尽管键盘和鼠标消息只是一个PM（或者Windows）程序可以接收的许多消息中的几个，大多数的其它消息都是键盘或者鼠标事件的结果。例如，菜单命令消息是使用者使用键盘或者鼠标进行菜单选择的结果。在处理菜单命令消息时，键盘或者鼠标消息并未完全被处理。

序列化消息队列的主要原因是允许使用者的预先「键入」键盘按键和预先「按入」鼠标按钮。如果一个键盘或者鼠标消息导致输入焦点从一个窗口切换到另一个窗口，那么接下来的键盘消息应该进入拥有新的输入焦点的窗口中去。因此，系统不知道将下一个使用者输入消息发送到何处，直到前一个消息被处理完为止。

目前的共识是不应该让一个应用系统有可能占用整个系统，而这需要非序列化的消息队列，32位版本的Windows支持这种消息队列。如果一个程序正在忙着处理一项冗长作业，那么您可以将输入焦点切换到另一个程序中。

多线程解决方案

我讨论OS/2的Presentation Manager，只是因为它是第一个为早期的Windows程序写作者（比如我自己）介绍多线程的环境。有趣的是，PM实作多线程的局限为程序写作者提供了应该如何架构多线程程序的必要线索。即使这些限制在32位的Windows中已经大幅减少，但是从更有限的环境中学到的经验仍然是非常有效的。因此，让我们继续讨论下去。

在一个多线程环境中，程序可以将它们自己分隔为同时执行的片段（叫做执行绪）。对执行绪的支持是解决PM中存在的序列化消息队列的最好方法，并且在Windows中线程有更实际的意义。

就程序代码来说，一个线程简单地被表示为可能呼叫程序中其它函数的函数。程序从其主线程开始执行，这个主执行绪是在传统的C程序中叫做main的函数，而在Windows中是WinMain。一旦执行起来，程序可以通过在系统呼叫CreateThread中指定初始线程函数的名称来建立新的线程的执行。操作系统在执行绪之间优先权式地切换控件，和它在程序之间切换控制权的方法非常类似。

在OS/2的Presentation Manager中，每个线程可以建立一个消息队列，也可以不建立。如果希望从线程建立窗口，那么一个PM线程必须建立消息队列。否则，如果只是进行许多的数据处理或者图形输出，那么线程不需要建立消息队列。因为无消息队列的程序不处理消息，所以它们将不会当住系统。唯一的限制是一个无消息队列线程无法向一个消息队列线程中的窗口发送消息，或者呼叫任何发送消息的函数（不过，它们可以将消息递送给消息队列线程）。

这样，PM程序写作者学会了如何将它们的程序分隔为一个消息队列线程（在其中建立所有的窗口并处理传送给窗口的消息）和一个或者多个无消息队列线程，在其中执行冗长的背景工作。PM程序写作者还了解到「1/10秒规则」，大体上，程序写作者被告知，一个消息队列线程处理任何消息都不应该超过1/10秒，任何花费更长时间的事情都应该在另一个线程中完成。如果所有的程序写作者都遵循这一规则，那么将没有PM程序会将系统当住超过1/10秒。

多线程架构

我已经说过PM的限制让程序写作者理解如何在图形环境中执行的程序里头使用多个执行绪提供了必要的线索。因此在这里我将为您的程序建议一种架构：您的主执行绪建立您程序所需要的所有窗口，并在其中包含所有的窗口消息处理程序，以便处理这些窗口的所有消息；所有其它执行绪只进行一些背景处理，除了和主执行绪通讯，它们不和使用者进行交流。

可以把这种架构想象成：主线程处理使用者输入（和其它消息），并建立程序中的其它线程，这些附加的线程完成与使用者无关的工作。

换句话说，您程序的主线程是一个老板，而您的其它线程是老板的职员。老板将大的工作丢给职员处理，而他自己保持和外界的联系。因为那些线程仅仅是职员，所以其它线程不会举行它们自

己的记者招待会。它们会认真地完成自己的工作，将结果报告给老板，并等待他们的下一个任务。

一个程序中的线程是同一程序的不同部分，因此他们共享程序的资源，如内存和打开的文件。因为线程共享程序的内存，所以他们还共享静态变量。然而，每个线程都有他们自己的堆栈，因此动态变量对每个线程是唯一的。每个线程还有各自的处理器状态（和数学协处理器状态），这个状态在进行线程切换期间被储存和恢复。

线程间的「争吵」

正确地设计、写作和测试一个复杂的多线程应用程序显然是Windows程序写作者可能遇到的最困难的工作之一。因为优先权式多任务系统可以在任何时刻中断一个线程，并将控制权切换到另一个线程中，在两个线程之间可能有无法预料的随机交互作用的情况。

多线程程序中的一个常见的错误被称为「竞争状态（race condition）」，这发生在程序写作者假设一个线程在另一个线程需要某资料之前已经完成了某些处理（如准备数据）的时候。为了帮助协调线程的活动，操作系统要求各种形式的同步。一种是同步信号（semaphore），它允许程序写作者在程序代码中的某一点阻止一个线程的执行，直到另一个执行绪发信号让它继续为止。类似于同步信号的是「临界区域（critical section）」，它是程序代码中不可中断的部分。

但是同步信号还可能产生称为「死锁（deadlock）」的常见线程错误，这发生在两个线程互相阻止了另一个的执行，而继续执行的唯一办法又是它们继续向前执行。

幸运的是，32位程序比16位程序更能抵抗线程所涉及的某些问题。例如，假定一个线程执行下面的简单叙述：

lCount++ ;

其中lCount是由其它线程使用的一个32位的long型态变量，C中的这个叙述被编译为两条机械码指令，第一条将变量的低16位加1，而第二条指令将任何可能的进位加到高16位上。假定操作系统在这两个机械码指令之间中断了线程。如果lCount在第一条机械码指令之前是0x0000FFFF，那么lCount在线程被中断时为0，而这正是另一个线程将看到的值。只有当线程继续执行时，lCount才会增加到正确的值0x00010000。

这是那些偶尔会导致操作问题的错误之一。在16位程序中，解决此问题正确的方法是将叙述包含在一个临界区域中，在这期间线程不会被中断。然而，在一个32位程序中，该叙述是正确的，因为它被编译为一条机械码指令。

Windows的好处

32位Windows版本（包括Windows NT和Windows 98）有一个非序列化的消息队列。这种实作似乎非常好：如果一个程序正在花费一段长时间处理一个消息，那么鼠标位于该程序的窗口上时，鼠标光标将呈现为一个时钟，但是当将鼠标移到另一个程序的窗口上时，鼠标光标将变为正常的箭头形状。只需按一下就可以将另一个窗口提到前面来。

然而，使用者仍然不能使用正在处理大量工作的那个程序，因为那些工作会阻止程序接收其它消息，这不是我们所希望的。一个程序应该总是能随时处理消息的，所以这时就需要使用从属线程了。

在Windows NT和Windows 98中，没有消息队列线程和无消息队列线程的区别，每个线程在建立时都会有它自己的消息队列，从而减少了PM程序中关于线程的一些不便规定（然而，在大多数情况下，您仍然想通过一条专门处理消息的线程中的消息程序处理输入，而将冗长作业交给那些不包含窗口的线程处理，这种结构几乎总是最容易理解的，我们将看到这一点）。

还有更好的事情：Windows NT和Windows 98中有个函数允许线程杀死同一程序中的另一个

线程。当您开始编写多线程程序代码时，您将会发现这种功能在有时是很方便的。OS/2的早期版本没有「杀死线程」的函数。

最后的好消息（至少对这里的话题是好消息）是Windows NT和Windows 98实作了一些被称为「线程区域储存空间（TLS：thread local storage）」的功能。为了了解这一点，回顾一下我在前面提到过的，静态变量（对一个函数来说，既是整体又是区域变量）在线程之间是被共享的，因为它们位于程序的数据储存空间中。动态变量（对一个函数来说总是区域变量）对每一个线程则是唯一的，因为它们占据堆栈上的空间，而每个线程都有它自己的堆栈。

有时让两个或多个线程使用相同的函数，而让这些线程使用唯一于线程的静态变量，那会带来很大便利。这就是线程区域储存空间，其中涉及一些Windows函数呼叫，但是Microsoft还为C编译器进行扩展，使线程区域储存空间的使用更透明于程序写作者。

新改良过的！支持多线程了！

既然已经介绍了线程的现状，让我们来展望一下线程的未来。有时，有人会出现一种使用操作系统所提供的每一种功能特性的冲动。最坏的情况是，当您的老板走到您的桌前并说：「我听说这种新功能非常炫，让我们在自己的程序中用一些这种新功能吧。」然后您将花费一个星期的时间，试图去了解您的应用程序如何从这种新功能获益。

应该注意的是，在并不需要多线程的应用系统中加入多线程是没有任何意义的。如果您的程序显示沙漏光标的时间太长，或者如果它使用PeekMessage呼叫来避免沙漏光标的出现，那么请重新规划您的程序架构，使用多线程可能会是一个好主意。其它情形，您是在为难您自己，并可能会在程序代码中产生新的错误。

在某些情况下，沙漏光标的出现可能是完全适当的。我在前面提到过「1/10秒规则」，而将一个大文件加载内存可能会花费多于1/10秒的时间，这是否意味着文件加载例程应该在分离的线程中实作呢？没有必要。当使用者命令一个程序打开文件时，他或者她通常想立即完成该操作。将文件加载例程放在分离的线程中只会增加额外的负担。即使您想向您的朋友夸耀您在编写多线程程序，也完全不值得这样做！

Windows 的多线程处理

建立新的线程的API函数是CreateThread，它的语法如下：

```
hThread = CreateThread (&security_attributes, dwStackSize, ThreadProc,
                                    pParam, dwFlags, &idThread) ;
```

第一个参数是指向SECURITY_ATTRIBUTES型态的结构的指针。在Windows 98中忽略该参数。在Windows NT中，它被设为NULL。第二个参数是用于新线程的初始堆栈大小，默认值为0。在任何情况下，Windows根据需要动态延长堆栈的大小。

CreateThread的第三个参数是指向线程函数的指标。函数名称没有限制，但是必须以下列形式声明：

```
DWORD WINAPI ThreadProc (PVOID pParam) ;
```

CreateThread的第四个参数为传递给ThreadProc的参数。这样主线程和从属线程就可以共享数据。

CreateThread 的第五个参数通常为 0，但当建立的线程不马上执行时为旗标 CREATE_SUSPENDED。线程将暂停直到呼叫ResumeThread来恢复线程的执行为止。第六个参数是一个指标，指向接受执行绪ID值的变量。

大多数Windows程序写作者喜欢用在PROCESS.H表头文件中声明的C执行时期链接库函数 _beginthread。它的语法如下：

hThread = _beginthread (ThreadProc, uiStackSize, pParam) ;

它更简单，对于大多数应用程序很完美，这个线程函数的语法为：

void __cdecl ThreadProc (void * pParam) ;

再论随机矩形

程序20-1 RNDRCTMT是第五章里的RANDRECT程序的多线程版本，您将回忆起RANDRECT使用的是PeekMessage循环来显示一系列的随机矩形。

程序20-1 RNDRCTMT

RNDRCTMT.C

```
/*-------------------------------------------------------------------------
RNDRCTMT.C -- Displays Random Rectangles
(c) Charles Petzold, 1998
-------------------------------------------------------------------------*/
#include <windows.h>
#include <process.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
HWND hwnd ;
int cxClient, cyClient ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("RndRctMT") ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow ( szAppName, TEXT ("Random Rectangles"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
```

```
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

VOID Thread (PVOID pvoid)
{
  HBRUSH hBrush ;
  HDC hdc ;
  int xLeft, xRight, yTop, yBottom, iRed, iGreen, iBlue ;

  while (TRUE)
  {
    if (cxClient != 0 || cyClient != 0)
    {
      xLeft = rand () % cxClient ;
      xRight = rand () % cxClient ;
      yTop = rand () % cyClient ;
      yBottom = rand () % cyClient ;
      iRed = rand () & 255 ;
      iGreen = rand () & 255 ;
      iBlue = rand () & 255 ;

      hdc = GetDC (hwnd) ;
      hBrush = CreateSolidBrush (RGB (iRed, iGreen, iBlue)) ;
      SelectObject (hdc, hBrush) ;

      Rectangle (hdc,min (xLeft, xRight), min (yTop, yBottom),
        max (xLeft, xRight), max (yTop, yBottom)) ;

      ReleaseDC (hwnd, hdc) ;
      DeleteObject (hBrush) ;
    }
  }
}
LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_CREATE:
    _beginthread (Thread, 0, NULL) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在建立多线程的Windows程序时，需要在「Project Settings」对话框中做一些修改。选择「C/C++」页面标签，然后在「Category」下拉式清单方块中选择「Code Generation」。在「Use Run-Time Library」下拉式清单方块中，可以看到用于「Release」设定的「Single-Threaded」和用于Debug设定的「Debug Single-Threaded」。将这些分别改为「Multithreaded」和「Debug Multithreaded」。这将把编译器旗标改为/MT，它是编译器在编译多线程的应用程序所需要的。具体地说，编译器将在.OBJ文件中插入LIBCMT.LIB文件名，而不是LIBC.LIB。连结程序使用这个名称与执行期链接库函数连结。

LIBC.LIB和LIBCMT.LIB文件包含C语言链接库函数，有些C语言链接库函数包含静态数据。例如，

由于strtok函数可能被连续地多次呼叫，所以它在静态内存中储存了一个指标。在多线程程序中，每个线程必须在strtok函数中有它自己的静态指针。因此，这个函数的多线程版本稍微不同于单线程的strtok函数。

同时请注意，我在RNDRCTMT.C中包含了表头文件PROCESS.H，这个文件定义一个名为_beginthread的函数，它启动一个新的线程。只有定义了_MT标识符，才会声明这个函数，这是/MT旗标的另一个结果。

在RNDRCTMT.C的WinMain函数中，由CreateWindow传回的hwnd值被储存在一个整体变量中，因此cxClient和cyClient值也可以由窗口消息处理程序的WM_SIZE消息获得。

窗口消息处理程序以最容易的方法呼叫_beginthread – 简单地以线程函数的地址（称为Thread）作为第一个参数，其它参数使用0，线程函数传回VOID并有一个参数，该参数是一个指向VOID的指标。在RNDRCTMT中的Thread函数不使用这个参数。

在呼叫了_beginthread函数之后，线程函数（以及该线程函数可能呼叫的其它任何函数）中的程序代码和程序中的其它程序代码同时执行。两个或者多个执行绪使用一个程序中的同一函数，在这种情况下，动态区域变量（储存在堆栈上）对每个执行绪是唯一的。对程序中的所有执行绪来说，所有的静态变量都是一样的。这就是窗口消息处理程序设定整体的cxClient和cyClient变量并由Thread函数使用的方式。

有时您需要唯一于各个线程的持续储存性数据。通常，这种数据是静态变量，但在Windows 98中，您可以使用「线程区域储存空间」，我将在本章后面进行讨论。

程序设计竞赛的问题

1986年10月3日，Microsoft举行了为期一天，针对计算机杂志出版社的技术编辑和作者的简短的记者招待会，来讨论他们当时的一组语言产品，包括他们的第一个交谈式开发环境，QuickBASIC 2.0。当时，Windows 1.0出现还不到一年，但是没有人知道我们什么时候能得到与该环境类似的东西（这花了好几年）。这一事件与众不同的部分原因是由于Microsoft的公关人员所举办的「Storm the Gates」程序设计竞赛。Bill Gates使用QuickBASIC 2.0，而计算机出版社的人员可以使用他们选择的任何语言产品。

竞赛的问题是从公众提出的题目中挑选出来的（挑选那些需要写大约半小时程序来解决的问题），问题如下：

建立一个包含四个窗口的多任务仿真程序。第一个窗口必须显示一系列的递增数，第二个必须显示一系列的递增质数，而第三个必须显示Fibonacci数列（Fibonacci数列以数字0和1开始，后头每一个数都是其前两个数的和 – 即0、1、1、2、3、5、8等等）。这三个窗口应该在数字达到窗口底部时或者进行滚动，或者自行清除窗口内容。第四个窗口必须显示任意半径的圆，而程序必须在按下一个Escape键时终止。

当然，在1986年10月，在DOS下执行的这样一个程序最多只能是模拟多任务而已，而且没有一个竞赛者具有足够的勇气 – 并且其中大多数也没有足够的知识 – 来为Windows编写这个程序。再者，如果真要这么做，当然不会只花半小时了！

参加这次竞赛的大多数人编写了一个程序来将屏幕分为四个区域，程序中包含一个循环，依次更新每个窗口，然后检查是否按下了Escape键。如同DOS环境下的传统习惯，程序占用了百分之百的CPU处理时间。

如果在Windows 1.0中写程序，那么结果将是类似程序20-2 MULTI1的结果。我说「类似」，是因为我编写的程序是32位的，但程序结构和相当多的程序代码 – 除了变量和函数参数定义以及

Unicode支持 – 都是相同的。

程序20-2 MULTI1

MULTI1.C

```
/*---------------------------------------------------------------------------
MULTI1.C -- Multitasking Demo
(c) Charles Petzold, 1998
---------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int cyChar ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Multi1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }
  hwnd = CreateWindow ( szAppName, TEXT ("Multitasking Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

int CheckBottom (HWND hwnd, int cyClient, int iLine)
{
  if (iLine * cyChar + cyChar > cyClient)
  {
    InvalidateRect (hwnd, NULL, TRUE) ;
    UpdateWindow (hwnd) ;
    iLine = 0 ;
  }
  return iLine ;
}

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

```
// Window 1: Display increasing sequence of numbers
// ------------------------------------------------------------------------
LRESULT APIENTRY WndProc1 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int iNum, iLine, cyClient ;
  HDC hdc ;
  TCHAR szBuffer[16] ;

  switch (message)
  {
  case WM_SIZE:
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_TIMER:
    if (iNum < 0)
      iNum = 0 ;

    iLine = CheckBottom (hwnd, cyClient, iLine) ;
    hdc = GetDC (hwnd) ;

    TextOut (hdc, 0, iLine * cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("%d"), iNum++)) ;

    ReleaseDC (hwnd, hdc) ;
    iLine++ ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// ------------------------------------------------------------------------
// Window 2: Display increasing sequence of prime numbers
// ------------------------------------------------------------------------
LRESULT APIENTRY WndProc2 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int iNum = 1, iLine, cyClient ;
  HDC hdc ;
  int i, iSqrt ;
  TCHAR szBuffer[16] ;

  switch (message)
  {
  case WM_SIZE:
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_TIMER:
    do {
      if (++iNum < 0)
        iNum = 0 ;

      iSqrt = (int) sqrt (iNum) ;

      for (i = 2 ; i <= iSqrt ; i++)
        if (iNum % i == 0)
          break ;
    }
    while (i <= iSqrt) ;

    iLine = CheckBottom (hwnd, cyClient, iLine) ;
    hdc = GetDC (hwnd) ;

    TextOut ( hdc, 0, iLine * cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("%d"), iNum)) ;
    ReleaseDC (hwnd, hdc) ;
    iLine++ ;
    return 0 ;
  }
```

```
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// ------------------------------------------------------------------------
// Window 3: Display increasing sequence of Fibonacci numbers
// ------------------------------------------------------------------------
LRESULT APIENTRY WndProc3 (HWND hwnd, UINT message, WPARAM wParam,LPARAM lParam)
{
  static int iNum = 0, iNext = 1, iLine, cyClient ;
  HDC hdc ;
  int iTemp ;
  TCHAR szBuffer[16] ;

  switch (message)
  {
  case WM_SIZE:
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_TIMER:
    if (iNum < 0)
    {
      iNum = 0 ;
      iNext = 1 ;
    }

    iLine = CheckBottom (hwnd, cyClient, iLine) ;
    hdc = GetDC (hwnd) ;

    TextOut ( hdc, 0, iLine * cyChar, szBuffer,
      wsprintf (szBuffer, "%d", iNum)) ;
    ReleaseDC (hwnd, hdc) ;
    iTemp = iNum ;
    iNum = iNext ;
    iNex += iTemp ;
    iLine++ ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// ------------------------------------------------------------------------
// Window 4: Display circles of random radii
// ------------------------------------------------------------------------
LRESULT APIENTRY WndProc4 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static int cxClient, cyClient ;
  HDC hdc ;
  int iDiameter ;

  switch (message)
  {
  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_TIMER:
    InvalidateRect (hwnd, NULL, TRUE) ;
    UpdateWindow (hwnd) ;

    iDiameter = rand() % (max (1, min (cxClient, cyClient))) ;
    hdc = GetDC (hwnd) ;

    Ellipse (hdc, (cxClient - iDiameter) / 2,
      (cyClient - iDiameter) / 2,
      (cxClient + iDiameter) / 2,
      (cyClient + iDiameter) / 2) ;
```

```
    ReleaseDC (hwnd, hdc) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
// ---------------------------------------------------------------------------
// Main window to create child windows
// ---------------------------------------------------------------------------
LRESULT APIENTRY WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndChild[4] ;
  static TCHAR * szChildClass[] = { TEXT ("Child1"), TEXT ("Child2"),
    TEXT ("Child3"), TEXT ("Child4") } ;
  static WNDPROC ChildProc[] = { WndProc1, WndProc2, WndProc3, WndProc4 } ;
  HINSTANCE hInstance ;
  int i, cxClient, cyClient ;
  WNDCLASS wndclass ;

  switch (message)
  {
  case WM_CREATE:
    hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;

    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = NULL ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;

    for (i = 0 ; i < 4 ; i++)
    {
      wndclass.lpfnWndProc = ChildProc[i] ;
      wndclass.lpszClassName = szChildClass[i] ;

      RegisterClass (&wndclass) ;

      hwndChild[i] = CreateWindow (szChildClass[i], NULL,
        WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE,
        0, 0, 0, 0,
        hwnd, (HMENU) i, hInstance, NULL) ;
    }

    cyChar = HIWORD (GetDialogBaseUnits ()) ;
    SetTimer (hwnd, 1, 10, NULL) ;
    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    for (i = 0 ; i < 4 ; i++)
      MoveWindow (hwndChild[i], (i % 2) * cxClient / 2,
      (i > 1) * cyClient / 2,
      cxClient / 2, cyClient / 2, TRUE) ;
    return 0 ;

  case WM_TIMER:
    for (i = 0 ; i < 4 ; i++)
      SendMessage (hwndChild[i], WM_TIMER, wParam, lParam) ;

    return 0 ;

  case WM_CHAR:
    if (wParam == '\x1B')
      DestroyWindow (hwnd) ;
```

```
    return 0 ;

  case WM_DESTROY:
    KillTimer (hwnd, 1) ;
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在这个程序里实际上没有什么我们没见过的东西。主窗口建立四个子窗口，每个子窗口占据显示区域的一个象限。主窗口还设定一个Windows定时器并发送WM_TIMER消息给四个子窗口中的每一个。

通常一个Windows程序应该保留足够的信息以便在WM_PAINT消息处理期间重建其窗口中的内容。MULTI1没有这么做，既然它绘制和清除窗口的速度如此之快，所以我认为那是不必要的。

WndProc2中的质数产生器的效率并不很高，但是有效。如果一个数除了1和它自身以外没有别的因子，那么这个数就是质数。当然，要检查一个数是否是质数并不要求使用小于被检查数的所有数来除这个数并检查余数，而只需使用所有小于被检查数的平方根的数。平方根计算是发表浮点数的原因，否则，该程序将是完全依据整数的程序。

MULTI1程序没有什么不好的地方。使用Windows定时器是在Windows的早期（和目前）版本中模拟多任务的一种好方法，然而，定时器的使用有时限制了程序的速度。如果程序可以在WM_TIMER消息处理中更新它的所有窗口而还有时间剩余下来的话，那就意味着它并没有充分利用我们的机器资源。

一种可能的解决方案是在单个WM_TIMER消息处理期间进行两次或者更多次的更新，但是到底多少次呢？这不得不依赖于机器的速度，而有很大的变动性。您当然不会想编写一个只能适用于25MHz的386或50MHz的486或100-GHz的Pentium VII上的程序吧。

多线程解决方案

让我们来看一看关于这个程序设计问题的一种多线程解决方案。如程序20-3 MULTI2所示。

程序20-3 MULTI2

MULTI2.C

```
/*-----------------------------------------------------------------------------
MULTI2.C -- Multitasking Demo
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#include <process.h>

typedef struct
{
  HWND hwnd ;
  int cxClient ;
  int cyClient ;
  int cyChar ;
  BOOL bKill ;
}
PARAMS, *PPARAMS ;
LRESULT APIENTRY WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("Multi2") ;
  HWND hwnd ;
  MSG msg ;
```

```
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Multitasking Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

int CheckBottom (HWND hwnd, int cyClient, int cyChar, int iLine)
{
  if (iLine * cyChar + cyChar > cyClient)
  {
    InvalidateRect (hwnd, NULL, TRUE) ;
    UpdateWindow (hwnd) ;
    iLine = 0 ;
  }
  return iLine ;
}

// ---------------------------------------------------------------------------
// Window 1: Display increasing sequence of numbers
// ---------------------------------------------------------------------------

void Thread1 (PVOID pvoid)
{
  HDC hdc ;
  int iNum = 0, iLine = 0 ;
  PPARAMS pparams ;
  TCHAR szBuffer[16] ;

  pparams = (PPARAMS) pvoid ;

  while (!pparams->bKill)
  {
    if (iNum < 0)
      iNum = 0 ;
    iLine = CheckBottom ( pparams->hwnd, pparams->cyClient,
      pparams->cyChar,iLine) ;

    hdc = GetDC (pparams->hwnd) ;
```

```
    TextOut ( hdc, 0, iLine * pparams->cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("%d"), iNum++)) ;

    ReleaseDC (pparams->hwnd, hdc) ;
    iLine++ ;
  }
  _endthread () ;
}

LRESULT APIENTRY WndProc1 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static PARAMS params ;
  switch (message)
  {
  case WM_CREATE:
    params.hwnd = hwnd ;
    params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
    _beginthread (Thread1, 0, &params) ;
    return 0 ;

  case WM_SIZE:
    params.cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_DESTROY:
    params.bKill = TRUE ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// ------------------------------------------------------------------------
// Window 2: Display increasing sequence of prime numbers
// ------------------------------------------------------------------------

void Thread2 (PVOID pvoid)
{
  HDC hdc ;
  int iNum = 1, iLine = 0, i, iSqrt ;
  PPARAMS pparams ;
  TCHAR szBuffer[16] ;

  pparams = (PPARAMS) pvoid ;
  while (!pparams->bKill)
  {
    do
    {
      if (++iNum < 0)
        iNum = 0 ;
      iSqrt = (int) sqrt (iNum) ;
      for (i = 2 ; i <= iSqrt ; i++)
        if (iNum % i == 0)
          break ;
    }
    while (i <= iSqrt) ;
    iLine = CheckBottom ( pparams->hwnd, pparams->cyClient,
      pparams->cyChar,iLine) ;

    hdc = GetDC (pparams->hwnd) ;

    TextOut ( hdc, 0, iLine * pparams->cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("%d"), iNum)) ;

    ReleaseDC (pparams->hwnd, hdc) ;
    iLine++ ;
  }
  _endthread () ;
}
```

```
LRESULT APIENTRY WndProc2 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static PARAMS params ;
  switch (message)
  {
  case WM_CREATE:
    params.hwnd = hwnd ;
    params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
    _beginthread (Thread2, 0, &params) ;
    return 0 ;

  case WM_SIZE:
    params.cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_DESTROY:
    params.bKill = TRUE ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// Window 3: Display increasing sequence of Fibonacci numbers
// ----------------------------------------------------------

void Thread3 (PVOID pvoid)
{
  HDC hdc ;
  int iNum = 0, iNext = 1, iLine = 0, iTemp ;
  PPARAMS pparams ;
  TCHAR szBuffer[16] ;

  pparams = (PPARAMS) pvoid ;
  while (!pparams->bKill)
  {
    if (iNum < 0)
    {
      iNum = 0 ;
      iNext = 1 ;
    }
    iLine = CheckBottom ( pparams->hwnd, pparams->cyClient,
      pparams->cyChar, iLine) ;

    hdc = GetDC (pparams->hwnd) ;

    TextOut (hdc, 0, iLine * pparams->cyChar, szBuffer,
      wsprintf (szBuffer, TEXT ("%d"), iNum)) ;

    ReleaseDC (pparams->hwnd, hdc) ;
    iTemp = iNum ;
    iNum = iNext ;
    iNext += iTemp ;
    iLine++ ;
  }
  _endthread () ;
}

LRESULT APIENTRY WndProc3 (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static PARAMS params ;
  switch (message)
  {
  case WM_CREATE:
    params.hwnd = hwnd ;
    params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
    _beginthread (Thread3, 0, &params) ;
    return 0 ;
```

```
  case WM_SIZE:
    params.cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_DESTROY:
    params.bKill = TRUE ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// ------------------------------------------------------------------------
// Window 4: Display circles of random radii
// ------------------------------------------------------------------------

void Thread4 (PVOID pvoid)
{
  HDC hdc ;
  int iDiameter ;
  PPARAMS pparams ;

  pparams = (PPARAMS) pvoid ;
  while (!pparams->bKill)
  {
    InvalidateRect (pparams->hwnd, NULL, TRUE) ;
    UpdateWindow (pparams->hwnd) ;

    iDiameter = rand() % (max (1,
      min (pparams->cxClient, pparams->cyClient))) ;

    hdc = GetDC (pparams->hwnd) ;

    Ellipse (hdc, (pparams->cxClient - iDiameter) / 2,
      (pparams->cyClient - iDiameter) / 2,
      (pparams->cxClient + iDiameter) / 2,
      (pparams->cyClient + iDiameter) / 2) ;

    ReleaseDC (pparams->hwnd, hdc) ;
  }
  _endthread () ;
}

LRESULT APIENTRY WndProc4 (HWND hwnd, UINT message,WPARAM wParam,LPARAM lParam)
{
  static PARAMS params ;
  switch (message)
  {
  case WM_CREATE:
    params.hwnd = hwnd ;
    params.cyChar = HIWORD (GetDialogBaseUnits ()) ;
    _beginthread (Thread4, 0, &params) ;
    return 0 ;

  case WM_SIZE:
    params.cxClient = LOWORD (lParam) ;
    params.cyClient = HIWORD (lParam) ;
    return 0 ;

  case WM_DESTROY:
    params.bKill = TRUE ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

// ------------------------------------------------------------------------
// Main window to create child windows
// ------------------------------------------------------------------------
```

```
LRESULT APIENTRY WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndChild[4] ;
  static TCHAR * szChildClass[] = { TEXT ("Child1"), TEXT ("Child2"),
    TEXT ("Child3"), TEXT ("Child4") } ;
  static WNDPROC ChildProc[] = { WndProc1, WndProc2, WndProc3, WndProc4 } ;
  HINSTANCE hInstance ;
  int i, cxClient, cyClient ;
  WNDCLASS wndclass ;

  switch (message)
  {
  case WM_CREATE:
    hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;
    wndclass.style = CS_HREDRAW | CS_VREDRAW ;
    wndclass.cbClsExtra = 0 ;
    wndclass.cbWndExtra = 0 ;
    wndclass.hInstance = hInstance ;
    wndclass.hIcon = NULL ;
    wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
    wndclass.lpszMenuName = NULL ;

    for (i = 0 ; i < 4 ; i++)
    {
      wndclass.lpfnWndProc = ChildProc[i] ;
      wndclass.lpszClassName = szChildClass[i] ;

      RegisterClass (&wndclass) ;

      hwndChild[i] = CreateWindow (szChildClass[i], NULL,
        WS_CHILDWINDOW | WS_BORDER | WS_VISIBLE,
        0, 0, 0, 0,
        hwnd, (HMENU) i, hInstance, NULL) ;
    }

    return 0 ;

  case WM_SIZE:
    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    for (i = 0 ; i < 4 ; i++)
      MoveWindow (hwndChild[i], (i % 2) * cxClient / 2,
      (i > 1) * cyClient / 2,
      cxClient / 2, cyClient / 2, TRUE) ;
    return 0 ;

  case WM_CHAR:
    if (wParam == '\x1B')
      DestroyWindow (hwnd) ;

    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

MULTI2.C的WinMain和WndProc函数非常类似于MULTI1.C中的同名函数。WndProc为四个窗口注册了四种窗口类别，建立了这些窗口，并在WM_SIZE消息处理期间缩放这些窗口。WndProc的唯一不同是它不再设定Windows定时器，也不再处理WM_TIMER消息。

MULTI2中较大的改变是每个子窗口消息处理程序透过在WM_CREATE消息处理期间呼叫_beginthread函数来建立另一个线程。总括来说，MULTI2程序有五个同时执行的执行绪，主执行

绪包含主窗口消息处理程序和四个子窗口消息处理程序，其余的四个执行绪使用名为Thread1、Thread2等的函数，这四个线程负责绘制四个窗口。

我在RNDRCTMT程序中给出的多线程程序代码没有使用_beginthread的第三个参数，这个参数允许一个建立另一个线程的线程在32位变量中将信息传递给其它线程。通常，这个变量是一个指针，而且是指向一个结构的指针，这允许原来的线程和新线程共享信息，而不必借助于整体变量。您可以看到，在MULTI2中没有整体变量。

对MULTI2程序，我在程序开头定义了一个名为PARAMS的结构和一个名为PPARAMS的指向结构的指针，这个结构有五个字段 – 窗口句柄、窗口的宽度和高度、字符的高度和名为bKill的布尔变数。最后的结构字段允许建立线程告知被建立线程何时终止。

让我们来看一看WndProc1，这是显示增加数序列的子窗口消息处理程序。窗口消息处理程序变得非常简单，唯一的区域变量是一个PARAMS结构。在WM_CREATE消息处理期间，它设定这个结构的hwnd和cyChar字段，呼叫_beginthread来建立一个使用Thread1函数的新线程，并传递给新线程一个指向该结构的指针。在WM_SIZE消息处理期间，WndProc1设定结构的cyClient字段，而在WM_DESTROY消息处理期间，它将bKill字段设定为TRUE。Thread1函数通过对_endthread的呼叫而告结束。这并不是绝对必要的，因为线程将在退出线程函数之后被清除。不过，要退出一个深陷入复杂的处理程序的线程时，_endthread是很有用的。

Thread1函数完成在窗口上的实际绘图，并且和程序的其它四个线程同时执行。函数接收指向PARAMS结构的一个指针，并进入一个while循环，不断检查bKill是TRUE还是FALSE。如果是FALSE，那么函数必须进行MULTI1.C中的WM_TIMER消息处理期间所作的同样处理 – 格式化数字、取得设备内容句柄并使用TextOut显示数字。

当您在Windows 98中执行MULTI2时，将会看到，窗口更新要比在MULTI1中快得多，这表示程序在更加有效地利用处理器的资源。在MULTI1和MULTI2之间还有另一种区别：通常，当您移动或者缩放一个窗口时，内定窗口消息处理程序进入一种模态循环，而窗口的所有输出都将停止。在MULTI2中，输出将继续。

有问题吗?

似乎MULTI2程序并没有达到它应该有的稳固性。我为什么会这样认为呢？让我们来看一看MULTI2.C中的一些多线程「缺陷」，以WndProc1和Thread1为例。

WndProc1在MULTI2的主线程中执行，而Thread1与它同时执行，Windows 98在这两个线程之间进行切换是不可预测的。假定Thread1正在执行，并且刚好执行了检查PARAMS结构的bKill字段是否为TRUE的程序代码。发现不为TRUE，但是这之后Windows 98将控制权切换到主线程，这时使用者终止了程序，WndProc1收到一个WM_DESTROY消息并将bKill参数设为TRUE。哦，这参数设定得太晚了！操作系统突然切换到Thread1中，而该函数会试图取得一个不存在的窗口的设备内容句柄。

事实证明，这不是一个问题。Windows 98够稳固，以致另一条线程呼叫的图形处理函数只是失败而已，而不会引起任何问题。

正确的多线程程序写作技术涉及线程同步的使用（尤其是临界区域的使用），我将马上加以详细地讨论。大体上，临界区域通过对EnterCriticalSection和LeaveCriticalSection的呼叫而加以界定。如果一个线程进入一个临界区域，那么另一个线程将无法再进入这个临界区域。后一个线程被阻档在对EnterCriticalSection的呼叫上，直到第一个线程呼叫LeaveCriticalSection时为止。

在MULTI2中的另一个可能存在的问题是，当另外一个线程显示其输出时，主线程可能会收到一个WM_ERASEBKGND或WM_PAINT消息。这里，使用临界区域有助于避免当两个程序试图在同一

个窗口上绘图时可能导致的任何问题。但是，经验显示，Windows 98很恰当地序列化了对图形绘制函数的存取。亦即，当另一个线程正在绘图的时候，一个线程不能在同一个窗口上绘图。

Windows 98文件提醒说，有一种未进行图形函数序列化的情形，这就是GDI对象（如画笔、画刷、字体、位图、区域和调色盘等）的使用。有可能发生一个线程清除了一个对象，而另一个线程仍然在使用它的情况。解决这个问题的方法要求使用临界区域，或者最好不要在线程之间共享GDI对象。

Sleep的好处

我曾经提到，我认为对一个多线程程序来说，最好的架构是主线程建立程序中的所有窗口，以及所有的窗口消息处理程序，并处理所有的窗口消息。其它线程完成背景工作或者冗长作业。

不过，假设您想在另一个线程中做动画。通常，Windows中的动画是使用WM_TIMER消息来实作的。如果这个线程没有建立窗口，那么它也不会收到这些消息。如果没有定时器，动画又可能会执行得太快。

解决方案是Sleep函数。实际上，线程呼叫Sleep函数来自动暂停执行，该函数唯一一个参数是以毫秒计的时间。Sleep函数呼叫在指定的时间过去以前不会传回控制权。在这段时间内，线程被暂停，并且不会被配置给时间片段（尽管该线程显然仍然要求在tick时给予一小段的处理时间，因为系统必须确定线程是否应该重新开始执行）。给Sleep一个值为0的参数将导致线程交回它尚未使用完的时间片段。

当一个线程呼叫Sleep时，只是该线程被暂停指定的时间。系统仍然执行其它的执行绪，这些执行绪和暂停的执行绪可以是在同一个程序中，也可以是在另一个程序中。我在第十四章中的SCRAMBLE程序中使用了Sleep函数，以放慢画面清除的操作。

通常，您不应该在您的主线程中使用Sleep函数，因为这会减慢对消息的处理速度，但是因为SCRAMBLE没有建立任何窗口，因此在那里使用Sleep应该没有问题。

线程同步

大约每年一次，在我公寓窗外的交通繁忙地段的红绿灯会停止工作。结果是造成交通的混乱，虽然轿车一般能避免撞上别的轿车，但是这些车经常挤在一起。

我用术语称两条路相交的十字路口为「临界区域」。一辆向南的车和一辆向西的车不可能同时通过一个十字路口而不撞着对方。依赖于交通流量，可以采用不同的方法来解决这个问题。对于视野清楚车辆稀少的路口，可以相信司机有处理的能力。车辆增多可能会要求一个停车号志，而更加繁忙的交通则将要求有红绿灯，红绿灯有助于协调路口的交通（当然，这些灯号必须正常工作）。

临界区域

在单工操作系统中，传统的计算机程序不需要红绿灯来帮助协调它们之间的行为。它们在执行时似乎独占了整条路，而且也确实是这样，没有什么会干扰它们的工作。

即使在多任务操作系统中，大多数的程序也似乎各自独立地在执行，但是可能会发生一些问题。例如，两个程序可能会需要同时从同一个文件中读或者对同一文件进行写。在这种情况下，操作系统提供了一种共享文件和记录上锁的技术来帮助解决这个问题。

然而，在支持多线程的操作系统中，情况会变得混乱而且存在潜在的危险。两个或多个线程共享某些数据的情况并不罕见。例如，一个线程可以更新一个或者多个变量，而另一个线程可以使用这些变量。有时这会引发一个问题，有时又不会（记住操作系统将控制权从一个线程切换到另一个

线程的操作，只能在机器码指令之间发生。如果只是一个整数被线程共享，那么对这个变量的改变通常发生在单个指令中，因此潜在的问题被最小化了）。

然而，假设线程共享几个变量或者数据结构。通常，这么多个变量或者结构的字段在它们之间必须是一致的。操作系统可以在更新这些变量的程序中间中断一个线程，那么使用这些变量的线程得到的将是不一致的数据。

结果是冲突发生了，并且通常不难想象这样的错误将对程序造成怎样的破坏。我们所需要的是类似于红绿灯的程序写作技术，以帮助我们对线程交通进行协调和同步，这就是临界区域。大体上，一个临界区域就是一块不可中断的程序代码。

有四个函数用于临界区域。要使用这些函数，您必须定义一个临界区域对象，这是一个型态为CRITICAL_SECTION的整体变量。例如：

CRITICAL_SECTION cs ;

这个CRITICAL_SECTION数据型态是一个结构，但是其中的字段只能由Windows内部使用。这个临界区域对象必须先被程序中的某个线程初始化，通过呼叫：

InitializeCriticalSection (&cs) ;

这样就建立了一个名为cs的临界区域对象。该函数的在线辅助说明包含下面的警告：「临界区域对象不能被移动或者复制，程序也不能修改该对象，但必须在逻辑上把它视为不透明的。」这句话，可以被解释为：「不要干扰它，甚至不要看它。」

当临界区域对象被初始化之后，线程可以通过下面的呼叫进入临界区域：

EnterCriticalSection (&cs) ;

在这时，线程被认为「拥有」临界区域对象。两个线程不可以同时拥有同一个临界区域对象，因此，如果一个线程进入了临界区域，那么下一个使用同一临界区域对象呼叫EnterCriticalSection的线程将在函数呼叫中被暂停。只有当第一个线程通过下面的呼叫离开临界区域时，函数才会传回控制权：

LeaveCriticalSection (&cs) ;

这时，在EnterCriticalSection呼叫中被停住的那个线程将拥有临界区域，其函数呼叫也将传回，允许线程继续执行。

当临界区域不再被程序所需要时，可以通过呼叫

DeleteCriticalSection (&cs) ;

将其删除，该函数释放所有被配置来维护此临界区域对象的系统资源。

这种临界区域技术涉及「互斥」（此术语在我们继续讨论线程同步时将再次出现）。在任何时刻，只有一个线程能拥有一个临界区域。因此，一个线程可以进入一个临界区域，设定一个结构的字段，然后退出临界区域。另一个使用该结构的线程在存取结构中的字段之前也要先进入该临界区域，然后再退出临界区域。

注意，您可以定义多个临界区域对象，比如cs1和cs2。例如，如果一个程序有四个线程，而前两个线程共享一些数据，那么它们可以使用一个临界区域对象，而另外两个线程共享一些其它的数据，那么它们可以使用另一个临界区域对象。

您在主线程中使用临界区域时应该小心。如果从属线程在它自己的临界区域中花费了一段很长的时间，那么它可能会将主线程的执行阻碍很长一段时间。从属执行绪可能只是使用临界区域复制

该结构的字段到自己的区域变量中。

临界区域的一个限制是它们只能用于在同一程序内的线程之间的协调。但是在某些情况下，您需要协调两个不同程序对同一资源的共享（如共享内存等）。在此其况下不能使用临界区域，但是可以使用一种被称为「互斥对象（mutex object）」的技术。「mutex」是个合成字，代表「mutual exclusion（互斥）」，它在这里精确地表达了我们的目的。我们想防止一个程序的线程在更新数据或者使用共享内存与其它资源时被中断。

事件信号

多线程通常是用于那些必须执行长时间处理的程序。我们可以将一个「大作业」定义为一个可能会违反1/10秒规则的程序。显然大作业包括文书处理程序中的拼写检查、数据库程序中的文件排序或者索引、电子表格的重新计算、打印，甚至包括复杂的绘图。当然，迄今为止我们知道，遵循1/10秒规则的最好方法是将大作业放到另一个线程去执行。这些额外的执行绪不会建立窗口，因此它们不受1/10秒规则的限制。

通常希望这些额外的线程在完成其任务时能够通知主线程，或者主线程能够停止其它线程正在进行的作业。这就是我们下面将要讨论的。

BIGJOB1程序

作为一个想象的大作业，我将使用一系列浮点运算，有时这种运算被称为「暴力的」性能测试指针。这种计算以一种间接的方式递增一个整数的值：它求一个数的平方，再对结果取平方根（得到原来的整数），然后使用log和exp函数（同样得到原来的整数），接着使用atan和tan函数（还是得到原来的整数），最后对结果加1。

