第十章　功能表及其他资源

大多数 Windows 程式都包含一个自订的图示，Windows 将该图示显示在应用程式视窗标题列的左上角。当程式被列在「开始」功能表中，被显示在萤幕底部的工作列中，被列在 Windows Explorer 中，或者作为快捷方式显示在桌面上时，Windows 也显示该程式的图示。有些程式——大部分是像小画家一类的图形绘制工具——也使用自订滑鼠游标来表示程式的不同操作。还有许多 Windows 程式使用功能表和对话方块。功能表、对话方块加上卷动列，这是标准 Windows 使用者介面的卖点。

图示、游标、功能表和对话方块都是相互关联的，它们是 Windows 的全部资源型态。资源即资料，它们被储存在程式的.EXE 档案中，但是它们并非驻留在程式的资料区域中。也就是说，资源不能从程式原始码中定义的变数直接存取，Windows 提供函式直接或间接地把它们载入记忆体以备使用。我们已经遇到了两个这样的函式，即 LoadIcon 和 LoadCursor，它们出现在范例程式，定义视窗类别结构的内容设定叙述中。它们从 Windows 中载入二进位图示和游标映象，并传回该图示或游标的代号。在本章中，我们先建立自己的图示，它会从程式自己的.EXE 档案中载入。

在本书中，我们将讨论这些资源：

- 图示
- 游标
- 字串
- 自订资源
- 功能表
- 键盘加速键
- 对话方块
- 点阵图

前六个资源在本章讨论，对话方块在第十一章讨论，而点阵图在第十四章讨论。

图示、游标、字串和自订资源

使用资源的好处之一，在於程式的许多元件能够连结编译进程式的.EXE 档案中。如果没有资源这一个概念，如图示图像之类的二进位档案可能会存放在单独的档案中，.EXE 会把它读入记忆体中使用。或者图示不得不在程式中以位

元组阵列的形式定义（这样就无法看到实际的图示图像了）。作为资源，图示储存在开发者电脑上可单独编辑的档案中，但在编译程序中被连结编译进.EXE 档案中。

将图示添加到程式

将资源添加到程式中需要 Visual C++ Developer Studio 的一些附加功能。对於图示来说，可以使用「Image Editor」（也称为「Graphics Editor」）来绘制图示的图像。该图像被储存在副档名为.ICO 的图示档案中。Developer Studio 还产生一个资源描述档（副档名为.RC 的档案，有时也称作资源定义档案），它列出了程式的所有资源和一个让程式引用资源的表头档案（RESOURCE.H）。

因此，您可以看到这些新档案是如何组织在一起的，让我们以建立名为 ICONDEMO 的新专案开始。像往常一样，在 Developer Studio 中从 File 功能表中选择 New ，然後依次选择 专案 页面标签和 Win32 Application 。在 Project Name 栏中键入 ICONDEMO 并单击 OK 。这时，Developer Studio 建立了用於支援工作区和专案的五个档案。这些档案包括文字档案 ICONDEMO.DSW、ICONDEMO.DSP 和 ICONDEMO.MAK（假设当您从 Tools 功能表选择 Open 後，在显示的 Open 对话方块中，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 原始码档案。从 File 功能表上选择 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 档案。键入程式 10-1 中的程式，或选择 Insert 功能表，然後选择 File As Text 选项，从本书附上的光碟中复制原始码。

程式 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) ;
}
```

如果您试著编译该程式,因为在程式开头引用的 RESOURCE.H 档案并不存在，所以会产生错误。然而，您不必直接建立 RESOURCE.H 档案，而是由 Developer Studio 为您建立一个。

您可以通过将资源描述档添加到专案中来做到这一点。从「File」功能表中选择「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 会建立两个文字档案：ICONDEMO.RC（资源描述档）和 RESOURCE.H（允许 C 原始码档案和资源描述档引用相同的已定义识别字）。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档和 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专案时，Developer Studio 才会储存这些档案的新版本。

资源描述档是文字档案。它包括这些资源的可用文字形式表达的描述，例如功能表和对话方块。资源描述档也包括对非文字资源的二进位档案的引用，例如图示和自订的滑鼠游标。

现在，已经存在 RESOURCE.H 档案，您可以试著重新编译一下 ICONDEMO。现在会出现一条错误讯息，指出 IDI_ICON 还没被定义。这个识别字第一次出现在下面的叙述中：

```
wndclass.hIcon  = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
```

在本书前面的程式中，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式使用的是标准的图示，而这里我们的目的是使用自订图示。

那么让我们建立一个图示吧！在 Developer Studio 的「File View」视窗中，您会看到两个档案——ICONDEMO.C 和 ICONDEMO.RC。您开启 CONDEMO.C 後，就可以编辑原始码。开启 ICONDEMO.RC 後，就可以把资源添加到档案中或编辑已存在的资源。要添加图示的话，请从「 Insert 」功能表上选择「 Resource 」选择您想添加的资源，也就是图示，然後再按下「 New 」按钮。

现在呈现的是一个空白的 32×32 图素的图示，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「萤幕颜色」跟「反萤幕颜色」。当一个图素在著色时选择了「萤幕颜色」时，它实际上是透明的。不管图示在什么表面上显示，图示未著色的部分会显示出底色。这样我们就可以建立非矩形的图示。

双击围绕图示的区域，会出现「Icon Properties」对话方块，该对话方块使您能够更改图示的 ID 和档案名称。Developer Studio 可能已经将 ID 设定为 IDI_ICON1，将它改为 IDI_ICON，这样 ICONDEMO 就可以引用图示（字首 IDI 代表「图示的 ID」）。同样地，将档案名改为 ICONDEMO.ICO。

现在选择一种有特色的颜色(如红色)并在图示上画一个大的 B(代表 BIG)，请注意不必像图 10-1 那么整齐。

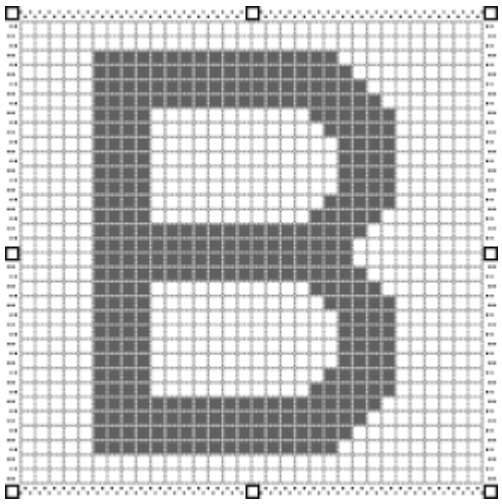


图 10-1 显示在 Developer Studio 中的标准（32×32） ICONDEMO 档案

此时程式应该能够编译并执行得很好了。Developer Studio 将在 ICONDEMO.RC 资源描述档中划一条横线，表示下面是带有识别字（IDI_ICON）的图示档案（ICONDEMO.ICO）。RESOURCE.H 表头档案中会包含 IDI_ICON 识别字的定义。

Developer Studio 通过资源编译器 RC.EXE 编译资源。文字资源描述档被转化为二进位形式，也就是具有副档名.RES 的档案。然後，该已编译的资源档案随同.OBJ 和.LIB 档案一起在 LINK 步骤中被指定连结。这就是资源被添加到最後产生出来的.EXE 档案中的方式。

当您执行 ICONDEMO 时，程式图示显示在标题列的左上角和工作列中。如果您将程式添加到「开始」功能表中，或在桌面上放置捷径，您也会在那儿看到该图示。

ICONDEMO 也在显示区域水平和垂直地重复显示该图示。程式使用叙述

```
hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
```

取得图示的代号。使用叙述

```
cxIcon = GetSystemMetrics (SM_CXICON) ;
cyIcon = GetSystemMetrics (SM_CYICON) ;
```

取得图示的大小。然後，程式通过多次呼叫

```
DrawIcon (hdc, x, y, hIcon) ;
```

显示图示，其中 x 和 y 是被显示图示其左上角的座标。

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 会回报图示的大小为 32×32 图素。这是我们在 Developer Studio 中建立的图示大小，它也是图示出现在桌面上和显示在 ICONDEMO 程式显示区域的大小。然而，这个大小并非显示在程式的标题列或工作列中的图示大小。小图示的大小可以由带有 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）」）。对於目前使用的大多数显示卡来说，小图

示的大小为 16×16 图素。

这会产生问题。当 Windows 将 32×32 的图示缩小为 16×16 的图示时，必需减少图素的行和列。这样，对於某些比较复杂的图示，就会失真。因此，我们应该为那些图像缩小就会变形的图示建立特殊的 16×16 图素的图示。在 Developer Studio 中图示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」对话方块，此时选择「Small（16×16）」。现在您可以画另一个图示。如图 10-2 所示，画一个「S」（表示「小」）。

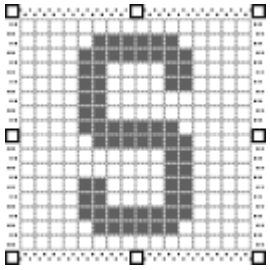


图 10-2 在 Developer Studio 中显示的小（16×16） ICONDEMO 档案

在该程式中您不必做任何事情。第二个图示图像被储存在相同的 ICONDEMO.ICO 档案中，并以相同的 IDI_ICON 识别字引用。在适当的时候，Windows 会自动使用该较小的图示，例如在标题列或工作列中。当在桌面上显示快捷方式，以及程式呼叫 DrawIcon 装饰显示区域时，Windows 会使用大图示。

在掌握这些知识之後，让我们看一看使用图示的详细情况。

取得图示代号

如果您仔细阅读 ICONDEMO.RC 和 RESOURCE.H 档案，会看到由 Developer Studio 产生用於维护档案的一些标记。然而，当编译资源描述档时，只有少数几行是重要的。这些从 ICONDEMO.RC 和 RESOURCE.H 档案中摘录下来的关键部分被列在程式 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
```

程式 10-2 ICONDEMO.RC 和 RESOURCE.H 档案的摘录

程式 10-2 所显示的 ICONDEMO.RC 和 RESOURCE.H 档案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 程式写作者就是这样做的。唯一不同的是 AFXRES.H，它是个表头档案，包含了在建立由机器产生的 MFC 专案时由 Developer Studio 使用的常用识别字。在本书中，我们不会用到 AFXRES.H。

ICONDEMO.RC 中的这行

```
IDI_ICON ICON DISCARDABLE "icondemo.ico"
```

是资源描述档的 ICON 叙述。该图示有一个数值识别字 IDI_ICON，等於 101。由 Developer Studio 添加的 DISCARDABLE 关键字指出，必要时 Windows 可以从记忆体中丢弃图示，以获得额外的空间。之後不需要程式任何特定的操作，Windows 就能够重新载入图示。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 才将档案名加上引号。

当资源编译程序将编译的资源储存在 ICONDEMO.RES 中，并且由连结程式将资源添加到 ICONDEMO.EXE 中以後，该资源就可以经由一个资源型态（RT_ICON）和一个识别字（IDI_ICON 或 101）来标识。程式可以通过呼叫 LoadIcon 函式取得此图示的代号：

```
hIcon = LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ICON)) ;
```

请注意 ICONDEMO 在两个地方呼叫这个函式，一次在定义视窗类别时，另一次在视窗讯息处理程式中取得图示的代号用於绘制。LoadIcon 传回 HICON 型态的值，它是图示的代号。

LoadIcon 的第一个参数，是指出资源来自哪个档案的执行实体代号。使用 hInstance 表示它来自程式自己的.EXE 档案。LoadIcon 的第二个参数实际上被定义为指向字串的指标。待会将会看到，可以使用字串而不是用数值识别字标识资源。巨集 MAKEINTRESOURCE（把整数转换成资源字串）生成指向非数字的指标，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，那么低字组就为图示的数值识别字。图示的识别字必须为 16 位元值。

本书前面的范例程式使用了预先定义的图示：

```
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示。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：图示的识别字能可以为字串吗？答案是可以。方法如下：在 Developer Studio 中，在 ICONDEMO 专案

的档案列表上，选择 IDONDEMO.RC 。您会看到顶端为「IconDemo Resource」的树状结构，然後是资源型态「Icon」，再下来是「IDI_ICON」。如果用滑鼠右键单击图示识别字，并从功能表上选择「 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串。我用这种方法指定资源名称，并在本书的其他地方也使用该方法。

我喜欢为图示（以及一些其他资源）使用文字名称，因为名称可以是程式的名称。例如，假定档案被命名为 MYPROG。如果您使用「Icon Properties」对话方块将图示的 ID 指定为「MyProg」（包括引号），资源描述档将包含下列叙述：

```
MYPROG ICON DISCARDABLE myprog.ico
```

然而，在 RESOURCE.H 中并没有#define 叙述，来指出 MYPROG 是数值识别字。资源描述档将假定 MYPROG 是字串识别字。

在 C 程式中，使用 LoadIcon 函式来取得图示代号。您可能已经有了表示程式名的字串：

```
static TCHAR szAppName [] = TEXT ("MyProg") ;
```

这意味著程式可以使用叙述：

```
hIcon = LoadIcon (hInstance, szAppName) ;
```

来载入图示，这比巨集 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识别字或字串。在「Icon Properties」对话方块中，在 ID 栏中输入数字。资源描述档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示。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 识别初始字元#作为 ASCII 形式中字元数值的开头。

在程式中使用图示

虽然 Windows 以几种方式用图示来代表程式，但是许多 Windows 程式仅在用 WNDCLASS 结构和 RegisterClass 定义视窗类别时指定一个图示。如我们所看到的，这样作用得很好，尤其当图示档案包含标准和较小的图像大小时，更是如此。Windows 在显示图示图像时，它会在图示档案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 有两个附加的栏位：cbSize 和 hIconSm。cbSize 栏位指出了 WNDCLASSEX 结构的大小，假设 hIconSm 被设定为小图示的图示代号。

这样，在 WNDCLASSEX 结构中，您可以设定与两个图示档案相关的两个图示代号——一个用於标准图示，一个用於小图示。

有这种必要吗？没有。正如我们看到的，Windows 已经从单个图示档案中提取了大小合适的图示图像。RegisterClassEx 似乎没有 RegisterClass 聪明。如果 hIconSm 栏位使用了包含多个图像的图示档案，则只有第一个图像能被利用。它可能是标准大小的图示，使用时才被缩小。RegisterClassEx 似乎是为了使用多个图示图像而设计的，每个图像只包含一种图示大小。因为现在可以将多个图示大小包括在同一个图示档案中，所以我建议使用 WNDCLASS 和 RegisterClass。

如果您想在程式执行的时候，动态地更改程式的图示，可以使用 SetClassLong 来达到目的。例如，如果您有与识别字 IDI_ALTICON 相关的第二个图示档案，则您可以使用以下的叙述将其切换到那个图示：