BIGJOB1程序如程序20-4所示。

程序20-4 BIGJOB1

BIGJOB1.C

```
/*--------------------------------------------------------------------------
BIGJOB1.C -- Multithreading Demo
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#include <process.h>

#define REP 1000000

#define STATUS_READY 0
#define STATUS_WORKING 1
#define STATUS_DONE 2

#define WM_CALC_DONE (WM_USER + 0)
#define WM_CALC_ABORTED (WM_USER + 1)

typedef struct
{
  HWND hwnd ;
  BOOL bContinue ;
}
PARAMS, *PPARAMS ;
LRESULT APIENTRY WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
```

```
  static TCHAR szAppName[] = TEXT ("BigJob1") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;
  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Multithreading Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void Thread (PVOID pvoid)
{
  double A = 1.0 ;
  INT i ;
  LONG lTime ;
  volatile PPARAMS pparams ;

  pparams = (PPARAMS) pvoid ;
  lTime = GetCurrentTime () ;
  for (i = 0 ; i < REP && pparams->bContinue ; i++)
    A = tan (atan (exp (log (sqrt (A * A))))) + 1.0 ;
  if (i == REP)
  {
    lTime = GetCurrentTime () - lTime ;
    SendMessage (pparams->hwnd, WM_CALC_DONE, 0, lTime) ;
  }
  else
    SendMessage (pparams->hwnd, WM_CALC_ABORTED, 0, 0) ;
  _endthread () ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static INT iStatus ;
  static LONG lTime ;
  static PARAMS params ;
  static TCHAR *szMessage[] = { TEXT ("Ready (left mouse button begins)"),
    TEXT ("Working (right mouse button ends)"),
    TEXT ("%d repetitions in %ld msec") } ;
  HDC hdc ;
```

```
  PAINTSTRUCT ps ;
  RECT rect ;
  TCHAR szBuffer[64] ;

  switch (message)
  {
  case WM_LBUTTONDOWN:
    if (iStatus == STATUS_WORKING)
    {
      MessageBeep (0) ;
      return 0 ;
    }

    iStatus = STATUS_WORKING ;

    params.hwnd = hwnd ;
    params.bContinue = TRUE ;

    _beginthread (Thread, 0, &params) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_RBUTTONDOWN:
    params.bContinue = FALSE ;
    return 0 ;

  case WM_CALC_DONE:
    lTime = lParam ;
    iStatus = STATUS_DONE ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_CALC_ABORTED:
    iStatus = STATUS_READY ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    wsprintf (szBuffer, szMessage[iStatus], REP, lTime) ;
    DrawText (hdc, szBuffer, -1, &rect,
      DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

这是一个相当简单的程序，但是我认为您将看到它如何展示在多线程程序中完成大作业的通用方法。为了使用BIGJOB1程序，在窗口的显示区域中按下鼠标左键，从而开始暴力的性能测试计算的1,000,000次重复，这在一台300MHz的Pentium II机器上将花费2秒。当完成计算时，花费的时间将显示在窗口上。当正在进行计算时，您可以通过在显示区域中按下鼠标右键来终止它。

让我们来看一看这是如何实作的：

窗口消息处理程序拥有了一个被叫做iStatus的静态变量（该变量可以被设定为在程序开始处定义的三个常数之一，常数以STATUS为前缀），该变量表示程序是否准备好进行一次计算，是否正在

进行一次计算，或者是否完成了计算。程序在WM_PAINT消息处理期间使用iStatus变量在显示区域的中央显示一个适当的字符串。

窗口消息处理程序还拥有一个静态结构（型态为PARAMS，也定义在程序的顶部），该结构是在窗口消息处理程序和其它线程之间的共享数据。结构只有两个字段 – hwnd（程序窗口的句柄）和bContinue，这是一个布尔变量，用于指示线程是否继续计算或者停止。

当您在显示区域中按下鼠标左键时，窗口消息处理程序将iStatus变量设为STATUS_WORKING，并设定PARAMS结构中的两个字段。结构的hwnd字段被设定为窗口句柄，当然，bContinue被设定为TRUE。

然后窗口过程调用_beginthread函数。线程函数Thread以呼叫GetCurrentTime开始，GetCurrentTime取得以毫秒计的Windows启动以来已经执行了的时间。然后它进入一个for循环，重复1,000,000次的暴力测试计算。还要注意，如果bContinue被设为了FALSE，那么线程将退出循环。

在for循环之后，线程函数检查它是否确实完成了1,000,000次计算。如果是，那么它再次呼叫GetCurrentTime获得所经过的时间，然后使用SendMessage向窗口消息处理程序发送一个由程序定义的WM_USER_DONE消息，并以经过的时间作为lParam参数。如果计算是在未完成之前被终止的（即，如果在循环期间PARAMS结构的bContinue字段变为FALSE），那么线程将发送给窗口消息处理程序一个WM_USER_ABORTED消息。然后，线程通过呼叫_endthread正常地结束。

在窗口消息处理程序中，当您在显示区域中按下鼠标右键时，PARAMS结构的bContinue字段被设为FALSE。这是如何在完成计算之前结束计算的方法。

注意Thread中的pparams变量定义为volatile，这种型态限定字向编译器指出变量可能会在实际的程序叙述外被修改（例如被另一个线程）。否则，最佳化的编译器会假设pparams->bContinue不能被for循环内的程序代码修改，没有必要在每层循环中检查变量。volatile关键词防止这样的最佳化进行。

窗口消息处理程序处理WM_USER_DONE消息时，首先储存经过的时间。对WM_USER_DONE和WM_USER_ABORTED消息的处理都是透过对InvalidateRect的呼叫产生WM_PAINT消息并在显示区域显示一个新的字符串。

提供一个方法（如结构中的bContinue字段）允许线程正常终止，通常是一个好主意。KillThread函数只有在正常终止线程比较困难时才应该使用，原因是线程可以配置资源，如内存等。如果当线程终止时没有释放所配置的内存，那么内存将仍然是被配置了的。线程不是程序：所配置的资源在一个程序的所有线程之间是共享的，因此当线程终止时，资源不会被自动释放。好的程序结构要求一个线程释放由它配置的所有资源。

您还应该知道当第二个线程仍在执行时，可以建立第三个执行绪。如果Windows在SendMessage呼叫和_endthread呼叫之间，将控制权从第二个线程切换到第一个线程，那么窗口消息处理程序就可能响应鼠标按键而建立一个新的线程，从而出现了上述的情况。这不是什么问题，但是如果这对您自己的应用来说是一个问题的话，那么您可能会考虑使用临界区域来避免线程之间的冲突。

事件对象

BIGJOB1在每次需要执行暴力测试计算时，就建立一个执行绪。执行绪在完成计算之后自动终止。

另一种可用的方法是在程序的整个生命周期内保持线程的执行，但是只在必要时才启动它。这

是一个应用事件对象的理想情况。

事件对象可以是「有信号的」（也称为「被设立的」）或「没信号的」（也称为「被重置的」）。您可以通过下面呼叫来建立事件对象：

hEvent = CreateEvent (&sa, fManual, fInitial, pszName) ;

第一个参数（指向一个SECURITY_ATTRIBUTES结构的指针）和最后一个参数（一个事件对象的名字）只有在事件对象被多个程序共享时才有意义。在同一程序中，这些参数通常被设定为NULL。如果您希望事件对象被初始化为有信号的，那么将fInitial参数设定为TRUE。而如果希望事件对象被初始化为无信号的，则将fInitial参数设定为FALSE。稍后，我将简短地描述fManual参数。

要设立一个现存的事件对象，呼叫

SetEvent (hEvent) ;

要重置一个事件对象，呼叫

ResetEvent (hEvent) ;

一个程序通常呼叫：

WaitForSingleObject (hEvent, dwTimeOut) ;

并且将第二个参数设定为INFINITE。如果事件对象目前是被设立的，那么函数将立即传回，否则，函数将暂停线程直到事件对象被设立。如果您将第二个参数设定为一个以毫秒计的超时时间值，这样函数也可能在事件对象被设立之前传回。

如果最初的CreateEvent呼叫的fManual参数被设定为FALSE，那么事件对象将在WaitForSingleObject函数传回时自动重置。这种功能特性通常使得事件对象没有必要使用ResetEvent函数。

现在，我们可以来看一看程序20-5所示的BIGJOB2.C程序。

程序20-5 BIGJOB2

BIGJOB2.C

```
/*--------------------------------------------------------------------------
BIGJOB2.C -- Multithreading Demo
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#include <process.h>

#define REP 1000000

#define STATUS_READY 0
#define STATUS_WORKING 1
#define STATUS_DONE 2

#define WM_CALC_DONE (WM_USER + 0)
#define WM_CALC_ABORTED (WM_USER + 1)

typedef struct
{
  HWND hwnd ;
  HANDLE hEvent ;
  BOOL bContinue ;
}
PARAMS, *PPARAMS ;
```

```
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("BigJob2") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Multithreading Demo"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void Thread (PVOID pvoid)
{
  double A = 1.0 ;
  INT i ;
  LONG lTime ;
  volatile PPARAMS pparams ;

  pparams = (PPARAMS) pvoid ;
  while (TRUE)
  {
    WaitForSingleObject (pparams->hEvent, INFINITE) ;
    lTime = GetCurrentTime () ;
    for (i = 0 ; i < REP && pparams->bContinue ; i++)
      A = tan (atan (exp (log (sqrt (A * A))))) + 1.0 ;
    if (i == REP)
    {
      lTime = GetCurrentTime () - lTime ;
      PostMessage (pparams->hwnd, WM_CALC_DONE, 0, lTime) ;
    }
    else
      PostMessage (pparams->hwnd, WM_CALC_ABORTED, 0, 0) ;
  }
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
```

```
{
  static HANDLE hEvent ;
  static INT iStatus ;
  static LONG lTime ;
  static PARAMS params ;
  static TCHAR *szMessage[] = { TEXT ("Ready (left mouse button begins)"),
    TEXT ("Working (right mouse button ends)"),
    TEXT ("%d repetitions in %ld msec") } ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;
  TCHAR szBuffer[64] ;

  switch (message)
  {
  case WM_CREATE:
    hEvent = CreateEvent (NULL, FALSE, FALSE, NULL) ;

    params.hwnd = hwnd ;
    params.hEvent = hEvent ;
    params.bContinue = FALSE ;

    _beginthread (Thread, 0, &params) ;

    return 0 ;

  case WM_LBUTTONDOWN:
    if (iStatus == STATUS_WORKING)
    {
      MessageBeep (0) ;
      return 0 ;
    }
    iStatus = STATUS_WORKING ;
    params.bContinue = TRUE ;

    SetEvent (hEvent) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_RBUTTONDOWN:
    params.bContinue = FALSE ;
    return 0 ;

  case WM_CALC_DONE:
    lTime = lParam ;
    iStatus = STATUS_DONE ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_CALC_ABORTED:
    iStatus = STATUS_READY ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    wsprintf ( szBuffer, szMessage[iStatus], REP, lTime) ;
    DrawText ( hdc, szBuffer, -1, &rect,
      DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
```

```
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

处理WM_CREATE消息时，窗口消息处理程序首先建立一个初始化为没信号的自动重置事件对象，然后建立线程。

Thread函数进入一个无限的while循环，在循环开始时首先呼叫WaitForSingleObject（注意PARAMS结构包括一个包含事件对象句柄的字段）。因为事件被初始化为重置的，所以线程的执行被阻挡在函数呼叫中。按下鼠标左键将导致窗口过程调用SetEvent，这将释放由WaitForSingleObject呼叫产生的第二个线程，并开始暴力测试计算。当计算完之后，线程再次呼叫WaitForSingleObject，但是由于第一次呼叫已经使事件对象重置，因此，线程将被暂停，直到再次按下鼠标。

在其它方面，程序几乎和BIGJOB1完全一样。

线程区域储存空间 （TLS）

多线程程序中的整体变量（以及任何被配置的内存）被程序中的所有线程共享。在一个函数中的局部静态变量也被使用函数的所有线程共享。一个函数中的局部动态变量是唯一于各个线程的，因为它们被储存在堆栈上，而每个线程有它自己的堆栈。

对各个线程唯一的持续性储存空间有存在的必要。例如，我在本章前面提到过的C中的strtok函数要求这种型态的储存空间。不幸的是，C语言不支持这类储存空间。但是Windows中提供了四个函数，它们实作了一种技术来做到这一点，并且Microsoft对C的扩充语法也支持它，这就叫做线程区域储存空间。

下面是API工作的方法：

首先，定义一个包含需要唯一于线程的所有数据的结构，例如：

```
typedef struct
{
  int a ;
  int b ;
}
DATA, * PDATA ;
```

主线程呼叫TlsAlloc获得一个索引值：

dwTlsIndex = TlsAlloc () ;

这个值可以储存在一个整体变量中或者通过参数结构传递给线程函数。

线程函数首先为该数据结构配置内存，并使用上面所获得的索引值呼叫TlsSetValue：

TlsSetValue (dwTlsIndex, GlobalAlloc (GPTR, sizeof (DATA)) ;

该函数将一个指标和某个线程及某个线程索引相关联。现在，任何需要使用这个指标的函数（包括最初的线程函数本身）都可以包含如下所示的程序代码：

PDATA pdata ;

...

pdata = (PDATA) TlsGetValue (dwTlsIndex) ;

现在函数可以设定或者使用pdata->a和pdata->b了。在线程函数终止以前，它释放配置的内存：

GlobalFree (TlsGetValue (dwTlsIndex)) ;

当使用该数据的所有线程都终止之时，主线程将释放索引：

TlsFree (dwTlsIndex) ;

这个程序刚开始可能令人有些迷惑，因此如果能看一看如何实作线程区域储存空间可能会有帮助（我不知道Windows实际上是如何实作的，但下面的方案是可能的）。首先，TlsAlloc可能只是配置一块内存（长度为0）并传回一个索引值，即指向这块内存的一个指针。每次使用该索引呼叫TlsSetValue时，通过重新配置将内存块增大8个字节。在这8个字节中储存的是呼叫函数的线程ID（通过GetCurrentThreadId来获得）以及传递给TlsSetValue函数的指标。TlsSetValue简单地使用线程ID来搜寻操作系统管理的线程区域储存空间地址表，然后传回指标。TlsFree将释放内存块。所以您看，这可能是一件容易得可以由您自己来实作的事情。不过，既然已经有工具为您做好了这些工作，那也不错。

Microsoft对C的扩充功能使这件工作更加容易。只要在要对每个线程都保留不同内容的变量前加上__declspec (thread)就好了。对于任何函数的外部静态变量，则为：

__declspec (thread) int iGlobal = 1 ;

对于函数内部的静态变量，则为：

__declspec (thread) static int iLocal = 2 ;

第二十一章 动态链接库

动态链接库（也称为DLL）是Microsoft Windows最重要的组成要素之一。大多数与Windows相关的磁盘文件如果不是程序模块，就是动态链接程序。迄今为止，我们都是在开发Windows应用程序；现在是尝试编写动态链接库的时候了。许多您已经学会的编写应用程序的规则同样适用于编写这些动态链接库模块，但也有一些重要的不同。

动态链接库的基本知识

正如前面所看到的，Windows应用程序是一个可执行文件，它通常建立一个或几个窗口，并使用消息循环接收使用者输入。通常，动态链接库并不能直接执行，也不接收消息。它们是一些独立的文件，其中包含能被程序或其它DLL呼叫来完成一定作业的函数。只有在其它模块呼叫动态链接库中的函数时，它才发挥作用。

所谓「动态链接」，是指Windows把一个模块中的函数呼叫连结到动态链接库模块中的实际函数上的程序。在程序开发中，您将各种目标模块(.OBJ)、执行时期链接库(.LIB)文件，以及经常是已编译的资源(.RES)文件连结在一起，以便建立Windows的.EXE文件，这时的连结是「静态连结」。动态链接与此不同，它发生在执行时期。

KERNEL32.DLL 、 USER32.DLL 和 GDI32.DLL 、 各 种 驱 动 程 序 文 件 如 KEYBOARD.DRV 、 SYSTEM.DRV和MOUSE.DRV和视频及打印机驱动程序都是动态链接库。这些动态链接库能被所有Windows应用程序使用。

有些动态链接库（如字体文件等）被称为「纯资源」。它们只包含数据（通常是资源的形式）而不包含程序代码。由此可见，动态链接库的目的之一就是提供能被许多不同的应用程序所使用的函数和资源。在一般的操作系统中，只有操作系统本身才包含其它应用程序能够呼叫来完成某一作业的例程。在Windows中，一个模块呼叫另一个模块函数的程序被推广了。结果使得编写一个动态链接库，也就是在扩充Windows。当然，也可认为动态链接库（包括构成Windows的那些动态链接库例程）是对使用者程序的扩充。

尽管一个动态链接库模块可能有其它扩展名（如.EXE或.FON），但标准扩展名是.DLL。只有带.DLL扩展名的动态链接库才能被Windows自动加载。如果文件有其它扩展名，则程序必须另外使用LoadLibrary或者LoadLibraryEx函数加载该模块。

您通常会发现，动态链接库在大型应用程序中最有意义。例如，假设要为Windows编写一个由几个不同的程序组成的大型财务软件包，就会发现这些应用程序会使用许多共同的例程。可以把这些公共例程放入一个一般性的目的码链接库（带.LIB扩展名）中，并在使用LINK静态连结时把它们加入各程序模块中。但这种方法是很浪费的，因为软件包中的每个程序都包含与公共例程相同的程序代码。而且，如果修改了链接库中的某个例程，就要重新连结使用此例程的所有程序。然而，如果把这些公共例程放到称为ACCOUNT.DLL的动态链接库中，就可解决这两个问题。只有动态链接库模块才包含所有程序都要用到的例程。这样能为储存文件节省磁盘空间，并且在同时执行多个应用程序时节省内存，而且，可以修改动态链接库模块而不用重新连结各个程序。

动态链接库实际上是可以独立存在的。例如，假设您编写了一系列3D绘图例程，并把它们放入

名为GDI3.DLL的DLL中。如果其它软件开发者对此链接库很感兴趣，您就可以授权他们将其加入他们的图形程序中。使用多个这样的图形程序的使用者只需要一个GDI3.DLL文件。

链接库：一词多义

动态链接库有着令人困惑的印象，部分原因是由于「链接库」这个词被放在几种不同的用语之后。除了动态链接库之外，我们也用它来称呼「目的码链接库」或「引用链接库」。

目的码链接库是带.LIB扩展名的文件。在使用连结程序进行静态连结时，它的程序代码就会加到程序的.EXE文件中。例如，在Microsoft Visual C++中，连同程序连结的一般C执行目的码链接库被称为LIBC.LIB。

引用链接库是目的码链接库文件的一种特殊形式。像目的码链接库一样，引用链接库有.LIB扩展名，并且被连结器用来确定程序代码中的函数呼叫来源。但引用链接库不含程序代码，而是为连结程序提供信息，以便在.EXE文件中建立动态链接时要用到的复位位表。包含在Microsoft编译器中的KERNEL32.LIB、USER32.LIB和GDI32.LIB文件是Windows函数的引用链接库。如果一个程序呼叫Rectangle函数，Rectangle将告诉LINK，该函数在GDI32.DLL动态链接库中。该信息被记录在.EXE文件中，使得程序执行时，Windows能够和GDI32.DLL动态链接库进行动态连结。

目的码链接库和引用链接库只用在程序开发期间使用，而动态链接库在执行期间使用。当一个使用动态链接库的程序执行时，该动态链接库必须在磁盘上。当Windows要执行一个使用了动态链接库的程序而需要加载该链接库时，动态链接库文件必须储存在含有该.EXE程序的目录下、目前的目录下、Windows系统目录下、Windows目录下，或者是在通过MS-DOS环境中的PATH可以存取到的目录下（Windows会按顺序搜索这些目录）。

一个简单的DLL

虽然动态链接库的整体概念是它们可以被多个应用程序所使用，但您通常最初设计的动态链接库只与一个应用程序相联系，可能是一个「测试」程序在使用DLL。

下面就是我们要做的。我们建立一个名为EDRLIB.DLL的DLL。文件名中的「EDR」代表「简便的绘图例程（easy drawing routines）」。这里的EDRLIB只含有一个函数（名称为EdrCenterText），但是您还可以将应用程序中其它简单的绘图函数添加进去。应用程序EDRTEST.EXE将通过呼叫EDRLIB.DLL中的函数来利用它。

要做到这一点，需要与我们以前所做的略有不同的方法，也包括Visual C++ 中我们没有看过的特性。在Visual C++ 中「工作空间（workspaces）」和「项目（projects）」不同。项目通常与建立的应用程序（.EXE）或者动态链接库（.DLL）相联系。一个工作空间可以包含一个或多个项目。迄今为止，我们所有的工作空间都只包含一个项目。我们现在就建立一个包含两个项目的工作空间EDRTEST – 一个用于建立EDRTEST.EXE，而另一个用于建立EDRLIB.DLL，即EDRTEST使用的动态链接库。

现在就开始。在Visual C++中，从「File」菜单选择「New」，然后选择「Workspaces」页面标签。（我们以前从来没有选择过。）在「Location」栏选择工作空间要储存的目录，然后在「Workspace Name」栏输入「EDRTEST」，按Enter键。

这样就建立了一个空的工作空间。Developer Studio还建立了一个名为EDRTEST的子目录，以及工作空间文件EDRTEST.DSW（就像两个其它文件）。

现在让我们在此工作空间里建立一个项目。从「File」菜单选择「New」，然后选择「Projects」页面标签。尽管过去您选择「Win32 Application」，但现在「Win32 Dynamic-Link Library」。另外，单击单选按钮「Add To Current Workspace」，这使得此项目是「EDRTEST」工作空间的一部

分。在「Project Name栏输入EDRLIB，但先不要按「OK」按钮。当您在Project Name栏输入EDRLIB时，Visual C++将改变「Location」栏，以显示EDRLIB作为EDRTEST的一个子目录。这不是我们要的，所以接着在「Location」栏删除EDRLIB子目录以便项目建立在EDRTEST目录。现在按「OK」。屏幕将显示一个对话框，询问您建立什么型态的DLL。选择「An Empty DLL Project」，然后按「Finish」。Visual C++将建立一个项目文件EDRLIB.DSP和一个构造文件EDRLIB.MAK（如果「Tools Options」对话框的B「uild页面卷标中选择了「Export Makefile」选项」。

现在您已经在此项目中添加了一对文件。从「File」菜单选择「New」，然后选择「Files」页面标签。选择「C/C++ Header File」，然后输入文件名EDRLIB.H。输入程序21-1所示的文件（或者从本书光盘中复制）。再次从「File」菜单中选择「New」，然后选择「Files」页面标签。这次选择「C++ Source File」，然后输入文件名EDRLIB.C。继续输入程序21-1所示的程序。

程序21-1 EDRLIB动态链接库

EDRLIB.H

```
/*--------------------------------------------------------------------------
EDRLIB.H header file
--------------------------------------------------------------------------*/
#ifdef __cplusplus
#define EXPORT extern "C" __declspec (dllexport)
#else
#define EXPORT __declspec (dllexport)
#endif

EXPORT BOOL CALLBACK EdrCenterTextA (HDC, PRECT, PCSTR) ;
EXPORT BOOL CALLBACK EdrCenterTextW (HDC, PRECT, PCWSTR) ;

#ifdef UNICODE
#define EdrCenterText EdrCenterTextW
#else
#define EdrCenterText EdrCenterTextA
#endif
```

EDRLIB.C

```
/*--------------------------------------------------------------------------
EDRLIB.C -- Easy Drawing Routine Library module
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include "edrlib.h"

int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
  return TRUE ;
}

EXPORT BOOL CALLBACK EdrCenterTextA ( HDC hdc, PRECT prc, PCSTR pString)
{
  int iLength ;
  SIZE size ;

  iLength = lstrlenA (pString) ;
  GetTextExtentPoint32A (hdc, pString, iLength, &size) ;
  return TextOutA (hdc,(prc->right - prc->left - size.cx) / 2,
    ( prc->bottom - prc->top - size.cy) / 2,
    pString, iLength) ;
}

EXPORT BOOL CALLBACK EdrCenterTextW (HDC hdc, PRECT prc, PCWSTR pString)
{
  int iLength ;
  SIZE size ;

  iLength = lstrlenW (pString) ;
  GetTextExtentPoint32W (hdc, pString, iLength, &size) ;
```

```
  return TextOutW (hdc, ( prc->right - prc->left - size.cx) / 2,
    ( prc->bottom - prc->top - size.cy) / 2,
    pString, iLength) ;
}
```

这里您可以按Release设定，或者也可以按Debug设定来建立EDRLIB.DLL。之后，RELEASE和DEBUG目录将包含EDRLIB.LIB（即动态链接库的引用链接库）和EDRLIB.DLL（动态链接库本身）。

纵观全书，我们建立的所有程序都可以根据UNICODE标识符来编译成使用Unicode或非Unicode字符串的程序代码。当您建立一个DLL时，它应该包括处理字符和字符串的Unicode和非Unicode版的所有函数。因此，EDRLIB.C就包含函数EdrCenterTextA（ANSI版）和EdrCenterTextW（宽字符版）。EdrCenterTextA定义为带有参数PCSTR（指向const字符串的指针），而EdrCenterTextW则定义为带有参数PCWSTR（指向const宽字符串的指针）。EdrCenterTextA函数将呼叫lstrlenA、GetTextExtentPoint32A和TextOutA。EdrCenterTextW将呼叫lstrlenW、GetTextExtentPoint32W和TextOutW。如果定义了UNICODE标识符，则EDRLIB.H将EdrCenterText定义为EdrCenterTextW，否则定义为EdrCenterTextA。这样的做法很像Windows表头文件。

EDRLIB.H也包含函数DllMain，取代了DLL中的WinMain。此函数用于执行初始化和未初始化（deinitialization），我将在下一节讨论。我们现在所需要的就是从DllMain传回TRUE。

在这两个文件中，最后一点神秘之处就是定义了EXPORT标识符。DLL中应用程序使用的函数必须是「输出（exported）」的。这跟税务或者商业制度无关，只是确保函数名添加到EDRLIB.LIB的一个关键词（以便连结程序在连结使用此函数的应用程序时，能够解析出函数名称），而且该函数在EDRLIB.DLL中也是看得到的。EXPORT标识符包括储存方式限定词__declspec（dllexport）以及在表头文件按C++模式编译时附加的「C」。这将防止编译器使用C++的名称轧压规则（name mangling）来处理函数名称，使C和C++程序都能使用这个DLL。

链接库入口／出口点

当动态链接库首次启动和结束时，我们呼叫了DllMain函数。DllMain的第一个参数是链接库的执行实体句柄。如果您的链接库使用需要执行实体句柄（诸如DialogBox）的资源，那么您应该将hInstance储存为一个整体变量。DllMain的最后一个参数由系统保留。

fdwReason参数可以是四个值之一，说明为什么Windows要呼叫DllMain函数。在下面的讨论中，请记住一个程序可以被加载多次，并在Windows下一起执行。每当一个程序加载时，它都被认为是一个独立的程序（process）。

fdwReason的一个值DLL_PROCESS_ATTACH表示动态链接库被映像到一个程序的地址空间。链接库可以根据这个线索进行初始化，为以后来自该程序的请求提供服务。例如，这类初始化可能包括内存配置。在一个程序的生命周期内，只有一次对DllMain的呼叫以DLL_PROCESS_ATTACH为参数。使用同一DLL的其它任何程序都将导致另一个使用DLL_PROCESS_ATTACH参数的DllMain呼叫，但这是对新程序的呼叫。

如果初始化成功，DllMain应该传回一个非0值。传回0将导致Windows不执行该程序。

当fdwReason的值为DLL_PROCESS_DETACH时，意味着程序不再需要DLL了，从而提供给链接库自己清除自己的机会。在32位的Windows下，这种处理并不是严格必须的，但这是一种良好的程序写作习惯。

类似地，当以DLL_THREAD_ATTACH为fdwReason参数呼叫DllMain时，意味着某个程序建立了一个新的线程。当线程中止时，Windows以DLL_THREAD_DETACH为fdwReason参数呼叫DllMain。请注意，如果动态链接库是在线程被建立之后和一个程序连结的，那么可能会得到一个

没有事先对应一个DLL_THREAD_ATTACH呼叫的DLL_THREAD_DETACH呼叫。

当使用一个DLL_THREAD_DETACH参数呼叫DllMain时，线程仍然存在。动态链接库甚至可以在这个程序期间发送线程消息。但是它不应该使用PostMessage，因为线程可能在此消息被处理到之前就已经退出执行了。

测试程序

现在让我们在EDRTEST工作空间里建立第二个项目，程序名称为EDRTEST，而且使用EDRLIB.DLL。在Visual C++中加载EDRTEST工作空间时，请从「File」菜单选择「New」，然后在「New」对话框中选择「Projects」页面标签。这次选择「Win32 Application」，并确保选中了「Add To Current Workspace」按钮。输入项目名称EDRTEST。再在「Locations」栏删除第二个EDRTEST子目录。按下「OK」，然后在下一个对话框选择「An Empty Project」，按「Finish」。

从「File」菜单再次选择「New」。选择「Files」页面标签然后选择「C++ Source File」。确保「Add To Project」清单方块显示「EDRTEST」而不是「EDRLIB」。输入文件名称EDRTEST.C，然后输入程序21-2所示的程序。此程序用EdrCenterText函数将显示区域中的字符串居中对齐。

程序21-2 EDRTEST

EDRTEST.C

```
/*------------------------------------------------------------------------
EDRTEST.C -- Program using EDRLIB dynamic-link library
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "edrlib.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("StrProg") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("DLL Demonstration Program"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  HDC hdc ;
  PAINTSTRUCT ps ;
  RECT rect ;

  switch (message)
  {
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    GetClientRect (hwnd, &rect) ;

    EdrCenterText (hdc, &rect,
      TEXT ("This string was displayed by a DLL")) ;

    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

注意，为了定义EdrCenterText函数，EDRTEST.C包括EDRLIB.H表头文件，此函数将在WM_PAINT消息处理期间呼叫。

在编译此程序之前，您可能希望做以下几件事。首先，在「Project」菜单选择「Select Active Project」。这时您将看到「EDRLIB」和「EDRTEST」，选择「EDRTEST」。在重新编译此工作空间时，您真正要重新编译的是程序。另外，在「Project」菜单中，选择「Dependencies」，在「Select Project To Modify」清单方块中选择「EDRTEST」。在「Dependent On The Following Project(s)」列表选中「EDRLIB」。此操作的意思是：EDRTEST需要EDRLIB动态链接库。以后每次重新编译EDRTEST时，如果必要的话，都将在编译和连结EDRTEST之前重新重新编译EDRLIB。

从「Project」菜单选择「Settings」，单击「General」标签。当您在左边的窗格中选择「EDRLIB」或者「EDRTEST」项目时，如果设定为「Win32 Release」，则显示在右边窗格中的「Intermediate Files」和「Output Files」将位于RELEASE目录；如果设定为「Win32 Debug」，则位于DEBUG目录。如果不是，请按此修改。这样可确保EDRLIB.DLL与EDRTEST.EXE在同一个目录中，而且程序在使用DLL时也不会产生问题。

在「Project Setting」对话框中依然选中「EDRTEST」，单击「C/C++」页面标签。按本书的惯例，在「Preprocessor Definitions」中，将「UNICODE」添加到Debug设定。

现在您就可以在「Debug」或「Release」设定中重新编译EDRTEST.EXE了。必要时，Visual C++将首先编译和连结EDRLIB。RELEASE和DEBUG目录都包含EDRLIB.LIB（引用链接库）和EDRLIB.DLL。当Developer Studio连结EDRTEST时，将自动包含引用链接库。

了解EDRTEST.EXE文件中不包含EdrCenterText程序代码很重要。事实上，要证明执行了EDRLIB.DLL文件和EdrCenterText函数很简单：执行EDRTEST.EXE需要EDRLIB.DLL。

执行EDRTEST.EXE时，Windows按外部链接库模块执行固定的函数。其中许多函数都在一般Windows动态链接库中。但Windows也看到程序从EDRLIB呼叫了函数，因此Windows将EDRLIB.DLL文件加载到内存，然后呼叫EDRLIB的初始化例程。EDRTEST呼叫EdrCenterText函数是动态链接到EDRLIB中函数的。

在EDRTEST.C原始码文件中包含EDRLIB.H与包含WINDOWS.H类似。连结EDRLIB.LIB与连结Windows引用链接库（例如USER32.LIB）类似。当您的程序执行时，它连结EDLIB.DLL的方式与连结USER32.DLL的方式相同。恭喜您！您已经扩展了Windows的功能！

在继续之前，我还要对动态链接库多说明一些：

首先，虽然我们将DLL作为Windows的延伸，但它也是您的应用程序的延伸。DLL所完成的每件工作对于应用程序来说都是应用程序所交代要完成的。例如，应用程序拥有DLL配置的全部内存、DLL建立的全部窗口以及DLL打开的所有文件。多个应用程序可以同时使用同一个DLL，但在Windows下，这些应用程序不会相互影响。

多个程序能够共享一个动态链接库中相同的程序代码。但是，DLL为每个程序所储存的数据都不同。每个程序都为DLL所使用的全部数据配置了自己的地址空间。我们将在下以节看到，共享内存需要额外的工作。

在DLL中共享内存

令人兴奋的是，Windows能够将同时使用同一个动态链接库的应用程序分开。不过，有时却不太令人满意。您可能希望写一个DLL，其中包含能够被不同应用程序或者同一个程序的不同例程所共享的内存。这包括使用共享内存。共享内存实际上是一种内存映像文件。

让我们测试一下，这项工作是如何在程序STRPROG（「字符串程序（string program）」）和动态链接库STRLIB（「字符串链接库（string library）」）中完成的。STRLIB有三个输出函数被STRPROG呼叫，我们只对此感兴趣，STRLIB中的一个函数使用了在STRPROG定义的callback函数。

STRLIB是一个动态链接库模块，它储存并排序了最多256个字符串。在STRLIB中，这些字符串均为大写，并由共享内存维护。利用STRLIB的三个函数，STRPROG能够添加字符串、删除字符串以及从STRLIB获得目前的所有字符串。STRPROG测试程序有两个菜单项（「Enter」和「Delete」），这两个菜单项将启动不同的对话框来添加或删除字符串。STRPROG在其显示区域列出目前储存在STRLIB中的所有字符串。

下面这个函数在STRLIB定义，它将一个字符串添加到STRLIB的共享内存。

EXPORT BOOL CALLBACK AddString (pStringIn)

参数pStringIn是字符串的指针。字符串在AddString函数中变成大写。如果在STRLIB的列表中有一个相同的字符串，那么此函数将添加一个字符串的复本。如果成功，AddString传回「TRUE」（非0），否则传回「FALSE」（0）。如果字符串的长度为0，或者不能配置储存字符串的内存，或者已经储存了256个字符串，则传回值将都是FALSE。

STRLIB函数从STRLIB的共享内存中删除一个字符串。

EXPORT BOOL CALLBACK DeleteString (pStringIn)

另外，参数pStringIn是一个字符串指针。如果有多个相同内容字符串，则删除第一个。如果成功，那么DeleteString传回「TRUE」（非0），否则传回「FALSE」（0）。传回「FALSE」表示字符串长度为0，或者找不到相同内容的字符串。

STRLIB函数使用了呼叫程序中的一个callback函数，以便列出目前储存在STRLIB共享内存中的

字符串：

EXPORT int CALLBACK GetStrings (pfnGetStrCallBack, pParam)

在呼叫程序中，callback函数必须像下面这样定义：

EXPORT BOOL CALLBACK GetStrCallBack (PSTR pString, PVOID pParam)

GetStrings的参数pfnGetStrCallBack指向callback函数。直到callback函数传回「FALSE」（0），GetStrings将为每个字符串都呼叫一次GetStrCallBack。GetStrings传回传递给callback函数的字符串数。pParam参数是一个远程指针，指向程序写作者定义的数据。

当然，此程序可以编译成Unicode程序，或者在STRLIB的支持下，编译成Unicode和非Unicode应用程序。与EDRLIB一样，所有的函数都有「A」和「W」两种版本。在内部，STRLIB以Unicode储存所有的字符串。如果非Unicode程序使用了STRLIB（也就是说，程序将呼叫AddStringA、DeleteStringA和GetStringsA），字符串将在Unicode和非Unicode之间转换。

与STRPROG和STRLIB项目相关的工作空间名为STRPROG。此文件按EDRTEST工作空间的方式组合。程序21-3显示了建立STRLIB.DLL动态链接库所必须的两个文件。

程序21-3 STRLI

STRLIB.H

```
/*--------------------------------------------------------------------------
STRLIB.H header file
--------------------------------------------------------------------------*/
#ifdef __cplusplus
#define EXPORT extern "C" __declspec (dllexport)
#else
#define EXPORT __declspec (dllexport)
#endif

// The maximum number of strings STRLIB will store and their lengths
#define MAX_STRINGS 256
#define MAX_LENGTH 63

// The callback function type definition uses generic strings

typedef BOOL (CALLBACK * GETSTRCB) (PCTSTR, PVOID) ;

// Each function has ANSI and Unicode versions

EXPORT BOOL CALLBACK AddStringA (PCSTR) ;
EXPORT BOOL CALLBACK AddStringW (PCWSTR) ;

EXPORT BOOL CALLBACK DeleteStringA (PCSTR) ;
EXPORT BOOL CALLBACK DeleteStringW (PCWSTR) ;

EXPORT int CALLBACK GetStringsA (GETSTRCB, PVOID) ;
EXPORT int CALLBACK GetStringsW (GETSTRCB, PVOID) ;

// Use the correct version depending on the UNICODE identifier

#ifdef UNICODE
#define AddString AddStringW
#define DeleteString  DeleteStringW
#define GetStrings  GetStringsW
#else
#define AddString AddStringA
#define DeleteString  DeleteStringA
#define GetStrings  GetStringsA
#endif
```

STRLIB.C

```
/*--------------------------------------------------------------------------
STRLIB.C - Library module for STRPROG program
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/

#include <windows.h>
#include <wchar.h> // for wide-character string functions
#include "strlib.h"

// shared memory section (requires /SECTION:shared,RWS in link options)
#pragma data_seg ("shared")
int iTotal = 0 ;
WCHAR szStrings [MAX_STRINGS][MAX_LENGTH + 1] = { '\0' } ;
#pragma data_seg ()

#pragma comment(linker,"/SECTION:shared,RWS")

int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
  return TRUE ;
}

EXPORT BOOL CALLBACK AddStringA (PCSTR pStringIn)
{
  BOOL bReturn ;
  int iLength ;
  PWSTR pWideStr ;

  // Convert string to Unicode and call AddStringW
  iLength = MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, NULL, 0) ;
  pWideStr = (PWSTR)malloc (iLength) ;
  MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, pWideStr, iLength) ;
  bReturn = AddStringW (pWideStr) ;
  free (pWideStr) ;

  return bReturn ;
}

EXPORT BOOL CALLBACK AddStringW (PCWSTR pStringIn)
{
  PWSTR pString ;
  int i, iLength ;

  if (iTotal == MAX_STRINGS - 1)
    return FALSE ;
  if ((iLength = wcslen (pStringIn)) == 0)
    return FALSE ;
  // Allocate memory for storing string, copy it, convert to uppercase
  pString = (PWSTR)malloc (sizeof (WCHAR) * (1 + iLength)) ;
  wcscpy (pString, pStringIn) ;
  _wcsupr (pString) ;

  // Alphabetize the strings
  for (i = iTotal ; i > 0 ; i--)
  {
    if (wcscmp (pString, szStrings[i - 1]) >= 0)
      break ;
    wcscpy (szStrings[i], szStrings[i - 1]) ;
  }
  wcscpy (szStrings[i], pString) ;
  iTotal++ ;

  free (pString) ;
  return TRUE ;
}

EXPORT BOOL CALLBACK DeleteStringA (PCSTR pStringIn)
{
  BOOL bReturn ;
```

```
  int iLength ;
  PWSTR pWideStr ;

  // Convert string to Unicode and call DeleteStringW

  iLength = MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, NULL, 0) ;
  pWideStr = (PWSTR)malloc (iLength) ;
  MultiByteToWideChar (CP_ACP, 0, pStringIn, -1, pWideStr, iLength) ;
  bReturn = DeleteStringW (pWideStr) ;
  free (pWideStr) ;

  return bReturn ;
}

EXPORT BOOL CALLBACK DeleteStringW (PCWSTR pStringIn)
{
  int i, j ;
  if (0 == wcslen (pStringIn))
    return FALSE ;
  for (i = 0 ; i < iTotal ; i++)
  {
    if (_wcsicmp (szStrings[i], pStringIn) == 0)
      break ;
  }
  // If given string not in list, return without taking action
  if (i == iTotal)
    return FALSE ;
  // Else adjust list downward
  for (j = i ; j < iTotal ; j++)
    wcscpy (szStrings[j], szStrings[j + 1]) ;
  szStrings[iTotal--][0] = '\0' ;
  return TRUE ;
}

EXPORT int CALLBACK GetStringsA (GETSTRCB pfnGetStrCallBack, PVOID pParam)
{
  BOOL bReturn ;
  int i, iLength ;
  PSTR pAnsiStr ;

  for (i = 0 ; i < iTotal ; i++)
  {
    // Convert string from Unicode
    iLength = WideCharToMultiByte ( CP_ACP, 0, szStrings[i], -1, NULL, 0, NULL, NULL) ;
    pAnsiStr = (PSTR)malloc (iLength) ;
    WideCharToMultiByte ( CP_ACP, 0, szStrings[i], -1, pAnsiStr, iLength, NULL, NULL) ;

    // Call callback function

    bReturn = pfnGetStrCallBack (pAnsiStr, pParam) ;

    if (bReturn == FALSE)
      return i + 1 ;

    free (pAnsiStr) ;
  }
  return iTotal ;
}

EXPORT int CALLBACK GetStringsW (GETSTRCB pfnGetStrCallBack, PVOID pParam)
{
  BOOL bReturn ;
  int i ;

  for (i = 0 ; i < iTotal ; i++)
  {
    bReturn = pfnGetStrCallBack (szStrings[i], pParam) ;
    if (bReturn == FALSE)
      return i + 1 ;
```

```
  }
  return iTotal ;
}
```

除了DllMain函数以外，STRLIB中只有六个函数供其它函数输出用。所有这些函数都按EXPORT定义。这会使LINK在STRLIB.LIB引用链接库中列出它们。

STRPROG程序

STRPROG程序如程序21-4所示，其内容相当浅显易懂。两个菜单选项(Enter和Delete)启动一个对话框，让您输入一个字符串，然后STRPROG呼叫AddString或者DeleteString。当程序需要更新它的显示区域时，呼叫GetStrings并使用函数GetStrCallBack来列出所列举的字符串。

程序21-4 STRPROG

STRPROG.C

```
/*------------------------------------------------------------------------
STRPROG.C - Program using STRLIB dynamic-link library
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "strlib.h"
#include "resource.h"

typedef struct
{
  HDC hdc ;
  int xText ;
  int yText ;
  int xStart ;
  int yStart ;
  int xIncr ;
  int yIncr ;
  int xMax ;
  int yMax ;
}
CBPARAM ;
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("StrProg") ;
TCHAR szString [MAX_LENGTH + 1] ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("DLL Demonstration Program"),
    WS_OVERLAPPEDWINDOW,
```

```
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

BOOL CALLBACK DlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG:
    SendDlgItemMessage (hDlg, IDC_STRING, EM_LIMITTEXT, MAX_LENGTH, 0) ;
    return TRUE ;

  case WM_COMMAND:
    switch (wParam)
    {
    case IDOK:
      GetDlgItemText (hDlg, IDC_STRING, szString, MAX_LENGTH) ;
      EndDialog (hDlg, TRUE) ;
      return TRUE ;

    case IDCANCEL:
      EndDialog (hDlg, FALSE) ;
      return TRUE ;
    }
  }
  return FALSE ;
}

BOOL CALLBACK GetStrCallBack (PTSTR pString, CBPARAM * pcbp)
{
  TextOut ( pcbp->hdc, pcbp->xText, pcbp->yText,
    pString, lstrlen (pString)) ;

  if ((pcbp->yText += pcbp->yIncr) > pcbp->yMax)
  {
    pcbp->yText = pcbp->yStart ;
    if ((pcbp->xText += pcbp->xIncr) > pcbp->xMax)
      return FALSE ;
  }
  return TRUE ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HINSTANCE hInst ;
  static int cxChar, cyChar, cxClient, cyClient ;
  static UINT iDataChangeMsg ;
  CBPARAM cbparam ;
  HDC hdc ;
  PAINTSTRUCT ps ;
  TEXTMETRIC tm ;

  switch (message)
  {
  case WM_CREATE:
    hInst = ((LPCREATESTRUCT) lParam)->hInstance ;
    hdc = GetDC (hwnd) ;
    GetTextMetrics (hdc, &tm) ;
```

```
    cxChar = (int) tm.tmAveCharWidth ;
    cyChar = (int) (tm.tmHeight + tm.tmExternalLeading) ;
    ReleaseDC (hwnd, hdc) ;

    // Register message for notifying instances of data changes

    iDataChangeMsg = RegisterWindowMessage (TEXT ("StrProgDataChange")) ;
    return 0 ;
  case WM_COMMAND:
    switch (wParam)
    {
    case IDM_ENTER:
      if (DialogBox (hInst, TEXT ("EnterDlg"), hwnd, &DlgProc))
      {
        if (AddString (szString))
          PostMessage (HWND_BROADCAST, iDataChangeMsg, 0, 0) ;
        else
          MessageBeep (0) ;
      }
      break ;

    case IDM_DELETE:
      if (DialogBox (hInst, TEXT ("DeleteDlg"), hwnd, &DlgProc))
      {
        if (DeleteString (szString))
          PostMessage (HWND_BROADCAST, iDataChangeMsg, 0, 0) ;
        else
          MessageBeep (0) ;
      }
      break ;
    }
    return 0 ;

    case WM_SIZE:
      cxClient = (int) LOWORD (lParam) ;
      cyClient = (int) HIWORD (lParam) ;
      return 0 ;

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      cbparam.hdc = hdc ;
      cbparam.xText= cbparam.xStart = cxChar ;
      cbparam.yText= cbparam.yStart = cyChar ;
      cbparam.xIncr= cxChar * MAX_LENGTH ;
      cbparam.yIncr= cyChar ;
      cbparam.xMax = cbparam.xIncr * (1 + cxClient / cbparam.xIncr) ;
      cbparam.yMax = cyChar * (cyClient / cyChar - 1) ;

      GetStrings ((GETSTRCB) GetStrCallBack, (PVOID) &cbparam) ;

      EndPaint (hwnd, &ps) ;
      return 0 ;

    case WM_DESTROY:
      PostQuitMessage (0) ;
      return 0 ;

    default:
      if (message == iDataChangeMsg)
        InvalidateRect (hwnd, NULL, TRUE) ;
      break ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

STRPROG.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
ENTERDLG DIALOG DISCARDABLE 20, 20, 186, 47
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Enter"
FONT 8, "MS Sans Serif"
BEGIN
LTEXT   "&Enter:",IDC_STATIC,7,7,26,9
EDITTEXT   IDC_STRING,31,7,148,12,ES_AUTOHSCROLL
DEFPUSHBUTTON  "OK",IDOK,32,26,50,14
PUSHBUTTON  "Cancel",IDCANCEL,104,26,50,14
END
DELETEDLG DIALOG DISCARDABLE 20, 20, 186, 47
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Delete"
FONT 8, "MS Sans Serif"
BEGIN
LTEXT   "&Delete:",IDC_STATIC,7,7,26,9
EDITTEXT   IDC_STRING,31,7,148,12,ES_AUTOHSCROLL
DEFPUSHBUTTON  "OK",IDOK,32,26,50,14
PUSHBUTTON  "Cancel",IDCANCEL,104,26,50,14
END

//////////////////////////////////////////////////////////////////////////////
// Menu
STRPROG MENU DISCARDABLE
BEGIN
MENUITEM "&Enter!",   IDM_ENTER
MENUITEM "&Delete!",    IDM_DELETE
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by StrProg.rc
#define IDC_STRING    1000
#define IDM_ENTER    40001
#define IDM_DELETE    40002
#define IDC_STATIC    -1
```

STRPROG.C包含STRLIB.H表头文件，其中定义了STRPROG将使用的STRLIB中的三个函数。

当您执行STRPROG的多个执行实体的时候，本程序的奥妙之处就会显露出来。STRLIB将在共享内存中储存字符串及其指针，并允许STRPROG中的所有执行实体共享此数据。让我们看一下它是如何执行的吧。

在STRPROG执行实体之间共享数据

Windows在一个Win32程序的地址空间周围筑了一道墙。通常，一个程序的地址空间中的数据是私有的，对别的程序而言是不可见的。但是执行STRPROG的多个执行实体表示了STRLIB在程序的所有执行实体之间共享数据是毫无问题的。当您在一个STRPROG窗口中增加或者删除一个字符串时，这种改变将立即反映在其它的窗口中。

在全部例程之间，STRLIB共享两个变量：一个字符数组和一个整数（记录已储存的有效字符串的个数）。STRLIB将这两个变量储存在共享的一个特殊内存区段中：

```
#pragma data_seg ("shared")
int  iTotal = 0 ;
WCHAR  szStrings [MAX_STRINGS][MAX_LENGTH + 1] = { '\0' } ;
#pragma data_seg ()
```

第一个#pragma叙述建立数据段，这里命名为shared。您可以将这段命名为任何一个您喜欢的名字。在这里的#pragma叙述之后的所有初始化了的变量都放在shared数据段中。第二个#pragma叙述标示段的结束。对变量进行专门的初始化是很重要的，否则编译器将把它们放在普通的未初始化数据段中而不是放在shared中。

连结器必须知道有一个「shared」共享数据段。在「Project Settings」对话框选择「Link」页面卷标。选中「STRLIB」时在「Project Options」字段（在Release和Debug设定中均可），包含下面的连结叙述：

/SECTION:shared,RWS

字母RWS表示段具有读、写和共享属性。或者，您也可以直接用DLL原始码指定连结选项，就像我们在STRLIB.C那样：

#pragma comment(linker,"/SECTION:shared,RWS")

共享的内存段允许iTotal变量和szStrings字符串数组在STRLIB的所有例程之间共享。因为MAX_STRINGS等于256，而MAX_LENGTH等于63，所以，共享内存段的长度为32,772字节 – iTotal变量需要4字节，256个指针中的每一个都需要128字节。

使用共享内存段可能是在多个应用程序间共享数据的最简单的方法。如果需要动态配置共享内存空间，您应该查看内存映像文件对象的用法，文件在/Platform SDK/Windows Base Services/Interprocess Communication/File Mapping。

各式各样的 DLL 讨论

如前所述，动态链接库模块不接收消息，但是，动态链接库模块可呼叫GetMessage和PeekMessage。实际上，从消息队列中得到的消息是发给呼叫链接库函数的程序的。一般来说，链接库是替呼叫它的程序工作的，这是一项对链接库所呼叫的大多数Windows函数都适用的规则。

动态链接库可以从链接库文件或者从呼叫链接库的程序文件中加载资源（如图标、字符串和位图）。加载资源的函数需要执行实体句柄。如果链接库使用它自己的执行实体句柄（初始化期间传给链接库的），则链接库能从它自己的文件中获得资源。为了从呼叫程序的.EXE文件中得到资源，程序链接库函数需要呼叫该函数的程序的执行实体句柄。

在链接库中登录窗口类别和建立窗口需要一点技巧。窗口类别结构和CreateWindow呼叫都需要执行实体句柄。尽管在建立窗口类别和窗口时可使用动态链接库模块的执行实体句柄，但在链接库建立窗口时，窗口消息仍会发送到呼叫链接库中程序的消息队列。如果使用者必须在链接库中建立窗口类别和窗口，最好的方法可能是使用呼叫程序的执行实体句柄。

因为模态对话框的消息是在程序的消息循环之外接收到的，因此使用者可以在链接库中呼叫DialogBox来建立模态对话框。执行实体句柄可以是链接库句柄，并且DialogBox的hwndParent参数可以为NULL。

不用输入引用信息的动态链接

除了在第一次把使用者程序加载内存时，由Windows执行动态链接外，程序执行时也可以把程序同动态链接库模块连结到一起。例如，您通常会这样呼叫Rectangle函数：

Rectangle (hdc, xLeft, yTop, xRight, yBottom) ;

因为程序和GDI32.LIB引用链接库连结，该链接库提供了Rectangle的地址，因此这种方法有

效。

您也可以用更迂回的方法呼叫Rectangle。首先用typedef为Rectangle定义一个函数型态：

typedef BOOL (WINAPI * PFNRECT) (HDC, int, int, int, int) ;

然后定义两个变量：

HANDLE hLibrary ;

PFNRECT pfnRectangle ;

现在将hLibrary设定为链接库句柄，将lpfnRectangle设定为Rectangle函数的地址：

hLibrary = LoadLibrary (TEXT ("GDI32.DLL"))

pfnRectangle = (PFNPRECT) GetProcAddress (hLibrary, TEXT ("Rectangle"))

如果找不到链接库文件或者发生其它一些错误，LoadLibrary函数传回NULL。现在您可以呼叫函数然后释放链接库：

pfnRectangle (hdc, xLeft, yTop, xRight, yBottom) ;

FreeLibrary (hLibrary) ;

尽管这项执行时期动态链接的技术并没有为Rectangle函数增加多大好处，但它肯定是有用的，如果直到执行时还不知道程序动态链接库模块的名称，这时就需要使用它。

上面的程序代码使用了LoadLibrary和FreeLibrary函数。Windows为所有的动态链接库模块提供「引用计数」，LoadLibrary使引用计数递增。当Windows加载任何使用了链接库的程序时，引用计数也会递增。FreeLibrary使引用计数递减，在使用了链接库的程序执行实体结束时也是如此。当引用计数为零时，Windows将从内存中把链接库删除掉，因为不再需要它了。

纯资源链接库

可由Windows程序或其它链接库使用的动态链接库中的任何函数都必须被输出。然而，DLL也可以不包含任何输出函数。那么，DLL到底包含什么呢？答案是资源。

假设使用者正在使用需要几幅位图的Windows应用程序进行工作。通常要在程序的资源描述文件中列出资源，并用LoadBitmap函数把它们加载内存。但使用者可能希望建立若干套位图，每一套均适用于Windows所使用的不同显示卡。将不同套的位图存放到不同文件中可能是明智的，因为只需要在硬盘上保留一套位图。这些文件就是纯资源文件。

程序21-5说明如何建立包含9幅位图的名为BITLIB.DLL的纯资源链接库文件。BITLIB.RC文件列出了所有独立的位图文件并为每个文件赋予一个序号。为了建立BITLIB.DLL，需要9幅名为BITMAP1.BMP、BITMAP2.BMP等等的位图。您可以使用附带的光盘上提供的位图或者在Visual C++中建立这些位图。它们与ID从1到9相对应。

程序21-5 BITLIB

BITLIB.C