```
SetClassLong (hwnd, GCL_HICON,
    LoadIcon (hInstance, MAKEINTRESOURCE (IDI_ALTICON))) ;
```

如果不想储存程式图示的代号，但要使用 DrawIcon 函式在别处显示它，可以使用 GetClassLong 获得代号。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 中对同一个图示呼叫了 LoadIcon 两次。这不会产生问题，也没有使用额外的记忆体。LoadIcon 是取得代号但不需要清除代号的少数几个函式之一。实际上有一个 DestroyIcon 函式，但它与 CreateIcon、CreateIconIndirect 和 CreateIconFromResource 连在一起使用。这些函式使程式能够动态地建立图示图像。

使用自订游标

在程式中使用自订的滑鼠游标与使用自订的图示相似，只是大多数程式写作者总是使用 Windows 提供的游标。自订游标一般为单色，大小为 32 32 图素。在 Developer Studio 中建立游标与建立图示的方法相同（从「 Insert 」功能表上选择「 Resource 」，然後单击「 Cursor 」），但不要忘记定义热点。

可以在物件类别定义中设定自订游标，叙述为：

```
wndclass.hCursor = LoadCursor (hInstance, MAKEINTRESOURCE (IDC_CURSOR)) ;
```

如果游标用文字名称定义，则为：

```
wndclass.hCursor = LoadCursor (hInstance, szCursor) ;
```

每当滑鼠位於根据这个类别建立的视窗上时，就会显示与 IDC_CURSOR 或 szCursor 相对应的滑鼠游标。

如果使用了子视窗，那么您可能希望游标随著所在视窗的不同而有所区别。如果程式为这些子视窗定义了视窗类别，就可以在每个视窗类别中适当地设定 hCursor 栏位，让每个视窗类别使用不同的游标。如果使用了预先定义的子视窗控制项，就可以使用以下方法改变视窗类别的 hCursor 栏位：

```
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视窗，就可以使用 SetCursor 来改变滑鼠游标：

```
SetCursor (hCursor) ;
```

在处理 WM_MOUSEMOVE 讯息处理期间，您应该呼叫 SetCursor；否则，当游标移动时，Windows 将使用视窗类别中定义的游标来重画游标。文件指出，如果没有改变游标，则 SetCursor 速度将会很快。

字串资源

把字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数的一般字串时，并没有碰到任何问题。

字串资源主要是为了让程式转换成其他语言时更为方便。正如後面两章中将看到的一样，功能表和对话方块也是资源描述档的一部分。如果使用字串资源而不是将字串直接放入原始码中，那么程式所使用的所有文字将在同一档案——资源描述档中。如果转换了资源描述档中的文字，那么建立程式的另一种语言版本所需做的一切就是重新连结程式。这种方法比重新组织原始码安全得多（然而，除了下一个范例程式，我在本书的其他程式中不使用字串表，原因是字串表使程式码看起来更为模糊和复杂）。

您可以在「 Insert 」功能表中选择「 Resource 」，再选择「 String Table 」，建立一个字串表。字串会显示在萤幕右边的列表中。通过双击字串就可以选中它。针对每个字串，您可以指定识别字和字串的内容。

在资源描述中，字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"
      其他字串定义
END
```

如果您在替早期版本的 Windows 写程式，并在文字编辑器中手动建立这个

字串表（用 Developer Studio 来做这件事当然更容易得多了），您可以用左右大括弧代替 BEGIN 和 END 叙述。

资源描述可以包含多个字串表，但是每个 ID 必须唯一表示一个字串。每个字串占一行，最多 4097 个字元。\t 可以作为跳位字元，\n 则作为 linefeed 字元号。DrawText 和 MessageBox 函式能够识别这些控制符号。

您的程式可以使用 LoadString 呼叫把字串复制到程式资料段的缓冲区中：

```
LoadString (hInstance, id, szBuffer, iMaxLength) ;
```

参数 id 是 ID，它加在资源描述档中每个字串的前面；szBuffer 是指向接收字串的字元阵列的指标；iMaxLength 是送入 szBuffer 中的最大字元数。函式传回字串中的字元数。

每个字串前面的 ID 一般是定义在表头档案中的巨集识别字。许多 Windows 程式写作者使用字首 IDS_ 来表示字串的 ID。有时，档案名称或其他资讯需要在字串显示时插入到字串中。在这种情况下，您可以将 C 的格式化字元放入字串，并把它用於 wsprintf 中作为一个格式化字串。

所有资源文字——包括字串表中的文字——以 Unicode 格式储存在.RES 编译资源档案以及最终的.EXE 档案中。LoadStringW 函式直接载入 Unicode 文字。LoadStringA 函式（仅在 Windows 98 下有效）完成由 Unicode 到本地内码表的文字转换。

让我们来看一个程式，它使用三个字串，在讯息方块中显示三条错误资讯。RESOURCE.H 表头档案为这些资讯定义了三个识别字：

```
#define IDS_FILENOTFOUND          1
#define IDS_FILETOOBIG          2
#define IDS_FILEREADONLY          3
```

资源描述档具有此字串表：

```
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 原始码档案也包含这个表头档案，并定义了一个显示讯息方块的函式（我假定 szAppName 是一个包含程式名称的整体变数）。

```
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」资讯的讯息方块，程式呼叫：

```
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——程式写作者，而不是那个使用您程式的幸运者）。自订资源让连结.EXE 档案中的各种资料更为方便，对取得程式中的资料也是如此。资料可以是您需要的任何格式。程式用於存取自订资源的 Windows 函式促使 Windows 将资料载入记忆体并传回指向它的指标。然後您就可以对程式做任何操作。您会发现对於储存和存取各种自己的资料，这要比把资料储存在外部档案中，再使用档案输入函式存取它要方便得多。

例如，您有一个档案叫做 BINDATA.BIN，它包含程式需要显示的一些资料。您可以选择这个档案的格式。如果在 MYPROG 专案中有 MYPROG.RC 资源描述档，您就可以在 Developer Studio 中从「 Insert 」功能表中选择「 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。然後，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位资料的视窗。但是您不必输入什么，用滑鼠右键单击 IDR_BINTYPE1 名称，并选择 Properties ，然後就可以输入一个档案名称：例如，BINDATA.BIN。

资源描述档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。有了图示後，您可以对资源名称使用文字的名称，而不是数字的识别字。

当您编译并连结程式，整个 BINDATA.BIN 档案会被并入 MYPROG.EXE 档案中。

在程式的初始化（比如，在处理 WM_CREATE 讯息时）期间，您可以获得资源的代号：

```
hResource =      LoadResource (   hInstance,
                 FindResource (   hInstance, TEXT ("BINTYPE"),
                                                MAKEINTRESOURCE
(IDR_BINTYPE1))) ;
```

变数 hResource 定义为 HGLOBAL 型态，它是指向记忆体区块的代号。不管它的名称是什么，LoadResource 不会立即将资源载入记忆体。把 LoadResource

和 FindResource 函式如上例般合在一起使用，在实质上就类似於 LoadIcon 和 LoadCursor 函式的做法。事实上，LoadIcon 和 LoadCursor 函式就用到了 LoadResource 和 FindResource 函式。

当您需要存取文字时，呼叫 LockResource：

```
pData = LockResource (hResource) ;
```

LockResource 将资源载入记忆体（如果还没有载入的话），然後它会传回一个指向资源的指标。当结束对资源的使用时，您可以从记忆体中释放它：

```
FreeResource (hResource) ;
```

当您的程式终止时，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——一个图示、一个字串表和一个自订的资源——的范例程式。程式 10-3 所示的 POEPOEM 程式在其显示区域显示 Edgar Allan Poe 的「Annabel Lee」文字。自订的资源是档案 POEPOEM.TXT，它包含了一段诗文，此文字档案以反斜线（\）结束。

程式 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
```

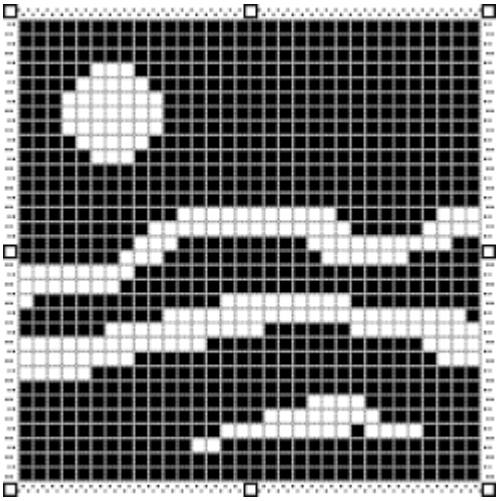
POEPOEM.TXT

```
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 资源描述档中，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 取得资源代号。使用 LockResource 锁定资源，并且使用一个小程式将档案末尾的反斜线（\）换成 0，这有利於後面 WM_PAINT 讯息处理期间使用的 DrawText 函式。

注意，这里使用的是子视窗的卷动列，而不是视窗卷动列，这是因为子视窗卷动列有一个自动的键盘介面，因此在 POEPOEM 中没有处理 WM_KEYDOWN。

POEPOEM 还使用三个字串，它们的 ID 在 RESOURCE.H 表头档案中定义。在程式的开始，IDS_APPNAME 和 IDS_CAPTIONPOEPOEM 字串由 LoadString 载入记忆体：

```
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 必须将资源字串由 Unicode 转化为 ANSI，而 LoadStringW 仅是直接载入它），LoadStringW 在 Windows 98 下不被支援。这意味著在 Windows 98 下，当 RegisterClassW 函式失败时，MessageBoxW 函式（Windows 98 支援）就不能使用 LoadStringW 载入程式的字串。由於这个原因，程式使用 LoadStringA 载入 IDS_APPNAME 和 IDS_ERRMSG 字串，并使用 MessageBoxA 显示自订的讯息方块：

```
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 字串变数是指向 char 的指标。

既然我们已经定义了用於 POEPOEM 的所有字串资源，那么翻译者将程式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。

功能表

您还记得 Monty Python 有关乳酪店的幽默短剧吗？那故事内容是这样的：一个客人走进乳酪店想买某种乳酪。当然，店里没有这种乳酪。因此他又问有没有另一种乳酪，然後再问另一种，再问另一种，不断的问店家有没有另一种乳酪（最後总共问了 40 种的乳酪），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功能表的使用来避免。一个功能表是一列可用的选项，它告诉饥饿的用餐者，厨房可以提供哪些服务，并且——对於 Windows 程式来说——还告诉使用者一个应用程式能够执行哪些操作。

功能表可能是 Windows 程式提供的一致使用者介面中最重要的部分，而在您的程式中增加功能表，是Windows程式设计中相对简单的部分。您在Developer Studio 中定义功能表。每个可选的功能表项被赋予唯一的 ID。您在视窗类别结构中指定功能表名称。当使用者选择一个功能表项时，Windows 给您的程式发送包含该 ID 的 WM_COMMAND 讯息。

讨论完功能表後，我还将讨论键盘加速键，它们是一些键的组合，主要用於启动功能表功能。

功能表概念

视窗的功能表列紧接在标题列的下方显示，这个功能表列有时被称为「主功能表」或「顶层功能表」。列在顶层功能表的项目通常是下拉式功能表，也叫做「突现式功能表」或「子功能表」。您也可以定义多重嵌套的突现式功能表，也就是说，在突现式功能表上的项目可以存取另一个突现式功能表。有时突现式功能表上的项目呼叫对话方块以获得更多的资讯（对话方块在下一章介绍）。在标题列的最左端，很多父视窗都显示程式的小图示，这个图示可以启动系统功能表。它实际上是另一个突现式功能表。

突现式功能表的各项可以是「被选中的」，这意味著 Windows 在功能表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功能表中选中了哪些选项。这些选项之间可以是互斥的，也可以不互斥。顶层功能表项不能被选中。

顶层功能表或突现式功能表项可以被「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。被启用或禁用的功能表项在使用者看来是一样的，但是无效化的功能表项是使用灰色文字来显示的。

从使用者的角度来看，启用、禁用和无效化的功能表项都是可以「选择的」（被选择的功能表项目会被加高亮度显示），也就是说，使用者可以使用滑鼠选择被禁用的功能表项，将反相显示游标列移动到禁用的功能表项上，或者使用功能表项的关键字母来选择该功能表项。然而，从程式写作者的角度来看，启用、禁用和无效化功能表项的功能是不同的。Windows 只为启用的功能表项向程式发送 WM_COMMAND 讯息。要让选项变得无效，可以把那些功能表项禁用和无效化。如果您想让使用者知道选择是无效的，那么您可以让一个功能表项无效化。

功能表结构

当您建立或改变程式中的功能表时，把顶层功能表和每一个突现式功能表想像成各自独立的功能表是有用的。顶层功能表有一个功能表代号，在顶层功能表中的每一个突现式功能表也有它自己的功能表代号。系统功能表（也是一个突现式功能表）也有功能表代号。

功能表中的每一项都有三个特性。第一个特性是功能表中显示什么，它可以是字串或点阵图。第二个特性是 WM_COMMAND 讯息中 Windows 发送给程式的功能表 ID，或者是在使用者选择功能表项时 Windows 显示的突现式功能表的代号。第三个特性是功能表项的属性，包括是否被禁用、无效化或被选中。

定义功能表

要使用 Developer Studio 来给程式资源描述档添加功能表，可以从 Insert 功能表中选择 Resource 并选择 Menu （或者您可能已经知道了）。然後，您可以用交谈式的方式定义功能表。功能表中每一项都有一个相关的 Menu Item Properties 对话方块，指出该项目的字串。如果选中了 Pop-up 核取方块，该项目就会呼叫一个突现式功能表，并且没有 ID 与此项目相联系。如果没有选中 Pop-up 核取方块，该项目被选中时就会产生带有特定 ID 的 WM_COMMAND 讯息。这两类功能表项分别出现在资源描述档的 POPUP 和 MENUITEM 叙述中。

当您为功能表中的项目键入文字时，可以键入一个「&」符号，指出後面一个字元在 Windows 显示功能表时要加底线。这种底线字元是在您使用 Alt 键选择功能表项时 Windows 要寻找的比对字元。如果在文字中不包括「&」符号，就不显示任何底线，Windows 会将功能表项文字的第一个字母用於 Alt 键查找。

如果在 Menu Items Properties 对话方块中选中 Grayed 选项，则功能表项是不能启动的，它的文字是灰色的，该项不产生 WM_COMMAND 讯息。如果选

中 Inactive 选项，则功能表项也是不能启动的，也不产生 WM_COMMAND 讯息，但是它的文字显示正常。 Checked 选项在功能表项边上放置一个选中标记。 Separator 选项在突现式功能表上产生一个分栏的横线。

在突现式功能表的项目上，可以在字串中使用跳位字元\t。紧接著\t 的文字被放置在距离突现式功能表的第一列右边新的一列上。在本章後面，会看到在使用键盘加速键时它起的作用。字串中的\a 使跟著它的文字向右对齐。

您指定的 ID 值是 Windows 发送给视窗讯息处理程式中功能表讯息中的数值。在功能表中 ID 值应该是唯一的。按照惯例，我使用以 IDM(「ID for a Menu」)开头的识别字。

在程式中引用功能表

大多数 Windows 应用程式在资源描述档中只有一个功能表。您可以给功能表起一个与程式名称相同的文字的名称。程式写作者经常将程式名用於功能表名称，以便相同的字串可以用於视窗类别、程式的图示名称和功能表名称。然後，程式在视窗的定义中为功能表引用该名称：

```
wndclass.lpszMenuName = szAppName ;
```