```
/*------------------------------------------------------------
BITLIB.C --  Code entry point for BITLIB dynamic-link library
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
  return TRUE ;
}
```

BITLIB.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Bitmap
1   BITMAP DISCARDABLE  "bitmap1.bmp"
2    BITMAP DISCARDABLE  "bitmap2.bmp"
3    BITMAP DISCARDABLE  "bitmap3.bmp"
4    BITMAP DISCARDABLE  "bitmap4.bmp"
5    BITMAP DISCARDABLE  "bitmap5.bmp"
6   BITMAP DISCARDABLE  "bitmap6.bmp"
7    BITMAP DISCARDABLE  "bitmap7.bmp"
8    BITMAP DISCARDABLE  "bitmap8.bmp"
9    BITMAP DISCARDABLE  "bitmap9.bmp"
```

在名为SHOWBIT的工作空间中建立BITLIB项目。在名为SHOWBIT的另一个项目中，建立程序21-6所示的SHOWBIT程序，这与前面的一样。不过，不要使BITLIB依赖于SHOWBIT；否则，连结程序中将需要BITLIB.LIB文件，并且因为BITLIB没有任何输出函数，它也不会建立BITLIB.LIB。事实上，要分别重新编译BITLIB和SHOWBIT，可以交替设定其中一个为「Active Project」然后再重新编译。

SHOWBIT.C从BITLIB读取位图资源，然后在其显示区域显示。按键盘上的任意键可以循环显示。

程序21-6 SHOWBIT

SHOWBIT.C

```
/*------------------------------------------------------------------------
SHOWBIT.C -- Shows bitmaps in BITLIB dynamic-link library
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;

TCHAR szAppName [] = TEXT ("ShowBit") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName,
    TEXT ("Show Bitmaps from BITLIB (Press Key)"),
    WS_OVERLAPPEDWINDOW,
```

```
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

void DrawBitmap (HDC hdc, int xStart, int yStart, HBITMAP hBitmap)
{
  BITMAP bm ;
  HDC hMemDC ;
  POINT pt ;

  hMemDC = CreateCompatibleDC (hdc) ;
  SelectObject (hMemDC, hBitmap) ;
  GetObject (hBitmap, sizeof (BITMAP), &bm) ;
  pt.x = bm.bmWidth ;
  pt.y = bm.bmHeight ;

  BitBlt (hdc, xStart, yStart, pt.x, pt.y, hMemDC, 0, 0, SRCCOPY) ;
  DeleteDC (hMemDC) ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HINSTANCE hLibrary ;
  static int iCurrent = 1 ;
  HBITMAP hBitmap ;
  HDC hdc ;
  PAINTSTRUCT ps ;

  switch (message)
  {
  case WM_CREATE:
    if ((hLibrary = LoadLibrary (TEXT ("BITLIB.DLL"))) == NULL)
    {
      MessageBox ( hwnd, TEXT ("Can't load BITLIB.DLL."),
        szAppName, 0) ;
      return -1 ;
    }
    return 0 ;

  case WM_CHAR:
    if (hLibrary)
    {
      iCurrent ++ ;
      InvalidateRect (hwnd, NULL, TRUE) ;
    }
    return 0 ;
  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;

    if (hLibrary)
    {
      hBitmap = LoadBitmap (hLibrary, MAKEINTRESOURCE (iCurrent)) ;

      if (!hBitmap)
      {
        iCurrent = 1 ;
```

```
      hBitmap = LoadBitmap (hLibrary,
        MAKEINTRESOURCE (iCurrent)) ;
    }
    if (hBitmap)
    {
      DrawBitmap (hdc, 0, 0, hBitmap) ;
      DeleteObject (hBitmap) ;
    }
  }
  EndPaint (hwnd, &ps) ;
  return 0 ;

 case WM_DESTROY:
  if (hLibrary)
    FreeLibrary (hLibrary) ;

  PostQuitMessage (0) ;
  return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

在处理WM_CREATE消息处理期间，SHOWBIT获得了BITLIB.DLL的句柄：

if ((hLibrary = LoadLibrary (TEXT ("BITLIB.DLL"))) == NULL)

如果BITLIB.DLL与SHOWBIT.EXE不在同一个目录，Windows将按本章前面讨论的方法搜索。如果LoadLibrary传回NULL，SHOWBIT显示一个消息框来报告错误，并从WM_CREATE消息传回-1。这将导致WinMain中的CreateWindow呼叫传回NULL，而且程序终止程序。

SHOWBIT透过链接库句柄和位图号码来呼叫LoadBitmap，从而得到一个位图句柄：

hBitmap = LoadBitmap (hLibrary, MAKEINTRESOURCE (iCurrent)) ;

如果号码iCurrent对应的位图无效或者没有足够的内存加载位图，则传回一个错误。

在处理WM_DESTROY消息时，SHOWBIT释放链接库：

FreeLibrary (hLibrary) ;

当SHOWBIT的最后一个执行实体终止时，BITLIB.DLL的引用计数变为0，并且释放所占用的内存。这就是实作「图片剪辑」程序的一种简单方法，所谓的「图片剪辑」程序就是能够将预先建立的位图（或者metafile、增强型metafile）加载到剪贴簿，以供其它程序使用的程序。

第二十二章 声音与音乐

在Microsoft Windows中，声音、音乐与视频的综合运用是一个重要的进步。对多媒体的支持起源于1991年所谓的Microsoft Windows多媒体延伸功能（Multimedia Extensions to Microsoft Windows）。1992年，Windows 3.1的发布使得对多媒体的支持成为另一类API。最近几年，CD-ROM驱动器和声卡 – 在90年代初期还很少见 – 已成为新PC的标准配备。现在，几乎所有的人们都深信：多媒体在很大程度上有益于Windows的可视化图形，从而使计算机摆脱了其只是处理数字和文字的机器的传统角色。

WINDOWS和多媒体

从某种意义上来说，多媒体就是透过与设备无关的函数呼叫来获得对各种硬件的存取。让我们首先看一下硬件，然后再看看Windows多媒体API的结构。

多媒体硬件

或许最常用的多媒体硬件就是波形声音设备，也就是平常所说的声卡。波形声音设备将麦克风的输入或其它声音输入转换为数字取样，并将其储存到内存或者储存到以.WAV为扩展名的磁盘文件中。波形声音设备还将波形转换回模拟声音，以便通过PC扩音器来播放。

声卡通常还包含MIDI设备。MIDI是符合工业标准的乐器数字化接口（Musical Instrument Digital Interface）。这类硬件播放音符以响应短的二进制命令消息。MIDI硬件通常还可以通过电缆连结到如音乐键盘等的MIDI输入设备上。通常，外部的MIDI合成器也能够添加到声卡上。

现在，大多数PC上的CD-ROM驱动器都具备播放普通音乐CD的能力。这就是平常所说的「CD声音」。来自波形声音设备、MIDI设备以及CD声音设备的输出，一般在使用者的控制下用「音量控制」程序混合在一起。

另外几种普遍的多媒体「设备」不需要额外的硬件。Windows视频设备（也称作AVI视频设备）播放扩展名为.AVI（audio-video interleave：声音视频插格）的电影或动画文件。「ActiveMovie控件」可以播放其它型态的电影，包括QuickTime和MPEG。PC上的显示卡需要特定的硬件来协助播放这些电影。

还有个别PC使用者使用某种Pioneer雷射影碟机或者Sony VISCA系列录放机。这些设备都有串行端口接口，因此可由PC软件来控制。某些显示卡具有一种称为「窗口影像（video in a window）」的功能，此功能允许一个外部的视频信号与其它应用程序一起出现在Windows的屏幕上。这也可认为是一种多媒体设备。

API概述

在Windows中，API支持的多媒体功能主要分成两个集合。它们通常称为「低阶」和「高阶」界面。

低阶接口是一系列函数，这些函数以简短的说明性前缀开头，而且在/Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Functions（与高阶函数一起）中列出。

低阶的波形声音输入输出函数的前缀是waveIn和waveOut。我们将在本章看到这些函数。另外，本章还讨论用midiOut函数来控制MIDI输出设备。这些API还包括midiIn和midiStream函数。

本章还使用前缀为time的函数，这些函数允许设定一个高分辨率的定时器例程，其定时器的时间间隔速率最低能够到1毫秒。此程序主要用于播放MIDI音乐。其它几组函数包括声音压缩、视频压缩以及动画和视频序列，可惜的是本章不包括这些函数。

您还会注意到多媒体函数列表中七个带有前缀mci的函数，它们允许存取媒体控制接口（MCI：Media Control Interface）。这是一个高阶的开放接口，用于控制多媒体PC中所有的多媒体硬件。MCI包括所有多媒体硬件都共有的许多命令，因为多媒体的许多方面都以磁带录音机这类设备播放/记录方式为模型。您为输入或输出而「打开」一台设备，进而可以「录音」（对于输入）或者「播放」（对于输出），并且结束后可以「关闭」设备。

MCI本身分为两种形式。一种形式下，可以向MCI发送消息，这类似于Windows消息。这些消息包括位编码标记和C数据结构。另一种形式下，可以向MCI发送文字字符串。这个程序主要用于描述命令语言，此语言具有灵活的字符串处理函数，但支持呼叫Windows API的函数不多。字符串命令版的MCI还有利于交互研究和学习MCI，我们马上就举一个例子。MCI中的设备名称包括CD声音（cdaudio）、波形音响（waveaudio）、MIDI编曲器（sequencer）、影碟机（videodisc）、vcr、overlay（窗口中的模拟视频）、dat（digital audio tape：数字式录频磁带）以及数字视频（digitalvideo）。MCI设备分为「简单型」和「混合型」。简单型设备（如CD声音）不使用文件。混合型设备（如波形音响）则使用文件。使用波形音响时，这些文件的扩展名是.WAV。

存取多媒体硬件的另一种方法包括DirectX API，它超出了本书的范围。

另外两个高阶多媒体函数也值得一提：MessageBeep和PlaySound，它们在第三章有示范。MessageBeep播放「控制台」的「声音」中指定的声音。PlaySound可播放磁盘上、内存中或者作为资源加载的.WAV文件。本章的后面还会用到PlaySound函数。

用TESTMCI研究MCI

在Windows多媒体的早期，软件开发套件含有一个名为MCITEST的C程序，它允许程序写作者交谈式输入MCI命令并学习这些命令的工作方式。这个程序，至少是C语言版，显然已经消失了。因此，我又重新建立了它，即程序22-1所示的TESTMCI程序。虽然我不认为目前程序代码与旧的程序代码有什么区别，但现在的使用者接口还是依据以前的MCITEST程序，并且没有使用现在的程序代码。

程序22-1 TESTMCI

TESTMCI.C

```
/*------------------------------------------------------------------------
TESTMCI.C -- MCI Command String Tester
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define ID_TIMER 1
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("TestMci") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
```

```
  }
  return 0 ;
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndEdit ;
  int iCharBeg, iCharEnd, iLineBeg, iLineEnd, iChar, iLine, iLength ;
  MCIERROR error ;
  RECT rect ;
  TCHAR szCommand [1024], szReturn [1024],
    szError [1024], szBuffer [32] ;

  switch (message)
  {
  case WM_INITDIALOG:
    // Center the window on screen

    GetWindowRect (hwnd, &rect) ;
    SetWindowPos (hwnd, NULL,
      (GetSystemMetrics (SM_CXSCREEN) - rect.right + rect.left) / 2,
      (GetSystemMetrics (SM_CYSCREEN) - rect.bottom + rect.top) / 2,
      0, 0, SWP_NOZORDER | SWP_NOSIZE) ;

    hwndEdit = GetDlgItem (hwnd, IDC_MAIN_EDIT) ;
    SetFocus (hwndEdit) ;
    return FALSE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDOK:
      // Find the line numbers corresponding to the selection

      SendMessage (hwndEdit, EM_GETSEL, (WPARAM) &iCharBeg,
        (LPARAM) &iCharEnd) ;

      iLineBeg = SendMessage (hwndEdit, EM_LINEFROMCHAR, iCharBeg, 0) ;
      iLineEnd = SendMessage (hwndEdit, EM_LINEFROMCHAR, iCharEnd, 0) ;

      // Loop through all the lines

      for (iLine = iLineBeg ; iLine <= iLineEnd ; iLine++)
      {
        // Get the line and terminate it; ignore if blank

        * (WORD *) szCommand = sizeof (szCommand) / sizeof (TCHAR) ;

        iLength = SendMessage (hwndEdit, EM_GETLINE, iLine,
          (LPARAM) szCommand) ;
        szCommand [iLength] = '\0' ;

        if (iLength == 0)
          continue ;

        // Send the MCI command

        error =mciSendString (szCommand, szReturn,
          sizeof (szReturn) / sizeof (TCHAR), hwnd) ;

        // Set the Return String field

        SetDlgItemText (hwnd, IDC_RETURN_STRING, szReturn) ;

        // Set the Error String field (even if no error)

        mciGetErrorString (error, szError, sizeof (szError) / sizeof (TCHAR)) ;

        SetDlgItemText (hwnd, IDC_ERROR_STRING, szError) ;
```

```
    }
    // Send the caret to the end of the last selected line

    iChar = SendMessage (hwndEdit, EM_LINEINDEX, iLineEnd, 0) ;
    iChar += SendMessage (hwndEdit, EM_LINELENGTH, iCharEnd, 0) ;
    SendMessage (hwndEdit, EM_SETSEL, iChar, iChar) ;

    // Insert a carriage return/line feed combination

    SendMessage (hwndEdit, EM_REPLACESEL, FALSE,
      (LPARAM) TEXT ("\r\n")) ;
    SetFocus (hwndEdit) ;
    return TRUE ;

  case IDCANCEL:
    EndDialog (hwnd, 0) ;
    return TRUE ;
  case IDC_MAIN_EDIT:
    if (HIWORD (wParam) == EN_ERRSPACE)
    {
      MessageBox (hwnd, TEXT ("Error control out of space."),
        szAppName, MB_OK | MB_ICONINFORMATION) ;
      return TRUE ;
    }
    break ;
  }
  break ;

  case MM_MCINOTIFY:
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_MESSAGE), TRUE) ;

    wsprintf (szBuffer, TEXT ("Device ID = %i"), lParam) ;
    SetDlgItemText (hwnd, IDC_NOTIFY_ID, szBuffer) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ID), TRUE) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUCCESSFUL),
      wParam & MCI_NOTIFY_SUCCESSFUL) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUPERSEDED),
      wParam & MCI_NOTIFY_SUPERSEDED) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ABORTED),
      wParam & MCI_NOTIFY_ABORTED) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_FAILURE),
      wParam & MCI_NOTIFY_FAILURE) ;

    SetTimer (hwnd, ID_TIMER, 5000, NULL) ;
    return TRUE ;

  case WM_TIMER:
    KillTimer (hwnd, ID_TIMER) ;

    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_MESSAGE), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ID), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUCCESSFUL), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUPERSEDED), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ABORTED), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_FAILURE), FALSE) ;
    return TRUE ;

  case WM_SYSCOMMAND:
    switch (LOWORD (wParam))
    {
    case SC_CLOSE:
      EndDialog (hwnd, 0) ;
      return TRUE ;
    }
    break ;
```

```
 }
 return FALSE ;
}
```

TESTMCI.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
TESTMCI DIALOG DISCARDABLE 0, 0, 270, 276
STYLE  WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "MCI Tester"
FONT 8, "MS Sans Serif"
BEGIN
EDITTEXT   IDC_MAIN_EDIT,8,8,254,100,ES_MULTILINE | ES_AUTOHSCROLL |
WS_VSCROLL
LTEXT    "Return String:",IDC_STATIC,8,114,60,8
EDITTEXT   IDC_RETURN_STRING,8,126,120,50,ES_MULTILINE |
ES_AUTOVSCROLL | ES_READONLY | WS_GROUP | NOT WS_TABSTOP
LTEXT    "Error String:",IDC_STATIC,142,114,60,8
EDITTEXT   IDC_ERROR_STRING,142,126,120,50,ES_MULTILINE |
ES_AUTOVSCROLL | ES_READONLY | NOT WS_TABSTOP
GROUPBOX   "MM_MCINOTIFY Message",IDC_STATIC,9,186,254,58
LTEXT    "",IDC_NOTIFY_ID,26,198,100,8
LTEXT   "MCI_NOTIFY_SUCCESSFUL",IDC_NOTIFY_SUCCESSFUL,26,212,100,
8,WS_DISABLED
LTEXT   "MCI_NOTIFY_SUPERSEDED",IDC_NOTIFY_SUPERSEDED,26,226,100,
8,WS_DISABLED
LTEXT   "MCI_NOTIFY_ABORTED",IDC_NOTIFY_ABORTED,144,212,100,8,
WS_DISABLED
LTEXT   "MCI_NOTIFY_FAILURE",IDC_NOTIFY_FAILURE,144,226,100,8,
WS_DISABLED
DEFPUSHBUTTON  "OK",IDOK,57,255,50,14
PUSHBUTTON   "Close",IDCANCEL,162,255,50,14
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by TestMci.rc
#define IDC_MAIN_EDIT    1000
#define IDC_NOTIFY_MESSAGE    1005
#define IDC_NOTIFY_ID    1006
#define IDC_NOTIFY_SUCCESSFUL   1007
#define IDC_NOTIFY_SUPERSEDED    1008
#define IDC_NOTIFY_ABORTED    1009
#define IDC_NOTIFY_FAILURE    1010
#define IDC_SIGNAL_MESSAGE    1011
#define IDC_SIGNAL_ID    1012
#define IDC_SIGNAL_PARAM    1013
#define IDC_RETURN_STRING    1014
#define IDC_ERROR_STRING    1015
#define IDC_DEVICES    1016
#define IDC_STATIC    -1
```

与本章的大多数程序一样，TESTMCI使用非模态对话框作为它的主窗口。与本章所有的程序一样，TESTMCI要求WINMM.LIB引用链接库在Microsoft Visual C++「Projects Settings」对话框的「Links」页列出。

此程序用到了两个最重要的多媒体函数：mciSendString和mciGetErrorText。在TESTMCI的主编辑窗口输入一些内容然后按下Enter键（或「OK」按钮）后，程序将输入的字符串作为第一个参数传递给mciSendString命令：

```
error = mciSendString (szCommand, szReturn,
                  sizeof (szReturn) / sizeof (TCHAR), hwnd) ;
```

如果在编辑窗口选择了不止一行，则程序将按顺序将它们发送给mciSendString函数。第二个参数是字符串地址，此字符串取得从函数传回的信息。程序将此信息显示在窗口的「Return String」区域。从mciSendString传回的错误代码传递给mciGetErrorString函数，以获得文字错误说明；此说明显示在TESTMCI窗口的「Error String」区域。

MCITEXT和CD声音

通过控制CD-ROM驱动器和播放声音CD，您会对MCI命令字符串留下很好的印象。因为这些命令字符串一般都非常简单，并且更重要的是您可以听到一些音乐，所以这是好的起点。您可以在/Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Command Strings中获得MCI命令字符串的参考，以方便本练习。

请确认CD-ROM驱动器的声音输出已连结到扩音器或耳机，然后放入一张声音CD，如Bruce Springsteen的「Born to Run」。Windows 98中，「CD播放程序」将启动并开始播放此唱片。如果是这样的话，终止「CD播放程序」，然后可以叫出TESTMCI并且键入命令：

open cdaudio

然后按Enter键。其中open是MCI命令，cdaudio是MCI认定的CD-ROM驱动器的设备名称（假定您的系统中只有一个CD-ROM驱动器。要获得多个CD-ROM驱动器名称需使用sysinfo命令）。

TESTMCI中的「Return String」区域显示mciSendString函数中系统传回给程序的字符串。如果执行了open命令，则此值是1。TESTMCI在「Error String」区域中显示mciGetErrorString依据mciSendString传回值所传回的信息。如果mciSendString没有传回错误代码，则「Error String」区域显示文字"The specified command was carried out"。

假定执行了open命令，现在就可以输入：

play cdaudio

CD将开始播放唱片上的第一首乐曲「Thunder Road」。输入下面的命令可以暂停播放：

pause cdaudio

或者

stop cdaudio

对于CD声音设备来说，这些叙述的功能相同。您可用下面的叙述重新播放：

play cdaudio

迄今为止，我们使用的全部字符串都由命令和设备名称组成。其中有些命令带有选项。例如，键入：

status cdaudio position

根据收听时间的长短，「Return String」区域将显示类似下面的一些字符：

01:15:25

这是些什么？很显然不是小时、分钟和秒，因为CD没有那么长。要找出时间格式，请键入：

status cdaudio time format

现在「Return String」区域显示下面的字符串：

msf

这代表「分-秒-格」。CD声音中，每秒有75格。时间格式的讯格部分可在0到74之间的范围内变化。

状态命令有一连串的选项。使用下面的命令，您可以确定msf格式的CD全部长度：

status cdaudio length

对于「Born to Run」，「Return String」区域将显示：

39:28:19

这指的是39分28秒19格。

现在试一下

status cdaudio number of tracks

「Return String」区域将显示：

8

我们从CD封面上知道「Born to Run」CD上第五首乐曲是主题曲。MCI命令中的乐曲从1开始编号。要想知道乐曲「Born to Run」的长度，可以键入下面的命令：

status cdaudio length track 5

「Return String」区域将显示：

04:30:22

我们还可确定此乐曲从盘上的哪个位置开始：

status cdaudio position track 5

「Return String」区域将显示：

17:36:35

根据这条信息，我们可以直接跳到乐曲标题：

play cdaudio from 17:36:35 to 22:06:57

此命令只播放一首乐曲，然后停止。最后的值是由4:30:22（乐曲长度）加17:36:35得到的。或者，也可以用下面的命令确定：

status cdaudio position track 6

或者，也可以将时间格式设定为乐曲-分-秒-格：

set cdaudio time format tmsf

然后

play cdaudio from 5:0:0:0 to 6:0:0:0

或者，更简单地

play cdaudio from 5 to 6

如果时间的尾部是0，那么您可去掉它们。还可以用毫秒设定时间格式。

每个MCI命令字符串都可以在字符串的后面包括选项wait和notify（但不是同时使用）。例如，假设您只想播放「Born to Run」的前10秒，而且播放后，您还想让程序完成其它工作。您可按下

面的方法进行（假定您已经将时间格式设定为tmsf）：

play cdaudio from 5:0:0 to 5:0:10 wait

这种情况下，直到函数执行结束，也就是说，直到播放完「Born to Run」的前10秒，mciSendString函数才传回。

现在很明显，一般来说，在单线程的应用程序中这不是一件好事。如果不小心键入：

play cdaudio wait

直到整个唱片播放完以后，mciSendString函数才将控制权传回给程序。如果必须使用wait选项（在只要执行MCI描述文件而不管其它事情的时候，这么做很方便，与我将展示的一样），首先使用break命令。此命令可设定一个虚拟键码，此码将中断mciSendString命令并将控制权传回给程序。例如，要设定Escape键来实作此目的，可用：

break cdaudio on 27

这里，27是十进制的VK_ESCAPE值。

比wait选项更好的是notify选项：

play cdaudio from 5:0:0 to 5:0:10 notify

这种情况下，mciSendString函数立即传回，但如果该操作在MCI命令的尾部定义，则mciSendString函数的最后一个参数所指定句柄的窗口会收到MM_MCINOTIFY消息。TESTMCI程序在MM_MCINOTIFY框中显示此消息的结果。为避免与其它可能键入的命令混淆，TESTMCI程序在5秒后停止显示MM_MCINOTIFY消息的结果。

您可以同时使用wait和notify关键词，但没有理由这么做。不使用这两个关键词，内定的操作就既不是wait，也不是您通常所希望的notify。

用这些命令结束播放时，可键入下面的命令来停止CD：

stop cdaudio

如果在关闭之前没有停止CD-ROM设备，那么甚至在关闭设备之后还会继续播放CD。

另外，您还可以试试您的硬件允许或者不允许的一些命令：

eject cdaudio

最后按下面的方法关闭设备：

close cdaudio

虽然TESTMCI自己不能储存或加载文本文件，但可以在编辑控件和剪贴簿之间复制文字：先从TESTMCI选择一些内容，将其复制到剪贴簿（用Ctrl-C），再将这些文字从剪贴簿复制到「记事本」，然后储存。相反的操作，可以将一系列的MCI命令加载到TESTMCI。如果选择了一系列命令然后按下「OK」按钮（或者Enter键），则TESTMCI将每次执行一条命令。这就允许您编写MCI的「描述文件」，即MCI命令的简单列表。

例如，假设您想听歌曲「Jungleland」（唱片中的最后一首）、「Thunder Road」和「Born to Run」，并要按此顺序听，可以编写如下的描述命令：

open cdaudio

set cdaudio time format tmsf

break cdaudio on 27

```
play        cdaudio from 8 wait
play        cdaudio from 1 to 2 wait
play        cdaudio from 5 to 6 wait
stop        cdaudio
eject       cdaudio
close       cdaudio
```

不用wait关键词，就不能正常工作，因为mciSendString命令会立即传回，然后执行下一条命令。

此时，如何编写仿真CD播放程序的简单应用程序，就应该相当清楚了。程序可以确定乐曲数量、每个乐曲的长度并能显示允许使用者从任意位置开始播放（不过，请记住：mciSendString总是传回文字字符串信息，因此您需要编写解析处理程序来将这些字符串转换成数字）。可以肯定，这样的程序还要使用Windows定时器，以产生大约1秒的时间间隔。在WM_TIMER消息处理期间，程序将呼叫：

status cdaudio mode

来查看CD是暂停还是在播放。

status cdaudio position

命令允许程序更新显示以给使用者显示目前的位置。但可能还存在更令人感兴趣的事：如果程序知道音乐音调部分的节拍位置，那么就可以使屏幕上的图形与CD同步。这对于音乐指令或者建立自己的图形音乐视频程序极为有用。

波形声音

波形声音是最常用的Windows多媒体特性。波形声音设备可以通过麦克风捕捉声音，并将其转换为数值，然后把它们储存到内存或者磁盘上的波形文件中，波形文件的扩展名是.WAV。这样，声音就可以播放了。

声音与波形

在接触波形声音API之前，具备一些预备知识很重要，这些知识包括物理学、听觉以及声音进出计算机的程序。

声音就是振动。当声音改变了鼓膜上空气的压力时，我们就感觉到了声音。麦克风可以感应这些振动，并且将它们转换为电流。同样，电流再经过放大器和扩音器，就又变成了声音。传统上，声音以模拟方式储存（例如录音磁带和唱片），这些振动储存在磁气脉冲或者轮廓凹槽中。当声音转换为电流时，就可以用随时间振动的波形来表示。振动最自然的形式可以用正弦波表示，它的一个周期如图5-5所示。

正弦波有两个参数 – 振幅（也就是一个周期中的最大振幅）和频率。我们已知振幅就是音量，频率就是音调。一般来说人耳可感受的正弦波的范围是从20Hz（每秒周期）的低频声音到20,000Hz的高频声，但随着年龄的增长，对高频声音的感受能力会逐年退化。

人感受频率的能力与频率是对数关系而不是线性关系。也就是说，我们感受20Hz到40Hz的频率变化与感受40Hz到80Hz的频率变化是一样的。在音乐中，这种加倍的频率定义为八度音阶。因

此，人耳可感觉到大约10个八度音阶的声音。钢琴的范围是从27.5 Hz到4186 Hz之间，略小于7个八度音阶。

虽然正弦波代表了振动的大多数自然形式，但纯正弦波很少在现实生活中单独出现，而且，纯正弦波并不动听。大多数声音都很复杂。

任何周期的波形（即，一个循环波形）可以分解成多个正弦波，这些正弦波的频率都是整倍数。这就是所谓的Fourier级数，它以法国数学家和物理学家Jean Baptiste Joseph Fourier（1768-1830）的名字命名。周期的频率是基础。级数中其它正弦波的频率是基础频率的2倍、3倍、4倍（等等）。这些频率的声音称为泛音。基础频率也称作一级谐波。第一泛音是二级谐波，以此类推。

正弦波谐波的相对强度给每个周期的波形唯一的声音。这就是「音质」，它使得喇叭吹出喇叭声，钢琴弹出钢琴声。

人们一度认为电子合成乐器仅仅需要将声音分解成谐波并且与多个正弦波重组即可。不过，事实证明现实世界中的声音并不是这么简单。代表现实世界中声音的波形都没有严格的周期。乐器之间谐波的相对强度是不同的，并且谐波也随着每个音符的演奏时间改变。特别是乐器演奏音符的开始位置 – 我们称作起奏（attack） – 相当复杂，但这个位置又对我们感受音质至关重要。

由于近年来数字储存能力的提高，我们可以将声音直接以数字形式储存而不用复杂的重组。

脉冲编码调制（Pulse Code Modulation）

计算机处理的是数值，因此要使声音进入计算机，就必须设计一种能将声音与数字信号相互转换的机制。

不压缩数据就完成此功能的最常用方法称作「脉冲编码调制」（PCM：pulse code modulation）。PCM可用在光盘、数字式录音磁带以及Windows中。脉冲编码调制其实只是一种概念上很简单的处理步骤的奇怪代名词而已。

利用脉冲编码调制，波形可以按固定的周期频率取样，其频率通常是每秒几万次。对于每个样本都测量其波形的振幅。完成将振幅转换成数字信号工作的硬件是模拟数字转换器（ADC：analog-to-digital converter）。类似地，通过数字模拟转换器（DAC：digital-to-analog converter）可将数字信号转换回波形电子信号。但这样转换得到的波形与输入的并不完全相同。合成的波形具有由高频组成的尖锐边缘。因此，播放硬件通常在数字模拟转换器后还包括一个低通滤波器。此滤波器滤掉高频，并使合成后的波形更平滑。在输入端，低通滤波器位于ADC前面。

脉冲编码调制有两个参数：取样频率，即每秒内测量波形振幅的次数；样本大小，即用于储存振幅级的位数。与您想象的一样：取样频率越高，样本大小越大，原始声音的复制品才更好。不过，存在一个提高取样频率和样本大小的极点，超过这个极点也就超过了人类分辨声音的极限。另外，如果取样频率和样本大小过低，将导致不能精确地复制音乐以及其它声音。

取样频率

取样频率决定声音可被数字化和储存的最大频率。尤其是，取样频率必须是样本声音最高频率的两倍。这就是「Nyquist频率（Nyquist Frequency）」，以30年代研究取样程序的工程师Harry Nyquist的名字命名。

以过低的取样频率对正弦波取样时，合成的波形比最初的波形频率更低。这就是所说的失真信号。为避免失真信号的发生，在输入端使用低通滤波器以阻止频率大于半个取样频率的所有波形。在输出端，数字模拟转换器产生的粗糙的波形边缘实际上是由频率大于半个取样频率的波形组成的泛音。因此，位于输出端的低通滤波器也阻止频率大于半个取样频率的所有波形。

声音CD中使用的取样频率是每秒44,100个样本，或者称为44.1kHz。这个特有的数值是这样产生的：

人耳可听到最高20kHz的声音，因此要拦截人能听到的整个声音范围，就需要40kHz的取样频率。然而，由于低通滤波器具有频率下滑效应，所以取样频率应该再高出大约百分之十才行。现在，取样频率就达到了44kHz。这时，我们要与视频同时记录数字声音，于是取样频率就应该是美国、欧洲电视显示格速率的整数倍，这两种视频格速率分别是30Hz和25Hz。这就使取样频率升高到了44.1kHz。

取样频率为44.1kHz的光盘会产生大量的数据，这对于一些应用程序来说实在是太多了，例如对于录制声音而不是录制音乐时就是这样。把取样频率减半到22.05 kHz，可由一个10 kHz的泛音来简化复制声音的上半部分。再将其减半到11.025 kHz就向我们提供了5 kHz频率范围。44.1 kHz、22.05 kHz和11.025 kHz的取样频率，以及8 kHz都是波形声音设备普遍支持的标准。

因为钢琴的最高频率为4186 Hz，所以您可能会认为给钢琴录音时，11.025 kHz的取样频率就足够了。但4186 Hz只是钢琴最高的基础频率而已，滤掉大于5000Hz的所有正弦波将减少可被复制的泛音，而这样将不能精确地捕捉和复制钢琴的声音。

样本大小

脉冲编码调制的第二个参数是按位计算的样本大小。样本大小决定了可供录制和播放的最低音与最高音之间的区别。这就是通常所说的动态范围。

声音强度是波形振幅的平方（即每个正弦波一个周期中最大振幅的合成）。与频率一样，人对声音强度的感受也呈对数变化。

两个声音在强度上的区别是以贝尔（以电话发明人Alexander Graham Bell的名字命名）和分贝（dB）为单位进行测量的。1贝尔在声音强度上呈10倍增加。1dB就是以相同的乘法步骤成为1贝尔的十分之一。由此，1dB可增加声音强度的1.26倍（10的10次方根），或者增加波形振幅的1.12倍（10的20次方根）。1分贝是耳朵可感觉出的声强的最小变化。从开始能听到的声音极限到让人感到疼痛的声音极限之间的声强差大约是100 dB。

可用下面的公式来计算两个声音间的动态范围，单位是分贝：

$$dB = 20 \cdot \log\left(\frac{A_1}{A_2}\right)$$

其中A1和A2是两个声音的振幅。因为只可能有一个振幅，所以样本大小是1位，动态范围是0。

如果样本大小是8位，则最大振幅与最小振幅之间的比例就是256。这样，动态范围就是：

$$dB = 20 \cdot \log(256)$$

或者48分贝。48的动态范围大约相当于非常安静的房屋与电动割草机之间的差别。将样本大小加倍到16位产生的动态范围是：

$$dB = 20 \cdot \log(65536)$$

或者96分贝。这非常接近听觉极限和疼痛极限，而且人们认为这就是复制音乐的理想值。

Windows同时支持8位和16位的样本大小。储存8位的样本时，样本以无正负号字节处理，静音将储存为一个值为0x80的字符串。16位的样本以带正负号整数处理，这时静音将储存为一个值为0的字符串。

要计算未压缩声音所需的储存空间，可用以秒为单位的声音持续时间乘以取样频率。如果用16位样本而不是8位样本，则将其加倍，如果是录制立体声则再加倍。例如，1小时的CD声音（或者是在每个立体声样本占2字节、每秒44 ,100个样本的速度下进行3 600秒）需要635MB，这快要接近一张CD-ROM的储存量了。

在软件中产生正弦波

对于第一个关于波形声音的练习，我们不打算将声音储存到文件中或播放录制的声音。我们将使用低阶的波形声音API（即，前缀是waveOut的函数）来建立一个称作SINEWAVE的声音正弦波生成器。此程序以1 Hz的增量来生成从20Hz（人可感觉的最低值）到5,000Hz（与人感觉的最高值相差两个八度音阶）的正弦波。

我们知道，标准C执行时期链接库包括了一个sin函数，该函数传回一个弧度角的正弦值（2π弧度等于360度）。sin函数传回值的范围是从-1到1（早在第五章，我们就在SINEWAVE程序中使用过这个函数）。因此，应该很容易使用sin函数生成输出到波形声音硬件的正弦波数据。基本上是用代表波形（这时是正弦波）的数据来填充缓冲区，并将此缓冲区传递给API。（这比前面所讲的稍微有些复杂，但我将详细介绍）。波形声音硬件播放完缓冲区中的数据后，应将第二个缓冲区中的数据传递给它，并且以此类推。

第一次考虑这个问题（而且对PCM也一无所知）时，您大概会认为将一个周期的正弦波分成若干固定数量的样本 – 例如360个 – 才合理。对于20 Hz的正弦波，每秒输出7,200个样本。对于200 Hz的正弦波，每秒则要输出72,000个样本。这有可能实作，但实际上却不能这么做。对于5,000 Hz的正弦波，就需要每秒输出1,800,000个样本，这的确会增大DAC的负担！更重要的是，对于更高的频率，这种作法会比实际需要的精确度还高。

就脉冲编码调制而言，取样频率是个常数。假定取样频率是SINEWAVE程序中使用的11,025Hz。如果要生成一个2,756.25Hz（确切地说是四分之一的取样频率）的正弦波，则正弦波的每个周期就有4个样本。对于25Hz的正弦波，每个周期就有441个样本。通常，每周期的样本数等于取样频率除以要得到的正弦波频率。一旦知道了每周期的样本数，用2π弧度除以此数，然后用sin函数来获得每周期的样本。然后再反复对一个周期进行取样，从而建立一个连续的波形。

问题是每周期的样本数可能带有小数，因此在使用时这种方法并不是很好。每个周期的尾部都会有间断。

使它正常工作的关键是保留一个静态的「相位角」变数。此角初始化为0。第一个样本是0度正弦。随后，相位角增加一个值，该值等于2π乘以频率再除以取样频率。用此相位角作为第二个样本，并且按此方法继续。一旦相位角超过2π弧度，则减去2π弧度，而不要把相位角再初始化为0。

例如，假定要用11,025Hz的取样频率来生成1,000Hz的正弦波。即每周期有大约11个样本。为便于理解，此处相位角按度数给出 – 大约前一个半周期的相位角是：0、32.65、65.31、97.96、130.61、163.27、195.92、228.57、261.22、293.88、326.53、359.18、31.84、64.49、97.14、129.80、162.45、195.10，以此类推。存入缓冲区的波形数据是这些角度的正弦值，并已缩放到每样本的位数。为后来的缓冲区建立数据时，可继续增加最后的相位角，而不要将它初始化为0。

如程序22-2所示，FillBuffer函数完成这项工作 – 与SINEWAVE程序的其余部分一起完成。

程序22-2 SINEWAVE

SINEWAVE.C

```
/*------------------------------------------------------------------------
SINEWAVE.C -- Multimedia Windows Sine Wave Generator
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#include "resource.h"

#define SAMPLE_RATE 11025
#define FREQ_MIN 20
#define FREQ_MAX 5000
#define FREQ_INIT 440
#define OUT_BUFFER_SIZE 4096
#define PI 3.14159

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("SineWave") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
  }
  return 0 ;
}

VOID FillBuffer (PBYTE pBuffer, int iFreq)
{
  static double fAngle ;
  int i ;

  for (i = 0 ; i < OUT_BUFFER_SIZE ; i++)
  {
    pBuffer [i] = (BYTE) (127 + 127 * sin (fAngle)) ;
    fAngle += 2 * PI * iFreq / SAMPLE_RATE ;
    if ( fAngle > 2 * PI)
      fAngle -= 2 * PI ;
  }
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL bShutOff, bClosing ;
  static HWAVEOUT hWaveOut ;
  static HWND hwndScroll ;
  static int iFreq = FREQ_INIT ;
  static PBYTE pBuffer1, pBuffer2 ;
  static PWAVEHDR pWaveHdr1, pWaveHdr2 ;
  static WAVEFORMATEX waveformat ;
  int  iDummy ;

  switch (message)
  {
  case WM_INITDIALOG:
    hwndScroll = GetDlgItem (hwnd, IDC_SCROLL) ;
    SetScrollRange(hwndScroll, SB_CTL, FREQ_MIN, FREQ_MAX, FALSE) ;
    SetScrollPos (hwndScroll, SB_CTL, FREQ_INIT, TRUE) ;
    SetDlgItemInt (hwnd, IDC_TEXT, FREQ_INIT, FALSE) ;

    return TRUE ;

  case WM_HSCROLL:
    switch (LOWORD (wParam))
```

```
    {
    case SB_LINELEFT: iFreq -= 1 ; break ;
    case SB_LINERIGHT: iFreq += 1 ; break ;
    case SB_PAGELEFT: iFreq /= 2 ; break ;
    case SB_PAGERIGHT: iFreq *= 2 ; break ;

    case SB_THUMBTRACK:
      iFreq = HIWORD (wParam) ;
      break ;

    case SB_TOP:
      GetScrollRange (hwndScroll, SB_CTL, &iFreq, &iDummy) ;
      break ;

    case SB_BOTTOM:
      GetScrollRange (hwndScroll, SB_CTL, &iDummy, &iFreq) ;
      break ;
    }

    iFreq = max (FREQ_MIN, min (FREQ_MAX, iFreq)) ;
    SetScrollPos (hwndScroll, SB_CTL, iFreq, TRUE) ;
    SetDlgItemInt (hwnd, IDC_TEXT, iFreq, FALSE) ;
    return TRUE ;

    case WM_COMMAND:
      switch (LOWORD (wParam))
      {
      case IDC_ONOFF:
        // If turning on waveform, hWaveOut is NULL

        if (hWaveOut == NULL)
        {
          // Allocate memory for 2 headers and 2 buffers

          pWaveHdr1 = malloc (sizeof (WAVEHDR)) ;
          pWaveHdr2 = malloc (sizeof (WAVEHDR)) ;
          pBuffer1 = malloc (OUT_BUFFER_SIZE) ;
          pBuffer2 = malloc (OUT_BUFFER_SIZE) ;

          if (!pWaveHdr1 || !pWaveHdr2 || !pBuffer1 || !pBuffer2)
          {
            if (!pWaveHdr1) free (pWaveHdr1) ;
            if (!pWaveHdr2) free (pWaveHdr2) ;
            if (!pBuffer1) free (pBuffer1) ;
            if (!pBuffer2) free (pBuffer2) ;

            MessageBeep (MB_ICONEXCLAMATION) ;
            MessageBox (hwnd, TEXT ("Error allocating memory!"),
              szAppName, MB_ICONEXCLAMATION | MB_OK) ;
            return TRUE ;
          }

          // Variable to indicate Off button pressed

          bShutOff = FALSE ;

          // Open waveform audio for output

          waveformat.wFormatTag = WAVE_FORMAT_PCM ;
          waveformat.nChannels = 1 ;
          waveformat.nSamplesPerSec = SAMPLE_RATE ;
          waveformat.nAvgBytesPerSec = SAMPLE_RATE ;
          waveformat.nBlockAlign = 1 ;
          waveformat.wBitsPerSample = 8 ;
          waveformat.cbSize = 0 ;

          if (waveOutOpen (&hWaveOut, WAVE_MAPPER, &waveformat,
            DWORD) hwnd, 0, CALLBACK_WINDOW)!= MMSYSERR_NOERROR)
          {
```

```
          free (pWaveHdr1) ;
          free (pWaveHdr2) ;
          free (pBuffer1) ;
          free (pBuffer2) ;

          hWaveOut = NULL ;
          MessageBeep (MB_ICONEXCLAMATION) ;
          MessageBox (hwnd, TEXT ("Error opening waveform audio device!"),
            szAppName, MB_ICONEXCLAMATION | MB_OK) ;
          return TRUE ;
        }

        // Set up headers and prepare them

        pWaveHdr1->lpData = pBuffer1 ;
        pWaveHdr1->dwBufferLength = OUT_BUFFER_SIZE ;
        pWaveHdr1->dwBytesRecorded = 0 ;
        pWaveHdr1->dwUser = 0 ;
        pWaveHdr1->dwFlags = 0 ;
        pWaveHdr1->dwLoops = 1 ;
        pWaveHdr1->lpNext = NULL ;
        pWaveHdr1->reserved = 0 ;

        waveOutPrepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

        pWaveHdr2->lpData = pBuffer2 ;
        pWaveHdr2->dwBufferLength = OUT_BUFFER_SIZE ;
        pWaveHdr2->dwBytesRecorded = 0 ;
        pWaveHdr2->dwUser = 0 ;
        pWaveHdr2->dwFlags = 0 ;
        pWaveHdr2->dwLoops = 1 ;
        pWaveHdr2->lpNext = NULL ;
        pWaveHdr2->reserved = 0 ;

        waveOutPrepareHeader (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;
      }
      // If turning off waveform, reset waveform audio
      else
      {
        bShutOff = TRUE ;
        waveOutReset (hWaveOut) ;
      }
      return TRUE ;
    }
    break ;

    // Message generated from waveOutOpen call

    case MM_WOM_OPEN:
      SetDlgItemText (hwnd, IDC_ONOFF, TEXT ("Turn Off")) ;

      // Send two buffers to waveform output device

      FillBuffer (pBuffer1, iFreq) ;
      waveOutWrite (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

      FillBuffer (pBuffer2, iFreq) ;
      waveOutWrite (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;
      return TRUE ;

      // Message generated when a buffer is finished

    case MM_WOM_DONE:
      if (bShutOff)
      {
        waveOutClose (hWaveOut) ;
        return TRUE ;
      }
```

```
      // Fill and send out a new buffer

      FillBuffer (((PWAVEHDR) lParam)->lpData, iFreq) ;
      waveOutWrite (hWaveOut, (PWAVEHDR) lParam, sizeof (WAVEHDR)) ;
      return TRUE ;

    case MM_WOM_CLOSE:
      waveOutUnprepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;
      waveOutUnprepareHeader (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;

      free (pWaveHdr1) ;
      free (pWaveHdr2) ;
      free (pBuffer1) ;
      free (pBuffer2) ;

      hWaveOut = NULL ;
      SetDlgItemText (hwnd, IDC_ONOFF, TEXT ("Turn On")) ;

      if (bClosing)
        EndDialog (hwnd, 0) ;

      return TRUE ;

    case WM_SYSCOMMAND:
      switch (wParam)
      {
      case SC_CLOSE:
        if (hWaveOut != NULL)
        {
          bShutOff = TRUE ;
          bClosing = TRUE ;

          waveOutReset (hWaveOut) ;
        }
        else
          EndDialog (hwnd, 0) ;

        return TRUE ;
      }
      break ;
 }
 return FALSE ;
}
```