虽然存取功能表资源的最常用方法是在视窗类别中指定功能表，您也可以使用其他方法。Windows 应用程式可以使用 LoadMenu 函式将功能表资源载入记忆体中，如同 LoadIcon 和 LoadCursor 函式一样。LoadMenu 传回一个功能表代号。如果您在资源描述档中为功能表使用了名称，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)) ;
```

然後，您可以将这个功能表代号作为 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 呼叫中指定的功能表可以覆盖视窗类别中指定的任何功能表。如果 CreateWindow 的第九个参数是 NULL，那么您可以把视窗类别中的功能表看作是这种视窗类别的视窗内定使用的功能表。这样，您可以为依据同一视窗类别建立的几个视窗使用不同的功能表。

您也可以在视窗类别中指定 NULL 功能表，并且在 CreateWindow 呼叫中也指定 NULL 功能表，然後在视窗被建立後再给视窗指定一个功能表：

```
SetMenu (hwnd, hMenu) ;
```

这种形式使您可以动态地修改视窗的功能表。在本章後面的 NOPOPUPS 程式

中我们将会看到这方面的例子。

当视窗被清除时，与视窗相关的所有功能表都将被清除。与视窗不相关的功能表在程式结束前通过呼叫 DestroyMenu 主动清除。

功能表和讯息

当使用者选择一个功能表项时，Windows 通常向视窗讯息处理程式发送几个不同的讯息。在大多数情况下， 您的程式可以忽略大部分讯息，只需把它们传递给 DefWindowProc 即可。WM_INITMENU 就是这一类的讯息，它具有下列参数：

- wParam： 主功能表代号
- lParam： 0

wParam 值是您的主功能表代号，即使使用者选择的是系统功能表中的项目。Windows 程式通常忽略 WM_INITMENU 讯息。尽管在选中该项之前的讯息已经给程式提供了修改功能表的机会，但是我们觉得此刻改变顶层功能表是会扰乱使用者的。

程式也会接收到 WM_MENUSELECT 讯息。随著使用者在功能表项中移动游标或者滑鼠，程式会收到许多 WM_MENUSELECT 讯息。这对实作那些包含对功能表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：功能表 ID 或者突现式功能表代号
- HIWORD (wParam)：选择旗标
- lParam: 包含被选中项目的功能表代号

WM_MENUSELECT 是一个功能表追踪讯息，wParam 的值告诉您目前选择的是功能表中的哪一项（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。如果您需要根据对功能表项的选择来改变视窗显示区域的内容，那么您可以使用 WM_MENUSELECT 讯息。许多程式把该讯息发送给 DefWindowProc。

当 Windows 准备显示一个突现式功能表时，它给视窗讯息处理程式发送一个 WM_INITMENUPOPUP 讯息，参数如下：

- wParam： 突现式功能表代号
- LOWORD (lParam)：突现式功能表索引
- HIWORD (lParam)： 系统功能表为 1，其他为 0

如果您需要在显示突现式功能表之前启用或者禁用功能表项，那么这个讯息就很重要。例如，假定程式使用突现式功能表上的 Paste 命令从 剪贴簿 复制文字，当您收到突现式功能表中的 WM_INITMENUPOPUP 讯息时，应确定剪贴簿

内是否有文字存在。如果没有，那么应该使 Paste 功能表项无效化。我们将在本章後面修改的 POPPAD 程式中看到这样的例子。

最重要的功能表讯息是 WM_COMMAND，它表示使用者已经从功能表中选中了一个被启用的功能表项。第八章中的 WM_COMMAND 讯息也可以由子视窗控制项产生。如果您碰巧为功能表和子视窗控制项使用同一 ID 码，那么您可以通过 lParam 的值来区别它们，功能表项的 lParam 其值为 0，请参见表 10-1。

表 10-1

	功能表	控制项
LOWORD (wParam):	功能表 ID	控制项 ID
HIWORD (wParam):	0	通知码
lParam:	0	子视窗代号

WM_SYSCOMMAND 讯息类似於 WM_COMMAND 讯息，只是 WM_SYSCOMMAND 表示使用者从系统功能表中选择一个启用的功能表项：

- wParam: 功能表 ID
- lParam: 0

然而，如果 WM_SYSCOMMAND 讯息是由按滑鼠按键产生的，LOWORD（lParam）和 HIWORD（lParam）将包含滑鼠游标位置的 x 和 y 萤幕座标。

对於 WM_SYSCOMMAND，功能表 ID 指示系统功能表中的哪一项被选中。对於预先定义的系统功能表项，较低的那四个位元应该和 0xFFF0 进行 AND 运算来遮罩掉，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。

如果您在系统功能表中添加功能表项，那么 wParam 的低字组将是您定义的功能表 ID。为了避免与预先定义的功能表 ID 相冲突，应用程式应该使用小於 0xF000 的值，这对於将一般的 WM_SYSCOMMAND 讯息发送给 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功能表命令。

我们将讨论的最後一个讯息是 WM_MENUCHAR。实际上，它根本不是功能表讯息。在下列两种情况之一发生时，Windows 会把这个讯息发送到视窗讯息处理程式：如果使用者按下 Alt 和一个与功能表项不匹配的字元时，或者在显示突现式功能表而使用者按下一个与突现式功能表里的项目不匹配的字元键时。随 WM_MENUCHAR 讯息一起发送的参数如下所示：

- LOWORD (wParam): 字元代码（ASCII 或 Unicode）
- HIWORD (wParam): 选择码

- lParam: 功能表代号

选择码是：

- 0 不显示突现式功能表
- MF_POPUP 显示突现式功能表
- MF_SYSMENU 显示系统突现式功能表

Windows 程式通常把该讯息传递给 DefWindowProc，它一般给 Windows 传回 0，这会使 Windows 发出哔声。在第十四章 GRAFMENU 程式中会看到 WM_MENUCHAR 讯息的使用。

范例程式

让我们来看一个简单的例子。程式 10-4 所示的 MENUDEMO 程式，在主功能表中有五个选择项——File、Edit、Background、Timer 和 Help，每一项都与一个突现式功能表相连。MENUDEMO 只完成了最简单、最通用的功能表处理操作，包括拦截 WM_COMMAND 讯息和检查 wParam 的低字组。

程式 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 资源描述档给了您定义功能表的提示。功能表的名称为「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 」对话方块中选中「 Separator 」框产生的。注意功能表中有一个项目具有「 Checked 」选项，

另一个具有「 Grayed 」选项。还有，「 Background 」突现式功能表中的五个项目应该按顺序输入，确保识别字是以数值的顺序，本程式需要这样。所有功能表项的识别字定义在 RESOURCE.H 中。

当收到突现式功能表「 File 」和「 Edit 」各项有关的 WM_COMMAND 讯息时，MENUDEMO 程式只使系统发出哔声。「 Background 」突现式功能表列出 MENUDEMO 用来给背景著色的五种现有画刷。在 MENUDEMO.RC 资源描述档中，「 White 」功能表项（功能表 ID 为 IDM_BKGND_WHITE）被标以「 CHECKED 」，它在功能表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 」突现式功能表上的五种画刷相互排斥。当 MENUDEMO.C 收到一个 WM_COMMAND 讯息，而该讯息中的 wParam 是「 Background 」突现式功能表上的五项之一时，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，首先要得到功能表代号：

```
hMenu = GetMenu (hwnd) ;
```

CheckMenuItem 函式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) ;
```

视窗类别中的背景颜色於是被替换为新的背景颜色，视窗显示区域变为无效状态，Windows 使用新的背景颜色清除视窗。

Timer 突现式功能表列出了两个选项——「Start」和「Stop」。开始时，「Stop」选项变为灰色的（就像在资源描述档中的功能表定义一样）。当您选择「Start」选项时，MENUDEMO 试图启动一个计时器，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 讯息，并且 wParam 等於 IDM_TIMER_STOP 时，MENUDEMO 程式会停止计数，启用「 Start 」项，然後无效化「 Stop 」选项：

```
EnableMenuItem (hMenu, IDM_TIMER_START, MF_ENABLED) ;
EnableMenuItem (hMenu, IDM_TIMER_STOP,  MF_GRAYED) ;
```

请注意，在计时器执行时，MENUDEMO 程式不可能收到 wParam 等於 IDM_TIMER_START 的 WM_COMMAND 讯息。同样地，在计时器关闭时，MENUDEMO 程式也不可能收到 wParam 等於 IDM_TIMER_STOP 的 WM_COMMAND 讯息。

当 MENUDEMO 收到一个 WM_COMMAND 讯息，而该讯息的参数 wParam 等於 IDM_APP_ABOUT 或 IDM_APP_HELP 时，MENUDEMO 程式显示一个讯息方块（在下一

章中，我们将把讯息方块变为对话方块）。

当 MENUDEMO 程式收到一个 WM_COMMAND 讯息，其参数 wParam 等於 IDM_APP_EXIT 时，它给自己发送一个 WM_CLOSE 讯息。这个讯息与 DefWindowProc 收到 WM_SYSCOMMAND 讯息且 wParam 等於 SC_CLOSE 时发送给视窗讯息处理程式的讯息相同。我们将在本章後面介绍 POPPAD2 时再仔细研究这个问题。

功能表设计规范

在 MENUDEMO 中的「 File 」和「 Edit 」突现式功能表的格式与其他 Windows 程式中的格式非常类似。Windows 的目的之一是为使用者提供一种易懂的介面，而不要求使用者为每个程式重新学习基本操作方式。如果「 File 」和「 Edit 」功能表在每个 Windows 程式中看起来都一样，并且都使用同样的字母和 Alt 键来进行选择，那么当然有助於减轻使用者的学习负担。

除了「 File 」和「 Edit 」突现式功能表外，大多数 Windows 程式的功能表都是不同的。当设计一个功能表时，您应该看一看现有的 Windows 程式以尽量保持一致。当然，如果您认为别的程式是不对的，而您知道正确的方法，那么没有人能够阻止您。同时记住，修改一个功能表，通常只需要修改资源描述档而不必修改您的程式码。即使以後要改变功能表项的位置，也不会有多大的问题。

虽然您的程式功能表在顶层可以有 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，那么请在字串後面加一个惊叹号，表示功能表项不会启动突现式功能表。

较难的一种功能表定义方法

在程式的资源描述档中定义功能表，通常是在您的视窗中添加功能表的最简单方法，但不是唯一的方法。如果您没有使用资源描述档，那么可以使用 CreateMenu 和 AppendMenu 两个函式在程式中建立功能表。在您定义完功能表後，您可以将功能表代号发送给 CreateWindow，或者使用 SetMenu 来设定视窗的功能表。

以下是具体的做法。CreateMenu 简单地把一个代号传回给新功能表：

```
hMenu = CreateMenu () ;
```

功能表一开始为空。AppendMenu 将功能表项插入功能表中。您必须为顶层功能表项和每一个突现式功能表提供不同的功能表代号。突现式功能表是单独构成的，然後将突现式功能表代号插入顶层功能表。程式 10-5 中所示的程式码就是用这种方法建立功能表的，实际上，这个功能表与 MENUDEMO 程式中的功能

表相同。为了简化说明，代码使用 ASCII 字串。

程式 10-5　不使用资源描述档建立与 MENUDEMO 程式相同功能表的 C 程式码

```
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：使用资源描述档功能表模板来制作功能表，会更容易而且更清楚。我并不鼓励您使用这里的方法定义功能表，而只是提供了一种实作功能表的方法。当然，您可以使用包含所有功能表项字串、ID 和旗标等的结构阵列来压缩程式码大小。不过，如果您这么做了，那么您还可以利用 Windows 定义功能表的第三种方法。LoadMenuIndirect 函式接受一个指向 MENUITEMTEMPLATE 型态的结构指标，并传回功能表的代号，该函式在载入资源描述档中的常规功能表模板後，在 Windows 中构造功能表，读者不妨自己尝试一下。

浮动突现式功能表

您还可以在没有顶层功能表列的情况下使用功能表，也就是说，您可以使突现式功能表出现在萤幕顶层的任何位置。一种方法是使用滑鼠右键来启动突现式功能表。程式 10-6 所示的 POPMENU 说明了这种方法。

程式 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
```

资源描述档 POPMENU.RC 定义的功能表与 MENUDEMO.RC 中的功能表非常相似。不同的是，在顶层功能表中只包含一项——一个突现式功能表「MyMenu」，它呼叫「File」、「Edit」、「Background」和「Help」选项。这四个选项垂直一行地出现在突现式功能表上，而不是水平一列地出现在主功能表上。

在 WndProc 中的 WM_CREATE 处理期间,POPMENU 取得此突现式功能表的代号，就是带有文字「MyMenu」的那个突现式功能表：

```
hMenu = LoadMenu (hInst, szAppName) ;
hMenu = GetSubMenu (hMenu, 0) ;
```

在 WM_RBUTTONUP 讯息处理期间，POPMENU 提供了滑鼠指标的位置，将此位置转换为萤幕座标，再将座标值传递给 TrackPopupMenu：

```
point.x = LOWORD (lParam) ;
point.y = HIWORD (lParam) ;
ClientToScreen (hwnd, &point) ;
```

```
TrackPopupMenu (hMenu, TPM_RIGHTBUTTON, point.x, point.y,
                                        0, hwnd, NULL) ;
```

然後，Windows 显示出具有「File」、「Edit」、「Background」和「Help」项的突现式功能表。选择其中任何一项都可以使嵌套的突现式功能表显示在右边，功能表函式与一般的功能表一样。

如果要使用与该程式的主功能表相同的功能表并带有 TrackPopupMenu，您会遇到一些问题，因为函式需要突现式功能表代号。在「Microsoft Knowledge Base」文章 ID Q99806 有提供一些资讯。

使用系统功能表

使用 WS_SYSMENU 样式建立的父视窗，在其标题列的左侧有一个系统功能表按钮。如果您愿意，可以修改这个功能表。在 Windows 程式设计的早期，程式写作者一般把「About」功能表项放入系统功能表。虽然这种方法不常见，但是修改系统功能表往往是一种在短程式中添加功能表的快速偷懒方法。这里唯一的限制是：在系统功能表中增加的命令其 ID 值必须小於 0xF000；否则它们将会与 Windows 系统功能表命令所使用的 ID 值相冲突。还要记住，当您为这些新功能表项在视窗讯息处理程式中处理 WM_SYSCOMMAND 讯息时，您必须把其他的 WM_SYSCOMMAND 讯息发送给 DefWindowProc。如果您不这样做，那么实际上是禁用了系统功能表上的所有正常选项。

程式 10-7 中所示的 POORMENU（「设计不当的个人功能表」）在系统功能表中加入了一个分隔条和三个命令，最後一个命令将删除这些附加的功能表项。

程式 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) ;
}
```

三个功能表 ID 在 POORMENU.C 的开始部分定义：

```
#define IDM_ABOUT           1
#define IDM_HELP            2
#define IDM_REMOVE          3
```

在程式视窗建立之後，POORMENU 得到一个系统功能表的代号：