SINEWAVE.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
SINEWAVE DIALOG DISCARDABLE 100, 100, 200, 50
STYLE  WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Sine Wave Generator"
FONT 8, "MS Sans Serif"
BEGIN
SCROLLBAR  IDC_SCROLL,8,8,150,12
RTEXT   "440",IDC_TEXT,160,10,20,8
LTEXT   "Hz",IDC_STATIC,182,10,12,8
PUSHBUTTON  "Turn On",IDC_ONOFF,80,28,40,14
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by SineWave.rc
```

```
#define IDC_STATIC     -1
#define IDC_SCROLL    1000
#define IDC_TEXT     1001
#define IDC_ONOFF    1002
```

注意，FillBuffer例程中用到的OUT_BUFFER_SIZE、SAMPLE_RATE和PI标识符在程序的顶部定义。FillBuffer的iFreq参数是需要的频率，单位是Hz。还要注意，sin函数的结果调整到了0到254的范围之间。对于每个样本，sin函数的fAngle参数都增加一个值，该值的大小是2π弧度乘以需要的频率再除以取样频率。

SINEWAVE的窗口包含三个控件：一个用于选择频率的水平滚动条，一个用于显示目前所选频率的静态文字区域，以及一个标记为「Turn On」的按钮。按下此按钮后，您将从连结声卡的扩音器中听到正弦波的声音，同时按钮上的文字将变成「Turn Off」。用键盘或者鼠标移动滚动条可以改变频率。要关闭声音，可以再次按下按钮。

SINEWAVE程序代码初始化滚动条，以便频率在WM_INITDIALOG消息处理期间最低是20Hz，最高是5000Hz。初始化时，滚动条设定为440 Hz。用音乐术语来说就是中音上面的A，它在管弦乐队演奏时用来调音。DlgProc在接收WM_HSCROLL消息处理期间改变静态变量iFreq。注意，Page Left和Page Right将导致DlgProc增加或者减少一个八度音阶。

当DlgProc从按钮收到一个WM_COMMAND消息时，它首先配置4个内存块－2个用于WAVEHDR结构，我们马上讨论。另两个用于缓冲区储存波形数据，我们将这两个缓冲区称为pBuffer1和pBuffer2。

通过呼叫waveOutOpen函数，SINEWAVE打开波形声音设备以便输出，waveOutOpen函数使用下面的参数：

waveOutOpen (&hWaveOut, wDeviceID, &waveformat, dwCallBack,

dwCallBackData, dwFlags) ;

将第一个参数设定为指向HWAVEOUT（handle to waveform audio output：波形声音输出句柄）型态的变量。从函数传回时，此变量将设定为一个句柄，后面的波形输出呼叫中将使用该句柄。

waveOutOpen的第二个参数是设备ID。它允许函数可以在安装多个声卡的机器上使用。参数的范围在0到系统所安装的波形输出设备数之间。呼叫waveOutGetNumDevs可以获得波形输出设备数，而呼叫waveOutGetDevCaps可以找出每个波形输出设备。如果想消除设备问号，那么您可以用常数WAVE_MAPPER（定义为-1）来选择设备，该设备在「控制台」的「多媒体」中「音效」页面卷标里的「喜欢使用的设备」中指定。另外，如果首选设备不能满足您的需要，而其它设备可以，那么系统将选择其它设备。

第三个参数是指向WAVEFORMATEX结构的指针（后面将详细介绍）。第四个参数是窗口句柄或指向动态链接库中callback函数的指标，用来表示接收波形输出消息的窗口或者callback函数。使用callback函数时，可在第五个参数中指定程序定义的数据。dwFlags参数可设为CALLBACK_WINDOW或CALLBACK_FUNCTION，以表示第四个参数的型态。您也可用WAVE_FORMAT_QUERY标记来检查能否打开设备（实际上并不打开它）。还有其它几个标记可用。

waveOutOpen的第三个参数定义为指向WAVEFORMATEX型态结构的指针，此结构在MMSYSTEM.H中定义如下：

```
typedef struct waveformat_tag
{
  WORD wFormatTag ; // waveform format = WAVE_FORMAT_PCM
  WORD nChannels ; // number of channels = 1 or 2
  DWORD nSamplesPerSec ; // sample rate
```

```
  DWORD nAvgBytesPerSec ; // bytes per second
  WORD nBlockAlign ; // block alignment
  WORD wBitsPerSample ; // bits per samples = 8 or 16
  WORD cbSize ; // 0 for PCM
}
WAVEFORMATEX, * PWAVEFORMATEX ;
```

您可用此结构指定取样频率（nSamplesPerSec）和取样精确度（nBitsPerSample），以及选择单声道或立体声（nChannels）。结构中有些信息看起来是多余的，但该结构也可用于非PCM的取样方式。在非PCM取样方式下，此结构的最后一个字段设定为非0值，并带有其它信息。

对于PCM取样方式，nBlockAlign字段设定为nChannels乘以wBitsPerSample再除以8所得到的数值，它表示每次取样的总字节数。nAvgBytesPerSec字段设定为nSamplesPerSec和nBlockAlign的乘积。

SINEWAVE初始化WAVEFORMATEX结构的字段，并呼叫waveOutOpen函数：

```
waveOutOpen ( &hWaveOut, WAVE_MAPPER, &waveformat,
          (DWORD) hwnd, 0, CALLBACK_WINDOW)
```

如果呼叫成功，则waveOutOpen函数传回MMSYSERR_NOERROR（定义为0），否则传回非0的错误代码。如果waveOutOpen的传回值非0，则SINEWAVE清除窗口，并显示一个标识错误的消息框。

现在设备打开了，SINEWAVE继续初始化两个WAVEHDR结构的字段，这两个结构用于在API中传递缓冲。WAVEHDR定义如下：

```
typedef struct wavehdr_tag
{
  LPSTR lpData; // pointer to data buffer
  DWORD dwBufferLength; // length of data buffer
  DWORD dwBytesRecorded; // used for recorded
  DWORD dwUser; // for program use
  DWORD dwFlags; // flags
  DWORD dwLoops; // number of repetitions
  struct wavehdr_tag FAR *lpNext; // reserved
  DWORD reserved; // reserved
}
WAVEHDR, *PWAVEHDR ;
```

SINEWAVE将lpData字段设定为包含数据的缓冲区地址，dwBufferLength字段设定为此缓冲区的大小，dwLoops字段设定为1，其它字段都设定为0或NULL。如果要重复循环播放声音，可设定dwFlags和dwLoops字段。

SINEWAVE下一步为两个信息表头呼叫waveOutPrepareHeader函数，以防止结构和缓冲区与磁盘发生数据交换。

到此为止，所有的这些准备都是响应单击开启声音的按钮。但在程序的消息队列里已经有一个消息在等待响应。因为我们已经在函数waveOutOpen中指定要用一个窗口消息处理程序来接收波形输出消息，所以waveOutOpen函数向程序的消息队列发送了MM_WOM_OPEN消息，wParam消息参数设定为波形输出句柄。要处理MM_WOM_OPEN消息，SINEWAVE呼叫FillBuffer函数两次，并用正弦波形数据填充pBuffer缓冲区。然后SINEWAVE把两个WAVEHDR结构传送给waveOutWrite，此函数将数据传送到波形输出硬件，才真正开始播放声音。

当波形硬件播放完waveOutWrite函数传送来的数据后，就向窗口发送MM_WOM_DONE消息，其中wParam参数是波形输出句柄，lParam是指向WAVEHDR结构的指针。SINEWAVE在处理此消息时，将计算缓冲区的新数据，并呼叫waveOutWrite来重新提交缓冲区。

编写SINEWAVE程序时也可以只用一个WAVEHDR结构和一个缓冲区。不过，这样在播放完数据后将会有很短暂的停顿，以等待程序处理MM_WOM_DONE消息来提交新的缓冲区。SINEWAVE

使用的「双缓冲」技术避免了声音的不连续。

当使用者单击「Turn Off」按钮关闭声音时，DlgProc接收到另一个WM_COMMAND消息。对此消息，DlgProc把bShutOff变量设定为TRUE，并呼叫waveOutReset函数。此函数停止处理声音并发送一条MM_WOM_DONE消息。bShutOff为TRUE时，SINEWAVE透过呼叫waveOutClose来处理MM_WOM_DONE，从而产生一条MM_WOM_CLOSE消息。处理MM_WOM_CLOSE通常包括清除程序。SINEWAVE为两个WAVEHDR结构而呼叫waveOutUnprepareHeader、释放所有的内存块并把按钮上的文字改回「Turn On」。

如果硬件继续播放缓冲区的声音数据，那么它自己呼叫waveOutClose就没有作用。您必须先呼叫waveOutReset来停止播放并产生MM_WOM_DONE消息。当wParam是SC_CLOSE时，DlgProc也处理WM_SYSCOMMAND消息，这是因为使用者从系统菜单中选择了「Close」。如果波形声音继续播放，DlgProc则呼叫waveOutReset。无论如何，最后总要呼叫EndDialog来结束程序。

数位录音机

Windows提供了一个称为「录音程序」来录制和播放数字声音。程序22-3所示的程序（RECORD1）不如「录音程序」完善，因为它不含有任何文件I/O，也不允许声音编辑。然而，这个程序显示了使用低阶波形声音API来录制和回放声音的基本方法。

程序22-3 RECORD1

RECORD1.C

```
/*------------------------------------------------------------------------
RECORD1.C -- Waveform Audio Recorder
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define INP_BUFFER_SIZE 16384
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record1") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
  }
  return 0 ;
}

void ReverseMemory (BYTE * pBuffer, int iLength)
{
  BYTE b ;
  int i ;

  for (i = 0 ; i < iLength / 2 ; i++)
  {
    b = pBuffer [i] ;
    pBuffer [i] = pBuffer [iLength - i - 1] ;
    pBuffer [iLength - i - 1] = b ;
  }
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL bRecording, bPlaying, bReverse, bPaused,
    bEnding, bTerminating ;
```

```
static DWORD dwDataLength, dwRepetitions = 1 ;
static HWAVEIN hWaveIn ;
static HWAVEOUT hWaveOut ;
static PBYTE pBuffer1, pBuffer2, pSaveBuffer, pNewBuffer ;
static PWAVEHDR pWaveHdr1, pWaveHdr2 ;
static TCHAR szOpenError[] = TEXT ("Error opening waveform audio!");
static TCHAR szMemError [] = TEXT ("Error allocating memory!") ;
static WAVEFORMATEX waveform ;

switch (message)
{
case WM_INITDIALOG:
  // Allocate memory for wave header

  pWaveHdr1 = malloc (sizeof (WAVEHDR)) ;
  pWaveHdr2 = malloc (sizeof (WAVEHDR)) ;

  // Allocate memory for save buffer

  pSaveBuffer = malloc (1) ;
  return TRUE ;

case WM_COMMAND:
  switch (LOWORD (wParam))
  {
  case IDC_RECORD_BEG:
    // Allocate buffer memory

    pBuffer1 = malloc (INP_BUFFER_SIZE) ;
    pBuffer2 = malloc (INP_BUFFER_SIZE) ;

    if (!pBuffer1 || !pBuffer2)
    {
      if (pBuffer1) free (pBuffer1) ;
      if (pBuffer2) free (pBuffer2) ;

      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szMemError, szAppName,
        MB_ICONEXCLAMATION | MB_OK) ;
      return TRUE ;
    }

    // Open waveform audio for input

    waveform.wFormatTag = WAVE_FORMAT_PCM ;
    waveform.nChannels = 1 ;
    waveform.nSamplesPerSec = 11025 ;
    waveform.nAvgBytesPerSec = 11025 ;
    waveform.nBlockAlign = 1 ;
    waveform.wBitsPerSample = 8 ;
    waveform.cbSize = 0 ;

    if (waveInOpen (&hWaveIn, WAVE_MAPPER, &waveform,
      (DWORD) hwnd, 0, CALLBACK_WINDOW))
    {
      free (pBuffer1) ;
      free (pBuffer2) ;
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szOpenError, szAppName,
        MB_ICONEXCLAMATION | MB_OK) ;
    }
    // Set up headers and prepare them

    pWaveHdr1->lpData = pBuffer1 ;
    pWaveHdr1->dwBufferLength = INP_BUFFER_SIZE ;
    pWaveHdr1->dwBytesRecorded = 0 ;
    pWaveHdr1->dwUser = 0 ;
    pWaveHdr1->dwFlags = 0 ;
    pWaveHdr1->dwLoops = 1 ;
```

```
    pWaveHdr1->lpNext = NULL ;
    pWaveHdr1->reserved = 0 ;
    waveInPrepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;

    pWaveHdr2->lpData = pBuffer2 ;
    pWaveHdr2->dwBufferLength = INP_BUFFER_SIZE ;
    pWaveHdr2->dwBytesRecorded = 0 ;
    pWaveHdr2->dwUser = 0 ;
    pWaveHdr2->dwFlags = 0 ;
    pWaveHdr2->dwLoops = 1 ;
    pWaveHdr2->lpNext = NULL ;
    pWaveHdr2->reserved = 0 ;

    waveInPrepareHeader (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;
    return TRUE ;

  case IDC_RECORD_END:
    // Reset input to return last buffer

    bEnding = TRUE ;
    waveInReset (hWaveIn) ;
    return TRUE ;

  case IDC_PLAY_BEG:
    // Open waveform audio for output

    waveform.wFormatTag = WAVE_FORMAT_PCM ;
    waveform.nChannels = 1 ;
    waveform.nSamplesPerSec = 11025 ;
    waveform.nAvgBytesPerSec = 11025 ;
    waveform.nBlockAlign = 1 ;
    waveform.wBitsPerSample = 8 ;
    waveform.cbSize = 0 ;

    if (waveOutOpen (&hWaveOut, WAVE_MAPPER, &waveform,
      (DWORD) hwnd, 0, CALLBACK_WINDOW))
    {
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szOpenError, szAppName,
        MB_ICONEXCLAMATION | MB_OK) ;
    }
    return TRUE ;

  case IDC_PLAY_PAUSE:
    // Pause or restart output

    if (!bPaused)
    {
      waveOutPause (hWaveOut) ;
      SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
      bPaused = TRUE ;
    }
    else
    {
      waveOutRestart (hWaveOut) ;
      SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
      bPaused = FALSE ;
    }
    return TRUE ;

  case IDC_PLAY_END:
    // Reset output for close preparation

    bEnding = TRUE ;
    waveOutReset (hWaveOut) ;
    return TRUE ;

  case IDC_PLAY_REV:
    // Reverse save buffer and play
```

```
    bReverse = TRUE ;
    ReverseMemory (pSaveBuffer, dwDataLength) ;

    SendMessage (hwnd, WM_COMMAND, IDC_PLAY_BEG, 0) ;
    return TRUE ;

  case IDC_PLAY_REP:
    // Set infinite repetitions and play

    dwRepetitions = -1 ;
    SendMessage (hwnd, WM_COMMAND, IDC_PLAY_BEG, 0) ;
    return TRUE ;

  case IDC_PLAY_SPEED:
    // Open waveform audio for fast output

    waveform.wFormatTag = WAVE_FORMAT_PCM ;
    waveform.nChannels = 1 ;
    waveform.nSamplesPerSec =22050 ;
    waveform.nAvgBytesPerSec= 22050 ;
    waveform.nBlockAlign = 1 ;
    waveform.wBitsPerSample = 8 ;
    waveform.cbSize = 0 ;
    if (waveOutOpen (&hWaveOut, 0, &waveform, (DWORD) hwnd, 0,
      CALLBACK_WINDOW))
    {
      essageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szOpenError, szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    }
    return TRUE ;
}
break ;

case MM_WIM_OPEN:
  // Shrink down the save buffer

  pSaveBuffer = realloc (pSaveBuffer, 1) ;

  // Enable and disable buttons

  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), FALSE) ;
  SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

  // Add the buffers

  waveInAddBuffer (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
  waveInAddBuffer (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

  // Begin sampling

  bRecording = TRUE ;
  bEnding = FALSE ;
  dwDataLength = 0 ;
  waveInStart (hWaveIn) ;
  return TRUE ;

case MM_WIM_DATA:

  // Reallocate save buffer memory

  pNewBuffer = realloc (pSaveBuffer, dwDataLength +
```

```
    ((PWAVEHDR) lParam)->dwBytesRecorded) ;

  if (pNewBuffer == NULL)
  {
    waveInClose (hWaveIn) ;
    MessageBeep (MB_ICONEXCLAMATION) ;
    MessageBox (hwnd, szMemError, szAppName,
      MB_ICONEXCLAMATION | MB_OK) ;
    return TRUE ;
  }

  pSaveBuffer = pNewBuffer ;
  CopyMemory (pSaveBuffer + dwDataLength, ((PWAVEHDR) lParam)->lpData,
    ((PWAVEHDR) lParam)->dwBytesRecorded) ;

  dwDataLength += ((PWAVEHDR) lParam)->dwBytesRecorded ;

  if (bEnding)
  {
    waveInClose (hWaveIn) ;
    return TRUE ;
  }

  // Send out a new buffer

  waveInAddBuffer (hWaveIn, (PWAVEHDR) lParam, sizeof (WAVEHDR)) ;
  return TRUE ;

case MM_WIM_CLOSE:
  // Free the buffer memory

  waveInUnprepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
  waveInUnprepareHeader (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

  free (pBuffer1) ;
  free (pBuffer2) ;

  // Enable and disable buttons

  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE) ;
  SetFocus (GetDlgItem (hwnd, IDC_RECORD_BEG)) ;

  if (dwDataLength > 0)
  {
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), TRUE) ;
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;
  }
  bRecording = FALSE ;

  if (bTerminating)
    SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

  return TRUE ;

case MM_WOM_OPEN:
  // Enable and disable buttons

  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), TRUE);
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), TRUE);
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), FALSE) ;
```

```
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), FALSE) ;
  SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

  // Set up header

  pWaveHdr1->lpData = pSaveBuffer ;
  pWaveHdr1->dwBufferLength = dwDataLength ;
  pWaveHdr1->dwBytesRecorded = 0 ;
  pWaveHdr1->dwUser = 0 ;
  pWaveHdr1->dwFlags = WHDR_BEGINLOOP | WHDR_ENDLOOP ;
  pWaveHdr1->dwLoops = dwRepetitions ;
  pWaveHdr1->lpNext = NULL ;
  pWaveHdr1->reserved = 0 ;

  // Prepare and write

  waveOutPrepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;
  waveOutWrite (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

  bEnding = FALSE ;
  bPlaying = TRUE ;
  return TRUE ;

case MM_WOM_DONE:
  waveOutUnprepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;
  waveOutClose (hWaveOut) ;
  return TRUE ;

case MM_WOM_CLOSE:
  // Enable and disable buttons

  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP), TRUE) ;
  EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED), TRUE) ;
  SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

  SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
  bPaused = FALSE ;
  dwRepetitions = 1 ;
  bPlaying = FALSE ;

  if (bReverse)
  {
    ReverseMemory (pSaveBuffer, dwDataLength) ;
    bReverse = FALSE ;
  }

  if (bTerminating)
    SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

  return TRUE ;

case WM_SYSCOMMAND:
  switch (LOWORD (wParam))
  {
  case SC_CLOSE:
    if (bRecording)
    {
      bTerminating = TRUE ;
      bEnding = TRUE ;
      waveInReset (hWaveIn) ;
      return TRUE ;
    }
```

```
    if (bPlaying)
    {
      bTerminating = TRUE ;
      bEnding = TRUE ;
      waveOutReset (hWaveOut) ;
      return TRUE ;
    }

    free (pWaveHdr1) ;
    free (pWaveHdr2) ;
    free (pSaveBuffer) ;
    EndDialog (hwnd, 0) ;
    return TRUE ;
  }
  break ;
 }
 return FALSE ;
}
```

RECORD.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
RECORD  DIALOG DISCARDABLE 100, 100, 152, 74
STYLE   WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION  "Waveform Audio Recorder"
FONT 8,  "MS Sans Serif"
BEGIN
PUSHBUTTON   "Record",IDC_RECORD_BEG,28,8,40,14
PUSHBUTTON   "End",IDC_RECORD_END,76,8,40,14,WS_DISABLED
PUSHBUTTON   "Play",IDC_PLAY_BEG,8,30,40,14,WS_DISABLED
PUSHBUTTON   "Pause",IDC_PLAY_PAUSE,56,30,40,14,WS_DISABLED
PUSHBUTTON   "End",IDC_PLAY_END,104,30,40,14,WS_DISABLED
PUSHBUTTON   "Reverse",IDC_PLAY_REV,8,52,40,14,WS_DISABLED
PUSHBUTTON   "Repeat",IDC_PLAY_REP,56,52,40,14,WS_DISABLED
PUSHBUTTON   "Speedup",IDC_PLAY_SPEED,104,52,40,14,WS_DISABLED
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Record.rc
#define IDC_RECORD_BEG    1000
#define IDC_RECORD_END    1001
#define IDC_PLAY_BEG    1002
#define IDC_PLAY_PAUSE    1003
#define IDC_PLAY_END    1004
#define IDC_PLAY_REV    1005
#define IDC_PLAY_REP    1006
#define IDC_PLAY_SPEED    1007
```

RECORD.RC和RESOURCE.H文件也在RECORD2和RECORD3程序中使用。

RECORD1窗口有8个按钮。第一次执行RECORD1时，只有「Record」按钮有效。按下「Record」后，就开始录音，这时「Record」按钮无效，而「End」按钮有效。按下「End」可停止录音。这时，「Play」、「 Reverse」、「 Repeat」和「Speedup」也都有效，选择任一个按钮都可重放声音：「Play」表示正常播放；「Reverse」表示反向播放；「Repeat」表示无限的重复播放（好像循环录音带）；「Speedup」以正常速度的两倍来播放。要停止播放，您可以选择「End」按钮，而按下「Pause」按钮可停止播放。按下后，「Pause」按钮将变为「Resume」按钮，用于继续播放声音。如果要录制另一段声音，新录制的声音将替换内存里现有的声音。

任何时候，有效按钮都是可以执行有效操作的按钮。这需要在RECORD1原始码中包括对EnableWindow的多次呼叫，但是程序并不检查具体的按钮操作是否有效。显然，这使得程序操作更为直观。

RECORD1用了许多快捷方式来简化程序代码。首先，如果安装了多个波形声音硬设备，则RECORD1只使用内定设备。其次，程序按标准的11.025 kHz的取样频率和8位的取样精确度来录音和放音，而不管设备能否提供更高的取样频率和取样精确度。唯一的例外是加速功能，加速时RECORD1按22.050kHz的取样频率播放声音，这样不仅播放速度提高了一倍，而且频率也提高了一个音阶。

录制声音既包括为输入而打开波形声音硬件，还包括将缓冲区传递给API，以便接收声音数据。

RECORD1设有几个内存块。其中三个很小，至少在初始化时很小，并且在DlgProc的WM_INITDIALOG消息处理期间进行配置。程序配置两个WAVEHDR结构，分别由指针pWaveHdr1和pWaveHdr2指向。这两个结构用于将缓冲区传递给波形API。pSaveBuffer指标指向储存整个录音的缓冲区，最初配置时只有一个字节。然后，随着录音的进行，该缓冲区不断增大，以适应所有的声音数据（如果录音时间过长，则RECORD1能够在录制程序中及时发现内存溢出，并允许您重放成功储存的声音）。由于这个缓冲区用来储存堆积的声音数据，所以我将其称为「储存缓冲区（save buffer）」。指标pBuffer1和pBuffer2指向的另外两个内存块，大小是16K，它们在记录接收的声音数据时配置。录音结束后释放这些内存块。

8个按钮中的每一个都向REPORT1窗口的对话程序DlgProc产生WM_COMMAND消息。最初只有「Record」按钮有效。按下此按钮将产生WM_COMMAND消息，其中wParam参数等于IDC_RECORD_BEG。为处理这个消息，RECORD1配置两个16K的缓冲区来接收声音数据，初始化WAVEFORMATEX结构的字段，并将此结构传递给waveInOpen函数，然后设定两个WAVEHDR结构。

waveInOpen函数产生一条MM_WIM_OPEN消息。在此消息处理期间，RECORD1把储存缓冲区的大小缩减到1个字节，以准备接收数据（当然，第一次录音时，储存缓冲区的大小就是1个字节，但以后录制时，就可能大多了）。在MM_WIM_OPEN消息处理期间，RECORD1也将适当的按钮设定为有效和无效。然后，程序用waveInAddBuffer把两个WAVEHDR结构和缓冲区传送给API。这时会设定某些标记，然后呼叫waveInStart开始录音。

采用11.025kHz的取样频率和8位的取样精确度时，16K的缓冲区可储存大约1.5秒的声音。这时，RECORD1接收MM_WIM_DATA消息。在响应此消息处理期间，程序将根据变量dwDataLength和WAVEHDR结构中的字段dwBytesRecorded对缓冲区重新配置。如果配置失败，RECORD1呼叫waveInClose来停止录音。

如果重新配置成功，则RECORD1把16K缓冲区里的数据复制到储存缓冲区，然后再次呼叫waveInAddBuffer。此程序将持续到RECORD1用完储存缓冲区的内存，或使用者按下「End」按钮为止。

「End」按钮产生WM_COMMAND消息，其中wParam等于IDC_RECORD_END。处理这个消息很简单，RECORD1把bEnding标记设定为TRUE并呼叫waveInReset。waveInReset函数使录音停止，并产生MM_WIM_DATA消息，该消息含有部分填充的缓冲区。除了呼叫waveInClose来关闭波形输入设备外，RECORD1对这个消息正常响应。

waveInClose产生MM_WIM_CLOSE消息。RECORD1响应此消息时，释放16K输入缓冲区，并使相应的按钮有效或无效。尤其是，当储存缓冲区里存有数据（除非第一次配置就失败，否则一般都含有数据）时，播放按钮将有效。

录音以后，储存缓冲区里将含有这些声音数据。当使用者选择「Play」按钮时，DlgProc就接收一个WM_COMMAND消息，其中wParam等于IDC_PLAY_BEG。响应时，程序将初始化WAVEFORMATEX结构的字段，并呼叫waveOutOpen。

waveOutOpen呼叫再次产生MM_WOM_OPEN消息，在此消息处理期间，RECORD1把相应的按钮设为有效或无效（只允许使用「Pause」和「End」），用储存缓冲区来初始化WAVEHDR结构的字段，呼叫waveOutPrepareHeader来准备要播放的声音，然后呼叫waveOutWrite开始播放。

一般情况下，直到播放完储存缓冲区里的所有数据才停止。这时产生MM_WOM_DONE 消息。如果还有缓冲区要播放，则程序会在这时将它们传递给API。由于RECORD1只播放一个大缓冲区，因此程序不再简单地准备标题，而是呼叫waveOutClose。waveOutClose函数产生MM_WOM_CLOSE消息。在此消息处理期间，RECORD1使相应的按钮有效或无效，并允许声音再次播放或者录制新声音。

程序中还有一个「End」按钮，利用此按钮，使用者可以在播放完储存缓冲区之前的任何时刻停止播放。「End」按钮产生一个WM_COMMAND消息，其中wParam等于IDC_PLAY_END，响应时，程序呼叫waveOutReset，此函数产生一条正常处理的MM_WOM_DONE消息。

RECORD1的窗口中还包括一个「Pause」按钮。处理此按钮很简单：第一次按时下，RECORD1呼叫waveOutPause来暂停播放，并将按钮上的文字改为「Resume」。按下「Resume」按钮时，通过呼叫waveOutRestart来继续播放。

为了使程序更有趣，窗口中还包括另外三个按钮：「Reverse」、「Repeat」和「Speedup」。这些按钮都产生WM_COMMAND消息，其中wParam的值分别等于IDC_PLAY_REV、IDC_PLAY_REP和IDC_PLAY_SPEED。

倒放声音就是把储存缓冲区里的数据按字节顺序反向，然后再正常播放。RECORD1中有一个称为ReverseMemory的小函数使字节反向。在WM_COMMAND消息处理期间，程序在播放块之前呼叫此函数，并在MM_WOM_CLOSE消息的后期再次呼叫此函数，以便将其恢复到正常状态。

「Repeat」按钮将往复不停地播放声音。由于API支持重复播放声音，所以这并不复杂。只要将WAVEHDR结构的dwLoops字段设为重复次数，将dwFlags字段设为WHDR_BEGINLOOP和WHDR_ENDLOOP，分别表示循环时缓冲区的开始部分和结束部分。因为RECORD1只使用一个缓冲区来播放声音，所以这两个标记组合到了dwFlags字段。

要实作两倍速播放也很容易。在准备为输出而打开波形声音期间，初始化WAVEFORMATEX结构的字段时，只需将nSamplesPerSec和nAvgBytesPerSec字段设定为22050，而不是11025。

另一种MCI界面

您可能已经发现，RECORD1很复杂。特别是在处理波形声音函数呼叫和它们产生的消息间的交互时，更复杂。处理可能出现的内存不足的情况也是如此。但这也许正是它称为低阶界面的原因。我在本章的前面提到过，Windows也提供高阶媒体控制接口（Media Control Interface）。

对波形声音来说，低阶接口与MCI之间的主要区别在于MCI用波形文件记录声音数据，并通过读取文件来播放声音。由于在播放声音之前要读取文件、处理文件然后再写入文件，所以让RECODE1来实作「特殊效果」很困难。这是典型的折衷选择问题：功能齐全或是使用方便？低阶接口很灵活，但MCI（其中的大部分）更方便。

MCI有两种不同但又相关的实作形式。一种形式用消息和数据结构将命令发送给多媒体设备，然后再从那里接收信息。另一种形式使用ASCII文字字符串。建立文字命令的接口最初是为了让多

媒体设备接受简单的描述命令语言的控制。但它也提供非常容易的交谈式控制，请参见本章前面，TESTMCI程序的展示。

RECORD2程序，如程序22-4所示，使用MCI形式的消息和数据结构来实作另一个数字声音录音机和播放器。虽然它使用的对话框模板与RECORD1一样，但并没有实作三个特殊效果的按钮。

程序22-4 RECORD2

RECORD2.C

```
/*------------------------------------------------------------------------
RECORD2.C -- Waveform Audio Recorder
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\record1\\resource.h"

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
  }
  return 0 ;
}

void ShowError (HWND hwnd, DWORD dwError)
{
  TCHAR szErrorStr [1024] ;
  mciGetErrorString (dwError, szErrorStr, sizeof (szErrorStr) / sizeof (TCHAR)) ;
  MessageBeep (MB_ICONEXCLAMATION) ;
  MessageBox (hwnd, szErrorStr, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL bRecording, bPlaying, bPaused ;
  static TCHAR szFileName[] = TEXT ("record2.wav") ;
  static WORD wDeviceID ;
  DWORD dwError ;
  MCI_GENERIC_PARMS mciGeneric ;
  MCI_OPEN_PARMS mciOpen ;
  MCI_PLAY_PARMS mciPlay ;
  MCI_RECORD_PARMS mciRecord ;
  MCI_SAVE_PARMS mciSave ;

  switch (message)
  {
  case WM_COMMAND:
    switch (wParam)
    {
    case IDC_RECORD_BEG:
      // Delete existing waveform file

      DeleteFile (szFileName) ;

      // Open waveform audio

      mciOpen.dwCallback = 0 ;
      mciOpen.wDeviceID = 0 ;
      mciOpen.lpstrDeviceType = TEXT ("waveaudio") ;
      mciOpen.lpstrElementName = TEXT ("") ;
      mciOpen.lpstrAlias = NULL ;
      dwError = mciSendCommand (0, MCI_OPEN,
```

```
      MCI_WAIT | MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
      (DWORD) (LPMCI_OPEN_PARMS) &mciOpen) ;
    if (dwError != 0)
    {
      ShowError (hwnd, dwError) ;
      return TRUE ;
    }
    // Save the Device ID

    wDeviceID = mciOpen.wDeviceID ;

    // Begin recording

    mciRecord.dwCallback = (DWORD) hwnd ;
    mciRecord.dwFrom = 0 ;
    mciRecord.dwTo = 0 ;

    mciSendCommand (wDeviceID, MCI_RECORD, MCI_NOTIFY,
      (DWORD) (LPMCI_RECORD_PARMS) &mciRecord) ;

    // Enable and disable buttons

    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
    SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

    bRecording = TRUE ;
    return TRUE ;

  case IDC_RECORD_END:
    // Stop recording

    mciGeneric.dwCallback = 0 ;

    mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT,
      (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

    // Save the file

    mciSave.dwCallback = 0 ;
    mciSave.lpfilename = szFileName ;

    mciSendCommand (wDeviceID, MCI_SAVE, MCI_WAIT | MCI_SAVE_FILE,
      (DWORD) (LPMCI_SAVE_PARMS) &mciSave) ;

    // Close the waveform device

    mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT,
      (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

    // Enable and disable buttons

    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

    bRecording = FALSE ;
    return TRUE ;

  case IDC_PLAY_BEG:
    // Open waveform audio

    mciOpen.dwCallback = 0 ;
```

```
    mciOpen.wDeviceID = 0 ;
    mciOpen.lpstrDeviceType = NULL ;
    mciOpen.lpstrElementName = szFileName ;
    mciOpen.lpstrAlias = NULL ;

    dwError = mciSendCommand ( 0, MCI_OPEN,
      MCI_WAIT | MCI_OPEN_ELEMENT,
      (DWORD) (LPMCI_OPEN_PARMS) &mciOpen) ;

    if (dwError != 0)
    {
      ShowError (hwnd, dwError) ;
      return TRUE ;
    }
    // Save the Device ID

    wDeviceID = mciOpen.wDeviceID ;

    // Begin playing

    mciPlay.dwCallback = (DWORD) hwnd ;
    mciPlay.dwFrom = 0 ;
    mciPlay.dwTo = 0 ;

    mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY,
      (DWORD) (LPMCI_PLAY_PARMS) &mciPlay) ;

    // Enable and disable buttons

    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), TRUE) ;
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

    bPlaying = TRUE ;
    return TRUE ;

  case IDC_PLAY_PAUSE:
    if (!bPaused)
      // Pause the play
    {
      mciGeneric.dwCallback = 0 ;

      mciSendCommand (wDeviceID, MCI_PAUSE, MCI_WAIT,
        (DWORD) (LPMCI_GENERIC_PARMS) & mciGeneric);

      SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
      Paused = TRUE ;
    }
    else
      // Begin playing again
    {
      mciPlay.dwCallback = (DWORD) hwnd ;
      mciPlay.dwFrom = 0 ;
      mciPlay.dwTo = 0 ;

      mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY,
        (DWORD) (LPMCI_PLAY_PARMS) &mciPlay) ;

      SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
      bPaused = FALSE ;
    }

    return TRUE ;

  case IDC_PLAY_END:
    // Stop and close
```

```
    mciGenericParms.dwCallback = 0 ;

    mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT,
      (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms) ;

    mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT,
      (DWORD) (LPMCI_GENERIC_PARMS) &mciGenericParms) ;

    // Enable and disable buttons

    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

    bPlaying = FALSE ;
    bPaused = FALSE ;
    return TRUE ;
}
break ;

case MM_MCINOTIFY:
  switch (wParam)
  {
  case MCI_NOTIFY_SUCCESSFUL:
    if (bPlaying)
      SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0) ;

    if (bRecording)
      SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0);

    return TRUE ;
  }
  break ;

  case WM_SYSCOMMAND:
    switch (wParam)
    {
    case SC_CLOSE:
      if (bRecording)
        SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0L) ;
      if (bPlaying)
        SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0L) ;

      EndDialog (hwnd, 0) ;
      return TRUE ;
    }
    break ;
}
return FALSE ;
}
```

RECORD2只使用两个MCI函数呼叫，其中最重要的呼叫如下所示：

error = mciSendCommand (wDeviceID, message, dwFlags, dwParam)

第一个参数是设备的识别数字（ID），您可以按句柄来使用ID。打开设备时就可以获得ID，并在随后的mciSendCommand呼叫中使用。第二个参数是前缀为MCI的常数。这些称为MCI命令消息，RECORD2展示了其中的七个：MCI_OPEN、MCI_RECORD、MCI_STOP、MCI_SAVE、MCI_PLAY、MCI_PAUSE和MCI_CLOSE。

dwFlags参数通常由0或者多个位旗标常数（由C的位OR运算子合成）组成。这些通常用来表示不同的选项。一些选项是某个命令消息所特有的，而另一些对所有的消息都是通用的。dwParam

参数通常是指向一个数据结构的长指针，该结构表示选项以及由设备获得的信息。许多MCI消息都与数据结构有关，而且这些数据结构对于消息来说都是唯一的。

如果mciSendCommand函数呼叫成功，则传回0值，否则传回错误代码。要向使用者报告此错误，可用下面的函数获得描述错误的文字字符串：

mciGetErrorString (error, szBuffer, dwLength)

此函数在程序TESTMCI中也用到过。

按下「Record」按钮后，RECORD2的窗口消息处理程序就收到一个WM_COMMAND消息，其中wParam等于IDC_RECORD_BEG。RECORD2从打开设备开始，包括设定MCI_OPEN_PARMS结构的字段，并用MCI_OPEN命令消息呼叫mciSendCommand。录音时，lpstrDeviceType字段设定为字符串「waveaudio」以说明设备型态，lpstrElementName字段设定为长度为0的字符串。MCI驱动程序使用内定的取样频率和取样精确度，但是您可以用MCI_SET命令进行修改。录音程序中，声音数据先储存在硬盘上的临时文件中，最后再转化成标准的波形文件。本章的后面将介绍波形文件的格式。播放录制的声音时，MCI使用波形文件中定义的取样频率和取样精确度。

如果RECORD2不能打开设备，则用mciGetErrorString和MessageBox提示错误信息。否则从mciSendCommand呼叫传回，MCI_OPEN_PARMS结构的wDeviceID字段包含有设备ID，以供后面的呼叫使用。

要开始录音，RECORD2就呼叫mciSendCommand，以MCI_RECORD命令消息和MCI_WAVE_RECORD_PARMS数据结构为参数。当然，您也可以将此结构（并使用表示这些字段已设定的位旗标）的dwFromz和dwTo字段进行设定，以便将声音插入现有的波形文件，其文件名在MCI_OPEN_PARMS结构的lpstrElementName字段指定。内定状态下，任何新的声音都插入在现有文件的开始位置。

RECORD2将MCI_WAVE_RECORD_PARMS结构的dwCallback字段设定为程序的窗口句柄，并在mciSendCommand呼叫中包含MCI_NOTIFY标记。这导致录音结束后向窗口消息处理程序发送一条通知消息。我将简要讨论一下这条通知消息。

录音结束后，按下前一个「End」按钮来停止录音，这时产生一个WM_COMMAND消息，其中wParam等于IDC_RECORD_END。响应时，窗口消息处理程序将呼叫mciSendCommand三次：MCI_STOP命令消息用于停止录音；MCI_SAVE命令消息用于把临时文件中的声音数据传递到MCI_SAVE_PARMS结构中指定的文件（「record2.wav」）；MCI_CLOSE命令消息用于删除所有的临时文件、释放已经建立的内存块并关闭设备。

播放时，MCI_OPEN_PARMS结构的lpstrElementName字段设定为文件名「record2.wav」。mciSendCommand第三个参数中所包含的MCI_OPEN_ELEMENT标记表示lpstrElementName字段是一个有效的文件名。通过文件的扩展名称.WAV，MCI知道使用者要打开一个波形声音设备。如果存在多个波形硬件，则打开第一个（设定MCI_OPEN_PARMS结构的lpstrDeviceType字段，也可以打开其它波形设备）。

播放将包括带有MCI_PLAY命令消息和MCI_PLAY_PARMS结构的mciSendCommand呼叫。虽然波形文件的任意部分都可以播放，但RECORD2只播放整个文件。

RECORD2还包括一个「Pause」按钮来暂停播放声音文件。这个按钮产生一个WM_COMMAND消息，其中wParam等于IDC_PLAY_PAUSE。响应时，程序将呼叫mciSendCommand，并以MCI_PAUSE命令消息和MCI_GENERIC_PARMS结构作为参数。MCI_GENERIC_PARMS结构用于这样一些消息：它们除了需要用于通知的可选窗口句柄外，不需要任何信息。如果播放已经暂停，则通过再次使用MCI_PLAY命令消息呼叫mciSendCommand

继续播放。

按下第二个「End」按钮也可以停止播放。这时产生wParam等于IDC_PLAY_END的WM_COMMAND消息。响应时，窗口消息处理程序将呼叫mciSendCommand两次：第一次使用MCI_STOP命令消息；第二次使用MCI_CLOSE命令消息。

现在有一个问题：虽然可以通过按下「End」按钮来手工终止播放，但您可能需要播放整个文件。程序如何知道文件播放完的时间呢？这是MCI通知消息的任务。

当带有MCI_RECORD和MCI_PLAY消息来呼叫mciSendCommand时，RECORD2将包括MCI_NOTIFY标记，并将数据结构的dwCallback字段设定为程序窗口句柄。这样就产生一个通知消息，称为MM_MCINOTIFY，并在某些环境下传递给窗口消息处理程序。消息参数wParam是一个状态代码，而lParam是设备ID。

带有MCI_STOP或者MCI_PAUSE命令消息来呼叫mciSendCommand时，您将接收到一个MM_MCINOTIFY消息，其中wParam等于MCI_NOTIFY_ABORTED。当您按下「Pause」按钮或者两个「End」按钮中的一个时，就会出现这种情况。由于对这些按钮已进行过适当的处理，所以RECORD2可以忽略这种情况。播放时，您会在声音文件结束后接收到MM_MCINOTIFY消息，其中wParam等于MCI_NOTIFY_SUCCESSFUL。这种情况下，窗口消息处理程序给自己发送一个WM_COMMAND消息，其中wParam等于IDC_PLAY_END，来仿真使用者按下「End」按钮。然后窗口消息处理程序作出正常响应：停止播放，关闭设备。

录音时，如果用于储存临时文件的硬盘空间不够，您就会接收一个MM_MCINOTIFY消息，其中wParam等于MCI_NOTIFY_SUCCESSFUL（虽然现在还不能说它很完美，但其功能已经很齐全了）。响应时，窗口消息处理程序给自己发送一个WM_COMMAND消息，其中wParam等于IDC_RECORD_END，然后与正常情况下一样：停止录音、储存文件并关闭设备。

MCI命令字符串的方法

Windows的多媒体接口曾经包含函数mciExecute，其语法如下：

bSuccess = mciExecute (szCommand) ;

其中唯一的参数是MCI命令字符串。函数传回布尔值－如果呼叫成功，则传回非0值，否则传回0。在功能上，mciExecute函数相同于呼叫后三个参数为NULL或0的mciSendString（TESTMCI中使用的依据字符串的MCI函数），然后在发生错误时呼叫mciGetErrorString和MessageBox。

虽然mciExecute不再是API的一部分，但我还是在RECORD3版的数字录音机中使用了这个函数。和RECORD2一样，RECORD3程序也使用RECORD1中的资源描述文件RECORD.RC和RESOURCE.H，如程序22-5所示。

程序22-5 RECORD3

RECORD3.C

```
/*------------------------------------------------------------------------
RECORD3.C -- Waveform Audio Recorder
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include "..\\record1\\resource.h"

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
```

```
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
  }
  return 0 ;
}

BOOL mciExecute (LPCTSTR szCommand)
{
  MCIERROR error ;
  TCHAR szErrorStr [1024] ;

  if (error = mciSendString (szCommand, NULL, 0, NULL))
  {
    mciGetErrorString (error, szErrorStr, sizeof (szErrorStr) / sizeof (TCHAR)) ;
    MessageBeep (MB_ICONEXCLAMATION) ;
    MessageBox ( NULL, szErrorStr, TEXT ("MCI Error"),
      MB_OK | MB_ICONEXCLAMATION) ;
  }
  return error == 0 ;
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL bRecording, bPlaying, bPaused ;
  switch (message)
  {
  case WM_COMMAND:
    switch (wParam)
    {
    case IDC_RECORD_BEG:
      // Delete existing waveform file

      DeleteFile (TEXT ("record3.wav")) ;

      // Open waveform audio and record

      if (!mciExecute (TEXT ("open new type waveaudio alias mysound")))
        return TRUE ;

      mciExecute (TEXT ("record mysound")) ;

      // Enable and disable buttons

      EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
      EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
      EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE);
      EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
      EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
      SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

      bRecording = TRUE ;
      return TRUE ;

    case IDC_RECORD_END:
      // Stop, save, and close recording

      mciExecute (TEXT ("stop mysound")) ;
      mciExecute (TEXT ("save mysound record3.wav")) ;
      mciExecute (TEXT ("close mysound")) ;

      // Enable and disable buttons

      EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
      EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
      EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
      EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
      EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
      SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;
```

```
    bRecording = FALSE ;
    return TRUE ;

  case IDC_PLAY_BEG:
    // Open waveform audio and play

    if (!mciExecute (TEXT ("open record3.wav alias mysound")))
      return TRUE ;

    mciExecute (TEXT ("play mysound")) ;

    // Enable and disable buttons

    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), TRUE) ;
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

    bPlaying = TRUE ;
    return TRUE ;

  case IDC_PLAY_PAUSE:
    if (!bPaused)
      // Pause the play
    {
      mciExecute (TEXT ("pause mysound")) ;
      SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
      bPaused = TRUE ;
    }
    else
      // Begin playing again
    {
      mciExecute (TEXT ("play mysound")) ;
      SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
      bPaused = FALSE ;
    }

    return TRUE ;

  case IDC_PLAY_END:
    // Stop and close

    mciExecute (TEXT ("stop mysound")) ;
    mciExecute (TEXT ("close mysound")) ;

    // Enable and disable buttons
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG), TRUE) ;
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END), FALSE);
    SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

    bPlaying = FALSE ;
    bPaused = FALSE ;
    return TRUE ;
  }
  break ;

case WM_SYSCOMMAND:
  switch (wParam)
  {
  case SC_CLOSE:
    if (bRecording)
      SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0L);
```

```
    if (bPlaying)
      SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0L) ;

    EndDialog (hwnd, 0) ;
    return TRUE ;
  }
  break ;
 }
 return FALSE ;
}
```

在研究消息导向和文字导向的MCI接口时，您会发现它们非常相近。很容易就可以猜测出MCI将命令字符串转换为相应的命令消息和数据结构。RECORD3可以使用像RECORD2一样使用MM_MCINOTIFY消息，但是它没有选择mciExecute函数的好处，它的缺点是程序不知道什么时候播放完波形文件。因此，这些按钮不能自动改变状态。您必须人工按下「End」按钮，以便让程序知道它已经准备再次录音或播放。

注意MCI的open命令中alias关键词的用法。它允许所有后来的MCI命令使用别名来引用设备。

波形声音文件格式

如果在十六进制转储程序下研究未压缩的.WAV文件（即PCM），您会发现它们具有表22-1所示的格式。

表22-1 .WAV文件格式

偏移量	字节	资料
0000	4	「RIFF」
0004	4	波形块的大小（文件大小减8）
0008	4	「WAVE」
000C	4	「fmt 」
0010	4	格式块的大小（16字节）
0014	2	wf.wFormatTag = WAVE_FORMAT_PCM = 1
0016	2	wf.nChannels
0018	4	wf.nSamplesPerSec
001C	4	wf.nAvgBytesPerSec
0020	2	wf.nBlockAlign
0022	2	wf.wBitsPerSample
0024	4	「data」
0028	4	波形资料的大小
002C		波形资料

这是一种扩充自RIFF（Resource Interchange File Format：资源交换文件格式）的格式。RIFF是用于多媒体数据文件的万用格式，它是一种标记文件格式。在这种格式下，文件由数据「块」组成，而这些数据块则由前面4个字符的ASCII名称和4字节（32位）的数据块大小来确认。数据块大小值不包括名称和大小所需要的8字节。

波形声音文件以文字字符串「RIFF」开始，用来标识这是一个RIFF文件。字符串后面是一个32位的数据块大小，表示文件其余部分的大小，或者是小于8字节的文件大小。

数据块以文字字符串「WAVE」开始，用来标识这是一个波形声音块，后面是文字字符串「fmt」－注意用空白使之成为4字符的字符串－用来标识包含波形声音数据格式的子数据块。「fmt」字符

串的后面是格式信息大小，这里是16字节。格式信息是WAVEFORMATEX结构的前16个字节，或者，像最初定义时一样，是包含WAVEFORMAT结构的PCMWAVEFORMAT结构。

nChannels字段的值是1或2，分别对应于单声道和立体声。nSamplesPerSec字段是每秒的样本数；标准值是每秒11,025、22,050和44 100个样本。nAvgBytesPerSec字段是取样速率，单位是每秒样本数乘以信道数，再乘以以位为单位的每个样本的大小，然后除以8并往上取整数。标准样本大小是8位和16位。nBlockAlign字段是信道数乘以以位为单位的样本大小，然后除以8并往上取整数。最后，该格式以wBitsPerSample字段结束，该字段是信道数乘以以位为单位的样本大小。

格式信息的后面是文字字符串「data」，然后是32位的数据大小，最后是波形数据本身。这些数据是按相同格式进行简单连结的样本，这与低阶波形声音设备上所使用的格式相同。如果样本大小是8位，或者更少，那么每个样本有1字节用于单声道，或者有2字节用于立体声。如果样本大小在9到16位之间，则每个样本就有2字节用于单声道，或者4字节用于立体声。对于立体声波形数据，每个样本都由左值及其后面的右值组成。

对于8位或不到8位的样本大小，样本字节被解释为无正负号值。例如，对于8位的样本大小，静音等于0x80字节的字符串。对于9位或更多的样本大小，样本被解释为有正负号值，这时静音的字符串等于值0。

用于读取标记文件的一个重要规则是忽略不准备处理的数据块。尽管波形声音文件需要「fmt」和「data」子数据块（按照此顺序），但它还包含其它子数据块。尤其是，波形声音文件可能包含一个标记为「INFO」的子数据块，和提供波形声音文件信息的子数据块的子数据块。

迭加合成实验

许多年来 – 至少从毕达哥拉斯的年代起 – 人们就已经试图分析音调。起初好像非常简单，但随后就变得复杂了。抱歉，我将重复一些已经说过的有关声音的问题。

音调，除了一些撞击声以外，都有特殊的音调或频率。这个频率可以在人类能够感受到的频谱范围内，也就是从20Hz到20,000Hz以内。例如，钢琴的频率范围在27.5Hz到4186Hz之间。音调的另一个特征是音量或响度。这与产生音调的波形的所有振幅相对应。响度的变化用分贝度量。迄今为止，一切都很好。

然后有一件难办的事称做「音质」。非常简单，音质就是声音的性质，利用它，我们可以区分按相同音调相同音量演奏的钢琴、小提琴和喇叭。

法国数学家Fourier发现一些周期性的波形 – 不论多么复杂 – 它们都可以表示为许多频率是基础频率整数倍的正弦波形。这个基础频率，也称作第一个谐波，是波形周期的频率。第一个泛音，也称作二级谐波，是基本频率的两倍；第二个泛音，或者三级谐波的频率是基本频率的三倍，依次类推。谐波振幅的相互关系形成了波形的形状。

例如，方波可以表示为许多的正弦波，其中偶数谐波（即2、4、6等等）的振幅都是0，而奇数谐波（即1、3、5等等）的振幅都按1、1/3、1/5比例依次类推。在锯齿波中，所有的泛音都出现，而振幅都按1、1/2、1/3、1/4比例依此类推。

对于德国科学家Hermann Helmholtz（1821-1894），这是了解音质的关键。在他的名著《On the Sensations of Tone》（1885年，1954年由Dover Press再版）中，Helmholtz假定耳朵和大脑将复杂的声音分解为正弦波，而这些正弦波相关的强度就是我们所感受的音质。不幸的是，事情还没有这么简单。

随着1968年Wendy Carlos的唱片《Switched on Bach》的发布，电子音乐合成器引起了公众的广泛注意。那时使用的合成器（例如Moog）是模拟合成器。这些合成器使用模拟电路来产生

各种声音波形，例如方波、三角波形和锯齿波形。要使这些波形听起来更像真实的乐器，它们取决于单个音符的变化程序。波形的所有振幅以「包络（envelope）」形成。当音符开始时，振幅由0开始增加，通常增加非常快。这就是所谓的起奏。然后当音符持续时，振幅保持为常数，这时称为持续。音符结束时，振幅降为0，这时称为释放。

波形通过滤波器，滤波器将削弱一些谐波，并将简单波形转换得更复杂、更有乐感。这些滤波器的切断频率由包络控制，以便声音的谐波内容在音符的程序中改变。

因为这些合成器以丰富的波形格式调和开始，而且一些谐波通过滤波器进行了削弱，这种形式的合成称为「负合成」。

即使在负合成期间，许多人也还会在电子音乐中发现迭加合成是下一个大问题。

在迭加合成中，您可以从许多整数倍正弦波生成器开始，选择整数倍以便于每个正弦波都对应一个谐波。每个谐波的振幅都由一个包络单独控制。使用模拟电路的迭加合成不实用，因为对单个音符就需要8和24之间数目的正弦波生成器，而与这些正弦波生成器相关的频率必须精确的互相对齐。模拟波形生成器稳定性很差，而且容易发生频率漂移。

不过，由数字合成器（可以数字化地使用对照表产生波形）和计算机产生的波形，频率漂移并不是个问题，因而迭加合成也就切实可行了。因此总的来说：在录制真实的乐曲时，可以用Fourier分解法将其分解成多个谐波。然后就可以确定每个谐波的相对强度，再用多个正弦波数字化地产生声音。

如果开始实验时用Fourier分析法分析实际的音调，并从多个正弦波来产生这些音调，那么人们将发现音质并不像Helmholtz所认为的那样简单。

最大的问题是真实音调的谐波之间并没有精确的整数关系。事实上，「谐波」一词对于实际的音调来说并不十分适当。各种正弦波组成都不和谐，或者更准确地说是「泛音」。

人们发现，实际音调泛音之间的不和谐在创造「真实的」声音时很重要。静态和谐会产生「电流」声。每个泛音都在单个音符上改变振幅和频率。泛音中，相对频率和振幅的关系对于不同的泛音以及来自相同乐器的不同强度是不同的。实际音调中最复杂的部分发生在音符的起奏部分，这时比较不和谐。人们发现音符的这个复杂的起奏位置对于人类感受音质很重要。

简而言之，实际乐器的声音比任何想象的都更复杂。分析音调的观点，以及后面用于控制泛音的振幅和频率的相对简单的包络观点显然都不实用。

实际乐曲的一些分析法发表于早期（1977到1978年间）的《Computer Music Journal》（当时由People's Computer Company发行，现在由MIT Press发行）由James A. Moorer、John Grey和John Strawn Some编写了第三部分丛书《Lexicon of Analyzed Tones》，该书显示了在小提琴、双簧管、单簧管和喇叭上演奏一个音符（小于半秒种）的泛音的振幅和频率图形。所用的音符是中音C上的降E。小提琴用20个泛音，双簧管和单簧管用21个，而喇叭用12个。实际上，《Computer Music Journal》的Volume II、Number 2（1978年9月）包含了用线段来近似双簧管、单簧管和喇叭的不同频率和振幅的包络。

因此，利用Windows上支持的声音波形功能，下面的程序很简单：将这些数字键入程序、为每个泛音都产生多个正弦波样本、添加这些样本并将其发送给波形声音声卡，因此把20年前原始录制的声音重新制造出来也很容易。ADDSYNTH（「迭加合成」）如程序22-6所示。

程序22-6 ADDSYNTH

ADDSYNTH.C

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

```
ADDSYNTH.C -- Additive Synthesis Sound Generation
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <math.h>
#include "addsynth.h"
#include "resource.h"

#define ID_TIMER 1
#define SAMPLE_RATE 22050
#define MAX_PARTIALS 21
#define PI 3.14159

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("AddSynth") ;
// Sine wave generator
// -------------------

double SineGenerator (double dFreq, double * pdAngle)
{
  double dAmp ;
  dAmp = sin (* pdAngle) ;
  * pdAngle += 2 * PI * dFreq / SAMPLE_RATE ;

  if (* pdAngle >= 2 * PI)
    * pdAngle -= 2 * PI ;

  return dAmp ;
}

// Fill a buffer with composite waveform
// -------------------------------------

VOID FillBuffer (INS ins, PBYTE pBuffer, int iNumSamples)
{
  static double dAngle [MAX_PARTIALS] ;
  double dAmp, dFrq, dComp, dFrac ;
  int i, iPrt, iMsecTime, iCompMaxAmp, iMaxAmp, iSmp ;
  // Calculate the composite maximum amplitude

  iCompMaxAmp = 0 ;
  for (iPrt = 0 ; iPrt < ins.iNumPartials ; iPrt++)
  {
    iMaxAmp = 0 ;
    for (i = 0 ; i < ins.pprt[iPrt].iNumAmp ; i++)
      iMaxAmp = max (iMaxAmp, ins.pprt[iPrt].pEnvAmp[i].iValue) ;
    iCompMaxAmp += iMaxAmp ;
  }

  // Loop through each sample
  for (iSmp = 0 ; iSmp < iNumSamples ; iSmp++)
  {
    dComp = 0 ;
    iMsecTime = (int) (1000 * iSmp / SAMPLE_RATE) ;

    // Loop through each partial
    for (iPrt = 0 ; iPrt < ins.iNumPartials ; iPrt++)
    {
      dAmp = 0 ;
      dFrq = 0 ;

      for (i = 0 ; i < ins.pprt[iPrt].iNumAmp - 1 ; i++)
      {
        if (iMsecTime >= ins.pprt[iPrt].pEnvAmp[i ].iTime &&
          iMsecTime <= ins.pprt[iPrt].pEnvAmp[i+1].iTime)
        {
          dFrac = (double) (iMsecTime -
            ins.pprt[iPrt].pEnvAmp[i ].iTime) /
            (ins.pprt[iPrt].pEnvAmp[i+1].iTime -
```

```
          ins.pprt[iPrt].pEnvAmp[i ].iTime) ;

        dAmp = dFrac * ins.pprt[iPrt].pEnvAmp[i+1].iValue +
          (1-dFrac) * ins.pprt[iPrt].pEnvAmp[i ].iValue ;
        break ;
       }
     }

     for (i = 0 ; i < ins.pprt[iPrt].iNumFrq - 1 ; i++)
     {
       if (iMsecTime >= ins.pprt[iPrt].pEnvFrq[i ].iTime &&
         iMsecTime <= ins.pprt[iPrt].pEnvFrq[i+1].iTime)
       {
         dFrac = (double) (iMsecTime -ins.pprt[iPrt].pEnvFrq[i ].iTime) /
           (ins.pprt[iPrt].pEnvFrq[i+1].iTime -
           ins.pprt[iPrt].pEnvFrq[i ].iTime) ;
         dFrq = dFrac * ins.pprt[iPrt].pEnvFrq[i+1].iValue + (1-dFrac) *
           ins.pprt[iPrt].pEnvFrq[i ].iValue ;
         break ;
       }
     }
     dComp += dAmp * SineGenerator (dFrq, dAngle + iPrt) ;
    }
    pBuffer[iSmp] = (BYTE) (127 + 127 * dComp / iCompMaxAmp) ;
  }
}

// Make a waveform file
// --------------------------------------------------------------------

BOOL MakeWaveFile (INS ins, TCHAR * szFileName)
{
  DWORD dwWritten ;
  HANDLE hFile ;
  int iChunkSize, iPcmSize, iNumSamples ;
  PBYTE pBuffer ;
  WAVEFORMATEX waveform ;

  hFile = CreateFile (szFileName, GENERIC_WRITE, 0, NULL,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;
  if (hFile == NULL)
    return FALSE ;
  iNumSamples = ((long) ins.iMsecTime * SAMPLE_RATE / 1000 + 1) / 2 * 2 ;
  iPcmSize = sizeof (PCMWAVEFORMAT) ;
  iChunkSize = 12 + iPcmSize + 8 + iNumSamples ;

  if (NULL == (pBuffer = malloc (iNumSamples)))
  {
    CloseHandle (hFile) ;
    return FALSE ;
  }

  FillBuffer (ins, pBuffer, iNumSamples) ;
  waveform.wFormatTag = WAVE_FORMAT_PCM ;
  waveform.nChannels = 1 ;
  waveform.nSamplesPerSec = SAMPLE_RATE ;
  waveform.nAvgBytesPerSec = SAMPLE_RATE ;
  waveform.nBlockAlign = 1 ;
  waveform.wBitsPerSample = 8 ;
  waveform.cbSize = 0 ;

  WriteFile (hFile, "RIFF", 4, &dwWritten, NULL) ;
  WriteFile (hFile, &iChunkSize, 4, &dwWritten, NULL) ;
  WriteFile (hFile, "WAVEfmt ", 8, &dwWritten, NULL) ;
  WriteFile (hFile, &iPcmSize, 4, &dwWritten, NULL) ;
  WriteFile (hFile, &waveform, sizeof (WAVEFORMATEX) - 2, &dwWritten, NULL) ;
  WriteFile (hFile, "data", 4, &dwWritten, NULL) ;
  WriteFile (hFile, &iNumSamples, 4, &dwWritten, NULL) ;
  WriteFile (hFile, pBuffer, iNumSamples, &dwWritten, NULL) ;
```

```
  CloseHandle (hFile) ;
  free (pBuffer) ;

  if ((int) dwWritten != iNumSamples)
  {
    DeleteFile (szFileName) ;
    return FALSE ;
  }
  return TRUE ;
}

void TestAndCreateFile ( HWND hwnd, INS ins, TCHAR * szFileName,
                    int idButton)
{
  TCHAR szMessage [64] ;
  if (-1 != GetFileAttributes (szFileName))
    EnableWindow (GetDlgItem (hwnd, idButton), TRUE) ;
  else
  {
    if (MakeWaveFile (ins, szFileName))
      EnableWindow (GetDlgItem (hwnd, idButton), TRUE) ;
    else
    {
      wsprintf (szMessage, TEXT ("Could not create %x."), szFileName) ;
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szMessage, szAppName,
        MB_OK | MB_ICONEXCLAMATION) ;
    }
  }
}

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
  }
  return 0 ;
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static TCHAR * szTrum = TEXT ("Trumpet.wav") ;
  static TCHAR * szOboe = TEXT ("Oboe.wav") ;
  static TCHAR * szClar = TEXT ("Clarinet.wav") ;

  switch (message)
  {
  case WM_INITDIALOG:
    SetTimer (hwnd, ID_TIMER, 1, NULL) ;
    return TRUE ;

  case WM_TIMER:
    KillTimer (hwnd, ID_TIMER) ;
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    TestAndCreateFile (hwnd, insTrum, szTrum, IDC_TRUMPET) ;
    TestAndCreateFile (hwnd, insOboe, szOboe, IDC_OBOE) ;
    TestAndCreateFile (hwnd, insClar, szClar, IDC_CLARINET) ;

    SetDlgItemText (hwnd, IDC_TEXT, TEXT (" ")) ;
    SetFocus (GetDlgItem (hwnd, IDC_TRUMPET)) ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
```

```
    return TRUE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDC_TRUMPET:
      PlaySound (szTrum, NULL, SND_FILENAME | SND_SYNC) ;
      return TRUE ;

    case IDC_OBOE:
      PlaySound (szOboe, NULL, SND_FILENAME | SND_SYNC) ;
      return TRUE ;

    case IDC_CLARINET:
      PlaySound (szClar, NULL, SND_FILENAME |SND_SYNC) ;
      return TRUE ;
    }
    break ;

    case WM_SYSCOMMAND:
      switch (LOWORD (wParam))
      {
      case SC_CLOSE:
        EndDialog (hwnd, 0) ;
        return TRUE ;
      }
      break ;
  }
  return FALSE ;
}
```

ADDSYNTH.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Dialog
ADDSYNTH DIALOG DISCARDABLE 100, 100, 176, 49
STYLE  WS_MINIMIZEBOX | WS_CAPTION | WS_SYSMENU
CAPTION "Additive Synthesis"
FONT 8,  "MS Sans Serif"
BEGIN
PUSHBUTTON  "Trumpet",IDC_TRUMPET,8,8,48,16
PUSHBUTTON  "Oboe",IDC_OBOE,64,8,48,16
PUSHBUTTON  "Clarinet",IDC_CLARINET,120,8,48,16
LTEXT  "Preparing Data...",IDC_TEXT,8,32,100,8
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by AddSynth.rc
#define  IDC_TRUMPET     1000
#define  IDC_OBOE     1001
#define  IDC_CLARINET     1002
#define  IDC_TEXT     1003
```

这里没有给出附加文件ADDSYNTH.H，因为它包含几百行令人讨厌的叙述，您将在本书附上的光盘上找到它。在ADDSYNTH.H的开始位置，我定义了三个结构，用于储存包络数据。每个振幅和频率分别储存到型态ENV的结构数组中。这些数字对由时间（毫秒）和振幅值（按任意度量单位）或频率（以周期/秒为单位）组成。这些数组的长度可变，其变化范围从6到14。假定振幅和频率值之间直接相关。

每种乐器都包括一个泛音集（喇叭用12个，双簧管和单簧管分别使用21个），这些泛音集储存在型态PRT的结构数组中。PRT结构储存振幅和频率包络的点数，以及指向ENV数组的指针。INS结构包括音调的总时间（以毫秒为单位）、泛音数以及指向储存泛音的PRT数组的指针。

ADDSYNTH有三个标记为「Trumpet」、「Oboe」和「Clarinet」的按钮。PC的速度还没有快到足以实时计算所有的迭加合成，因此第一次执行ADDSYNTH时，这些按钮将失效，直到程序计算完样本并建立了TRUMPET.WAV、OBOE.WAV和CLARINET.WAV声音文件后，按钮才启动，而且可以使用PlaySound函数播放这三种声音。下次执行时，程序将检查波形文件是否存在，而不需重新建立。

ADDSYNTH中的FillBuffer函数完成了大多数工作。FillBuffer从计算合成最大振幅的总数开始。为此，它在乐器的泛音中循环，以找出每个泛音的最大振幅，然后将所有的最大振幅加起来。此值后来用于将样本缩放到8位的样本大小。

然后FillBuffer计算每个样本的值。每个样本都对应于一段以毫秒为单位的时间，该时间取决于取样频率（实际上，在22.05 kHz的取样频率下，每22个样本对应于相同的毫秒时间值）。然后，FillBuffer在泛音中循环。对于频率和振幅，它找出与毫秒时间值对应的包络线段，并执行线性插补。

频率值与相位角值一起传递给SineGenerator函数。本章前面讨论过，产生数字化的正弦波形需要保持相位角值，并依据频率值增加。从SineGenerator函数传回时，正弦值将乘以泛音的振幅并累加。样本的所有泛音都加在起来之后，样本就缩放到字节大小。

起床号波形声音

WAKEUP，如程序22-7所示，是原始码文件看起来不是很完整的程序之一。程序窗口看起来像对话框，但是没有资源描述文件（我们已经知道如何编写），并且程序使用一个波形档案，但在光盘上却没有这样的档案。不过，程序非常有趣：它播放的声音很大，并且非常令人讨厌。WAKEUP是我的闹钟，能够唤醒我继续工作。

程序22-7 WAKEUP

WAKEUP.C

```
/*------------------------------------------------------------------------
WAKEUP.C -- Alarm Clock Program
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <commctrl.h>

// ID values for 3 child windows
#define ID_TIMEPICK 0
#define ID_CHECKBOX 1
#define ID_PUSHBTN 2

// Timer ID

#define ID_TIMER 1

// Number of 100-nanosecond increments (ie FILETIME ticks) in an hour
#define FTTICKSPERHOUR (60 * 60 * (LONGLONG) 10000000)
// Defines and structure for waveform "file"
#define SAMPRATE 11025
#define NUMSAMPS (3 * SAMPRATE)
#define HALFSAMPS (NUMSAMPS / 2)

typedef struct
{
  char chRiff[4] ;
  DWORD dwRiffSize ;
```

```
  char chWave[4] ;
  char chFmt [4] ;
  DWORD dwFmtSize ;
  PCMWAVEFORMAT pwf ;
  char chData[4] ;
  DWORD dwDataSize ;
  BYTE byData[0] ;
}
WAVEFORM ;
// The window proc and the subclass proc
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT CALLBACK SubProc (HWND, UINT, WPARAM, LPARAM) ;

// Original window procedure addresses for the subclassed windows

WNDPROC SubbedProc [3] ;

// The current child window with the input focus

HWND hwndFocus ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInst,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName [] = TEXT ("WakeUp") ;
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  wndclass.style = 0 ;
  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 = (HBRUSH) (1 + COLOR_BTNFACE) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, szAppName,
    WS_OVERLAPPED | WS_CAPTION |
    WS_SYSMENU | WS_MINIMIZEBOX,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static HWND hwndDTP, hwndCheck, hwndPush ;
  static WAVEFORM waveform = { "RIFF", NUMSAMPS + 0x24,
```

```
  "WAVE", "fmt ",
  sizeof (PCMWAVEFORMAT), 1, 1, SAMPRATE,
  SAMPRATE, 1, 8, "data", NUMSAMPS } ;
static WAVEFORM * pwaveform ;
FILETIME ft ;
HINSTANCE hInstance ;
INITCOMMONCONTROLSEX icex ;
int i, cxChar, cyChar ;
LARGE_INTEGER li ;
SYSTEMTIME st ;

switch (message)
{
case WM_CREATE:
  // Some initialization stuff

  hInstance = (HINSTANCE) GetWindowLong (hwnd, GWL_HINSTANCE) ;

  icex.dwSize = sizeof (icex) ;
  icex.dwICC = ICC_DATE_CLASSES ;
  InitCommonControlsEx (&icex) ;

  // Create the waveform file with alternating square waves

  pwaveform = malloc (sizeof (WAVEFORM) + NUMSAMPS) ;
  * pwaveform = waveform ;

  for (i = 0 ; i < HALFSAMPS ; i++)
    if (i % 600 < 300)
      if (i % 16 < 8)
        pwaveform->byData[i] = 25 ;
      else
        pwaveform->byData[i] = 230 ;
      else
        if (i % 8 < 4)
          pwaveform->byData[i] = 25 ;
        else
          pwaveform->byData[i] = 230 ;
        // Get character size and set a fixed window size.
        cxChar = LOWORD (GetDialogBaseUnits ()) ;
        cyChar = HIWORD (GetDialogBaseUnits ()) ;

        SetWindowPos (hwnd, NULL, 0, 0, 42 * cxChar, 10 * cyChar / 3 + 2 *
          GetSystemMetrics (SM_CYBORDER) +GetSystemMetrics (SM_CYCAPTION)
          ,SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE) ;

        // Create the three child windows

        hwndDTP = CreateWindow (DATETIMEPICK_CLASS, TEXT (""),
          WS_BORDER | WS_CHILD | WS_VISIBLE | DTS_TIMEFORMAT,
          2 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
          hwnd, (HMENU) ID_TIMEPICK, hInstance, NULL) ;
        hwndCheck = CreateWindow (TEXT ("Button"), TEXT ("Set Alarm"),
          WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
          16 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
          hwnd, (HMENU) ID_CHECKBOX, hInstance, NULL) ;

        hwndPush = CreateWindow (TEXT ("Button"), TEXT ("Turn Off"),
          WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON | WS_DISABLED,
          28 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
          hwnd, (HMENU) ID_PUSHBTN, hInstance, NULL) ;

        hwndFocus = hwndDTP ;

        // Subclass the three child windows

        SubbedProc [ID_TIMEPICK] = (WNDPROC)
          SetWindowLong (hwndDTP, GWL_WNDPROC, (LONG) SubProc) ;
        SubbedProc [ID_CHECKBOX] = (WNDPROC)
```

```
          SetWindowLong (hwndCheck, GWL_WNDPROC, (LONG) SubProc);
        SubbedProc [ID_PUSHBTN] = (WNDPROC)
          SetWindowLong (hwndPush, GWL_WNDPROC, (LONG) SubProc) ;

        // Set the date and time picker control to the current time
        // plus 9 hours, rounded down to next lowest hour

        GetLocalTime (&st) ;
        SystemTimeToFileTime (&st, &ft) ;
        li = * (LARGE_INTEGER *) &ft ;
        li.QuadPart += 9 * FTTICKSPERHOUR ;
        ft = * (FILETIME *) &li ;
        FileTimeToSystemTime (&ft, &st) ;
        st.wMinute = st.wSecond = st.wMilliseconds = 0 ;
        SendMessage (hwndDTP, DTM_SETSYSTEMTIME, 0, (LPARAM) &st) ;
        return 0 ;

case WM_SETFOCUS:
  SetFocus (hwndFocus) ;
  return 0 ;

case WM_COMMAND:
  switch (LOWORD (wParam)) // control ID
  {
  case ID_CHECKBOX:

    // When the user checks the "Set Alarm" button, get the
    // time in the date and time control and subtract from
    // it the current PC time.

    if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
    {
      SendMessage (hwndDTP, DTM_GETSYSTEMTIME, 0, (LPARAM) &st) ;
      SystemTimeToFileTime (&st, &ft) ;
      li = * (LARGE_INTEGER *) &ft ;

      GetLocalTime (&st) ;
      SystemTimeToFileTime (&st, &ft) ;
      li.QuadPart -= ((LARGE_INTEGER *) &ft)->QuadPart ;

      // Make sure the time is between 0 and 24 hours!
      // These little adjustments let us completely ignore
      // the date part of the SYSTEMTIME structures.

      while ( li.QuadPart < 0)
        li.QuadPart += 24 * FTTICKSPERHOUR ;

      li.QuadPart %= 24 * FTTICKSPERHOUR ;

      // Set a one-shot timer! (See you in the morning.)

      SetTimer (hwnd, ID_TIMER, (int) (li.QuadPart / 10000), 0) ;
    }
    // If button is being unchecked, kill the timer.

    else
      KillTimer (hwnd, ID_TIMER) ;

    return 0 ;

    // The "Turn Off" button turns off the ringing alarm, and also
    // unchecks the "Set Alarm" button and disables itself.

  case ID_PUSHBTN:
    PlaySound (NULL, NULL, 0) ;
    SendMessage (hwndCheck, BM_SETCHECK, 0, 0) ;
    EnableWindow (hwndDTP, TRUE) ;
    EnableWindow (hwndCheck, TRUE) ;
    EnableWindow (hwndPush, FALSE) ;
```

```
      SetFocus (hwndDTP) ;
      return 0 ;
    }
    return 0 ;

    // The WM_NOTIFY message comes from the date and time picker.
    // If the user has checked "Set Alarm" and then gone back to
    // change the alarm time, there might be a discrepancy between
    // the displayed time and the one-shot timer. So the program
    // unchecks "Set Alarm" and kills any outstanding timer.

    case WM_NOTIFY:
      switch (wParam) // control ID
      {
      case ID_TIMEPICK:
        switch (((NMHDR *) lParam)->code) // notification code
        {
        case DTN_DATETIMECHANGE:
          if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
          {
            KillTimer (hwnd, ID_TIMER) ;
            SendMessage (hwndCheck, BM_SETCHECK, 0, 0) ;
          }
          return 0 ;
        }
      }
      return 0 ;

      // The WM_COMMAND message comes from the two buttons.

      case WM_TIMER:

        // When the timer message comes, kill the timer (because we only
        // want a one-shot) and start the annoying alarm noise going.

        KillTimer ( hwnd, ID_TIMER) ;
        PlaySound ( (PTSTR) pwaveform, NULL,
          SND_MEMORY | SND_LOOP | SND_ASYNC);

        // Let the sleepy user turn off the timer by slapping the
        // space bar. If the window is minimized, it's restored; then
        // it's brought to the forefront; then the pushbutton is enabled
        // and given the input focus.

        EnableWindow (hwndDTP, FALSE) ;
        EnableWindow (hwndCheck, FALSE) ;
        EnableWindow (hwndPush, TRUE) ;

        hwndFocus = hwndPush ;
        ShowWindow (hwnd, SW_RESTORE) ;
        SetForegroundWindow (hwnd) ;
        return 0 ;

        // Clean up if the alarm is ringing or the timer is still set.

      case WM_DESTROY:
        free (pwaveform) ;

        if (IsWindowEnabled (hwndPush))
          PlaySound (NULL, NULL, 0) ;

        if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
          KillTimer (hwnd, ID_TIMER) ;

        PostQuitMessage (0) ;
        return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

```
LRESULT CALLBACK SubProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  int idNext, id = GetWindowLong (hwnd, GWL_ID) ;
  switch (message)
  {
  case WM_CHAR:
    if (wParam == '\t')
    {
      idNext = id ;

      do
      idNext = (idNext +
        (GetKeyState (VK_SHIFT) < 0 ? 2 : 1)) % 3 ;
      while (!IsWindowEnabled (GetDlgItem (GetParent (hwnd), idNext)));

      SetFocus (GetDlgItem (GetParent (hwnd), idNext)) ;
      return 0 ;
    }
    break ;

  case WM_SETFOCUS:
    hwndFocus = hwnd ;
    break ;
  }
  return CallWindowProc ( SubbedProc [id], hwnd, message, wParam,lParam) ;
}
```

WAKEUP使用的波形只有两个方波，但是变化迅速。实际的波形在WndProc的WM_CREATE消息处理期间计算。所有的波形文件都储存在内存中。指向这个内存块的指针传递给PlaySound函数，该函数使用SND_MEMORY、SND_LOOP和SND_ASYNC参数。

WAKEUP使用称为「Date-Time Picker」的通用控件。这个控件用来让使用者选择指定的日期和时间（WAKEUP只使用时间挑选功能）。程序可以使用SYSTEMTIME结构来获得和设定时间，在获得和设定PC自身时钟时也使用该结构。要多方面了解Date-Time Picker，请试着建立不带有任何DTS样式旗标的窗口。

注意WM_CREATE消息结束时的处理方式：程序假定您在睡觉之前执行它，并希望它在8小时之后来唤醒您。

现在很明显，可以从GetLocalTime函数在SYSTEMTIME结构获得目前时间，而且可以「手工」增加时间。但在一般情况下，此计算将涉及检查大于24小时的结果时间，这意味着您必须增加天数字段，然后可能涉及增加月（因此还必须有用于每月天数和闰年检查的逻辑），最后您可能还要增加年。

事实上，推荐的方法（来自/Platform SDK/Windows Base Services/General Library/Time/Time Reference/Time Structures/SYSTEMTIME）是将SYSTEMTIME转换为FILETIME结构（使用SystemTimeToFileTime），将FILETIME结构强制转换为LARGE_INTEGER结构，在大整数上执行计算，再强制转换回FILETIME结构，然后转换回SYSTEMTIME结构（使用FileTimeToSystemTime）。

顾名思义，FILETIME结构用于获得和设定文件最后一次更新的时间。此结构如下：

```
type struct _FILETIME  // ft
{
  DWORD dwLowDateTime ;
  DWORD dwHighDateTime ;
}
FILETIME ;
```

这两个字段一起表示了从1601年1月1日起每隔1000亿分之一秒所显示的64位值。

Microsoft C/C++编译器支持64位整数作为ANSI C的非标准延伸语法。数据型态是__int64。您可以对__int64型态执行所有的常规算术运算，并且有一些执行时期链接库函数也支持它们。Windows的WINNT.H表头文件定义如下：

typedef __int64 LONGLONG ;

typedef unsigned __int64 DWORDLONG ;

在Windows中，这有时称为「四字组」，或者更普遍地称为「大整数」。也有一个union定义如下：

```
typedef union _LARGE_INTEGER
{
  struct
  {
    DWORD LowPart ;
    LONG HighPart ;
  } ;
  LONGLONG QuadPart ;
}
LARGE_INTEGER ;
```

这是/Platform SDK/Windows Base Services/General Library/Large Integer Operations中的全部文件。此union允许您使用32位或者64位的大整数。

MIDI 和音乐

由电子音乐合成器制造者协会在19世纪80年代早期开发了「乐器数字化接口」(MIDI：Musical Instrument Digital Interface)。MIDI是用于将它们中的电子乐器与计算机连结起来的协议，也是电子音乐领域中相当重要的标准。MIDI规范由MIDI Manufacturers Association（MMA）维护，它的网站是http://www.midi.org。

使用MIDI

MIDI为透过电缆来传递数字命令定义了传输协议。MIDI电缆使用5针DIN接头，但是只使用了三个接头。一个是屏蔽，一个是回路，而第三个传输数据。MIDI协议在每秒31,250位的速度下是单向的。数据的每个字节都由一个开始位开始，以一个停止位结束，用于每秒3,125字节的有效传输速率。

重要的是要了解真实的声音 – 不论是模拟格式还是数字格式 – 不是经由MIDI电缆传输的。通过电缆传输的通常都是简单的命令消息，长度一般是1、2或3字节。

简单的MIDI设定可以包括两片MIDI兼容硬件。一个是本身不发声，但是单独产生MIDI消息的MIDI键盘。此键盘有一个有标记有「MIDI Out」的MIDI端口。用MIDI电缆将这个埠与MIDI声音合成器的「MIDI In」埠连结起来。合成器看起来很像前面有几个按钮的小盒子。

按下键盘上的一个键时（假定是中音C)，键盘就将3个字节发送给MIDI Out端口。在十六进制中，这些字节是：

90 3C 40

第一个字节（90）显示Note On消息。第二个字节是键号，其中3C是中音C。第三个字节是敲按键的速度，此速度范围是从1到127。我们恰巧使用了一个对速度不敏感的键盘，因此它发送平均速度值。这个3字节的消息顺着MIDI电缆进入合成器的Midi In埠。通过播放中音C的音调来响应合成器。

释放键时，键盘会将另一个3字节消息发送给MIDI Out端口：

90 3C 00

这与Note On命令相同，但带有0速字节。这个字节值0表示Note Off命令，意味着应该关闭音符。合成器通过停止声音来响应。

如果合成器有复调音乐的能力（即，同时播放多个音符的能力），那么您就可以在键盘上演奏和弦。键盘产生多条Note On消息，并且合成器将播放所有的音符。当您释放和弦时，键盘就将多条Note Off消息发送给合成器。

一般来说，这种设定中的键盘称为「MIDI控制器」，它负责产生MIDI消息来控制合成器。MIDI控制器看起来不像键盘。MIDI控制器包括下面几种：看起来像单簧管或萨克斯管的MID管乐控制器、MIDI吉他控制器、MIDI弦乐控制器和MIDI鼓控制器。至少所有这些控制器都产生3字节的Note On和Note Off消息。

胜过类似的键盘或传统乐器，控制器也可以是「编曲器」，它是在内存中储存Note On和Note Off消息顺序，然后再播放的硬件。现在单机编曲器已经比几年前少见多了，因为它们已经被计算机所替代。安装MIDI卡的计算机也可以生成Note On和Note Off消息来控制合成器。MIDI编辑软件，允许您在屏幕上作曲，还可以储存来自MIDI控制器的MIDI消息，并处理这些消息，然后将MIDI消息发送给合成器。

合成器有时也称为「声音模块（sound module）」或「音源器（tone generator）」。MIDI不指定如何真正产生这些声音的方法。合成器可以使用任何一种声音生成技术。

实际上，只有非常简单的MIDI控制器（例如管乐控制器）才只有MIDI Out电缆埠。通常键盘都有内建合成器，并且有三个MIDI电缆端口，分别标记为「MIDI In」、「MIDI Out」和「MIDI Thru」。MIDI In端口接受MIDI消息，从而播放键盘的内部合成器。MIDI Out端口将MIDI消息从键盘发送到外部合成器。MIDI Thru埠是一个输出埠，它复制MIDI In端口的输入信号 – 无论从MIDI In埠获得什么都发送给MIDI Thru埠（MIDI Thru埠不包括从MIDI Out埠发送的任何信息）。

透过电缆连结MIDI硬件只有两种方法：将一个硬件上的MIDI Out连结到另一个的MIDI In，或者将MIDI Thru与MIDI In连结。MIDI Thru端口允许连结一系列的MIDI合成器。

程序更改

合成器能制作哪种声音？是钢琴声、小提琴声、喇叭声还是飞碟声？通常合成器能够生成的各种声音都储存在ROM或者其它地方。它们通常称为「声音」、「乐器」或者「音色」。（「音色」一词来自模拟合成器的时代，当时通过将音色和弦插入合成器前面的插孔中来设定不同的声音）。

在MIDI中，合成器能够生成的各种声音称为「程序」。改变这个程序需要向合成器发送MIDI Program Change消息

C0 pp

其中，pp的范围是0到127。通常MIDI键盘的顶部是一系列有限的按钮，这些按钮将产生Program Change消息。透过按下这些按钮，您可以从键盘控制合成器的声音。这些按钮号通常由1开始，而不是由0开始，因此程序句柄1与Program Change字节的0对应。

MIDI规格没有说明程序句柄与乐器的对应关系。例如，著名的Yamaha DX7合成器上的前三个程序分别称为「Warm Strings」、「 Mellow Horn」和「Pick Guitar」。而在Yamaha TX81Z音调发生器上，它们是Grand Piano、Upright Piano和Deep Grand。在Roland MT-32声音模块上，它们是Acoustic Piano 1、Acoustic Piano 2和Acoustic Piano 3。因此，如果不希望在从键盘制作程序改变时感到吃惊，那么最好了解一下乐器声与您将使用的合成器的程序句柄的对应关系。

这对于包含Program Change消息的MIDI文件来说是一个实际问题 – 这些文件并不是设备无关的，因为它们的内容在不同的合成器上听起来是不一样的。然而，在最近几年，「General MIDI」（GM）标准已经把这些程序句柄标准化。Windows支援General MIDI。如果合成器与General MIDI规格不一致，那么程序转换可使它仿真General MIDI合成器。

MIDI通道

迄今为止，我已经讨论了两条MIDI消息，第一条是Note On：

90 kk vv

其中，kk是键号 (0到127)，v v是速度 (0到127)。0速度表示Note Off命令。第二条是Program Change：

C0 pp

其中，pp的范围是从0到127。这些是典型的MIDI消息。第一个字节称作「状态」字节。根据字节的状态，它通常后跟0、1或2字节的「数据」(我即将说明的「系统专有」消息除外)。从数据字节中分辨出状态字节很容易：高位总是1用于状态字节，0用于数据字节。

然而，我还没有讨论过这两个消息的普通格式。Note On消息的普通格式如下：

9n kk vv

而Program Change是：

Cn pp

在这两种情况下，n表示状态字节的低四位，其变化范围是0到15。这就是MIDI「通道」。通道一般从1开始编号，因此，如果n为0，则代表通道1。

使用16个不同通道允许一条MIDI电缆传输16种不同声音的消息。通常，您将发现MIDI消息的特殊字符串以Program Change消息开始，为所用的不同信道设定声音，而字符串的后面是多条Note On和Note Off命令。再后面可能是其它的Program Change命令。但任何时候，每个通道都只与一种声音联系。

让我们作一个简单范例：假定我已经讨论过的键盘控制能够同时产生用于两条不同信道 – 信道1和信道2 – 的MIDI消息。透过按下键盘上的按钮将两条Program Change消息发送给合成器：

C0 01

C1 05

现在设定信道1用于程序2，并设定信道2用于程序6（回忆信道句柄和程序句柄都是基于1的，但消息中的编码是基于0的)。现在按下键盘上的键时，就发送两条Note On消息，一条用于一个通道：

90 kk vv

91 kk vv

这就允许您和谐地同时播放两种乐器的声音。

另一种方法是「分开」键盘。低键可以在信道1上产生Note On消息，高键可以在信道2上产生Note On消息。这就允许您在一个键盘上独立播放两种乐器的声音。

当您考虑PC上的MIDI编曲软件时，使用16个通道将更为有利。每个通道都代表不同的乐器。如果有能够独立播放16种不同乐器的合成器，那么您就可以编写用于16个波段的管弦乐曲，而且只使用一条MIDI电缆将MIDI卡与合成器连结起来。

MIDI消息

尽管Note On和Program Change消息在任何MIDI执行中都是最重要的消息，但并不是所有的MIDI都可以执行。表22-2是MIDI规格中定义的MIDI信道消息表。我在前面提到过，状态字节的高位总是设定着，而状态字节后面的数据字节的高位都等于0。这意味着状态字节的范围是0x80到0xFF，而数据字节的范围是0到0x7F。

表22-2 MIDI信道消息（n =信道句柄，从0到15）

MIDI消息	数据字节	值
Note Off	8n kk vv	kk = 键号（0-127） vv = 速度（0-127）
Note On	9n kk vv	kk = 键号（0-127） vv = 速度（1-127，0 = note off）
Polyphonic After Touch	An kk tt	kk = 键号（0-127） tt = 按下之后 （0-127）
Control Change	Bn cc xx	cc = 控制器（0-121） xx = 值（0-127）
Channel Mode Local Control	Bn 7A xx	xx = 0（关），127（开）
All Notes Off	Bn 7B 00	
Omni Mode Off	Bn 7C 00	
Omni Mode On	Bn 7D 00	
Mono Mode On	Bn 7E cc	cc = 频道数
Poly Mode On	Bn 7F 00	
Program Change	Cn pp	pp = 程序（0-127）
Channel After Touch	Dn tt	tt = 按下之后（0-127）
Pitch Wheel Change	En ll hh	ll = 低7位（0-127） hh = 高7位（0-127）

虽然没有严格的要求，键号通常还是与西方音乐的传统音符相对应（例如，对于打击声音，每个键号码可以是不同的打击乐器）。当键号与钢琴类的键盘对应时，键60（十进制）是中音C。MIDI键号在普通的88键钢琴范围的基础上向下扩展了21个音符，向上扩展了19个音符。速度句柄是按下某键的速度，在钢琴上它控制声音的响度与和谐特征。特殊的声音可以依这种方式或其它方式来响应键的速度。

前面展示的例子使用带有0速度字节的Note On消息来表示Note Off命令。对于键盘（或者其它控制器）还有一个单独的Note Off命令，该命令实作释放键的速度，不过，非常少见。

还有两个「接触后」消息。接触后是一些键盘的特征，按下某个键以后，再用力按下键可以在某些方式上改变声音。一个消息（状态字节0xDn）是将接触后应用于通道中目前演奏的所有音符，这是最常见的。状态字节0xAn表示独立应用每个单独键的接触后。

通常，键盘上都有一些用于进一步控制声音的刻度盘或开关。这些设备称为「控制器」，所有

变化都由状态字节0xBn表示。通过从0到121的号码确认控制器。0xBn状态字节也用于Channel Mode消息，这些消息显示了合成器如何在通道中响应同时发生的音符。

一个非常重要的控制器是上下转换音调的轮，它有一个单独的MIDI消息，其状态字节是0xEn。

表22-2中所缺少的是状态字节以从F0到FF开始的消息。这些消息称为系统消息，因为它们适用于整个MIDI系统，而不是部分通道。系统消息通常用于同步的目的、触发编曲器、重新设定硬件以及获得信息。

许多MIDI控制器连续发送状态字节0xFE，该字节称为Active Sensing消息。这简单地表示了MIDI控制器仍依附于系统。

一条重要的系统消息是以状态字节0xF0开始的「系统专用」消息。此消息用于将数据块按厂商与合成器所依靠的格式传递给合成器（例如，用这种方法可以将新的声音定义从计算机传递给合成器）。系统专用消息只是可以包含多于2个数据字节的唯一消息。实际上，数据字节数是变化的，而每个数据字节的高位都设定为0。状态字节0xF7表示系统专用消息的结尾。

系统专用消息也用于从合成器转储数据（例如，声音定义）。这些数据都是通过MIDI Out端口来自合成器。如果要用设备无关的方式对MIDI编写程序，则应该尽可能避免使用系统专用消息。但是它们对于定义新的合成器声音是非常有用的。

MIDI文件（扩展名是.MDI）是带有定时信息的MIDI信息集，可以用MCI播放MIDI文件。不过，我将在本章的后面讨论低阶midiOut函数。

MIDI编曲简介

低阶MIDI的API包括前缀为midiIn和midiOut的函数，它们分别用于读取来自外部控制器的MIDI序列和在内部或外部的合成器上播放音乐。尽管其名称为「低阶」，但使用这些函数时并不需要了解MIDI卡上的硬件接口。

要在播放音乐的准备期间打开一个MIDI输出设备，可以呼叫midiOutOpen函数：

```
error = midiOutOpen (&hMidiOut, wDeviceID, dwCallBack,
                     dwCallBackData, dwFlags) ;
```

如果呼叫成功，则函数传回0，否则传回错误代码。如果参数设定正确，则常见的一种错误就是MIDI设备已被其它程序使用。

该函数的第一个参数是指向HMIDIOUT型态变量的指针，它接收后面用于MIDI输出函数的MIDI输出句柄。第二个参数是设备ID。要使用真实的MIDI设备，这个参数范围可以是从0到小于由midiOutGetNumDevs传回的数值。您还可以使用MIDIMAPPER，它在MMSYSTEM.H中定义为-1。大多数情况下，函数的后三个参数设定为NULL或0。

一旦打开一个MIDI输出设备并获得了其句柄，您就可以向该设备发送MIDI消息。此时可以呼叫：

error = midiOutShortMsg (hMidiOut, dwMessage) ;

第一个参数是从midiOutOpen函数获得的句柄。第二个参数是包装在32位DWORD中的1字节、2字节或者3字节的消息。我在前面讨论过，MIDI消息以状态字节开始，后面是0、1或2字节的数据。在dwMessage中，状态字节是最不重要的，第一个数据字节次之，第二个数据字节再次之，最重要的字节是0。

例如，要在MIDI通道5上以0x7F的速度演奏中音C（音符是0x3C），则需要3字节的Note On消息：

0x95 0x3C 0x7F

midiOutShortMsg的参数dwMessage等于0x007F3C95。

三个基础的MIDI消息是Program Change（可为某一特定通道而改变乐器声音）、Note On和Note Off。打开一个MIDI输出设备后，应该从一条Program Change消息开始，然后发送相同数量的Note On和Note Off消息。

当您一直演奏您想演奏的音乐时，您可以重置MIDI输出设备以确保关闭所有的音符：

midiOutReset (hMidiOut) ;

然后关闭设备：

midiOutClose (hMidiOut) ;

使用低阶的MIDI输出API时，midiOutOpen、midiOutShortMsg、midiOutReset和midiOutClose是您需要的四个基础函数。

现在让我们演奏一段音乐。BACHTOCC，如程序22-8所示，演奏了J. S. Bach著名的风琴演奏的D小调《Toccata and Fugue》中托卡塔部分的第一小节。