```
hMenu = GetSystemMenu (hwnd, FALSE) ;
```

第一次呼叫 GetSystemMenu 时，您应该为修改功能表作准备，将第二个参数设定为 FALSE。

使用四个 AppendMenu 呼叫来实作对功能表的修改：

```
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」功能表项将使 POORMENU 删除这些附加的功能表项，这只要把第二个参数设定为 TRUE，再次呼叫 GetSystemMenu 即可：

```
GetSystemMenu (hwnd, TRUE) ;
```

标准系统功能表有下列选项：Restore、Move、Size、Minimize、Maximize 和 Close。它们产生 wParam 分别等於 SC_RESTORE、SC_MOVE、SC_SIZE、SC_MINIMUM、SC_MAXIMUM 和 SC_CLOSE 的 WM_SYSCOMMAND 讯息。尽管 Windows 程式一般不这样做，但是您可以自己处理这些讯息，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功能表的标准选项。Windows 文件中还介绍了一些系统功能表的标准附加项目，这些附加项目使用识别字 SC_NEXTWINDOW、SC_PREVWINDOW、SC_VSCROLL、SC_HSCROLL 和 SC_ARRANGE。您也许会发现，在一些应用程式中将这些命令加入系统功能表是合适的。

改变功能表

我们已经看到了如何使用 AppendMenu 函式为程式定义功能表以及将功能表项加入到系统功能表中。在 Windows 3.0 之前，您不得不被迫使用 ChangeMenu 函式来完成这种工作。ChangeMenu 函式有很多功能，至少在当时，整个 Windows 中它是最复杂的函式之一。现在，许多函式都比 ChangeMenu 函式还要复杂，并且 ChangeMenu 的功能被分解为五个新的函式：

- AppendMenu 在功能表尾部添加一个新的功能表项目
- DeleteMenu 删除功能表中一个现有的功能表项并清除该项目
- InsertMenu 在功能表中插入一个新项目
- ModifyMenu 修改一个现有的功能表项目
- RemoveMenu 从功能表中移走某一项目

如果功能表项是一个突现式功能表，那么 DeleteMenu 和 RemoveMenu 之间的区别就很重要。DeleteMenu 清除突现式功能表，但 RemoveMenu 不清除它。

其他功能表命令

下面是在使用功能表时一些有用的函式。

当您改变顶层功能表项时，直到 Windows 重画功能表列时才显示所做的改变。您可以通过下列呼叫来强迫执行功能表更新：

```
DrawMenuBar (hwnd) ;
```

注意，DrawMenuBar 的参数是视窗代号而不是功能表代号。

您可以使用下列命令来获得突现式功能表的代号：

```
hMenuPopup = GetSubMenu (hMenu, iPosition) ;
```

其中 iPosition 是 hMenu 指示的顶层功能表中突现式功能表项的索引（开始为 0）。然後您可以在其他函式中使用突现式功能表代号（例如在 AppendMenu 函式中）。

您可以使用下列命令获得顶层功能表或者突现式功能表中目前的项数：

```
iCount = GetMenuItemCount (hMenu) ;
```

您可以取得突现式功能表项的功能表 ID：

```
id = GetMenuItemID (hMenuPopup, iPosition) ;
```

其中 iPosition 是功能表项在突现式功能表中的位置（以 0 开始）。

在 MENUDEMO 中您已经看到如何选中、或者取消选中突现式功能表中的某一项：

```
CheckMenuItem (hMenu, id, iCheck) ;
```

在 MENUDEMO 中，hMenu 是顶层功能表的代号，id 是功能表 ID，而 iCheck 的值是 MF_CHECKED 或 MF_UNCHECKED。如果 hMenu 是突现式功能表代号，那么参数 id 是位置索引而不是功能表 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 函式与 CheckMenuItem 函式所完成的工作类似。如果您在具有突现式功能表的顶层功能表项上使用 EnableMenuItem，那么必须在第三个参数中使用 MF_BYPOSITION 识别字，因为功能表项没有功能表 ID。我们将在本章後面所示的 POPPAD2 程式中看到 EnableMenuItem 的一个例子。 HiliteMenuItem 也类似於 CheckMenuItem 和 EnableMenuItem，但是它使用的是 MF_HILITE 和 MF_UNHILITE。当您在功能表项之间移动时，Windows 使用反白显示方式加亮显示功能表项。您通常不需要使用 HiliteMenuItem。

您还需要对您的功能表做些什么呢？还记得我们在功能表中使用了哪些字串吗？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 是功能表 ID),也可以是 MF_BYPOSITION（其中的 id 是位置索引）。函式将字串的 iMaxCount 个位元组复制到 pString 中，并传回复制的位元组数。

或许您也想知道功能表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 识别字的检测来决定目前的属性。

也许现在您对功能表有了一些了解。这时您可能想知道，如果您不再需要功能表时又应该如何处理。您可以使用下面的命令来清除功能表：

```
DestroyMenu (hMenu) ;
```

从而使功能表代号无效。

建立功能表的非正统方法

现在让我们稍微偏离我们所讨论的主题。如果在您的程式中没有下拉式功能表，而是建立了多个没有突现式功能表的顶层功能表，并呼叫 SetMenu 在顶层功能表之间切换，那会是什么样的情形呢？就像 Lotus 1-2-3 中老式的文字模式功能表那样。程式 10-8 中的 NOPOPUPS 程式展示了处理这种情况。在这个程式中，「File」和「Edit」项与 MENUDEMO 程式中的类似，但是却以另一种顶层功能表显示出来。

程式 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，您建立了三个功能表，而不是一个。从「Insert」中选择「Resource」三次，每个功能表有一个不同的名称。当视窗讯息处理程式处理 WM_CREATE 讯息时，Windows 将每个功能表资源载入记忆体：

```
hMenuMain = LoadMenu (hInstance, TEXT ("MenuMain")) ;
hMenuFile = LoadMenu (hInstance, TEXT ("MenuFile")) ;
hMenuEdit = LoadMenu (hInstance, TEXT ("MenuEdit")) ;
```

开始时，程式只显示主功能表：

```
SetMenu (hwnd, hMenuMain) ;
```

主功能表使用字串「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 讯息被发送到视窗讯息处理程式。「File」和「Edit」功能表项以「FILE:」和「EDIT:」开始，表示它们是子功能表。每个功能表的最後一项都是字串「(Main)」，表示传回到主功能表。在这三个功能表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 ;