程序22-8 BACHTOCC

BACHTOCC.C

```
/*------------------------------------------------------------------------
BACHTOCC.C -- Bach Toccata in D Minor (First Bar)
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#define ID_TIMER 1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("BachTocc") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  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 ;
  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Bach Toccata in D Minor (First Bar)"),
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;
```

```
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                       int iData1, int iData2)
{
  DWORD dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
  return midiOutShortMsg (hMidi, dwMessage) ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static struct
  {
    int iDur ;
    int iNote [2] ;
  }
  noteseq [] = { 110, 69, 81, 110, 67, 79, 990, 69, 81, 220, -1, -1,
    110, 67, 79, 110, 65, 77, 110, 64, 76, 110, 62, 74,
    220, 61, 73, 440, 62, 74, 1980, -1, -1, 110, 57, 69,
    110, 55, 67, 990, 57, 69, 220, -1, -1, 220, 52, 64,
    220, 53, 65, 220, 49, 61, 440, 50, 62, 1980, -1, -1 } ;

  static HMIDIOUT hMidiOut ;
  static int iIndex ;
  int i ;

  switch (message)
  {
  case WM_CREATE:
    // Open MIDIMAPPER device

    if (midiOutOpen (&hMidiOut, MIDIMAPPER, 0, 0, 0))
    {
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox ( hwnd, TEXT ("Cannot open MIDI output device!"),
        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return -1 ;
    }
    // Send Program Change messages for "Church Organ"

    MidiOutMessage (hMidiOut, 0xC0, 0, 19, 0) ;
    MidiOutMessage (hMidiOut, 0xC0, 12, 19, 0) ;

    SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
    return 0 ;

  case WM_TIMER:
    // Loop for 2-note polyphony

    for (i = 0 ; i < 2 ; i++)
    {
      // Note Off messages for previous note

      if (iIndex != 0 && noteseq[iIndex - 1].iNote[i] != -1)
      {
        MidiOutMessage (hMidiOut, 0x80, 0, noteseq[iIndex - 1].iNote[i], 0) ;

        MidiOutMessage (hMidiOut, 0x80, 12, noteseq[iIndex - 1].iNote[i], 0) ;
      }
      // Note On messages for new note

      if (iIndex != sizeof (noteseq) / sizeof (noteseq[0]) &&
```

```
      noteseq[iIndex].iNote[i] != -1)
    {
      MidiOutMessage (hMidiOut, 0x90, 0, noteseq[iIndex].iNote[i], 127) ;

      MidiOutMessage (hMidiOut, 0x90, 12,noteseq[iIndex].iNote[i], 127) ;
    }
  }

  if (iIndex != sizeof (noteseq) / sizeof (noteseq[0]))
  {
    SetTimer (hwnd, ID_TIMER, noteseq[iIndex++].iDur - 1, NULL) ;
  }
  else
  {
    KillTimer (hwnd, ID_TIMER) ;
    DestroyWindow (hwnd) ;
  }
  return 0 ;

 case WM_DESTROY:
  midiOutReset (hMidiOut) ;
  midiOutClose (hMidiOut) ;
  PostQuitMessage (0) ;
  return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图22-1显示了Bach的D小调Toccata的第一小节。

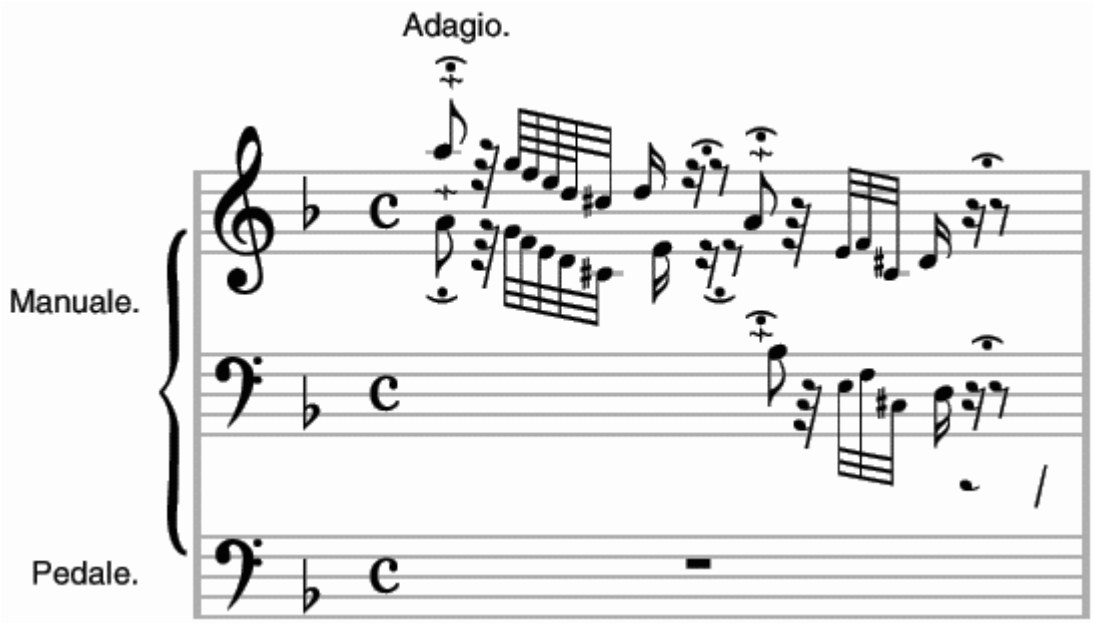


图22-1 Bach的D小调Toccata and Fugue的第一小节

在这里要做的就是把音乐转换成一系列的数值－基本键号和定时信息，其中定时信息表示发送Note On（对应于风琴键按下）和Note Off（释放键）消息的时间。由于风琴键盘对速度不敏感，所以我们用相同的速度来演奏所有的音符。另外一个简化是忽略断奏（即，在连续的音符之间留下一个很短的停顿，以达到尖硬的效果）和连奏（在连续的音符之间有更圆润的重迭）之间的区别。我们假定一个音符结束后面紧接着下一个音符开始。

如果看得懂乐谱，那么您就会注意到托卡塔曲以两个平行的八度音阶开始。因此BACHTOCC建立了一个数据结构noteseq来储存一系列的音符持续时间以及两个键号。不幸的是，音乐持续进

入第二小节就需要更特殊的方法来储存此信息。我将四分音符的持续时间定义为1760毫秒，也就是说，八分音符（在音符或者休止符上有一个符尾）的持续时间是880毫秒，十六分音符（两个符尾）是440毫秒，三十二分音符（三个符尾）是220毫秒，六十四分音符（四个符尾）是110毫秒。

这第一小节中有两个波音 – 一个在第一个音符处，另一个在小节的中间。这在乐谱上用带一条短竖线的曲线表示。在结构复杂的乐曲中，波音符号表示此音符实际应演奏为三个音符 – 标出的音符、比它低一个全音的音符，然后还是标出的音符。前两个音符演奏得要快，第三个音符要持续剩余的时间。例如，第一个音符是带波音的A，则应演奏为A、G、A。我将波音的前两个音符定义为六十四分音符，所以每个音符都持续110毫秒。

在第一小节还有四个延长符号。乐谱上表示为中间带点的半圆形。延长符号表示该音符在演奏时所持续的时间比标记的时间要长，通常由演奏者决定具体的时间。我对于延长符号延长了50%的时间。

可以看到，即使是转换一小段看来简单直接的乐曲，例如D小调《Toccata》的开头，也并不是件容易的事!

noteseq结构数组包含了这一小节中平行的音符和休止符的三个数字。音符持续时间的后面是用于平行八度音阶的两个MIDI键号。例如，第一个音符是A，持续时间是110毫秒。因为中音C的MIDI键号是60，所以中音C上面的A的键号是69，比A高一个八度音阶的键号是81。因此，noteseq数组的前三个数是110、69和81。我用音符值-1表示休止符。

WM_CREATE消息处理期间，BACHTOCC设定一个Windows定时器用于定时1000毫秒 – 表示乐曲从第1秒开始演奏 – 然后用MIDIMAPPER设备ID呼叫midiOutOpen。

BACHTOCC只需要一种乐器（风琴）的声音，所以只需要一个通道。为了简化MIDI消息的发送，BACHTOCC中还定义了一个小函数MidiOutMessage。此函数接收MIDI输出句柄、状态字节、信道句柄和两个字节数据。其功能是把这些数字打包到一条32位的消息并呼叫midiOutShortMsg。

在WM_CREATE消息处理程序的后期，BACHTOCC发送一条Program Change消息来选择「教堂风琴」的声音。在General MIDI声音配置中，教堂风琴声音在Program Change消息中用数字字节19表示。实际演奏的音符出现在WM_TIMER消息处理期间。用循环来处理两个音符的多音。如果前一个音符还在演奏，BACHTOCC就为该音符发送Note Off消息。然后，如果下一个音符不是休止符，则向通道0和12发送Note On消息。随后，重置Windows定时器，使其与noteseq结构中音符的持续时间一致。

音乐演奏完后，BACHTOCC删除窗口。在WM_DESTROY消息处理期间，程序呼叫midiOutReset和midiOutClose，然后终止程序。

尽管BACHTOCC合理地处理和计算声音（即使还不完全像真人演奏风琴），但一般情况下用Windows定时器按这种方式来演奏音乐并不管用。问题在于Windows定时器是依据PC的系统时钟，其分辨率不能满足音乐的要求。而且，Windows定时器不是同步的。这样，如果其它程序正忙于执行，则获得WM_TIMER消息就会有轻微的延迟。如果程序不能立即处理这些消息，就会放弃WM_TIMER消息，这时的声音听起来一团糟。

因此，当BACHTOCC显示了如何呼叫低阶MIDI输出函数时，使用Windows定时器显然不适合精确的音乐创作。所以，Windows还提供了一系列附加的定时器函数，使用低阶的MIDI输出函数时可以利用这些函数。这些函数的前缀为time，您可以利用它们将定时器的分辨率设定到最小1毫秒。我将在本章结尾的DRUM程序向您展示使用这些函数的方法。

通过键盘演奏MIDI合成器

因为大多数PC使用者可能都没有连结在机器上的MIDI键盘，所以可以用每个人都有的键盘（上面全部的字母键和数据键）来代替。程序22-9所示的程序KBMIDI允许您用PC键盘来演奏电子音乐合成器－不管是连结在声卡上的，还是挂接在MIDI Out埠的外部合成器。KBMIDI让您完全控制MIDI输出设备（即内部或外部的合成器）、MIDI通道和乐器声音。除了演奏时的趣味性以外，我还发现此程序对于开发Windows如何实作MIDI支持很有用。

程序22-9 KBMIDI

KBMIDI.C

```
/*------------------------------------------------------------------------
KBMIDI.C -- Keyboard MIDI Player
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
// Defines for Menu IDs
// --------------------

#define IDM_OPEN 0x100
#define IDM_CLOSE 0x101
#define IDM_DEVICE 0x200
#define IDM_CHANNEL 0x300
#define IDM_VOICE 0x400

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
TCHAR szAppName [] = TEXT ("KBMidi") ;
HMIDIOUT hMidiOut ;
int iDevice = MIDIMAPPER, iChannel = 0, iVoice = 0, iVelocity = 64 ;
int cxCaps, cyChar, xOffset, yOffset ;

// Structures and data for showing families and instruments on menu
// ----------------------------------------------------------------

typedef struct
{
  TCHAR * szInst ;
  int iVoice ;
}
INSTRUMENT ;
typedef struct
{
  TCHAR * szFam ;
  INSTRUMENT inst [8] ;
}
FAMILY ;
FAMILY fam [16] = {

  TEXT ("Piano"),

    TEXT ("Acoustic Grand Piano"), 0,
    TEXT ("Bright Acoustic Piano"),1,
    TEXT ("Electric Grand Piano"), 2,
    TEXT ("Honky-tonk Piano"), 3,
    TEXT ("Rhodes Piano"), 4,
    TEXT ("Chorused Piano"), 5,
    TEXT ("Harpsichord"), 6,
    TEXT ("Clavinet"), 7,
    TEXT ("Chromatic Percussion"),
    TEXT ("Celesta"), 8,
    TEXT ("Glockenspiel"), 9,
    TEXT ("Music Box"), 10,
    TEXT ("Vibraphone"), 11,
    TEXT ("Marimba"), 12,
    TEXT ("Xylophone"), 13,
    TEXT ("Tubular Bells"), 14,
    TEXT ("Dulcimer"), 15,
    TEXT ("Organ"),
```

```
    TEXT ("Hammond Organ"), 16,
    TEXT ("Percussive Organ"), 17,
    TEXT ("Rock Organ"), 18,
    TEXT ("Church Organ"), 19,
    TEXT ("Reed Organ"), 20,
    TEXT ("Accordian"), 21,
    TEXT ("Harmonica"), 22,
    TEXT ("Tango Accordian"), 23,
    TEXT ("Guitar"),
    TEXT ("Acoustic Guitar (nylon)"), 24,
    TEXT ("Acoustic Guitar (steel)"), 25,
    TEXT ("Electric Guitar (jazz)"), 26,
    TEXT ("Electric Guitar (clean)"), 27,
    TEXT ("Electric Guitar (muted)"), 28,
    TEXT ("Overdriven Guitar"), 29,
    TEXT ("Distortion Guitar"), 30,
    TEXT ("Guitar Harmonics"), 31,
    TEXT ("Bass"),
    TEXT ("Acoustic Bass"), 32,
    TEXT ("Electric Bass (finger)"), 33,
    TEXT ("Electric Bass (pick)"), 34,
    TEXT ("Fretless Bass"), 35,
    TEXT ("Slap Bass 1"), 36,
    TEXT ("Slap Bass 2"), 37,
    TEXT ("Synth Bass 1"), 38,
    TEXT ("Synth Bass 2"), 39,
    TEXT ("Strings"),
    TEXT ("Violin"), 40,
    TEXT ("Viola"), 41,
    TEXT ("Cello"), 42,
    TEXT ("Contrabass"), 43,
    TEXT ("Tremolo Strings"), 44,
    TEXT ("Pizzicato Strings"), 45,
    TEXT ("Orchestral Harp"), 46,
    TEXT ("Timpani"), 47,
    TEXT ("Ensemble"),
    TEXT ("String Ensemble 1"), 48,
    TEXT ("String Ensemble 2"), 49,
    TEXT ("Synth Strings 1"), 50,
    TEXT ("Synth Strings 2"), 51,
    TEXT ("Choir Aahs"), 52,
    TEXT ("Voice Oohs"), 53,
    TEXT ("Synth Voice"), 54,
    TEXT ("Orchestra Hit"), 55,
    TEXT ("Brass"),
    TEXT ("Trumpet"), 56,
    TEXT ("Trombone"), 57,
    TEXT ("Tuba"), 58,
    TEXT ("Muted Trumpet"), 59,
    TEXT ("French Horn"), 60,
    TEXT ("Brass Section"), 61,
    TEXT ("Synth Brass 1"), 62,
    TEXT ("Synth Brass 2"), 63,
    TEXT ("Reed"),
    TEXT ("Soprano Sax"), 64,
    TEXT ("Alto Sax"), 65,
    TEXT ("Tenor Sax"), 66,
    TEXT ("Baritone Sax"), 67,
    TEXT ("Oboe"), 68,
    TEXT ("English Horn"), 69,
    TEXT ("Bassoon"), 70,
    TEXT ("Clarinet"), 71,
    TEXT ("Pipe"),
    TEXT ("Piccolo"), 72,
    TEXT ("Flute "), 73,
    TEXT ("Recorder"), 74,
    TEXT ("Pan Flute"), 75,
    TEXT ("Bottle Blow"), 76,
    TEXT ("Shakuhachi"), 77,
```

```
    TEXT ("Whistle"), 78,
    TEXT ("Ocarina"), 79,
    TEXT ("Synth Lead"),
    TEXT ("Lead 1 (square)"), 80,
    TEXT ("Lead 2 (sawtooth)"), 81,
    TEXT ("Lead 3 (caliope lead)"), 82,
    TEXT ("Lead 4 (chiff lead)"), 83,
    TEXT ("Lead 5 (charang)"), 84,
    TEXT ("Lead 6 (voice)"), 85,
    TEXT ("Lead 7 (fifths)"), 86,
    TEXT ("Lead 8 (brass + lead)"), 87,
    TEXT ("Synth Pad"),
    TEXT ("Pad 1 (new age)"), 88,
    TEXT ("Pad 2 (warm)"), 89,
    TEXT ("Pad 3 (polysynth)"), 90,
    TEXT ("Pad 4 (choir)"), 91,
    TEXT ("Pad 5 (bowed)"), 92,
    TEXT ("Pad 6 (metallic)"), 93,
    TEXT ("Pad 7 (halo)"), 94,
    TEXT ("Pad 8 (sweep)"), 95,
    TEXT ("Synth Effects"),
    TEXT ("FX 1 (rain)"), 96,
    TEXT ("FX 2 (soundtrack)"), 97,
    TEXT ("FX 3 (crystal)"), 98,
    TEXT ("FX 4 (atmosphere)"), 99,
    TEXT ("FX 5 (brightness)"), 100,
    TEXT ("FX 6 (goblins)"), 101,
    TEXT ("FX 7 (echoes)"), 102,
    TEXT ("FX 8 (sci-fi)"), 103,
    TEXT ("Ethnic"),
    TEXT ("Sitar"), 104,
    TEXT ("Banjo"), 105,
    TEXT ("Shamisen"), 106,
    TEXT ("Koto"), 107,
    TEXT ("Kalimba"), 108,
    TEXT ("Bagpipe"), 109,
    TEXT ("Fiddle"), 110,
    TEXT ("Shanai"), 111,
    TEXT ("Percussive"),
    TEXT ("Tinkle Bell"), 112,
    TEXT ("Agogo"), 113,
    TEXT ("Steel Drums"), 114,
    TEXT ("Woodblock"), 115,
    TEXT ("Taiko Drum"), 116,
    TEXT ("Melodic Tom"), 117,
    TEXT ("Synth Drum"), 118,
    TEXT ("Reverse Cymbal"), 119,
    TEXT ("Sound Effects"),
    TEXT ("Guitar Fret Noise"), 120,
    TEXT ("Breath Noise"), 121,
    TEXT ("Seashore"), 122,
    TEXT ("Bird Tweet"), 123,
    TEXT ("Telephone Ring"), 124,
    TEXT ("Helicopter"), 125,
    TEXT ("Applause"), 126,
    TEXT ("Gunshot"), 127
} ;

// Data for translating scan codes to octaves and notes
// ---------------------------------------------------

#define NUMSCANS (sizeof key / sizeof key[0])
struct
{
  int iOctave ;
  int iNote ;
  int yPos ;
  int xPos ;
  TCHAR * szKey ;
```

```
}
key [] =
{
  // Scan Char Oct Note
  // ---- ---- --- ----
  -1, -1, 1, -1, NULL, // 0 None
    -1, -1, -1, -1, NULL, // 1 Esc
    -1, -1, 0, 0, TEXT (""), // 2 1
    5, 1, 0, 2, TEXT ("C#"), // 3 2 5 C#
    5, 3, 0, 4, TEXT ("D#"), // 4 3 5 D#
    -1, -1, 0, 6, TEXT (""), // 5 4
    5, 6, 0, 8, TEXT ("F#"), // 6 5 5 F#
    5, 8, 0, 10, TEXT ("G#"), // 7 6 5 G#
    5, 10, 0, 12, TEXT ("A#"), // 8 7 5 A#
    -1, -1, 0, 14, TEXT (""), // 9 8
    6, 1, 0, 16, TEXT ("C#"), // 10 9 6 C#
    6, 3, 0, 18, TEXT ("D#"), // 11 0 6 D#
    -1, -1, 0, 20, TEXT (""), // 12 -
    6, 6, 0, 22, TEXT ("F#"), // 13 = 6 F#
    -1, -1, -1, -1, NULL, // 14 Back

    -1, -1, -1, -1, NULL, // 15 Tab
    5, 0, 1, 1, TEXT ("C"), // 16 q 5 C
    5, 2, 1, 3, TEXT ("D"), // 17 w 5 D
    5, 4, 1, 5, TEXT ("E"), // 18 e 5 E
    5, 5, 1, 7, TEXT ("F"), // 19 r 5 F
    5, 7, 1, 9, TEXT ("G"), // 20 t 5 G
    5, 9, 1, 11, TEXT ("A"), // 21 y 5 A
    5, 11, 1, 13, TEXT ("B"), // 22 u 5 B
    6, 0, 1, 15, TEXT ("C"), // 23 i 6 C
    6, 2, 1, 17, TEXT ("D"), // 24 o 6 D
    6, 4, 1, 19, TEXT ("E"), // 25 p 6 E
    6, 5, 1, 21, TEXT ("F"), // 26 [ 6 F
    6, 7, 1, 23, TEXT ("G"), // 27 ] 6 G
    -1, -1, -1, -1, NULL, // 28 Ent
    -1, -1, -1, -1, NULL, // 29 Ctrl
    3, 8, 2, 2, TEXT ("G#"), // 30 a 3 G#
    3, 10, 2, 4, TEXT ("A#"), // 31 s 3 A#
    -1, -1, 2, 6, TEXT (""), // 32 d
    4, 1, 2, 8, TEXT ("C#"), // 33 f 4 C#
    4, 3, 2, 10, TEXT ("D#"), // 34 g 4 D#
    -1, -1, 2, 12, TEXT (""), // 35 h
    4, 6, 2, 14, TEXT ("F#"), // 36 j 4 F#
    4, 8, 2, 16, TEXT ("G#"), // 37 k 4 G#
    4, 10, 2, 18, TEXT ("A#"), // 38 l 4 A#
    -1, -1, 2, 20, TEXT (""), // 39 ;
    5, 1, 2, 22, TEXT ("C#"), // 40 ' 5 C#
    -1, -1, -1, -1, NULL, // 41 `
    -1, -1, -1, -1, NULL, // 42 Shift
    -1, -1, -1, -1, NULL, // 43 \ (not line continuation)
    3, 9, 3, 3, TEXT ("A"), // 44 z 3 A
    3, 11, 3, 5, TEXT ("B"), // 45 x 3 B
    4, 0, 3, 7, TEXT ("C"), // 46 c 4 C
    4, 2, 3, 9, TEXT ("D"), // 47 v 4 D
    4, 4, 3, 11, TEXT ("E"), // 48 b 4 E
    4, 5, 3, 13, TEXT ("F"), // 49 n 4 F
    4, 7, 3, 15, TEXT ("G"), // 50 m 4 G
    4, 9, 3, 17, TEXT ("A"), // 51 , 4 A
    4, 11, 3, 19, TEXT ("B"), // 52 . 4 B
    5, 0, 3, 21, TEXT ("C") // 53 / 5 C
} ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  MSG msg;
  HWND hwnd ;
  WNDCLASS wndclass ;
```

```
  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 = (HBRUSH)GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Keyboard MIDI Player"),
    WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  if (!hwnd)
    return 0 ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd);

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

// Create the program's menu (called from WndProc, WM_CREATE)
// ----------------------------------------------------------------------

HMENU CreateTheMenu (int iNumDevs)
{
  TCHAR szBuffer [32] ;
  HMENU hMenu, hMenuPopup, hMenuSubPopup ;
  int i, iFam, iIns ;
  MIDIOUTCAPS moc ;

  hMenu = CreateMenu () ;
  // Create "On/Off" popup menu
  hMenuPopup = CreateMenu () ;
  AppendMenu (hMenuPopup, MF_STRING, IDM_OPEN, TEXT ("&Open")) ;
  AppendMenu (hMenuPopup, MF_STRING | MF_CHECKED, IDM_CLOSE,
    TEXT ("&Closed")) ;
  AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
    TEXT ("&Status")) ;

  // Create "Device" popup menu

  hMenuPopup = CreateMenu () ;
  // Put MIDI Mapper on menu if it's installed
  if (!midiOutGetDevCaps (MIDIMAPPER, &moc, sizeof (moc)))
    AppendMenu (hMenuPopup, MF_STRING, IDM_DEVICE + (int) MIDIMAPPER,
    moc.szPname) ;
  else
    iDevice = 0 ;
  // Add the rest of the MIDI devices
  for (i = 0 ; i < iNumDevs ; i++)
  {
    midiOutGetDevCaps (i, &moc, sizeof (moc)) ;
```

```
    AppendMenu (hMenuPopup, MF_STRING, IDM_DEVICE + i, moc.szPname) ;
  }

  CheckMenuItem (hMenuPopup, 0, MF_BYPOSITION | MF_CHECKED) ;
  AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
    TEXT ("&Device")) ;
  // Create "Channel" popup menu
  hMenuPopup = CreateMenu () ;
  for (i = 0 ; i < 16 ; i++)
  {
    wsprintf (szBuffer, TEXT ("%d"), i + 1) ;
    AppendMenu (hMenuPopup, MF_STRING | (i ? MF_UNCHECKED : MF_CHECKED),
      IDM_CHANNEL + i, szBuffer) ;
  }

  AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
    TEXT ("&Channel")) ;
  // Create "Voice" popup menu
  hMenuPopup = CreateMenu () ;
  for (iFam = 0 ; iFam < 16 ; iFam++)
  {
    hMenuSubPopup = CreateMenu () ;
    for (iIns = 0 ; iIns < 8 ; iIns++)
    {
      wsprintf (szBuffer, TEXT ("&%d.\t%s"), iIns + 1,
        fam[iFam].inst[iIns].szInst) ;
      AppendMenu (hMenuSubPopup,
        MF_STRING | (fam[iFam].inst[iIns].iVoice ?
MF_UNCHECKED : MF_CHECKED),
              fam[iFam].inst[iIns].iVoice + IDM_VOICE,
              szBuffer) ;
    }

    wsprintf (szBuffer, TEXT ("&%c.\t%s"), 'A' + iFam,
      fam[iFam].szFam) ;
    AppendMenu (hMenuPopup, MF_STRING | MF_POPUP, (UINT) hMenuSubPopup,
      szBuffer) ;
  }
  AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
    TEXT ("&Voice")) ;
  return hMenu ;
}

// Routines for simplifying MIDI output
// ------------------------------------

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                  int iData1, int iData2)
{
  DWORD dwMessage ;
  dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
  return midiOutShortMsg (hMidi, dwMessage) ;
}

DWORD MidiNoteOff (HMIDIOUT hMidi, int iChannel, int iOct, int iNote, int iVel)
{
  return MidiOutMessage (hMidi, 0x080, iChannel, 12 * iOct + iNote, iVel) ;
}

DWORD MidiNoteOn (HMIDIOUT hMidi, int iChannel, int iOct, int iNote, int iVel)
{
  return MidiOutMessage ( hMidi, 0x090, iChannel, 12 * iOct + iNote, iVel) ;
}

DWORD MidiSetPatch (HMIDIOUT hMidi, int iChannel, int iVoice)
{
  return MidiOutMessage (hMidi, 0x0C0, iChannel, iVoice, 0) ;
}
```

```
DWORD MidiPitchBend (HMIDIOUT hMidi, int iChannel, int iBend)
{
  return MidiOutMessage (hMidi, 0x0E0, iChannel, iBend & 0x7F, iBend >> 7) ;
}

// Draw a single key on window
// ---------------------------------

VOID DrawKey (HDC hdc, int iScanCode, BOOL fInvert)
{
  RECT rc ;
  rc.left = 3 * cxCaps * key[iScanCode].xPos / 2 + xOffset ;
  rc.top = 3 * cyChar * key[iScanCode].yPos / 2 + yOffset ;
  rc.right = rc.left + 3 * cxCaps ;
  rc.bottom = rc.top + 3 * cyChar / 2 ;

  SetTextColor (hdc, fInvert ? 0x00FFFFFFul : 0x00000000ul) ;
  SetBkColor (hdc, fInvert ? 0x00000000ul : 0x00FFFFFFul) ;

  FillRect (hdc, &rc, (HBRUSH)GetStockObject (fInvert ? BLACK_BRUSH : WHITE_BRUSH)) ;
  DrawText (hdc, key[iScanCode].szKey, -1, &rc,
    DT_SINGLELINE | DT_CENTER | DT_VCENTER) ;
  FrameRect (hdc, &rc, (HBRUSH)GetStockObject (BLACK_BRUSH)) ;
}

// Process a Key Up or Key Down message
// ------------------------------------

VOID ProcessKey (HDC hdc, UINT message, LPARAM lParam)
{
  int iScanCode, iOctave, iNote ;
  iScanCode = 0x0FF & HIWORD (lParam) ;
  if (iScanCode >= NUMSCANS) // No scan codes over 53
    return ;

  if ((iOctave = key[iScanCode].iOctave) == -1) // Non-music key
    return ;

  if (GetKeyState (VK_SHIFT) < 0)
    iOctave += 0x20000000 & lParam ? 2 : 1 ;
  if (GetKeyState (VK_CONTROL) < 0)
    iOctave -= 0x20000000 & lParam ? 2 : 1 ;
  iNote = key[iScanCode].iNote ;
  if (message == WM_KEYUP) // For key up
  {
    MidiNoteOff (hMidiOut, iChannel, iOctave, iNote, 0) ; // Note off
    DrawKey (hdc, iScanCode, FALSE) ;
    return ;
  }

  if (0x40000000 & lParam) // ignore typematics
    return ;

  MidiNoteOn (hMidiOut, iChannel, iOctave, iNote, iVelocity) ; // Note on
  DrawKey (hdc, iScanCode, TRUE) ; // Draw the inverted key
}

// Window Procedure
// ----------------------

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL bOpened = FALSE ;
  HDC hdc ;
  HMENU hMenu ;
  int i, iNumDevs, iPitchBend, cxClient, cyClient ;
  MIDIOUTCAPS moc ;
  PAINTSTRUCT ps ;
  SIZE size ;
```

```
TCHAR szBuffer [16] ;

switch (message)
{
case WM_CREATE:
  // Get size of capital letters in system font

  hdc = GetDC (hwnd) ;

  GetTextExtentPoint (hdc, TEXT ("M"), 1, &size) ;
  cxCaps = size.cx ;
  cyChar = size.cy ;

  ReleaseDC (hwnd, hdc) ;

  // Initialize "Volume" scroll bar

  SetScrollRange (hwnd, SB_HORZ, 1, 127, FALSE) ;
  SetScrollPos (hwnd, SB_HORZ, iVelocity, TRUE) ;

  // Initialize "Pitch Bend" scroll bar

  SetScrollRange (hwnd, SB_VERT, 0, 16383, FALSE) ;
  SetScrollPos (hwnd, SB_VERT, 8192, TRUE) ;

  // Get number of MIDI output devices and set up menu

  if (0 == (iNumDevs = midiOutGetNumDevs ()))
  {
    MessageBeep (MB_ICONSTOP) ;
    MessageBox ( hwnd, TEXT ("No MIDI output devices!"),
      szAppName, MB_OK | MB_ICONSTOP) ;
    return -1 ;
  }
  SetMenu (hwnd, CreateTheMenu (iNumDevs)) ;
  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;

  xOffset = (cxClient - 25 * 3 * cxCaps / 2) / 2 ;
  yOffset = (cyClient - 11 * cyChar) / 2 + 5 * cyChar ;
  return 0 ;

case WM_COMMAND:
  hMenu = GetMenu (hwnd) ;

  // "Open" menu command

  if (LOWORD (wParam) == IDM_OPEN && !bOpened)
  {
    if (midiOutOpen (&hMidiOut, iDevice, 0, 0, 0))
    {
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, TEXT ("Cannot open MIDI device"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
    }
    else
    {
      CheckMenuItem (hMenu, IDM_OPEN, MF_CHECKED) ;
      CheckMenuItem (hMenu, IDM_CLOSE, MF_UNCHECKED) ;

      MidiSetPatch (hMidiOut, iChannel, iVoice) ;
      bOpened = TRUE ;
    }
  }

  // "Close" menu command
```

```
  else if (LOWORD (wParam) == IDM_CLOSE && bOpened)
  {
    CheckMenuItem (hMenu, IDM_OPEN, MF_UNCHECKED) ;
    CheckMenuItem (hMenu, IDM_CLOSE, MF_CHECKED) ;

    // Turn all keys off and close device
    for (i = 0 ; i < 16 ; i++)
      MidiOutMessage (hMidiOut, 0xB0, i, 123, 0) ;
    midiOutClose (hMidiOut) ;
    bOpened = FALSE ;
  }

  // Change MIDI "Device" menu command
  else if ( LOWORD (wParam) >= IDM_DEVICE - 1 &&
    LOWORD (wParam) < IDM_CHANNEL)
  {
    CheckMenuItem (hMenu, IDM_DEVICE + iDevice, MF_UNCHECKED) ;
    iDevice = LOWORD (wParam) - IDM_DEVICE ;
    CheckMenuItem (hMenu, IDM_DEVICE + iDevice, MF_CHECKED) ;

    // Close and reopen MIDI device

    if (bOpened)
    {
      SendMessage (hwnd, WM_COMMAND, IDM_CLOSE, 0L) ;
      SendMessage (hwnd, WM_COMMAND, IDM_OPEN, 0L) ;
    }
  }

  // Change MIDI "Channel" menu command

  else if ( LOWORD (wParam) >= IDM_CHANNEL &&
    LOWORD (wParam) < IDM_VOICE)
  {
    CheckMenuItem (hMenu, IDM_CHANNEL + iChannel, MF_UNCHECKED);
    iChannel = LOWORD (wParam) - IDM_CHANNEL ;
    CheckMenuItem (hMenu, IDM_CHANNEL + iChannel, MF_CHECKED) ;

    if (bOpened)
      MidiSetPatch (hMidiOut, iChannel, iVoice) ;
  }

  // Change MIDI "Voice" menu command

  else if (LOWORD (wParam) >= IDM_VOICE)
  {
    CheckMenuItem (hMenu, IDM_VOICE + iVoice, MF_UNCHECKED) ;
    iVoice = LOWORD (wParam) - IDM_VOICE ;
    CheckMenuItem (hMenu, IDM_VOICE + iVoice, MF_CHECKED) ;

    if (bOpened)
      MidiSetPatch (hMidiOut, iChannel, iVoice) ;
  }

  InvalidateRect (hwnd, NULL, TRUE) ;
  return 0 ;

  // Process a Key Up or Key Down message

case WM_KEYUP:
case WM_KEYDOWN:
  hdc = GetDC (hwnd) ;

  if (bOpened)
    ProcessKey (hdc, message, lParam) ;

  ReleaseDC (hwnd, hdc) ;
  return 0 ;
```

```
  // For Escape, turn off all notes and repaint

case WM_CHAR:
  if (bOpened && wParam == 27)
  {
    for (i = 0 ; i < 16 ; i++)
      MidiOutMessage (hMidiOut, 0xB0, i, 123, 0) ;

    InvalidateRect (hwnd, NULL, TRUE) ;
  }
  return 0 ;

  // Horizontal scroll: Velocity

case WM_HSCROLL:
  switch (LOWORD (wParam))
  {
  case SB_LINEUP: iVelocity -= 1 ; break ;
  case SB_LINEDOWN: iVelocity += 1 ; break ;
  case SB_PAGEUP: iVelocity -= 8 ; break ;
  case SB_PAGEDOWN: iVelocity += 8 ; break ;
  case SB_THUMBPOSITION: iVelocity = HIWORD (wParam) ; break ;
  default: return 0 ;
  }
  iVelocity = max (1, min (iVelocity, 127)) ;
  SetScrollPos (hwnd, SB_HORZ, iVelocity, TRUE) ;
  return 0 ;

  // Vertical scroll: Pitch Bend
  case WM_VSCROLL:
    switch (LOWORD (wParam))
    {
    case SB_THUMBTRACK: iPitchBend = 16383 - HIWORD (wParam) ; break ;
    case SB_THUMBPOSITION: iPitchBend = 8191 ; break ;
    default: return 0 ;
    }
    iPitchBend = max (0, min (iPitchBend, 16383)) ;
    SetScrollPos (hwnd, SB_VERT, 16383 - iPitchBend, TRUE) ;

    if (bOpened)
      MidiPitchBend (hMidiOut, iChannel, iPitchBend) ;
    return 0 ;

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      for (i = 0 ; i < NUMSCANS ; i++)
        if (key[i].xPos != -1)
          DrawKey (hdc, i, FALSE) ;

        midiOutGetDevCaps (iDevice, &moc, sizeof (MIDIOUTCAPS)) ;
        wsprintf (szBuffer, TEXT ("Channel %i"), iChannel + 1) ;

        TextOut ( hdc, cxCaps, 1 * cyChar,
          bOpened ? TEXT ("Open") : TEXT ("Closed"),
          bOpened ? 4 : 6) ;
        TextOut ( hdc, cxCaps, 2 * cyChar, moc.szPname,
          lstrlen (moc.szPname)) ;
        TextOut (hdc, cxCaps, 3 * cyChar, szBuffer, lstrlen (szBuffer)) ;
        TextOut(hdc, cxCaps, 4 * cyChar,
          fam[iVoice / 8].inst[iVoice % 8].szInst,
          lstrlen (fam[iVoice / 8].inst[iVoice % 8].szInst)) ;

        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_DESTROY :
      SendMessage (hwnd, WM_COMMAND, IDM_CLOSE, 0L) ;
      PostQuitMessage (0) ;
```

```
      return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

执行KBMIDI时，窗口显示了键盘上的键与传统钢琴或风琴按键的对应方式。左下角的Z键以110 Hz的频率演奏A。键盘的最下行，右边是中音C，倒数第二行为其升音或降音。上面两行键继续按此规律变化，从中音C到G#。这样，整个范围是三个八度音阶。另外，分别按Shift键和Ctrl键可使整个音域上升或下降1个八度音阶，这样有效的音域就是5个八度音阶。

不过，如果立即开始演奏，那么您将听不到任何声音。您必须先从「Status」菜单中选择「Open」，打开一个MIDI输出设备。如果埠打开成功，则按下一个键就向合成器发送一条MIDI Note On消息，释放键则产生一条Note Off消息。取决于键盘的按键特性，您可以同时演奏几个音符。

从「Status」菜单里选择「Close」来关闭MIDI设备。这对于需要在不终止KBMIDI程序的情况下执行Windows下的其它MIDI软件来说是很方便的。

「Device」菜单列出了已安装的MIDI输出设备，这些设备通过呼叫midiOutGetDevCaps函数获得。其中有些设备可能是MIDI Out埠连结的实际存在或不存在的外部合成器。列表还包括MIDI Mapper设备。这是从「控制台」的「多媒体」中选择的MIDI合成器。

「Channel」菜单用来选择从1到16的MIDI通道，内定状态下选择通道1。KBMIDI程序产生的所有MIDI消息都发送到所选的通道。

KBMIDI最后一个菜单项是「Voice」，它是一个双层菜单，用于选择128种乐器声音，这些声音在General MIDI规范中定义并在Windows中实作。这128种乐器声音分为16乐器组，每个乐器组有8种乐器。由于不同的MIDI键号对应于不同的泛音，所以这128种乐器声音也称为有旋律的声音。

General MIDI中还定义了大量无旋律的打击乐器。要演奏打击乐器，可以从「Channel」菜单选择通道10，还可以从「Voice」菜单选择第一种乐器声音（「Acoustic Grand Piano」）。这样，按不同的键就可以得到不同打击乐器的声音。从MIDI键号35（低于中音C两个八度音阶的B）到81（高于中音C近两个八度音阶的A），共有47种不同的打击乐器声音。在下面的DRUM程序中就利用了打击乐器通道。

KBMIDI程序有水平和垂直滚动条。由于PC键盘对按键速度不敏感，所以用水平滚动条来控制音符速度。一般来说，这与演奏音符的音量一致。设定完水平滚动条以后，所有的Note On消息都将使用这个速度。

垂直滚动条将产生一条称为「Pitch Bend」的MIDI消息。要使用此特性，请按下一个或多个键，然后用鼠标拖动滚动条。向上拖动滚动条音符频率将上升，向下拖动则频率下降。释放滚动条后将恢复正常的基音。

这两个滚动条要小心使用：因为拖动滚动条时，键盘消息将不进入程序的消息循环。因此，如果按下一个键后就开始拖动滚动条，然后在完成拖动之前就释放了该键，那么音符仍将发声。所以，拖动滚动条时不要按下或者释放任何键。对菜单也有类似的规则：按着键时不要进行菜单选择。另外，在按下与释放某个键期间，不要用Ctrl或Shift键来改变八度音阶。

如果一个或者多个音符出现「粘滞现象」，即释放后继续发声，那么请按下Esc键。按下此键将通过向MIDI合成器的16个通道发送16条All Notes Off消息，来关闭声音。

KBMIDI没有资源描述文件，而是通过搜索来建立的菜单。设备名称从midiOutGetDevCaps函数获得，乐器种类和名称则储存在程序的一个大数据结构中。

KBMIDI定义了几个小函数来简化MIDI消息。除了Pitch Bend消息以外，其它消息都在前面讨论过了。Pitch Bend消息用两个7位值组成一个14位的音调弯曲等级：0到0x1FFF之间的值降低基

音，0x2001到0x3FFF之间的值升高基音。

从「Status」菜单选择「Open」时，KBMIDI为选择的设备呼叫midiOutOpen；如呼叫成功，则呼叫MidiSetPatch函数。设备改变时，KBMIDI必须关闭前一个设备，必要时再打开新设备。当改变MIDI设备、MIDI通道、乐器声音时，KBMIDI也必须呼叫MidiSetPatch。

KBMIDI通过处理WM_KEYUP消息和WM_KEYDOWN消息来控制音符的发音。KBMIDI中用一个数组把键盘扫描码映像成八度音阶和音符。例如，美国英语键盘上Z键的扫描码是44，数组将其标记为八度音阶是3，音符是9（即A）。在KBMIDI的MidiNoteOn函数里，这些组合成了MIDI键号45（即12乘以3再加上9）。此数据结构也用于在窗口中画出键－每个键都有特定的水平和垂直位置，以及显示在矩形中的文字字符串。

水平滚动条的处理是很直接的：所有需要做的就是储存新的速度级并设定新的滚动条的位置。但是处理垂直滚动条以控制音调弯曲的操作稍有一点特殊，它处理的滚动条命令只有两个：用鼠标拖动滚动条时发生的SB_THUMBTRACK，以及释放滚动条时的SB_THUMBPOSITION。处理SB_THUMBPOSITION命令时，KBMIDI将滚动条位置设定为中间等级，并呼叫MidiPitchBend，其中参数值是8192。

MIDI击鼓器

有些打击乐器，如木琴或定音鼓，是「有旋律的」或「半音阶的」，因为它们可以用不同的音阶演奏乐曲。木琴用木板来对应不同的音阶，定音鼓也可以演奏曲调。这两种乐器及其它的有旋律的打击乐器都可以在KBMIDI的「Voice」菜单里选择。

但是，其它许多打击乐器都没有旋律，它们不能调音，而且通常含有太多的噪音，以致不能与某个基音相联系。在「General MIDI」规范中，这些没旋律的打击乐器声在通道10有效。不同的键号对应47种不同的打击乐器。

DRUM程序，如程序22-10所示，是一个计算机击鼓器。此程序让您用47种不同的打击乐器的声音来构造最大到32个音符的一个序列，然后在选择的速度和音量下反复演奏这个序列。

程序22-10 DRUM

DRUM.C

```
/*------------------------------------------------------------------------
DRUM.C -- MIDI Drum Machine
(c) Charles Petzold, 1998
------------------------------------------------------------------------*/
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "drumtime.h"
#include "drumfile.h"
#include "resource.h"

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK AboutProc (HWND, UINT, WPARAM, LPARAM) ;

void DrawRectangle (HDC, int, int, DWORD *, DWORD *) ;
void ErrorMessage (HWND, TCHAR *, TCHAR *) ;
void DoCaption (HWND, TCHAR *) ;
int AskAboutSave (HWND, TCHAR *) ;

TCHAR * szPerc [NUM_PERC] =
{
  TEXT ("Acoustic Bass Drum"), TEXT ("Bass Drum 1"),
    TEXT ("Side Stick"), TEXT ("Acoustic Snare"),
    TEXT ("Hand Clap"), TEXT ("Electric Snare"),
```

```
    TEXT ("Low Floor Tom"), TEXT ("Closed High Hat"),
    TEXT ("High Floor Tom"), TEXT ("Pedal High Hat"),
    TEXT ("Low Tom"), TEXT ("Open High Hat"),
    TEXT ("Low-Mid Tom"), TEXT ("High-Mid Tom"),
    TEXT ("Crash Cymbal 1"), TEXT ("High Tom"),
    TEXT ("Ride Cymbal 1"), TEXT ("Chinese Cymbal"),
    TEXT ("Ride Bell"), TEXT ("Tambourine"),
    TEXT ("Splash Cymbal"), TEXT ("Cowbell"),
    TEXT ("Crash Cymbal 2"), TEXT ("Vibraslap"),
    TEXT ("Ride Cymbal 2"), TEXT ("High Bongo"),
    TEXT ("Low Bongo"), TEXT ("Mute High Conga"),
    TEXT ("Open High Conga"), TEXT ("Low Conga"),
    TEXT ("High Timbale"), TEXT ("Low Timbale"),
    TEXT ("High Agogo"), TEXT ("Low Agogo"),
    TEXT ("Cabasa"), TEXT ("Maracas"),
    TEXT ("Short Whistle"), TEXT ("Long Whistle"),
    TEXT ("Short Guiro"), TEXT ("Long Guiro"),
    TEXT ("Claves"), TEXT ("High Wood Block"),
    TEXT ("Low Wood Block"), TEXT ("Mute Cuica"),
    TEXT ("Open Cuica"), TEXT ("Mute Triangle"),
    TEXT ("Open Triangle")
} ;

TCHAR szAppName [] = TEXT ("Drum") ;
TCHAR szUntitled [] = TEXT ("(Untitled)") ;
TCHAR szBuffer [80 + MAX_PATH] ;
HANDLE hInst ;
int cxChar, cyChar ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
  HWND hwnd ;
  MSG msg ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  wndclass.style = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (hInstance, szAppName) ;
  wndclass.hCursor = LoadCursor (NULL, IDC_ARROW) ;
  wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = szAppName ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, NULL,
    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
    WS_MINIMIZEBOX | WS_HSCROLL | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, szCmdLine) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
```

```
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static BOOL bNeedSave ;
  static DRUM drum ;
  static HMENU hMenu ;
  static int iTempo = 50, iIndexLast ;
  static TCHAR szFileName [MAX_PATH], szTitleName [MAX_PATH] ;
  HDC hdc ;
  int i, x, y ;
  PAINTSTRUCT ps ;
  POINT point ;
  RECT rect ;
  TCHAR * szError ;

  switch (message)
  {
  case WM_CREATE:
    // Initialize DRUM structure
    drum.iMsecPerBeat = 100 ;
    drum.iVelocity = 64 ;
    drum.iNumBeats = 32 ;

    DrumSetParams (&drum) ;

    // Other initialization
    cxChar = LOWORD (GetDialogBaseUnits ()) ;
    cyChar = HIWORD (GetDialogBaseUnits ()) ;

    GetWindowRect (hwnd, &rect) ;
    MoveWindow (hwnd, rect.left, rect.top,
      77 * cxChar, 29 * cyChar, FALSE) ;

    hMenu = GetMenu (hwnd) ;

    // Initialize "Volume" scroll bar
    SetScrollRange (hwnd, SB_HORZ, 1, 127, FALSE) ;
    SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;

    // Initialize "Tempo" scroll bar
    SetScrollRange (hwnd, SB_VERT, 0, 100, FALSE) ;
    SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;

    DoCaption (hwnd, szTitleName) ;
    return 0 ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDM_FILE_NEW:
      if ( bNeedSave && IDCANCEL == AskAboutSave (hwnd, szTitleName))
        return 0 ;

      // Clear drum pattern
      for (i = 0 ; i < NUM_PERC ; i++)
      {
        drum.dwSeqPerc [i] = 0 ;
        drum.dwSeqPian [i] = 0 ;
      }

      InvalidateRect (hwnd, NULL, FALSE) ;
      DrumSetParams (&drum) ;
      bNeedSave = FALSE ;
      return 0 ;

    case IDM_FILE_OPEN:
```

```
    // Save previous file
    if (bNeedSave && IDCANCEL ==
      AskAboutSave (hwnd, szTitleName))
      return 0 ;

    // Open the selected file
    if (DrumFileOpenDlg (hwnd, szFileName, szTitleName))
    {
      szError = DrumFileRead (&drum, szFileName) ;

      if (szError != NULL)
      {
        ErrorMessage (hwnd, szError, szTitleName) ;
        szTitleName [0] = '\0' ;
      }
      else
      {
        // Set new parameters
        iTempo = (int) (50 *
          (log10 (drum.iMsecPerBeat) - 1)) ;

        SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;
        SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;

        DrumSetParams (&drum) ;
        InvalidateRect (hwnd, NULL, FALSE) ;
        bNeedSave = FALSE ;
      }

      DoCaption (hwnd, szTitleName) ;
    }
    return 0 ;
  case IDM_FILE_SAVE:
  case IDM_FILE_SAVE_AS:
    // Save the selected file
    if ((LOWORD (wParam) == IDM_FILE_SAVE && szTitleName [0]) ||
      DrumFileSaveDlg (hwnd, szFileName, szTitleName))
    {
      szError = DrumFileWrite (&drum, szFileName) ;

      if (szError != NULL)
      {
        ErrorMessage (hwnd, szError, szTitleName) ;
        szTitleName [0] = '\0' ;
      }
      else
        bNeedSave = FALSE ;

      DoCaption (hwnd, szTitleName) ;
    }
    return 0 ;

  case IDM_APP_EXIT:
    SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;
    return 0 ;

  case IDM_SEQUENCE_RUNNING:
    // Begin sequence
    if (!DrumBeginSequence (hwnd))
    {
      ErrorMessage (hwnd,
        TEXT ("Could not start MIDI sequence -- ")
        TEXT ("MIDI Mapper device is unavailable!"),
        szTitleName) ;
    }
    else
    {
      CheckMenuItem (hMenu, IDM_SEQUENCE_RUNNING, MF_CHECKED) ;
      CheckMenuItem (hMenu, IDM_SEQUENCE_STOPPED, MF_UNCHECKED) ;
```

```
    }
    return 0 ;

  case IDM_SEQUENCE_STOPPED:
    // Finish at end of sequence
    DrumEndSequence (FALSE) ;
    return 0 ;

  case IDM_APP_ABOUT:
    DialogBox (hInst, TEXT ("AboutBox"), hwnd, AboutProc) ;
    return 0 ;
}
return 0 ;

case WM_LBUTTONDOWN:
case WM_RBUTTONDOWN:
  hdc = GetDC (hwnd) ;

  // Convert mouse coordinates to grid coordinates
  x = LOWORD (lParam) / cxChar - 40 ;
  y = 2 *HIWORD (lParam) / cyChar - 2 ;
  // Set a new number of beats of sequence
  if (x > 0 && x <= 32 && y < 0)
  {
    SetTextColor (hdc, RGB (255, 255, 255)) ;
    TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0, TEXT (":|"), 2);
    SetTextColor (hdc, RGB (0, 0, 0)) ;

    if (drum.iNumBeats % 4 == 0)
      TextOut ( hdc, (40 + drum.iNumBeats) * cxChar, 0,
      TEXT ("."), 1) ;

    drum.iNumBeats = x ;

    TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0, TEXT (":|"), 2);

    bNeedSave = TRUE ;
  }

  // Set or reset a percussion instrument beat
  if (x >= 0 && x < 32 && y >= 0 && y < NUM_PERC)
  {
    if (message == WM_LBUTTONDOWN)
      drum.dwSeqPerc[y] ^= (1 << x) ;
    else
      drum.dwSeqPian[y] ^= (1 << x) ;

    DrawRectangle (hdc, x, y, drum.dwSeqPerc, drum.dwSeqPian) ;

    bNeedSave = TRUE ;
  }

  ReleaseDC (hwnd, hdc) ;
  DrumSetParams (&drum) ;
  return 0 ;

case WM_HSCROLL:
  // Change the note velocity
  switch (LOWORD (wParam))
  {
  case SB_LINEUP: drum.iVelocity -= 1 ; break ;
  case SB_LINEDOWN: drum.iVelocity += 1 ; break ;
  case SB_PAGEUP: drum.iVelocity -= 8 ; break ;
  case SB_PAGEDOWN: drum.iVelocity += 8 ; break ;
  case SB_THUMBPOSITION:
    drum.iVelocity = HIWORD (wParam) ;
    break ;

  default:
```

```
    return 0 ;
  }

  drum.iVelocity = max (1, min (drum.iVelocity, 127)) ;
  SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;
  DrumSetParams (&drum) ;
  bNeedSave = TRUE ;
  return 0 ;

  case WM_VSCROLL:
    // Change the tempo
    switch (LOWORD (wParam))
    {
    case SB_LINEUP: iTempo -= 1 ; break ;
    case SB_LINEDOWN: iTempo += 1 ; break ;
    case SB_PAGEUP: iTempo -= 10 ; break ;
    case SB_PAGEDOWN: iTempo += 10 ; break ;
    case SB_THUMBPOSITION:
      iTempo = HIWORD (wParam) ;
      break ;

    default:
      return 0 ;
    }

    iTempo = max (0, min (iTempo, 100)) ;
    SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;

    drum.iMsecPerBeat = (WORD) (10 * pow (100, iTempo / 100.0)) ;

    DrumSetParams (&drum) ;
    bNeedSave = TRUE ;
    return 0 ;

    case WM_PAINT:
      hdc = BeginPaint (hwnd, &ps) ;

      SetTextAlign (hdc, TA_UPDATECP) ;
      SetBkMode (hdc, TRANSPARENT) ;

      // Draw the text strings and horizontal lines
      for (i = 0 ; i < NUM_PERC ; i++)
      {
        MoveToEx (hdc, i & 1 ? 20 * cxChar : cxChar,
          (2 * i + 3) * cyChar / 4, NULL) ;

        TextOut (hdc, 0, 0, szPerc [i], lstrlen (szPerc [i])) ;

        GetCurrentPositionEx (hdc, &point) ;

        MoveToEx (hdc, point.x + cxChar, point.y + cyChar / 2, NULL) ;
        LineTo (hdc, 39 * cxChar, point.y + cyChar / 2) ;
      }

      SetTextAlign (hdc, 0) ;

      // Draw rectangular grid, repeat mark, and beat marks
      for (x = 0 ; x < 32 ; x++)
      {
        for (y = 0 ; y < NUM_PERC ; y++)
          DrawRectangle (hdc, x, y, drum.dwSeqPerc, drum.dwSeqPian) ;

        SetTextColor (hdc, x == drum.iNumBeats - 1 ?
          RGB (0, 0, 0) : RGB (255, 255, 255)) ;

        TextOut (hdc, (41 + x) * cxChar, 0, TEXT (":|"), 2) ;

        SetTextColor (hdc, RGB (0, 0, 0)) ;
```

```
        if (x % 4 == 0)
          TextOut (hdc, (40 + x) * cxChar, 0, TEXT ("."), 1) ;
      }

      EndPaint (hwnd, &ps) ;
      return 0 ;

    case WM_USER_NOTIFY:
      // Draw the "bouncing ball"
      hdc = GetDC (hwnd) ;

      SelectObject (hdc, GetStockObject (NULL_PEN)) ;
      SelectObject (hdc, GetStockObject (WHITE_BRUSH)) ;

      for (i = 0 ; i < 2 ; i++)
      {
        x = iIndexLast ;
        y = NUM_PERC + 1 ;

        Ellipse (hdc, (x + 40) * cxChar, (2 * y + 3) * cyChar / 4,
          (x + 41) * cxChar, (2 * y + 5) * cyChar / 4);

        iIndexLast = wParam ;
        SelectObject (hdc, GetStockObject (BLACK_BRUSH)) ;
      }

      ReleaseDC (hwnd, hdc) ;
      return 0 ;

    case WM_USER_ERROR:
      ErrorMessage (hwnd, TEXT ("Can't set timer event for tempo"),
        szTitleName) ;
      // fall through
    case WM_USER_FINISHED:
      DrumEndSequence (TRUE) ;
      CheckMenuItem (hMenu, IDM_SEQUENCE_RUNNING, MF_UNCHECKED) ;
      CheckMenuItem (hMenu, IDM_SEQUENCE_STOPPED, MF_CHECKED) ;
      return 0 ;

    case WM_CLOSE:
      if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd, szTitleName))
        DestroyWindow (hwnd) ;

      return 0 ;

    case WM_QUERYENDSESSION:
      if (!bNeedSave || IDCANCEL != AskAboutSave (hwnd, szTitleName))
        return 1L ;

      return 0 ;

    case WM_DESTROY:
      DrumEndSequence (TRUE) ;
      PostQuitMessage (0) ;
      return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutProc ( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
  case WM_INITDIALOG:
    return TRUE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
```

```
    case IDOK:
      EndDialog (hDlg, 0) ;
      return TRUE ;
    }
    break ;
  }
  return FALSE ;
}

void DrawRectangle (HDC hdc, int x, int y, DWORD * dwSeqPerc,
                    DWORD * dwSeqPian)
{
  int iBrush ;
  if (dwSeqPerc [y] & dwSeqPian [y] & (1L << x))
    iBrush = BLACK_BRUSH ;
  else if (dwSeqPerc [y] & (1L << x))
    iBrush = DKGRAY_BRUSH ;
  else if (dwSeqPian [y] & (1L << x))
    iBrush = LTGRAY_BRUSH ;
  else
    iBrush = WHITE_BRUSH ;
  SelectObject (hdc, GetStockObject (iBrush)) ;
  Rectangle (hdc, (x + 40) * cxChar , (2 * y + 4) * cyChar / 4,
    (x + 41) * cxChar + 1, (2 * y + 6) * cyChar / 4 + 1) ;
}

void ErrorMessage (HWND hwnd, TCHAR * szError, TCHAR * szTitleName)
{
  wsprintf (szBuffer, szError,
    (LPSTR) (szTitleName [0] ? szTitleName : szUntitled)) ;
  MessageBeep (MB_ICONEXCLAMATION) ;
  MessageBox (hwnd, szBuffer, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

void DoCaption (HWND hwnd, TCHAR * szTitleName)
{
  wsprintf (szBuffer, TEXT ("MIDI Drum Machine - %s"),
    (LPSTR) (szTitleName [0] ? szTitleName : szUntitled)) ;
  SetWindowText (hwnd, szBuffer) ;
}

int AskAboutSave (HWND hwnd, TCHAR * szTitleName)
{
  int iReturn ;
  wsprintf (szBuffer, TEXT ("Save current changes in %s?"),
    (LPSTR) (szTitleName [0] ? szTitleName : szUntitled)) ;
  iReturn = MessageBox (hwnd, szBuffer, szAppName,
    MB_YESNOCANCEL | MB_ICONQUESTION) ;

  if (iReturn == IDYES)
    if (!SendMessage (hwnd, WM_COMMAND, IDM_FILE_SAVE, 0))
      iReturn = IDCANCEL ;
    return iReturn ;
}
```

DRUMTIME.H

```
/*------------------------------------------------------------------------
DRUMTIME.H Header File for Time Functions for DRUM Program
------------------------------------------------------------------------*/
#define NUM_PERC  47
#define WM_USER_NOTIFY (WM_USER + 1)
#define WM_USER_FINISHED (WM_USER + 2)
#define WM_USER_ERROR (WM_USER + 3)

#pragma pack(push, 2)
typedef struct
{
```

```
  short iMsecPerBeat ;
  short iVelocity ;
  short iNumBeats ;
  DWORD dwSeqPerc [NUM_PERC] ;
  DWORD dwSeqPian [NUM_PERC] ;
}
DRUM, * PDRUM ;
#pragma pack(pop)
void DrumSetParams (PDRUM) ;
BOOL DrumBeginSequence (HWND) ;
void DrumEndSequence (BOOL) ;
```

DRUMTIME.C

```
/*-----------------------------------------------------------------------------
DRUMFILE.C --Timer Routines for DRUM
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include "drumtime.h"

#define minmax(a,x,b) (min (max (x, a), b))
#define TIMER_RES 5
void CALLBACK DrumTimerFunc (UINT, UINT, DWORD, DWORD, DWORD) ;
BOOL bSequenceGoing, bEndSequence ;
DRUM drum ;
HMIDIOUT hMidiOut ;
HWND hwndNotify ;
int iIndex ;
UINT uTimerRes, uTimerID ;

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                  int iData1, int iData2)
{
  DWORD dwMessage ;
  dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
  return midiOutShortMsg (hMidi, dwMessage) ;
}

void DrumSetParams (PDRUM pdrum)
{
  CopyMemory (&drum, pdrum, sizeof (DRUM)) ;
}

BOOL DrumBeginSequence (HWND hwnd)
{
  TIMECAPS tc ;
  hwndNotify = hwnd ; // Save window handle for notification
  DrumEndSequence (TRUE) ; // Stop current sequence if running

  // Open the MIDI Mapper output port
  if (midiOutOpen (&hMidiOut, MIDIMAPPER, 0, 0, 0))
    return FALSE ;
  // Send Program Change messages for channels 9 and 0
  MidiOutMessage (hMidiOut, 0xC0, 9, 0, 0) ;
  MidiOutMessage (hMidiOut, 0xC0, 0, 0, 0) ;

  // Begin sequence by setting a timer event
  timeGetDevCaps (&tc, sizeof (TIMECAPS)) ;
  uTimerRes = minmax (tc.wPeriodMin, TIMER_RES, tc.wPeriodMax) ;
  timeBeginPeriod (uTimerRes) ;

  uTimerID = timeSetEvent(max ((UINT) uTimerRes, (UINT) drum.iMsecPerBeat),
    uTimerRes, DrumTimerFunc, 0, TIME_ONESHOT) ;

  if (uTimerID == 0)
  {
    timeEndPeriod (uTimerRes) ;
```

```
    midiOutClose (hMidiOut) ;
    return FALSE ;
  }

  iIndex = -1 ;
  bEndSequence = FALSE ;
  bSequenceGoing = TRUE ;

  return TRUE ;
}

void DrumEndSequence (BOOL bRightAway)
{
  if (bRightAway)
  {
    if (bSequenceGoing)
    {
      // stop the timer
      if (uTimerID)
        timeKillEvent (uTimerID) ;
      timeEndPeriod (uTimerRes) ;

      // turn off all notes
      MidiOutMessage (hMidiOut, 0xB0, 9, 123, 0) ;
      MidiOutMessage (hMidiOut, 0xB0, 0, 123, 0) ;
      // close the MIDI port midiOutClose (hMidiOut) ; bSequenceGoing = FALSE ;
    }
  }
  else
    bEndSequence = TRUE ;
}

void CALLBACK DrumTimerFunc ( UINT uID, UINT uMsg, DWORD dwUser,
                         DWORD dw1, DWORD dw2)
{
  static DWORD dwSeqPercLast [NUM_PERC], dwSeqPianLast [NUM_PERC] ;
  int i ;

  // Note Off messages for channels 9 and 0
  if (iIndex != -1)
  {
    for (i = 0 ; i < NUM_PERC ; i++)
    {
      if (dwSeqPercLast[i] & 1 << iIndex)
        MidiOutMessage (hMidiOut, 0x80, 9, i + 35, 0) ;
      if (dwSeqPianLast[i] & 1 << iIndex)
        MidiOutMessage (hMidiOut, 0x80, 0, i + 35, 0) ;
    }
  }

  // Increment index and notify window to advance bouncing ball
  iIndex = (iIndex + 1) % drum.iNumBeats ;
  PostMessage (hwndNotify, WM_USER_NOTIFY, iIndex, timeGetTime ()) ;

  // Check if ending the sequence
  if (bEndSequence && iIndex == 0)
  {
    PostMessage (hwndNotify, WM_USER_FINISHED, 0, 0L) ;
    return ;
  }

  // Note On messages for channels 9 and 0
  for (i = 0 ; i < NUM_PERC ; i++)
  {
    if (drum.dwSeqPerc[i] & 1 << iIndex)
      MidiOutMessage (hMidiOut, 0x90, 9, i + 35, drum.iVelocity) ;
    if (drum.dwSeqPian[i] & 1 << iIndex)
      MidiOutMessage (hMidiOut, 0x90, 0, i + 35, drum.iVelocity) ;
    dwSeqPercLast[i] = drum.dwSeqPerc[i] ;
```

```
    dwSeqPianLast[i] = drum.dwSeqPian[i] ;
  }
  // Set a new timer event
  uTimerID = timeSetEvent (max ((int) uTimerRes, drum.iMsecPerBeat),
    uTimerRes, DrumTimerFunc, 0, TIME_ONESHOT) ;
  if (uTimerID == 0)
  {
    PostMessage (hwndNotify, WM_USER_ERROR, 0, 0) ;
  }
}
```

DRUMFILE.H

```
/*-----------------------------------------------------------------------------
DRUMFILE.H Header File for File I/O Routines for DRUM
-----------------------------------------------------------------------------*/
BOOL DrumFileOpenDlg (HWND, TCHAR *, TCHAR *) ;
BOOL DrumFileSaveDlg (HWND, TCHAR *, TCHAR *) ;

TCHAR * DrumFileWrite (DRUM *, TCHAR *) ;
TCHAR * DrumFileRead (DRUM *, TCHAR *) ;
```

DRUMFILE.C

```
/*-----------------------------------------------------------------------------
DRUMFILE.C -- File I/O Routines for DRUM
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include <commdlg.h>
#include "drumtime.h"
#include "drumfile.h"
OPENFILENAME ofn = { sizeof (OPENFILENAME) } ;
TCHAR * szFilter[] = { TEXT ("Drum Files (*.DRM)"),
TEXT ("*.drm"), TEXT ("") } ;

TCHAR szDrumID []= TEXT ("DRUM") ;
TCHAR szListID [] = TEXT ("LIST") ;
TCHAR szInfoID [] = TEXT ("INFO") ;
TCHAR szSoftID [] = TEXT ("ISFT") ;
TCHAR szDateID [] = TEXT ("ISCD") ;
TCHAR szFmtID [] = TEXT ("fmt ") ;
TCHAR szDataID [] = TEXT ("data") ;
char szSoftware []= "DRUM by Charles Petzold, Programming Windows" ;

TCHAR szErrorNoCreate [] = TEXT ("File %s could not be opened for writing.");
TCHAR szErrorCannotWrite [] = TEXT ("File %s could not be written to. ") ;
TCHAR szErrorNotFound [] = TEXT ("File %s not found or cannot be opened.") ;
TCHAR szErrorNotDrum [] = TEXT ("File %s is not a standard DRUM file.") ;
TCHAR szErrorUnsupported [] = TEXT ("File %s is not a supported DRUM file.") ;
TCHAR szErrorCannotRead [] = TEXT ("File %s cannot be read.") ;

BOOL DrumFileOpenDlg (HWND hwnd, TCHAR * szFileName, TCHAR * szTitleName)
{
  ofn.hwndOwner = hwnd ;
  ofn.lpstrFilter = szFilter [0] ;
  ofn.lpstrFile = szFileName ;
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = szTitleName ;
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.Flags = OFN_CREATEPROMPT ;
  ofn.lpstrDefExt = TEXT ("drm") ;

  return GetOpenFileName (&ofn) ;
}
```

```
BOOL DrumFileSaveDlg ( HWND hwnd, TCHAR * szFileName,
                       TCHAR * szTitleName)
{
  ofn.hwndOwner = hwnd ;
  ofn.lpstrFilter = szFilter [0] ;
  ofn.lpstrFile = szFileName ;
  ofn.nMaxFile = MAX_PATH ;
  ofn.lpstrFileTitle = szTitleName ;
  ofn.nMaxFileTitle = MAX_PATH ;
  ofn.Flags = OFN_OVERWRITEPROMPT ;
  ofn.lpstrDefExt = TEXT ("drm") ;

  return GetSaveFileName (&ofn) ;
}

TCHAR * DrumFileWrite (DRUM * pdrum, TCHAR * szFileName)
{
  char szDateBuf [16] ;
  HMMIO hmmio ;
  int iFormat = 2 ;
  MMCKINFO mmckinfo [3] ;
  SYSTEMTIME st ;
  WORD wError = 0 ;

  memset (mmckinfo, 0, 3 * sizeof (MMCKINFO)) ;
  // Recreate the file for writing
  if ((hmmio = mmioOpen (szFileName, NULL,
    MMIO_CREATE | MMIO_WRITE | MMIO_ALLOCBUF)) == NULL)
    return szErrorNoCreate ;
  // Create a "RIFF" chunk with a "CPDR" type
  mmckinfo[0].fccType = mmioStringToFOURCC (szDrumID, 0) ;
  wError |= mmioCreateChunk (hmmio, &mmckinfo[0], MMIO_CREATERIFF) ;
  // Create "LIST" sub-chunk with an "INFO" type
  mmckinfo[1].fccType = mmioStringToFOURCC (szInfoID, 0) ;
  wError |= mmioCreateChunk (hmmio, &mmckinfo[1], MMIO_CREATELIST) ;
  // Create "ISFT" sub-sub-chunk
  mmckinfo[2].ckid = mmioStringToFOURCC (szSoftID, 0) ;
  wError |= mmioCreateChunk (hmmio, &mmckinfo[2], 0) ;
  wError |= (mmioWrite (hmmio, szSoftware, sizeof (szSoftware)) !=
    sizeof (szSoftware)) ;
  wError |= mmioAscend (hmmio, &mmckinfo[2], 0) ;
  // Create a time string
  GetLocalTime (&st) ;
  wsprintfA (szDateBuf, "%04d-%02d-%02d", st.wYear, st.wMonth, st.wDay) ;
  // Create "ISCD" sub-sub-chunk
  mmckinfo[2].ckid = mmioStringToFOURCC (szDateID, 0) ;
  wError |= mmioCreateChunk (hmmio, &mmckinfo[2], 0) ;
  wError |= (mmioWrite (hmmio, szDateBuf, (strlen (szDateBuf) + 1)) !=
    (int) (strlen (szDateBuf) + 1)) ;
  wError |= mmioAscend (hmmio, &mmckinfo[2], 0) ;
  wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;

  // Create "fmt " sub-chunk
  mmckinfo[1].ckid = mmioStringToFOURCC (szFmtID, 0) ;
  wError |= mmioCreateChunk (hmmio, &mmckinfo[1], 0) ;
  wError |= (mmioWrite (hmmio, (PSTR) &iFormat, sizeof (int)) !=
    sizeof (int)) ;
  wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;
  // Create the "data" sub-chunk
  mmckinfo[1].ckid = mmioStringToFOURCC (szDataID, 0) ;
  wError |= mmioCreateChunk (hmmio, &mmckinfo[1], 0) ;
  wError |= (mmioWrite (hmmio, (PSTR) pdrum, sizeof (DRUM)) !=
    sizeof (DRUM)) ;
  wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;
  wError |= mmioAscend (hmmio, &mmckinfo[0], 0) ;

  // Clean up and return
  wError |= mmioClose (hmmio, 0) ;
  if (wError)
```

```
  {
    mmioOpen (szFileName, NULL, MMIO_DELETE) ;
    return szErrorCannotWrite ;
  }
  return NULL ;
}

TCHAR * DrumFileRead (DRUM * pdrum, TCHAR * szFileName)
{
  DRUM drum ;
  HMMIO hmmio ;
  int i, iFormat ;
  MMCKINFO mmckinfo [3] ;

  ZeroMemory (mmckinfo, 2 * sizeof (MMCKINFO)) ;

  // Open the file
  if ((hmmio = mmioOpen (szFileName, NULL, MMIO_READ)) == NULL)
    return szErrorNotFound ;
  // Locate a "RIFF" chunk with a "DRUM" form-type
  mmckinfo[0].ckid = mmioStringToFOURCC (szDrumID, 0) ;
  if (mmioDescend (hmmio, &mmckinfo[0], NULL, MMIO_FINDRIFF))
  {
    mmioClose (hmmio, 0) ;
    return szErrorNotDrum ;
  }

  // Locate, read, and verify the "fmt " sub-chunk
  mmckinfo[1].ckid = mmioStringToFOURCC (szFmtID, 0) ;
  if (mmioDescend (hmmio, &mmckinfo[1], &mmckinfo[0], MMIO_FINDCHUNK))
  {
    mmioClose (hmmio, 0) ;
    return szErrorNotDrum ;
  }
  if (mmckinfo[1].cksize != sizeof (int))
  {
    mmioClose (hmmio, 0) ;
    return szErrorUnsupported ;
  }
  if (mmioRead (hmmio, (PSTR) &iFormat, sizeof (int)) != sizeof (int))
  {
    mmioClose (hmmio, 0) ;
    return szErrorCannotRead ;
  }

  if (iFormat != 1 && iFormat != 2)
  {
    mmioClose (hmmio, 0) ;
    return szErrorUnsupported ;
  }
  // Go to end of "fmt " sub-chunk
  mmioAscend (hmmio, &mmckinfo[1], 0) ;
  // Locate, read, and verify the "data" sub-chunk
  mmckinfo[1].ckid = mmioStringToFOURCC (szDataID, 0) ;
  if (mmioDescend (hmmio, &mmckinfo[1], &mmckinfo[0], MMIO_FINDCHUNK))
  {
    mmioClose (hmmio, 0) ;
    return szErrorNotDrum ;
  }
  if (mmckinfo[1].cksize != sizeof (DRUM))
  {
    mmioClose (hmmio, 0) ;
    return szErrorUnsupported ;
  }
  if (mmioRead (hmmio, (LPSTR) &drum, sizeof (DRUM)) != sizeof (DRUM))
  {
    mmioClose (hmmio, 0) ;
    return szErrorCannotRead ;
  }
```

```
  // Close the file
  mmioClose (hmmio, 0) ;
  // Convert format 1 to format 2 and copy the DRUM structure data
  if (iFormat == 1)
  {
    for (i = 0 ; i < NUM_PERC ; i++)
    {
      drum.dwSeqPerc [i] = drum.dwSeqPian [i] ;
      drum.dwSeqPian [i] = 0 ;
    }
  }

  memcpy (pdrum, &drum, sizeof (DRUM)) ;
  return NULL ;
}
```

DRUM.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Menu
DRUM  MENU DISCARDABLE
BEGIN
POPUP "&File"
BEGIN
MENUITEM "&New",  IDM_FILE_NEW
MENUITEM "&Open...", IDM_FILE_OPEN
MENUITEM "&Save",   IDM_FILE_SAVE
MENUITEM "Save &As...", IDM_FILE_SAVE_AS
MENUITEM SEPARATOR
MENUITEM "E&xit",  IDM_APP_EXIT
END
POPUP "&Sequence"
BEGIN
MENUITEM "&Running", IDM_SEQUENCE_RUNNING
MENUITEM "&Stopped", IDM_SEQUENCE_STOPPED
, CHECKED
END
POPUP "&Help"
BEGIN
MENUITEM "&About...", IDM_APP_ABOUT
END
END

//////////////////////////////////////////////////////////////////////////////
// Icon
DRUM   ICON   DISCARDABLE   "drum.ico"

//////////////////////////////////////////////////////////////////////////////
// Dialog
ABOUTBOX DIALOG DISCARDABLE 20, 20, 160, 164
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON "OK",IDOK,54,143,50,14
ICON  "DRUM",IDC_STATIC,8,8,21,20
CTEXT  "DRUM",IDC_STATIC,34,12,90,8
CTEXT  "MIDI Drum Machine",IDC_STATIC,7,36,144,8
CONTROL  "",IDC_STATIC,"Static",SS_BLACKFRAME,8,88,144,46
LTEXT  "Left Button:\t\tDrum sounds",IDC_STATIC,12,92,136,8
LTEXT  "Right Button:\t\tPiano sounds",IDC_STATIC,12,102,136,8
LTEXT   "Horizontal Scroll:\t\tVelocity",IDC_STATIC,12,112,136,8
LTEXT   "Vertical Scroll:\t\tTempo",IDC_STATIC,12,122,136,8
CTEXT  "Copyright (c) Charles Petzold, 1998",IDC_STATIC,8,48, 144,8
```

```
CTEXT  """Programming Windows,"" 5th Edition",IDC_STATIC,8,60, 144,8
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Drum.rc
#define  IDM_FILE_NEW      40001
#define  IDM_FILE_OPEN      40002
#define  IDM_FILE_SAVE      40003
#define  IDM_FILE_SAVE_AS     40004
#define  IDM_APP_EXIT      40005
#define  IDM_SEQUENCE_RUNNING    40006
#define  IDM_SEQUENCE_STOPPED    40007
#define  IDM_APP_ABOUT      40008
```

当第一次执行DRUM时，您将看到在窗口中有两列，左边一列按名称列出了47种不同的打击乐器。右边的网格是打击乐器的声音与时间的二维数组。每一个打击器都对应网格中的一列。32行就是32拍。如果要让这32拍出现在一个4/4拍的小节中（即每小节4个四分音符），那么每1拍对应一个三十二分音符。

从「Sequence」菜单选择「Running」时，程序将试图打开MIDI Mapper设备。如果失败，屏幕将出现一个消息框。否则，您将看到一个「跳动的小球」随演奏的节拍在网格底部跳过。

在网格的任何位置单击鼠标左键可以在此拍中演奏打击乐器的声音，这时区域将变成暗灰色。用鼠标右键还可以添加钢琴的拍子，这时区域将会变成亮灰色。如果按下两个键（同时或分别），此区域将变成黑色，而且可以同时听到打击乐器和钢琴的声音。再次单击其中的一个键或双键将关闭该拍中的声音。

网格上部是每4拍一个点。这些点使我们不用过多的计算就可以很简易地确定单击的位置。网格的右上角是一个冒号和一条竖线（:|），它们看起来像传统音乐符号中的反复记号。这个符号表示序列的长度。您可以通过单击鼠标来将反复记号放置于网格内的任意位置。该序列最多（但不包括）只能演奏反复记号以内的拍子。如果要建立华尔兹节奏，则应将反复记号设定为3拍的若干倍。

水平滚动条控制MIDI Note On消息中的速率字节。这虽然能改变一些合成器的音质，但一般会影响音量。程序起初将速率滚动条设定在中间位置。竖直滚动条控制拍子。这是对数刻度，范围从每拍1秒（滚动条在底部）到每拍10毫秒（滚动条在顶部）。程序最初将拍子设定为每拍100毫秒（1/10秒），这时滚动条在中间。

「File」菜单允许您储存和读取扩展名为.DRM的文件，这是我定义的一种格式。这些文件很小并采用了RIFF的文件格式，这是一种所有新的多媒体数据文件推荐使用的格式。「Help」菜单中的「About」选项显示一个对话框，该对话框用一段非常简明的摘要来说明鼠标在网格中的用法以及两个滚动条的功能。

最后，「Sequence」菜单中的「Stopped」选项用于目前序列结束后终止乐曲并关闭MIDI Mapper设备。

多媒体time函数

您可能会注意到DRUM.C没有呼叫任何多媒体函数。而所有的实际操作都发生在DRUMTIME模块中。

虽然普通的Windows定时器使用起来很简单，但它对实时时间应用却有灾难性的影响。就像我们在BACHTOCC程序中所看到的一样，演奏音乐就是这样的一种实时时间应用，对此Windows定时器是不合适的。为了提供在PC上演奏MIDI所需要的精确度，多媒体API还包括一个高分辨率的

定时器，此定时器通过7个前缀是time的函数实作。这些函数有一个是多余的，而DRUMTIME展示了其余6个函数的用途。定时器函数将处理执行在一个单独执行绪中的callback函数。系统将按照程序指定的定时器延迟时间来呼叫定时器。

处理多媒体定时器时，可以用毫秒指定两种不同的时间。第一个是延迟时间，第二个称为分辨率。您可以认为分辨率是容错误差。如果指定一个延迟100毫秒，而分辨率是10毫秒，则定时器的实际延迟范围在90到110毫秒之间。

使用定时器之前，应获得定时器的设备能力：

timeGetDevCaps (&timecaps, uSize) ;

第一个参数是TIMECAPS型态结构的指针，第二个参数是此结构的大小。TIMECAPS结构只有两个字段，wPeriodMin和wPeriodMax。这是定时器设备驱动程序所支持的最小和最大的分辨率值。如果呼叫timeGetDevCaps后再查看这些值，会发现wPeriodMin是1而wPeriodMax是65535，所以此函数并不是很重要。不过，得到这些分辨率值并用于其它定时器函数呼叫是个好主意。

下一步呼叫

timeBeginPeriod (uResolution) ;

来指出程序所需要的定时器分辨率的最低值。该值应在TIMECAPS结构所确定的范围之内。此呼叫允许为可能使用定时器的多个程序提供最好的定时器设备驱动程序。呼叫timeBeginPeriod及timeEndPeriod必须成对出现，我将在后面对timeEndPeriod作简短的描述。

现在可以真正设定一个定时器事件：

idTimer = timeSetEvent (uDelay, uResolution, CallBackFunc, dwData, uFlag) ;

如果发生错误，从呼叫传回的idTimer将是0。在呼叫的下面，将从Windows里用uDelay毫秒来呼叫CallBackFunc函数，其中允许的误差由uResolution指定。uResolution值必须大于或等于传递给timeBeginPeriod的分辨率。dwData是程序定义的数据，后来传递给CallBackFunc。最后一个参数可以是TIME_ONESHOT，也可以是TIME_PERIODIC。前者用于在uDelay毫秒数中获得一次CallBackFunc呼叫，而后者用于每个uDelay毫秒都获得一次CallBackFunc呼叫。

要在呼叫CallBackFunc之前终止只发生一次的定时器事件，或者暂停周期性的定时器事件，请呼叫

timeKillEvent (idTimer) ;

呼叫CallBackFunc后不必删除只发生一次的定时器事件。在程序中用完定时器以后，请呼叫

timeEndPeriod (wResolution) ;

其中的参数与传递给timeBeginPeriod的相同。

另两个函数的前缀是time。函数

dwSysTime = timeGetTime () ;

传回从Windows第一次启动到现在的系统时间，单位是毫秒。函数

timeGetSystemTime (&mmtime, uSize) ;

需要一个MMTIME结构的指针（与第一个参数一样），以及此结构的大小（与第二个参数一样）。虽然MMTIME结构可以在其它环境中用来得到非毫秒格式的系统时间，但此例中它都传回毫秒时间。所以timeGetSystemTime是多余的。

Callback 函数只限于它所能做的 Windows 函数呼叫中。Callback 函数可以呼叫 PostMessage，PostMessage 包含有四个定时器函数（timeSetEvent、timeKillEvent、timeGetTime 和多余的 timeGetSystemTime）、两个 MIDI 输出函数（midiOutShortMsg 和 midiOutLongMsg）以及调试函数OutputDebugStr。

很明显，设计多媒体定时器主要是用于MIDI序列而很少用于其它方面。当然，可以使用PostMessage来通知定时器事件的窗口消息处理程序，而且窗口消息处理程序可以做任何它想做的事，只是不能响应定时器callback自身的准确性。

Callback函数有五个参数，但只使用了其中两个参数：从timeSetEvent传回的定时器ID和最初作为参数传递给timeSetEvent的dwData值。

DRUM.C模块呼叫DRUMTIME.C中的DrumSetParams函数有很多次 – 建立DRUM窗口时、使用者在网格上单击或者移动滚动条时、从磁盘上加载.DRM文件时以及清除网格时。DrumSetParams的唯一的参数是指向DRUM型态结构的指针，此结构型态在DRUMTIME.H定义。该结构以毫秒为单位储存拍子时间、速度（通常对应于音量）、序列中的拍数以及用于储存网格（为打击乐器和钢琴声设定）的两套47个32字节的整数。这些32位整数中的每一位都对应序列的一拍。DRUM.C模块将在静态内存中维护一个DRUM型态的结构，并在呼叫DrumSetParams时向它传递一个指标。DrumSetParams只简单地复制此结构的内容。

要启动序列，DRUM呼叫DRUMTIME中的DrumBeginSequence函数。唯一的参数就是窗口句柄，其作用是通知。DrumBeginSequence打开MIDI Mapper输出设备，如果成功，则发送Program Change消息来为MIDI信道0和9选择乐器声音（这些通道是基于0的，所以9实际指的是MIDI通道10，即打击乐器通道。另一个通道用于钢琴声）。DrumBeginSequence 透过呼叫 timeGetDevCaps和timeBeginPeriod来继续工作。在TIMER_RES定义的理想定时器分辨率通常是5毫秒，但我定义了一个称作minmax的宏来计算从timeGetDevCaps传回的限制范围以内的分辨率。

下一个呼叫是timeSetEvent，用于确定拍子时间，计算分辨率、callback函数DrumTimerFunc以及TIME_ONESHOT常数。DRUMTIME用的是只发生一次的定时器，而不是周期性定时器，所以速度可以随序列的执行而动态变化。timeSetEvent呼叫之后，定时器设备驱动程序将在延迟时间结束以后呼叫DrumTimerFunc。

DrumTimerFunccallback是DRUMTIME.C中的函数，在DRUMTIME.C中有许多重要的操作。变量iIndex储存序列中目前的拍子。Callback从为目前演奏的声音发送MIDI Note Off消息开始。iIndex的初始值-1以防止第一次启动序列时发生这种情况。

接下来，iIndex递增并将其值连同使用者定义的一个WM_USER_NOTIFY消息一起传递给DRUM中的窗口句柄。wParam消息参数设定为iIndex，以便在DRUM.C中，WndProc能够移动网格底部的「跳动的小球」。

DrumTimerFunc将下列事件作为结束：把Note On消息发送给信道0和9的合成器上，并储存网格值以便下一次可以关闭声音，然后透过呼叫timeSetEvent来设定新的只发生一次的定时器事件。

要停止序列，DRUM呼叫DrumEndSequence，其中唯一的参数可以设定为TRUE或FALSE。如果是TRUE，则DrumEndSequence按下面的程序立即结束序列：删除所有待决的定时器事件，呼叫timeEndPeriod，向两个MIDI信道发送「all notes off」消息，然后关闭MIDI输出埠。当使用者决定终止程序时，DRUM用TRUE参数呼叫DrumEndSequence。

然而，当使用者在DRUM里的「Sequence」菜单中选择「Stop」时，程序将用FALSE作为参

数呼叫DrumEndSequence。这就允许序列在结束之前完成目前的循环。DrumEndSequence透过把bEndSequence整体变量设定为NULL来响应此呼叫。如果bEndSequence是TRUE，并且拍子的索引值设定为0，则DrumTimerFunc把使用者定义的WM_USER_FINISHED消息发送给WndProc。WndProc必须通过用TRUE作为参数呼叫DrumEndSequence来响应该消息，以便正确地结束定时器和MIDI埠的使用。

RIFF文件I/O

DRUM程序也可以储存和检索储存在DRUM结构中信息的文件。这些文件格式都是RIFF（Resource Interchange File Format：资源交换文件格式），即一般建议使用的多媒体文件型态。当然，您可以用标准文件I/O函数来读写RIFF文件，但更简便的方法是使用前缀是mmio（对「多媒体输入/输出」）的函数。

检查.WAV格式时我们发现，RIFF是标记文件格式，这意味着文件中的数据由不同长度的数据块组成。每个数据块都用一个标记来识别。一个标记就是一个4字节的ASCII字符串。这与32位整数的标记名称相比要容易些。标记的后面是数据块长度及其数据。因为文件中的信息不是位于文件开头固定的偏移量而是用标记定义，所以标记文件格式是通用的。这样，可以透过添加附加标记来增强文件格式。在读文件时，程序可以很容易地找到所需要的数据并跳过不需要的或者不理解的标记。

Windows中的RIFF文件由独立的数据块组成。一个数据块可以分为数据块类型、数据块大小以及数据本身。数据块类型是4字符的ASCII码标记，标记中间不能有空格，但末尾可以有。数据块大小是一个4字节（32位）的值，用于显示数据块的大小。数据本身必须占用偶数个字节，必要时可以在结尾补0。这样，数据块的每个部分都是从文件开头就字组对齐好了的。数据块大小不包括数据块类型和数据块大小所需要的8字节，并且不反映添加的数据。

对于一些数据块类型，数据块大小与特定文件无关，是相同的。在数据块是包含信息的固定长度的结构时，就是这种情况。其它情况下，数据块大小根据特定文件变化。

有两个特殊型态的数据块分别称为RIFF数据块和LIST数据块。其中，数据以一个4字符ASCII形式型态开始，后面是一个或多个子数据块。LIST数据块与RIFF数据块类似，只是数据以4字符的ASCII列表型态开始。RIFF数据块用于所有的RIFF文件，而LIST数据块只在文件内部用来合并相关子数据块。

一个RIFF文件就是一个RIFF数据块。因此，RIFF文件以字符串「RIFF」和一个表示文件长度减去8字节的32位值开始。（实际上，如果需要补充数据则文件可能会长一个字节。）

多媒体API包括16个前缀是mmio的函数，这些函数是专门为RIFF文件设计的。DRUMFILE.C中已经用到其中几个函数来读写DRUM数据文件。

要用mmio函数打开文件，则第一步是呼叫mmioOpen。函数传回一个文件句柄。mmioCreateChunk函数在文件中建立一个数据块，这使用MMCKINFO定义的数据块名称和特征。mmioWrite函数写入数据块。写完数据块以后，呼叫mmioAscend。传递给mmioAscend的MMCKINFO结构必须与前面通过传递给mmioCreateChunk来建立数据块的MMCKINFO结构相同。通过从目前文件指针中减去结构的dwDataOffset字段来执行mmioAscend函数，此文件指标现在位于数据块的结尾，并且此值储存在数据的前面。如果数据块在长度上不是2字节的倍数，则mmioAscend函数也填补数据。

RIFF文件由巢状组织的数据块套迭组成。为使mmioAscend正常工作，必须维护多个MMCKINFO结构，每个结构与文件中的一个曾级相联系。DRUM数据文件共有三级。因此，在DRUMFILE.C中的DrumFileWrite函数中，我为三个MMCKINFO结构定义了一个数组，可以分别标记为mmckinfo[0]、mmckinfo[1]和mmckinfo[2]。在第一次mmioCreateChunk呼叫中，mmckinfo[0]结构与DRUM形式型态一起用于建立RIFF型态的块。其后是第二次

mmioCreateChunk呼叫，它用mmckinfo[1]与INFO列表型态一起建立LIST型态的数据块。

第三次mmioCreateChunk呼叫用mmckinfo[2]建立一个ISFT型态的数据块，此数据块用于识别建立数据文件的软件。下面的mmioWrite呼叫用于写字符串szSoftware，呼叫mmioAscent可用mmckinfo[2]来填充此数据块的数据块大小字段。这是第一个完整的数据块。下一个数据块也在LIST数据块内。程序继续用另一个mmioCreateChunk来呼叫建立ISCD（creation data：建立数据）数据块，并再次使用mmckinfo[2]。在mmioWrite呼叫来写入数据块以后，使用mmckinfo[2]呼叫mmioAscend来填充数据块大小。现在写到了此数据块的结尾，也是LIST块的结尾。所以，要填充LIST数据块的数据块大小字段，可再次呼叫mmioAscend，这次使用mmckinfo[1]，它最初用于建立LIST数据块。

要建立「fmt」和「data」数据块，mmioCreateChunk使用mmckinfo[1]；mmioWrite呼叫的后面也使用mmckinfo[1]的mmioAscend。在这一点上，除了RIFF数据块本身以外，所有的数据块大小都填好了。这需要多次使用mmckinfo[0]来呼叫mmioAscend。虽然有多次呼叫，但只呼叫mmioClose一次。

看起来好像mmioAscend呼叫改变了目前的文件指标，而且它的确填充了数据块大小，但在函数传回时，在数据块结束（或可能因补充数据而增加1字节）以后，文件指针恢复到以前的位置。从应用的观点来看，所有的文件写入都是按从头到尾的顺序。

mmioOpen呼叫成功后，除了磁盘空间耗尽之外，不会发生其它错误。使用变量wError从mmioCreateChunk、mmioWrite、mmioAscend和mmioClose呼叫累计错误代码，如果磁盘空间不足则每个呼叫都会失败。如果发生了错误，则mmioOpen以MMIO_DELETE常数为参数来删除文件，并传回错误信息。

读RIFF文件与建立RIFF文件类似，只不过是呼叫mmioRead而不是mmioWrite，呼叫mmioDescend而不是mmioCreateChunk。「下降」（descend）到一个数据块，是指找到数据块位置，并把文件指针移动到数据块大小之后（或者在RIFF或LIST数据块类型的形式型态或者列表型态的后面）。从数据块「上升」指的是把文件指标移动到数据块的结尾。mmioDescend和mmioAscend函数都不能把文件指标移到文件的前一个位置。

DRUM以前的版本在1992年的《PC Magazine》发表。那时，Windows支持两个不同等级的MIDI合成器（称为「基本的」和「扩展的」）。那个程序写的文件有格式标识符1。本章的DRUM程序将格式标识符设定为2。不过，它可以读取并转换早期的格式。这在DrumFileRead例程中完成。

第二十三章 领略Internet

Internet – 全世界计算机透过不同协议交换信息的大型连结体 – 近几年重新定义了个人计算的几个领域。虽然拨接信息服务和电子邮件系统在Internet流行开来之前就已经存在，但它们通常局限于文字模式，并且根本没有连结而是各自分隔的。例如，每一种信息服务都需要拨不同的电话号码，用不同的使用者ID和密码登录。每一种电子邮件系统仅允许在特定系统的缴款使用者之间发送和接收邮件。

现在，往往只需要拨单一支电话就可以连结整个Internet，而且可以和有电子邮件地址的人进行全球通信。特别是在World Wide Web上，超文字、图形和多媒体（包括声音、音乐和视频）的使用已经扩展了在线信息的范围和功能。

如果要提供涵盖Windows中所有与Internet相关程序设计问题的彻底介绍，可能还需要再加上几本书才够。所以，本章实际上主要集中在如何让小型的Microsoft Windows应用程序能够有效地从Internet上取得信息的两个领域。这两个领域分别是Windows Sockets (Winsock) API和Windows Internet（WinInet）API支持的文件传输协议（FTP：File Transfer Protocol）的部分。

Windows Sockets

Socket是由University of California在Berkeley分校开发的概念，用于在UNIX操作系统上添加网络通讯支持。那里开发的API现在称为「Berkeley socket interface」。

Sockets和TCP/IP

Socket通常（但不专用于）与主宰Internet通信的传输控件协议/因特网协议（TCP/IP：Transmission Control Protocol/Internet Protocol）牵连在一起。因特网协定（IP：Internet Protocol），作为TCP/IP的组成部分之一，用来将数据打包成「数据封包（datagram）」，该资料封包包含用于标识数据来源和目的地的表头信息。而传输控制协议（TCP：Transmission Control Protocol）则提供了可靠的传输和检查IP数据封包正确性的方法。

在TCP/IP下，通讯端点由IP地址和端口号定义。IP地址包括4个字节，用于确定Internet上的服务器。IP地址通常按「由点连结的四个小于255的数字」的格式显示，例如「209.86.105.231」。埠号确定了特定的服务或服务器提供的服务。其中一些埠号已经标准化，以提供众所周知的服务。

当Socket与TCP/IP合用时，Socket就是TCP/IP的通讯端点。因此，Socket指定了IP地址和端口号。

网络时间服务

下面给出的范例程序与提供时间协议（Time Protocol）的Internet服务器相连结。此程序将获得目前准确的日期和时间，并用此信息设定您的PC时钟。

在美国，国家标准和技术协会（National Institute of Standards and Technology）（以前称为国家标准局（National Bureau of Standards））负责维护准确时间，该时间与世界各地的机构相联系。准确时间可用于无线电广播、电话号码、计算机拨号电话号码以及Internet，关于这些的所有文件都位于网站http://www.bldrdoc.gov/timefreq（网域名称「bldrdoc」指的是Boulder、

Colorado、NIST Time的位置和Frequency Division)。

我们只对NIST Network Time Service感兴趣，其详细的文件位于http://www.bldrdoc.gov/timefreq/service/nts.htm。此网页列出了十个提供NIST时间服务的服务器。例如，第一个名称为time-a.timefreq.bldrdoc.gov，其IP地址为132.163.135.130。

（我曾经编写过一个使用非Internet NIST计算机拨接服务的程序，并发表于《PC Magazine》，您也可以在Ziff-Davis的网站http://www.zdnet.com/pcmag/pctech/content/16/20/ut1620.001.html中找到。此程序对于想学习如何使用Windows Telephony API的人很有帮助。）

在Internet上有三个不同的时间服务，每一个都由Request for Comment（RFC）描述为Internet标准。日期协议（Daytime Protocol）（RFC-867）提供了一个ASCII字符串用于指出准确的日期和时间。该ASCII字符串的准确格式并不标准，但人们可以理解其中的含义。时间协议（RFC-868）提供了一个32位的数字，用来表示从1900年1月1日至今的秒数。该时间是UTC（不考虑字母顺序，它表示世界时间坐标（Coordinated Universal Time)），它类似于所谓的格林威治标准时间（Greenwich Mean Time）或者GMT – 英国格林威治时间。第三个协议称为网络时间协议（Network Time Protocol）（RFC-1305），该协议很复杂。

对于我们的目的，即包括分析Socket和不断更新PC时钟，时间协议RFC-868已经够用了。RFC-868只是一个两页的简短文件，主要是说用TCP获得准确时间的程序应该有如下步骤：

连结到提供此服务的服务器埠37。

接收32位的时间。

关闭连结。

现在我们已经知道了编写存取时间服务的Socket应用程序的每个细节。

NETTIME程序

Windows Sockets API，通常也称为WinSock，与Berkeley Sockets API兼容，因此，可以想象UNIX Socket程序代码可以顺利地拿到Windows上使用。Windows下更进一步的支持由对Berkeley Socket扩充的功能提供，其函数的形式是以WSA（「WinSock API」）为前缀。相关的概述和参考位于/Platform SDK/Networking and Distributed Services/Windows Sockets Version 2。

NETTIME，如程序23-1所示，展示了使用WinSock API的方法。

程序23-1 NETTIME

NETTIME.C

```
/*--------------------------------------------------------------------------
NETTIME.C -- Sets System Clock from Internet Services
(c) Charles Petzold, 1998
--------------------------------------------------------------------------*/
#include <windows.h>
#include "resource.h"

#define WM_SOCKET_NOTIFY (WM_USER + 1)
#define ID_TIMER 1

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK MainDlg (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK ServerDlg (HWND, UINT, WPARAM, LPARAM) ;

void ChangeSystemTime (HWND hwndEdit, ULONG ulTime) ;
void FormatUpdatedTime (HWND hwndEdit, SYSTEMTIME * pstOld,
```

```
                    SYSTEMTIME * pstNew) ;
void EditPrintf (HWND hwndEdit, TCHAR * szFormat, ...) ;
HINSTANCE hInst ;
HWND hwndModeless ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  static TCHAR szAppName[] = TEXT ("NetTime") ;
  HWND hwnd ;
  MSG msg ;
  RECT rect ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  wndclass.style = 0 ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
  wndclass.hCursor = NULL ;
  wndclass.hbrBackground = NULL ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox (NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow (szAppName, TEXT ("Set System Clock from Internet"),
    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
    WS_BORDER | WS_MINIMIZEBOX,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  // Create the modeless dialog box to go on top of the window
  hwndModeless = CreateDialog (hInstance, szAppName, hwnd, MainDlg) ;
  // Size the main parent window to the size of the dialog box.
  // Show both windows.

  GetWindowRect (hwndModeless, &rect) ;
  AdjustWindowRect (&rect, WS_CAPTION | WS_BORDER, FALSE) ;

  SetWindowPos (hwnd, NULL, 0, 0, rect.right - rect.left,
    rect.bottom - rect.top, SWP_NOMOVE) ;

  ShowWindow (hwndModeless, SW_SHOW) ;
  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  // Normal message loop when a modeless dialog box is used.
  while (GetMessage (&msg, NULL, 0, 0))
  {
    if (hwndModeless == 0 || !IsDialogMessage (hwndModeless, &msg))
    {
      TranslateMessage (&msg) ;
      DispatchMessage (&msg) ;
    }
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
  switch (message)
  {
  case WM_SETFOCUS:
    SetFocus (hwndModeless) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
  }
  return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK MainDlg ( HWND hwnd, UINT message, WPARAM wParam,
                   LPARAM lParam)
{
  static char szIPAddr[32] = { "132.163.135.130" } ;
  static HWND hwndButton, hwndEdit ;
  static SOCKET sock ;
  static struct sockaddr_in sa ;
  static TCHAR szOKLabel[32] ;
  int  iError, iSize ;
  unsigned long ulTime ;
  WORD  wEvent, wError ;
  WSADATA WSAData ;

  switch (message)
  {
  case WM_INITDIALOG:
    hwndButton = GetDlgItem (hwnd, IDOK) ;
    hwndEdit = GetDlgItem (hwnd, IDC_TEXTOUT) ;
    return TRUE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDC_SERVER:
      DialogBoxParam (hInst, TEXT ("Servers"), hwnd, ServerDlg, (LPARAM) szIPAddr) ;
      return TRUE ;

    case IDOK:
      // Call "WSAStartup" and display description text

      if (iError = WSAStartup (MAKEWORD(2,0), &WSAData))
      {
        EditPrintf (hwndEdit, TEXT ("Startup error #%i.\r\n"), iError) ;
        return TRUE ;
      }
      EditPrintf (hwndEdit, TEXT ("Started up %hs\r\n"),
        WSAData.szDescription);

      // Call "socket"

      sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP) ;

      if (sock == INVALID_SOCKET)
      {
        EditPrintf (hwndEdit, TEXT ("Socket creation error #%i.\r\n"), WSAGetLastError ()) ;
        WSACleanup () ;
        return TRUE ;
      }
      EditPrintf (hwndEdit, TEXT ("Socket %i created.\r\n"), sock) ;

      // Call "WSAAsyncSelect"

      if (SOCKET_ERROR == WSAAsyncSelect (sock, hwnd, WM_SOCKET_NOTIFY, FD_CONNECT | FD_READ))
      {
        EditPrintf ( hwndEdit, TEXT ("WSAAsyncSelect error #%i.\r\n"), WSAGetLastError ()) ;
        closesocket (sock) ;
```

```
      WSACleanup () ;
      return TRUE ;
    }

    // Call "connect" with IP address and time-server port

    sa.sin_family = AF_INET ;
    sa.sin_port = htons (IPPORT_TIMESERVER) ;
    sa.sin_addr.S_un.S_addr = inet_addr (szIPAddr) ;

    connect(sock, (SOCKADDR *) &sa, sizeof (sa)) ;

    // "connect" will return SOCKET_ERROR because even if it
    // succeeds, it will require blocking. The following only
    // reports unexpected errors.

    if (WSAEWOULDBLOCK != (iError = WSAGetLastError ()))
    {
      EditPrintf (hwndEdit, TEXT ("Connect error #%i.\r\n"), iError) ;
      closesocket (sock) ;
      WSACleanup () ;
      return TRUE ;
    }
    EditPrintf (hwndEdit, TEXT ("Connecting to %hs..."), szIPAddr) ;

    // The result of the "connect" call will be reported
    // through the WM_SOCKET_NOTIFY message.
    // Set timer and change the button to "Cancel"

    SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
    GetWindowText (hwndButton, szOKLabel, sizeof (szOKLabel) /sizeof (TCHAR)) ;
    SetWindowText (hwndButton, TEXT ("Cancel")) ;
    SetWindowLong (hwndButton, GWL_ID, IDCANCEL) ;
    return TRUE ;

  case IDCANCEL:
    closesocket (sock) ;
    sock = 0 ;
    WSACleanup () ;
    SetWindowText (hwndButton, szOKLabel) ;
    SetWindowLong (hwndButton, GWL_ID, IDOK) ;

    KillTimer (hwnd, ID_TIMER) ;
    EditPrintf (hwndEdit, TEXT ("\r\nSocket closed.\r\n")) ;
    return TRUE ;

  case IDC_CLOSE:
    if (sock)
      SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;

    DestroyWindow (GetParent (hwnd)) ;
    return TRUE ;
  }
  return FALSE ;

  case WM_TIMER:
    EditPrintf (hwndEdit, TEXT (".")) ;
    return TRUE ;

  case WM_SOCKET_NOTIFY:
    wEvent = WSAGETSELECTEVENT (lParam) ; // ie, LOWORD
    wError = WSAGETSELECTERROR (lParam) ; // ie, HIWORD

    // Process two events specified in WSAAsyncSelect

    switch (wEvent)
    {
      // This event occurs as a result of the "connect" call
```

```
    case FD_CONNECT:
      EditPrintf (hwndEdit, TEXT ("\r\n")) ;

      if (wError)
      {
        EditPrintf (hwndEdit, TEXT ("Connect error #%i."), wError) ;
        SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;
        return TRUE ;
      }
      EditPrintf (hwndEdit, TEXT ("Connected to %hs.\r\n"), szIPAddr) ;

      // Try to receive data. The call will generate an error
      // of WSAEWOULDBLOCK and an event of FD_READ

      recv (sock, (char *) &ulTime, 4, MSG_PEEK) ;
      EditPrintf (hwndEdit, TEXT ("Waiting to receive...")) ;
      return TRUE ;

      // This even occurs when the "recv" call can be made

    case FD_READ:
      KillTimer (hwnd, ID_TIMER) ;
      EditPrintf (hwndEdit, TEXT ("\r\n")) ;

      if (wError)
      {
        EditPrintf (hwndEdit, TEXT ("FD_READ error #%i."), wError) ;
        SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;
        return TRUE ;
      }
      // Get the time and swap the bytes

      iSize = recv (sock, (char *) &ulTime, 4, 0) ;
      ulTime = ntohl (ulTime) ;
      EditPrintf (hwndEdit,
        TEXT ("Received current time of %u seconds ")
        TEXT ("since Jan. 1 1900.\r\n"), ulTime) ;

      // Change the system time

      ChangeSystemTime (hwndEdit, ulTime) ;
      SendMessage (hwnd, WM_COMMAND, IDCANCEL, 0) ;
      return TRUE ;
    }
    return FALSE ;
 }
 return FALSE ;
}

BOOL CALLBACK ServerDlg ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static char * szServer ;
  static WORD wServer = IDC_SERVER1 ;
  char  szLabel [64] ;

  switch (message)
  {
  case WM_INITDIALOG:
    szServer = (char *) lParam ;
    CheckRadioButton (hwnd, IDC_SERVER1, IDC_SERVER10, wServer) ;
    return TRUE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDC_SERVER1:
    case IDC_SERVER2:
    case IDC_SERVER3:
    case IDC_SERVER4:
```

```
    case IDC_SERVER5:
    case IDC_SERVER6:
    case IDC_SERVER7:
    case IDC_SERVER8:
    case IDC_SERVER9:
    case IDC_SERVER10:
      wServer = LOWORD (wParam) ;
      return TRUE ;

    case IDOK:
      GetDlgItemTextA (hwnd, wServer, szLabel, sizeof (szLabel)) ;
      strtok (szLabel, "(") ;
      strcpy (szServer, strtok (NULL, ")")) ;
      EndDialog (hwnd, TRUE) ;
      return TRUE ;

    case IDCANCEL:
      EndDialog (hwnd, FALSE) ;
      return TRUE ;
    }
    break ;
  }
  return FALSE ;
}

void ChangeSystemTime (HWND hwndEdit, ULONG ulTime)
{
  FILETIME ftNew ;
  LARGE_INTEGER li ;
  SYSTEMTIME stOld, stNew ;

  GetLocalTime (&stOld) ;
  stNew.wYear  = 1900 ;
  stNew.wMonth  = 1 ;
  stNew.wDay  = 1 ;
  stNew.wHour  = 0 ;
  stNew.wMinute  = 0 ;
  stNew.wSecond  = 0 ;
  stNew.wMilliseconds = 0 ;

  SystemTimeToFileTime (&stNew, &ftNew) ;
  li = * (LARGE_INTEGER *) &ftNew ;
  li.QuadPart += (LONGLONG) 10000000 * ulTime ;
  ftNew = * (FILETIME *) &li ;
  FileTimeToSystemTime (&ftNew, &stNew) ;

  if (SetSystemTime (&stNew))
  {
    GetLocalTime (&stNew) ;
    FormatUpdatedTime (hwndEdit, &stOld, &stNew) ;
  }
  else
    EditPrintf (hwndEdit, TEXT ("Could NOT set new date and time.")) ;
}

void FormatUpdatedTime ( HWND hwndEdit, SYSTEMTIME * pstOld, SYSTEMTIME * pstNew)
{
  TCHAR szDateOld [64], szTimeOld [64], szDateNew [64], szTimeNew [64] ;
  GetDateFormat (LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE | DATE_SHORTDATE,
    pstOld, NULL, szDateOld, sizeof (szDateOld)) ;
  GetTimeFormat (LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE |
    TIME_NOTIMEMARKER | TIME_FORCE24HOURFORMAT,
    pstOld, NULL, szTimeOld, sizeof (szTimeOld)) ;

  GetDateFormat (LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE | DATE_SHORTDATE,
    pstNew, NULL, szDateNew, sizeof (szDateNew)) ;
  GetTimeFormat (LOCALE_USER_DEFAULT, LOCALE_NOUSEROVERRIDE |
    TIME_NOTIMEMARKER | TIME_FORCE24HOURFORMAT,
    pstNew, NULL, szTimeNew, sizeof (szTimeNew)) ;
```

```
  EditPrintf (hwndEdit, TEXT ("System date and time successfully changed ")
    TEXT ("from\r\n\t%s, %s.%03i to\r\n\t%s, %s.%03i."),
    szDateOld, szTimeOld, pstOld->wMilliseconds,
    szDateNew, szTimeNew, pstNew->wMilliseconds) ;
}

void EditPrintf (HWND hwndEdit, TCHAR * szFormat, ...)
{
  TCHAR szBuffer [1024] ;
  va_list pArgList ;

  va_start (pArgList, szFormat) ;
  wvsprintf (szBuffer, szFormat, pArgList) ;
  va_end (pArgList) ;
  SendMessage (hwndEdit, EM_SETSEL, (WPARAM) -1, (LPARAM) -1) ;
  SendMessage (hwndEdit, EM_REPLACESEL, FALSE, (LPARAM) szBuffer) ;
  SendMessage (hwndEdit, EM_SCROLLCARET, 0, 0) ;
}
```

NETTIME.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
//////////////////////////////////////////////////////////////////////////////
// Dialog
SERVERS DIALOG DISCARDABLE 20, 20, 274, 202
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "NIST Time Service Servers"
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON  "OK",IDOK,73,181,50,14
PUSHBUTTON  "Cancel",IDCANCEL,150,181,50,14
CONTROL
"time-a.timefreq.bldrdoc.gov (132.163.135.130) NIST, Boulder, Colorado",
IDC_SERVER1,"Button",BS_AUTORADIOBUTTON,9,7,256,16
CONTROL
"time-b.timefreq.bldrdoc.gov (132.163.135.131) NIST, Boulder, Colorado",
IDC_SERVER2,"Button",BS_AUTORADIOBUTTON,9,24,256,16
CONTROL
"time-c.timefreq.bldrdoc.gov (132.163.135.132) Boulder, Colorado",
IDC_SERVER3,"Button",BS_AUTORADIOBUTTON,9,41,256,16
CONTROL
"utcnist.colorado.edu (128.138.140.44) University of Colorado, Boulder",
IDC_SERVER4,"Button",BS_AUTORADIOBUTTON,9,58,256,16
CONTROL
"time.nist.gov (192.43.244.18) NCAR, Boulder, Colorado",
IDC_SERVER5,"Button",BS_AUTORADIOBUTTON,9,75,256,16
CONTROL
"time-a.nist.gov (129.6.16.35) NIST, Gaithersburg, Maryland",
IDC_SERVER6,"Button",BS_AUTORADIOBUTTON,9,92,256,16
CONTROL
"time-b.nist.gov (129.6.16.36) NIST, Gaithersburg, Maryland",
IDC_SERVER7,"Button",BS_AUTORADIOBUTTON,9,109,256,16
CONTROL
"time-nw.nist.gov (131.107.1.10) Microsoft, Redmond, Washington",
IDC_SERVER8,"Button",BS_AUTORADIOBUTTON,9,126,256,16
CONTROL
"utcnist.reston.mci.net (204.70.131.13) MCI, Reston, Virginia",
IDC_SERVER9,"Button",BS_AUTORADIOBUTTON,9,143,256,16

CONTROL
"nist1.data.com (209.0.72.7) Datum, San Jose, California",
IDC_SERVER10,"Button",BS_AUTORADIOBUTTON,9,160,256,16
END
NETTIME DIALOG DISCARDABLE 0, 0, 270, 150 STYLE WS_CHILD FONT 8, "MS Sans Serif"
BEGIN
```

```
DEFPUSHBUTTON  "Set Correct Time",IDOK,95,129,80,14
PUSHBUTTON  "Close",IDC_CLOSE,183,129,80,14
PUSHBUTTON  "Select Server...",IDC_SERVER,7,129,80,14
EDITTEXT   IDC_TEXTOUT,7,7,253,110,ES_MULTILINE | ES_AUTOVSCROLL |
ES_READONLY | WS_VSCROLL | NOT WS_TABSTOP
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by NetTime.rc
#define  IDC_TEXTOUT      101
#define  IDC_SERVER1      1001
#define  IDC_SERVER2      1002
#define  IDC_SERVER3      1003
#define  IDC_SERVER4      1004
#define  IDC_SERVER5      1005
#define  IDC_SERVER6      1006
#define  IDC_SERVER7      1007
#define  IDC_SERVER8      1008
#define  IDC_SERVER9      1009
#define  IDC_SERVER10      1010
#define  IDC_SERVER      1011
#define  IDC_CLOSE       1012
```

在结构上，NETTIME程序建立了一个依据NETTIME.RC中的NETTIME所建立的非系统模态对话框。程序重新定义了窗口的尺寸，以便非系统模态对话框可以覆盖程序的整个窗口显示区域。对话框包括一个只读编辑区（程序用于写入文字信息）、一个「Select Server」按钮、一个「Set Correct Time」按钮和一个「Close」按钮。「Close」按钮用于终止程序。

MainDlg中的szIPAddr变量用于储存服务器地址，内定是字符串「132.163.135.130」。「Select Server」按钮启动依据NETTIME.RC中的SERVERS模板建立的对话框。szIPAddr变量作为最后一个参数传递给DialogBoxParam。「Server」对话框列出了10个服务器（都是从NIST网站上逐字复制来的），这些服务器提供了我们感兴趣的服务。当使用者单击一个服务器时，ServerDlg将分析按钮文字，以获得相应的IP地址。新地址储存在szIPAddr变量中。

当使用者按下「Set Correct Time」按钮时，按钮将产生一个WM_COMMAND消息，其中wParam的低字组等于IDOK。MainDlg中的IDOK处理是大部分Socket初始行为发生的地方。

使用Windows Sockets API时，任何Windows程序必须呼叫的第一个函数是：

iError = WSAStartup (wVersion, &WSAData) ;

NETTIME将第一个参数设定为0x0200（表示2.0版本）。传回时，WSAData结构包含了Windows Sockets实作的相关信息，而且NETTIME将显示szDescription字符串，并简要提供了一些版本信息。

然后，NETTIME如下呼叫socket函数：

sock = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP) ;

第一个参数是一个地址种类，表示此处是某种Internet地址。第二个参数表示数据以数据流的形式传回，而不是以数据封包的形式传回（我们需要的数据只有4个字节长，而数据封包适用于较大的数据块）。最后一个参数是一个协议，我们指定使用的Internet协议是TCP。它是RFC-868所定义的两个协议之一。socket函数的传回值储存在SOCKET型态的变量中，以便后面的Socket函数的呼叫。

NETTIME下面呼叫的WSAAsynchSelect是另一个Windows特有的Socket函数。此函数用于避免因Internet响应过慢而造成应用程序当住。在WinSock文件中，有些函数与「阻碍性（blocking）」有关。也就是说，它们不能保证立即把控件权传回给程序。WSAAsyncSelect函数强制阻碍性的函

数转为非阻碍性的，即在函数执行完之前把控件传回给程序。函数的结果以消息的形式报告给应用程序。WSAAsyncSelect函数让应用程序指定消息和接收消息的窗口的数值。通常，函数的语法如下：

WSAAsyncSelect (sock, hwnd, message, iConditions) ;

为此任务，NETTIME使用程序定义的一个消息，该消息称为WM_SOCKET_NOTIFY。它也用WSAAsyncSelect的最后一个参数来指定消息发送的条件，特别在连结和接收资料时（FD_CONNECT | FD_READ)。

NETTIME呼叫的下一个WinSock函数是connect。此函数需要一个指向Socket地址结构的指针，对于不同的协议来说，此Socket地址结构是不同的。NETTIME使用为TCP/IP设计的结构版本：

```
struct sockaddr_in
{
  short   sin_family;
  u_short  sin_port;
  struct in_addr sin_addr;
  char   sin_zero[8];
} ;
```

其中in_addr是用于指定Internet地址，它可以用4个字节，或者2个无正负号短整数，或者1个无正负号长整数来表示。

NETTIME将sin_family字段设定为AF_INET，用于表示地址种类。将sin_port设定为埠号，这里是时间协议的埠号，RFC-868显示为37。但不要像我最初时那样，将此字段设为37。当大多数数字通过Internet时，结构的这个端口号字段必须是「big endian」的，即最高的字节排第一个。Intel微处理器是little endian。幸运的是，htons（「host-to-network short」）函数使字节翻转，因此NETTIME将sockaddr_in结构的sin_port字段设定为：

htons (IPPORT_TIMESERVER)

WINSOCK2.H中将常数定义为37。NETTIME用inet_addr函数将储存在szIPAddr字符串中的服务器地址转化为无正负号长整数，该整数用于设定结构的sin_addr字段。

如果应用程序在Windows 98下呼叫connect，而且目前Windows没有连结到Internet，那么将显示「拨号联机」对话框。这就是所谓的「自动拨号」。在Windows NT 4.0中没有实作「自动拨号」，因此如果在NT环境下执行，那么在执行NETTIME之前，就必须先连结上Internet。

connect函数通常已经会阻碍着后面程序的执行，这是因为连结成功以前需要花些时间。然而，由于NETTIME呼叫了WSAAsyncSelect，所以connect不会等待连结，事实上，它会立即传回SOCKET_ERROR的值。这并不是出现了错误，这只是表示现在还没有联机成功而已。NETTIME也不会检查这个传回值，只是呼叫WSAGetLastError而已。如果WSAGetLastError传回WSAEWOULDBLOCK（即函数的执行通常要受阻，但这里并没有受阻)，那就一切都还很正常。NETTIME将「Set Correct Time」按钮改成「Cancel」，并设定了一个1秒的定时器。WM_TIMER的处理方式只是在程序窗口中显示句点，以告诉使用者程序仍在执行，系统没有当掉。

连结最终完成时，MainDlg由WM_SOCKET_NOTIFY消息 – NETTIME在WSAAsyncSelect函数中指定的程序自订消息所通知。lParam的低字组等于FD_CONNECT，高字组表示错误。这时的错误可能是程序不能连结到指定的服务器。NETTIME还列出了其它9个服务器，供您选择，让您可以试试其它的服务器。

如果一切顺利，那么NETTIME将呼叫recv（「receive：接收」）函数来读取数据：

recv (sock, (char *) &ulTime, 4, MSG_PEEK) ;

这意味着，用4个字节来储存ulTime变量。最后一个参数表示只是读此数据，并不将其从输入队列中删除。像connect函数一样，recv传回一个错误代码，以表示函数通常受阻，但这时没有受阻。理论上来说（当然这不大可能），函数至少能传回数据的一部分，然后透过再次呼叫以获得其余的32个字节值。那就是呼叫recv函数时带有MSG_PEEK选项的原因。

与connect函数类似，recv函数也产生WM_SOCKET_NOTIFY消息，这时带有FD_READ的事件代码。NETTIME通过再次呼叫recv来对此响应，这时最后的参数是0，用于从队列中删除数据。我将简要讨论一下程序处理接收到的ulTime的方法。注意，NETTIME通过向自己发送WM_COMMAND消息来结束处理，该消息中wParam等于IDCANCEL。对话框程序通过呼叫closesocket和WSACleanup来响应。

再次呼叫NETTIME接收的32位的ulTime值是从1990年1月1日开始的0:00 UTC秒数。但最高顺序的字节是第一个字节，因此该值必须通过ntohl（「network-to-host long」）函数处理来调整字节顺序，以便Intel微处理器能够处理。然后，NETTIME呼叫ChangeSystemTime函数。

ChangeSystemTime首先取得目前的本地时间 – 即，使用者所在时区和日光节约时间的目前系统时间。将SYSTEMTIME结构设定为1900年1月1日午夜（0时）。并将这个SYSTEMTIME结构传递给SystemTimeToFileTime，将此结构转化为FILETIME结构。FILETIME实际上只是由两个32位的DWORD一起组成64位的整数，用来表示从1601年1月1日至今间隔为100奈秒（nanosecond）的间隔数。

ChangeSystemTime函数将FILETIME结构转化为LARGE_INTEGER。它是一个union，允许64位的值可以被当成两个32位的值使用，或者当成一个__int64数据型态的64位整数使用（__int64数据型态是Microsoft编译器对ANSI C标准的扩充）。因此，此值是1601年1月1日到1900年1月1日之间间隔为100奈秒的间隔数。这里，添加了1900年1月1日至今间隔为100奈秒的间隔数 – ulTime的10,000,000倍。

然后通过呼叫FileTimeToSystemTime将作为结果的FILETIME值转换回SYSTEMTIME结构。因为时间协议传回目前的UTC时间，所以NETTIME通过呼叫SetSystemTime来设定时间，SetSystemTime也依据UTC。基于显示的目的，程序呼叫GetLocalTime来获得更新时间。最初的本地时间和新的本地时间一起传递给FormatUpdatedTime，这个函数用GetTimeFormat函数和GetDateFormat函数将时间转化为ASCII字符串。

如果程序在Windows NT下执行，并且使用者没有取得设定时间的权限，那么SetSystemTime函数可能失败。如果SetSystemTime失败，则NETTIME将发出一个新时间未设定成功的消息来指出问题所在。

WinInet 和 FTP

WinInet（「Windows Internet」）API是一个高阶函数集，帮助程序写作者使用三个常见的Internet协议，这三个协议是：用于World Wide Web全球信息网的超文字传输协议（HTTP：Hypertext Transfer Protocol）、文件传输协议（FTP：File Transfer Protocol）和另一个称为Gopher的文件传输协议。WinInet函数的语法与常用的Windows文件函数的语法类似，这使得使用这些协议就像使用本地磁盘驱动器上的文件一样容易。WinInet API的文件位于/Platform SDK/Internet, Intranet, Extranet Services/Internet Tools and Technologies/WinInet API。

下面的范例程序将展示如何使用WinInet API的FTP部分。许多有网站的公司也都有「匿名FTP」服务器，这样使用者可以在不输入使用者名称和密码的情况下下载文件。例如，如果您在Internet

Explorer的地址栏输入ftp://ftp.microsoft.com，那么您就可以浏览FTP服务器上的目录并下载文件。如果进入ftp://ftp.cpetzold.com/cpetzold.com/ProgWin/UpdDemo，那么您将在我的匿名FTP服务器上发现与待会要提到的范例程序一块使用的文件列表。

虽然现今FTP服务对大多数的Web使用者来说并不是那么方便使用，但它仍然相当有用。例如，应用程序能利用FTP从匿名FTP服务器上取得数据，这些取得数据的运作程序几乎完全在台面下处理，而不需要使用者操心。这就是我们将讨论的UPDDEMO（「update demonstration：更新范例」）程序的构想。

FTP API概况

使用WinInet的程序必须在所有呼叫WinInet函数的源文件中包括表头文件WININET.H。程序还必须连结WININET.LIB。在Microsoft Visual C++中，您可以在「Project Settings」对话框的「Link」页面卷标中指定。执行时，程序将和WININET.DLL动态链接库连结。

在下面的论述中，我不会详细讨论函数的语法，因为某些函数有很多选项，这让它变得相当复杂。要掌握WinInet，您可以将UPDDEMO原始码当成食谱来看待。这时最重要的是了解有关的各个步骤以及FTP函数的范围。

要使用Windows Internet API，首先要呼叫InternetOpen。然后，使用WinInet支持的任何一种协议。InternetOpen给您一个Internet作业句柄，并储存到HINTERNET型态的变量中。用完WinInet API以后，应该通过呼叫InternetCloseHandle来关闭句柄。

要使用FTP，您接下来就要呼叫InternetConnect。此函数需要使用由InternetOpen建立Internet作业句柄，并且传回FTP作业的句柄。您可将此句柄作为名称开头为Ftp的所有函数的第一个参数。InternetConnect函数的参数指出要使用的FTP，还提供了服务器名称，例如，ftp.cpetzold.com。此函数还需要使用者名称和密码。如果存取匿名FTP服务器，这些参数可以设定为NULL。如果应用程序呼叫InternetConnect时PC并没有连结到Internet，Windows 98将显示「拨号联机」对话框。当使用FTP的应用程序结束时，呼叫InternetCloseHandle来关闭句柄。

这时可以开始呼叫有Ftp前缀的函数。您将发现这些函数与标准的Windows文件I/O函数很相似。为了避免与其它协议重复，一些以Internet为前缀的函数也可以处理FTP。

下面四个函数用于处理目录：

```
fSuccess = FtpCreateDirectory          (hFtpSession, szDirectory) ;
fSuccess = FtpRemoveDirectory          (hFtpSession, szDirectory) ;
fSuccess = FtpSetCurrentDirectory (hFtpSession, szDirectory) ;
fSuccess = FtpGetCurrentDirectory (hFtpSession, szDirectory, &dwCharacterCount) ;
```

注意，这些函数很像我们所熟悉的Windows提供用于处理本地文件系统的CreateDirectory、RemoveDirectory、SetCurrentDirectory和GetCurrentDirectory函数。

当然，存取匿名FTP的应用程序不能建立或删除目录。而且，程序也不能假定FTP目录具有和Windows文件系统相同的目录结构型态。特别是用相对路径名设定目录的程序，不能假定关于新的目录全名的一切。如果程序需要知道最后所在目录的整个名称，那么呼叫了SetCurrentDirectory之后必须再呼叫GetCurrentDirectory。GetCurrentDirectory的字符串参数至少包含MAX_PATH字符，并且最后一个参数应指向包含该值的变量。

下面两个函数让您删除或者重新命名文件（但不是在匿名FTP服务器上）：

```
fSuccess = FtpDeleteFile (hFtpSession, szFileName) ;
```

fSuccess = FtpRenameFile (hFtpSession, szOldName, szNewName) ;

经由先呼叫FtpFindFirstFile，可以查找文件（或与含有万用字符的文件名样式相符的多个文件）。此函数很像FindFirstFile函数，甚至都使用了相同的WIN32_FIND_DATA结构。该文件为列举出来的文件传回了一个句柄。您可以将此句柄传递给InternetFindNextFile函数以获得额外的文件名称信息。最后通过呼叫InternetCloseHandle来关闭句柄。

要打开文件，可以呼叫FtpFileOpen。这个函数传回一个文件句柄，此句柄可以用于InternetReadFile、InternetReadFileEx、InternetWrite和InternetSetFilePointer呼叫。最后可以通过呼叫最常用的InternetCloseHandle函数来关闭句柄。

最后，下面两个高级函数特别有用：FtpGetFile呼叫将文件从FTP服务器复制到本地内存，它合并了FtpFileOpen、FileCreate、InternetReadFile、WriteFile、InternetCloseHandle和CloseHandle呼叫。FtpGetFile的另一个参数是一个旗标，如果本地已经存在同名文件，那么该旗标将导致函数呼叫失败。FtpPutFile与此函数类似，用于将文件从本地内存复制到FTP服务器。

更新展示程序

UPDDEMO，如程序23-2所示，展示了用WinInet FTP函数在第二个线程执行期间从匿名FTP服务器上下载文件的方法。

程序23-2 UPDDEMO

UPDDEMO.C

```
/*-----------------------------------------------------------------------------
UPDDEMO.C -- Demonstrates Anonymous FTP Access
(c) Charles Petzold, 1998
-----------------------------------------------------------------------------*/
#include <windows.h>
#include <wininet.h>
#include <process.h>
#include "resource.h"

// User-defined messages used in WndProc

#define WM_USER_CHECKFILES (WM_USER + 1)
#define WM_USER_GETFILES (WM_USER + 2)

// Information for FTP download

#define FTPSERVER TEXT ("ftp.cpetzold.com")
#define DIRECTORY TEXT ("cpetzold.com/ProgWin/UpdDemo")
#define TEMPLATE TEXT ("UD??????.TXT")

// Structures used for storing filenames and contents
typedef struct
{
  TCHAR * szFilename ;
  char * szContents ;
}
FILEINFO ;
typedef struct
{
  int iNum ;
  FILEINFO info[1] ;
}
FILELIST ;
// Structure used for second thread
typedef struct
{
  BOOL bContinue ;
  HWND hwnd ;
}
```

```
PARAMS ;
// Declarations of all functions in program
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
VOID  FtpThread (PVOID) ;
VOID  ButtonSwitch (HWND, HWND, TCHAR *) ;
FILELIST * GetFileList (VOID) ;
int  Compare (const FILEINFO *, const FILEINFO *) ;

// A couple globals

HINSTANCE hInst ;
TCHAR szAppName[] = TEXT ("UpdDemo") ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
{
  HWND  hwnd ;
  MSG  msg ;
  WNDCLASS wndclass ;

  hInst = hInstance ;
  wndclass.style = 0 ;
  wndclass.lpfnWndProc = WndProc ;
  wndclass.cbClsExtra = 0 ;
  wndclass.cbWndExtra = 0 ;
  wndclass.hInstance = hInstance ;
  wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION) ;
  wndclass.hCursor = NULL ;
  wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
  wndclass.lpszMenuName = NULL ;
  wndclass.lpszClassName = szAppName ;

  if (!RegisterClass (&wndclass))
  {
    MessageBox ( NULL, TEXT ("This program requires Windows NT!"),
      szAppName, MB_ICONERROR) ;
    return 0 ;
  }

  hwnd = CreateWindow ( szAppName, TEXT ("Update Demo with Anonymous FTP"),
    WS_OVERLAPPEDWINDOW | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    NULL, NULL, hInstance, NULL) ;

  ShowWindow (hwnd, iCmdShow) ;
  UpdateWindow (hwnd) ;

  // After window is displayed, check if the latest file exists
  SendMessage (hwnd, WM_USER_CHECKFILES, 0, 0) ;
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
  }
  return msg.wParam ;
}

LRESULT CALLBACK WndProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static FILELIST * plist ;
  static int  cxClient, cyClient, cxChar, cyChar ;
  HDC  hdc ;
  int  i ;
  PAINTSTRUCT  ps ;
  SCROLLINFO  si ;
  SYSTEMTIME  st ;
  TCHAR  szFilename [MAX_PATH] ;
```

```
switch (message)
{
case WM_CREATE:
  cxChar = LOWORD (GetDialogBaseUnits ()) ;
  cyChar = HIWORD (GetDialogBaseUnits ()) ;
  return 0 ;

case WM_SIZE:
  cxClient = LOWORD (lParam) ;
  cyClient = HIWORD (lParam) ;

  si.cbSize = sizeof (SCROLLINFO) ;
  si.fMask = SIF_RANGE | SIF_PAGE ;
  si.nMin = 0 ;
  si.nMax = plist ? plist->iNum - 1 : 0 ;
  si.nPage = cyClient / cyChar ;

  SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
  return 0 ;

case WM_VSCROLL:
  si.cbSize = sizeof (SCROLLINFO) ;
  si.fMask = SIF_POS | SIF_RANGE | SIF_PAGE ;
  GetScrollInfo (hwnd, SB_VERT, &si) ;

  switch (LOWORD (wParam))
  {
  case SB_LINEDOWN: si.nPos += 1 ; break ;
  case SB_LINEUP: si.nPos -= 1 ; break ;
  case SB_PAGEDOWN: si.nPos += si.nPage ; break ;
  case SB_PAGEUP: si.nPos -= si.nPage ; break ;
  case SB_THUMBPOSITION: si.nPos = HIWORD (wParam) ; break ;
  default: return 0 ;
  }
  si.fMask = SIF_POS ;
  SetScrollInfo (hwnd, SB_VERT, &si, TRUE) ;
  InvalidateRect (hwnd, NULL, TRUE) ;
  return 0 ;

  case WM_USER_CHECKFILES:
    // Get the system date & form filename from year and month

    GetSystemTime (&st) ;
    wsprintf (szFilename, TEXT ("UD%04i%02i.TXT"), st.wYear, st.wMonth) ;

    // Check if the file exists; if so, read all the files

    if (GetFileAttributes (szFilename) != (DWORD) -1)
    {
      SendMessage (hwnd, WM_USER_GETFILES, 0, 0) ;
      return 0 ;
    }
    // Otherwise, get files from Internet.
    // But first check so we don't try to copy files to a CD-ROM!

    if (GetDriveType (NULL) == DRIVE_CDROM)
    {
      MessageBox (hwnd, TEXT ("Cannot run this program from CD-ROM!"),
        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
      return 0 ;
    }
    // Ask user if an Internet connection is desired

    if (IDYES == MessageBox (hwnd, TEXT ("Update information from Internet?"),
      szAppName, MB_YESNO | MB_ICONQUESTION))

      // Invoke dialog box
```

```
      DialogBox (hInst, szAppName, hwnd, DlgProc) ;

    // Update display

    SendMessage (hwnd, WM_USER_GETFILES, 0, 0) ;
    return 0 ;

  case WM_USER_GETFILES:
    SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
    ShowCursor (TRUE) ;

    // Read in all the disk files

    plist = GetFileList () ;

    ShowCursor (FALSE) ;
    SetCursor (LoadCursor (NULL, IDC_ARROW)) ;

    // Simulate a WM_SIZE message to alter scroll bar & repaint

    SendMessage (hwnd, WM_SIZE, 0, MAKELONG (cxClient, cyClient)) ;
    InvalidateRect (hwnd, NULL, TRUE) ;
    return 0 ;

  case WM_PAINT:
    hdc = BeginPaint (hwnd, &ps) ;
    SetTextAlign (hdc, TA_UPDATECP) ;

    si.cbSize = sizeof (SCROLLINFO) ;
    si.fMask = SIF_POS ;
    GetScrollInfo (hwnd, SB_VERT, &si) ;

    if (plist)
    {
      for (i = 0 ; i < plist->iNum ; i++)
      {
        MoveToEx (hdc, cxChar, (i - si.nPos) * cyChar, NULL) ;
        TextOut (hdc, 0, 0, plist->info[i].szFilename,
          lstrlen (plist->info[i].szFilename)) ;
        TextOut (hdc, 0, 0, TEXT (": "), 2) ;
        TextOutA (hdc, 0, 0, plist->info[i].szContents,
          strlen (plist->info[i].szContents)) ;
      }
    }
    EndPaint (hwnd, &ps) ;
    return 0 ;

  case WM_DESTROY:
    PostQuitMessage (0) ;
    return 0 ;
 }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK DlgProc ( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  static PARAMS params ;
  switch (message)
  {
  case WM_INITDIALOG:
    params.bContinue = TRUE ;
    params.hwnd = hwnd ;

    _beginthread (FtpThread, 0, &params) ;
    return TRUE ;

  case WM_COMMAND:
    switch (LOWORD (wParam))
    {
```

```
    case IDCANCEL: // button for user to abort download
      params.bContinue = FALSE ;
      return TRUE ;

    case IDOK: // button to make dialog box go away
      EndDialog (hwnd, 0) ;
      return TRUE ;
    }
  }
  return FALSE ;
}

/*---------------------------------------------------------------------------
FtpThread: Reads files from FTP server and copies them to local disk
---------------------------------------------------------------------------*/

void FtpThread (PVOID parg)
{
  BOOL  bSuccess ;
  HINTERNET  hIntSession, hFtpSession, hFind ;
  HWND  hwndStatus, hwndButton ;
  PARAMS * pparams ;
  TCHAR  szBuffer [64] ;
  WIN32_FIND_DATA finddata ;

  pparams = parg ;
  hwndStatus = GetDlgItem (pparams->hwnd, IDC_STATUS) ;
  hwndButton = GetDlgItem (pparams->hwnd, IDCANCEL) ;

  // Open an internet session

  hIntSession = InternetOpen (szAppName, INTERNET_OPEN_TYPE_PRECONFIG,
    NULL, NULL, INTERNET_FLAG_ASYNC) ;
  if (hIntSession == NULL)
  {
    wsprintf (szBuffer, TEXT ("InternetOpen error %i"), GetLastError ()) ;
    ButtonSwitch (hwndStatus, hwndButton, szBuffer) ;
    _endthread () ;
  }

  SetWindowText (hwndStatus, TEXT ("Internet session opened...")) ;
  // Check if user has pressed Cancel
  if (!pparams->bContinue)
  {
    InternetCloseHandle (hIntSession) ;
    ButtonSwitch (hwndStatus, hwndButton, NULL) ;
    _endthread () ;
  }

  // Open an FTP session.
  hFtpSession = InternetConnect (hIntSession, FTPSERVER, INTERNET_DEFAULT_FTP_PORT,
    NULL, NULL, INTERNET_SERVICE_FTP, 0, 0) ;
  if (hFtpSession == NULL)
  {
    InternetCloseHandle (hIntSession) ;
    wsprintf (szBuffer, TEXT ("InternetConnect error %i"),
      GetLastError ()) ;
    ButtonSwitch (hwndStatus, hwndButton, szBuffer) ;
    _endthread () ;
  }

  SetWindowText (hwndStatus, TEXT ("FTP Session opened...")) ;
  // Check if user has pressed Cancel
  if (!pparams->bContinue)
  {
    InternetCloseHandle (hFtpSession) ;
    InternetCloseHandle (hIntSession) ;
    ButtonSwitch (hwndStatus, hwndButton, NULL) ;
    _endthread () ;
```

```
  }

  // Set the directory
  bSuccess = FtpSetCurrentDirectory (hFtpSession, DIRECTORY) ;
  if (!bSuccess)
  {
    InternetCloseHandle (hFtpSession) ;
    InternetCloseHandle (hIntSession) ;
    wsprintf ( szBuffer, TEXT ("Cannot set directory to %s"),
      DIRECTORY) ;
    ButtonSwitch (hwndStatus, hwndButton, szBuffer) ;
    _endthread () ;
  }

  SetWindowText (hwndStatus, TEXT ("Directory found...")) ;
  // Check if user has pressed Cancel
  if (!pparams->bContinue)
  {
    InternetCloseHandle (hFtpSession) ;
    InternetCloseHandle (hIntSession) ;
    ButtonSwitch (hwndStatus, hwndButton, NULL) ;
    _endthread () ;
  }

  // Get the first file fitting the template
  hFind = FtpFindFirstFile (hFtpSession, TEMPLATE, &finddata, 0, 0) ;
  if (hFind == NULL)
  {
    InternetCloseHandle (hFtpSession) ;
    InternetCloseHandle (hIntSession) ;
    ButtonSwitch (hwndStatus, hwndButton, TEXT ("Cannot find files")) ;
    _endthread () ;
  }
  do
  {
    // Check if user has pressed Cancel
    if (!pparams->bContinue)
    {
      InternetCloseHandle (hFind) ;
      InternetCloseHandle (hFtpSession) ;
      InternetCloseHandle (hIntSession) ;
      ButtonSwitch (hwndStatus, hwndButton, NULL) ;
      _endthread () ;
    }
    // Copy file from internet to local hard disk, but fail
    // if the file already exists locally

    wsprintf (szBuffer, TEXT ("Reading file %s..."), finddata.cFileName) ;
    SetWindowText (hwndStatus, szBuffer) ;

    FtpGetFile ( hFtpSession,
      finddata.cFileName, finddata.cFileName, TRUE,
      FILE_ATTRIBUTE_NORMAL, FTP_TRANSFER_TYPE_BINARY, 0) ;
  }
  while (InternetFindNextFile (hFind, &finddata)) ;
  InternetCloseHandle (hFind) ;
  InternetCloseHandle (hFtpSession) ;
  InternetCloseHandle (hIntSession) ;

  ButtonSwitch (hwndStatus, hwndButton, TEXT ("Internet Download Complete"));
}

/*---------------------------------------------------------------------------
ButtonSwitch: Displays final status message and changes Cancel to OK
---------------------------------------------------------------------------*/
VOID ButtonSwitch (HWND hwndStatus, HWND hwndButton, TCHAR * szText)
{
  if (szText)
    SetWindowText (hwndStatus, szText) ;
```

```
  else
    SetWindowText (hwndStatus, TEXT ("Internet Session Cancelled")) ;
  SetWindowText (hwndButton, TEXT ("OK")) ;
  SetWindowLong (hwndButton, GWL_ID, IDOK) ;
}

/*--------------------------------------------------------------------------
GetFileList: Reads files from disk and saves their names and contents
--------------------------------------------------------------------------*/

FILELIST * GetFileList (void)
{
  DWORD  dwRead ;
  FILELIST * plist ;
  HANDLE  hFile, hFind ;
  int  iSize, iNum ;
  WIN32_FIND_DATA finddata ;

  hFind = FindFirstFile (TEMPLATE, &finddata) ;
  if (hFind == INVALID_HANDLE_VALUE)
    return NULL ;
  plist = NULL ;
  iNum = 0 ;

  do
  {
    // Open the file and get the size
    hFile = CreateFile (finddata.cFileName, GENERIC_READ, FILE_SHARE_READ,
      NULL, OPEN_EXISTING, 0, NULL) ;

    if (hFile == INVALID_HANDLE_VALUE)
      continue ;

    iSize = GetFileSize (hFile, NULL) ;

    if (iSize == (DWORD) -1)
    {
      CloseHandle (hFile) ;
      continue ;
    }
    // Realloc the FILELIST structure for a new entry

    plist = realloc (plist, sizeof (FILELIST) + iNum * sizeof (FILEINFO));

    // Allocate space and save the filename

    plist->info[iNum].szFilename = malloc (lstrlen (finddata.cFileName) +sizeof (TCHAR)) ;
    lstrcpy (plist->info[iNum].szFilename, finddata.cFileName) ;

    // Allocate space and save the contents

    plist->info[iNum].szContents = malloc (iSize + 1) ;
    ReadFile (hFile, plist->info[iNum].szContents, iSize, &dwRead, NULL);
    plist->info[iNum].szContents[iSize] = 0 ;

    CloseHandle (hFile) ;
    iNum ++ ;
  }
  while (FindNextFile (hFind, &finddata)) ;
  FindClose (hFind) ;
  // Sort the files by filename
  qsort (plist->info, iNum, sizeof (FILEINFO), Compare) ;
  plist->iNum = iNum ;
  return plist ;
}

/*--------------------------------------------------------------------------
Compare function for qsort
--------------------------------------------------------------------------*/
```

```
int Compare (const FILEINFO * pinfo1, const FILEINFO * pinfo2)
{
  return lstrcmp (pinfo2->szFilename, pinfo1->szFilename) ;
}
```

UPDDEMO.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
////////////////////////////////////////////////////////////////////////////
// Dialog
UPDDEMO DIALOG DISCARDABLE 20, 20, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Internet Download"
FONT 8, "MS Sans Serif"
BEGIN
PUSHBUTTON Cancel",IDCANCEL,69,74,50,14
CTEXT   "",IDC_STATUS,7,29,172,21
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by UpdDemo.rc
#define IDC_STATUS   40001
```

UPDDEMO使用的文件名称是UDyyyymm.TXT，其中yyyy是4位阿拉伯数字的年数（当然适用于2000），mm是2位阿拉伯数字的月数。这里假定程序可以享有每个月都有更新文件的好处。这些文件可能是整个月刊，而由于阅读效率上的考虑，让程序将其下载到本地储存媒体上。

因此，WinMain在呼叫ShowWindow和UpdateWindow来显示UPDDEMO主窗口以后，向WndProc发送程序定义的WM_USER_CHECKFILES消息。WndProc通过获得目前的年、月并检查该年月UDyyyymm.TXT文件所在的内定目录来处理此消息。这种文件的存在意义在于UPDDEMO会被完全更新（当然，事实并非如此。一些过时的文件将漏掉。如果要做得更完整，程序得进行更广泛的检测）。在这种情况下，UPDDEMO向自己发送一个WM_USER_GETFILES消息，它通过呼叫GetFileList函数来处理。这是UPDDEMO.C中稍长的一个函数，但它并不是特别有用，它所做的全部工作就是将所有的UDyyyymm.TXT文件读到动态配置的FILELIST型态结构中，该结构是在程序顶部定义的，然后让程序在其显示区域显示这些文件的内容。

如果UPDDEMO没有最新的文件，那么它必须透过Internet进行更新。程序首先询问使用者这样做是否「OK」。如果是，程序将显示一个简单的对话框，其中只有一个「Cancel」按钮和一个ID为IDC_STATUS的静态文字区。下载时，此静态文字区向使用者提供状态报告，并且允许使用者取消过于缓慢的更新作业。此对话程序的名称是DlgProc。

DlgProc很短，它建立了一个包括自身窗口句柄的PARAMS型态的结构以及一个名称为bContinue的BOOL变量，然后呼叫_beginthread来执行第二个执行绪。

FtpThread函数透过使用下面的呼叫来完成实际的传输：InternetOpen、InternetConnect、FtpSetCurrentDirectory 、 FtpFindFirstFile 、 InternetFindNextFile 、 FtpGetFile 和InternetCloseHandle（三次）。如同大多数程序代码，该线程函数如果略过错误检查、让使用者了解下一步的操作情况以及允许使用者随意取消整个显示的那些步骤，那么它将变得简洁许多。FtpThread函数透过用hwndStatus句柄呼叫SetWindowText来让使用者知道进展情况，这里指的是对话框中间的静态文字区。

线程可以依照下面的三种方式之一来终止：

第一种，FtpThread可能遇到从WinInet函数传回的错误。如果是这样，它将清除并编排错误字符串的格式，然后将此字符串（连同对话框文字区句柄和「Cancel」按钮的句柄一起）传递给ButtonSwitch。ButtonSwitch是一个小函数，它显示了文字字符串，并将「Cancel」按钮转换成「OK」按钮－不只是按钮上的文字字符串的转换，还包括控件ID的转换。这样就允许使用者按下「OK」按钮来结束对话框。

第二种方式，FtpThread能在没有任何错误的情况下完成任务，其处理方法和遇到错误时的方法一样，只不过对话框中显示的字符串为「Internet Download Complete」。

第三种方式，使用者可以在程序中选择取消下载。这时，DlgProc将PARAMS结构的bContinue字段设定为FALSE。FtpThread频繁地检查该值，如果bContinue等于FALSE，那么函数将做好应该进行的收拾工作，并以NULL文字参数呼叫ButtonSwitch，此参数表示显示了字符串「Internet Session Cancelled」。同样，使用者必须按下「OK」按钮来关闭对话框。

虽然UPDDEMO取得的每个文件只能显示一行，但我（本书的作者）可以用这个程序来告诉您（本书的读者）本书的更新内容以及其它信息，您也可以在网站上发现更详细的信息。因此，UPDDEMO成为我向您传送信息的方法，并且可以让本书的内容延续到最后一页之后。