      其他行程式
      }
      break ;
```

在 WM_DESTROY 讯息处理期间，NOPOPUPS 将程式的功能表设定为主功能表，并呼叫 DestroyMenu 来清除「File」和「Edit」功能表。当视窗被清除时，主功能表将被自动清除。

键盘加速键

加速键是产生 WM_COMMAND 讯息（有些情况下是 WM_SYSCOMMAND）的键组合。许多时候，程式使用加速键来重复常用功能表项的动作（然而，加速键还可以用於执行非功能表功能）。例如，许多 Windows 程式都有一个包含「Delete」或「Clear」选项的「Edit」功能表，这些程式习惯上都将 Del 键指定为该选项的加速键。使用者可以通过「 Alt 键」从功能表中选择「 Delete 」选项，或者只需按下加速键 Del 。当视窗讯息处理程式收到一个 WM_COMMAND 讯息时，它不必确定使用的是功能表还是加速键。

为什么要使用加速键

您也许会问：为什么我应该使用加速键?为什么不能直接拦截 WM_KEYDOWN

或 WM_CHAR 讯息而自己实作同样的功能表功能呢？好处又在哪里呢？对於一个单视窗应用程式，您当然可以拦截键盘讯息，但是使用加速键可以得到一些好处：您不需要把功能表和加速键的处理方式重写一遍。

对於有多个视窗和多个视窗讯息处理程式的应用程式来说，加速键是非常重要的。正如我们所看到的，Windows 将键盘讯息发送给目前活动视窗的视窗讯息处理程式。然而对於加速键， Windows 把 WM_COMMAND 讯息发送给视窗讯息处理程式，该视窗讯息处理程式的代号在 Windows 函式 TranslateAccelerator 中给出。通常这是主视窗，也是拥有功能表的视窗，这意味著无须每个视窗讯息处理程式都把加速键的操作处理程式重写一遍。

如果您在主视窗的显示区域中，使用了非系统模态对话方块（在下一章中会讨论）或者子视窗，那么这种好处就变得非常重要。如果定义一个特定的加速键以便在不同的视窗之间移动，那么，只需要一个视窗讯息处理程式有这个处理程式。子视窗就不会收到加速键引发的 WM_COMMAND 讯息。

安排加速键的几条规则

理论上，您可以使用任何虚拟键或者字元键连同 Shift 键、Ctrl 键或 Alt 键来定义加速键。然而，您应该尽力使应用程式之间协调一致，并且尽量避免干扰 Windows 的键盘使用。在加速键中，应该避免使用 Tab、Enter、Esc 和 Spacebar 键，因为这些键常常用於完成系统功能。

加速键最经常的用途是操作程式的「Edit」功能表中的各项。为这些功能表项推荐的加速键在 Windows 3.0 和 Windows 3.1 之间已有不同，因此通常都要支援如下所列的新旧两套加速键：

表 10-2

功能	旧加速键	新加速键
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资讯的功能键 F1。应该避免使用 F4、F5 和 F6 键，因为这些键常用在多重文件介面（MDI）程式中来完成特殊的功能（将在第十九章中讨论）。

加速键表

您可以在 Developer Studio 中定义加速键表。为了让程式中载入加速键表更为容易，给它和程式名相同的名称（与功能表和图示名也相同）。

每个加速键都有在 Accel Properties 对话方块中定义的 ID 和按键组合。如果您已经定义了功能表，则功能表 ID 会出现在下拉式清单方块中，因此不需要键入它们。

加速键可以是虚拟键或 ASCII 字元与 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 字元。也可以从下拉式清单方块中选取虚拟键。

当您为功能表项定义加速键时，应该将键的组合包含到功能表项的文字中。跳位字元（\t）将文字与加速键分割开，将加速键列在第二列。为了在功能表中为加速键做上标记，可以在文字「Ctrl」、「Shift」或「Alt」之後跟上一个「+」号和一个键名（例如，「Shift+F6」或「Ctrl+F6」）。

加速键表的载入

在您的程式中，您使用 LoadAccelerators 函式把加速键表载入记忆体，并获得该表的代号。 LoadAccelerators 叙述非常类似於 LoadIcon、LoadCursor 和 LoadMenu 叙述。

首先，把加速键表的代号定义为型态 HANDLE：

```
HANDLE hAccel ;
```

然後载入加速键表：

```
hAccel = LoadAccelerators (hInstance, TEXT ("MyAccelerators")) ;
```

正如图示、游标和功能表一样，您可以使用一个数值代替加速键表的名称，然後在 LoadAccelerators 叙述中和 MAKEINTRESOURCE 巨集一起使用该数值，或者把它放在双引号内，前面冠以字元「#」。

键盘代码转换

现在我们将讨论底下这三行程式码，在本书中，截至目前为止建立的所有 Windows 程式中都使用过它们。这些程式码是标准的讯息回圈：

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

下面把上头那段程式码加以修改，以便使用加速键：

```
while  (GetMessage (&msg, NULL, 0, 0))
```

```
{
      if (!TranslateAccelerator (hwnd, hAccel, &msg))
    {
                  TranslateMessage (&msg) ;
                  DispatchMessage (&msg) ;
      }
}
```

TranslateAccelerator 函式确认存放在 msg 讯息结构中的讯息是否为键盘讯息。如果是，该函式将找寻代号为 hAccel 的加速键表。如果找到了一个符合的，则呼叫代号为 hwnd 的视窗讯息处理程式。如果加速键 ID 与系统功能表的功能表项一致，则讯息就是 WM_SYSCOMMAND；否则，讯息为 WM_COMMAND。

当 TranslateAccelerator 传回时，如果讯息已经被转换（并且已经被发送给视窗讯息处理程式），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，而是经过回圈回到 GetMessage 呼叫中。

TranslateMessage 中的参数 hwnd 看起来有点累赘，因为讯息回圈中的其他三个函式都没有要求这个参数。此外，讯息结构本身（结构变数 msg）有一个叫做 hwnd 的成员，它是视窗代号。

该函式有些不同的原因在於：msg 结构的栏位由 GetMessage 呼叫填入。当 GetMessage 的第二个参数为 NULL 时，函式会找寻应用程式所有视窗的讯息。当 GetMessage 传回时，msg 结构的 hwnd 是将要获得讯息之视窗的视窗代号。然而，当 TranslateAccelerator 把键盘讯息转换为 WM_COMMAND 或 WM_SYSCOMMAND 讯息时，它使用函式的第一个参数指定的视窗代号 hwnd 来代替视窗代号 msg.hwnd。Windows 就是这样把所有加速键讯息发送给同一视窗讯息处理程式的，即使另一个应用视窗目前拥有输入焦点。当系统模态对话方块或者讯息方块拥有输入焦点时，TranslateAccelerator 不会转换键盘讯息，因为这些视窗的讯息是不经过程式的讯息回圈的。

在某些情况下，当您程式的另一个视窗（比如一个非系统模态对话方块）拥有输入焦点时，您也许不想转换加速键。您将在下一章中看到如何处理这种情况。

接收加速键讯息

当加速键与系统功能表中的功能表项相对应时，TranslateAccelerator 给视窗讯息处理程式发送一个 WM_SYSCOMMAND 讯息，否则，TranslateAccelerator 给视窗讯息处理程式发送一个 WM_COMMAND 讯息。下表所示为几种可能接收到的 WM_COMMAND 讯息，这些讯息用於加速键、功能表命令以及子视窗控制项：

表 10-3

	加速键	功能表	控制项
LOWORD (wParam)	加速键 ID	功能表 ID	控制项 ID
HIWORD (wParam)	1	0	通知码
lParam	0	0	子视窗代号

如果加速键与一个功能表项对应，那么视窗讯息处理程式还会收到 WM_INITMENU、WM_INITMENUPOPUP 和 WM_MENUSELECT 讯息，就好像选中了功能表选项一样。在处理 WM_INITMENUPOPUP 时，程式往往启用和禁用突现式功能表中的功能表项，因此，在使用加速键时，您仍然能够实作这类功能。如果加速键与一个禁用或者无效化的功能表项相对应，那么，TranslateAccelerator 函式就不会向视窗讯息处理程式发送 WM_COMMAND 或 WM_SYSCOMMAND 讯息。

如果活动视窗已经被最小化，那么 TranslateAccelerator 将为与启用的系统功能表项相对应的加速键向视窗讯息处理程式发送 WM_SYSCOMMAND 讯息，而不是 WM_COMMAND 讯息。TranslateAccelerator 也会为没有任何功能表项与之对应的加速键，来向视窗讯息处理程式发送 WM_COMMAND 讯息。

功能表与加速键应用程式 POPPAD

在第九章，我们建立了一个叫做 POPPAD1 的程式，它使用了子视窗编辑控制项来实作基本的笔记本功能。在这一章中，我们将加入「File」和「Edit」功能表，并称此程式为 POPPAD2。「Edit」功能表的功能表项的功能全部可用；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 如程式 10-9 所示。

程式 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
```

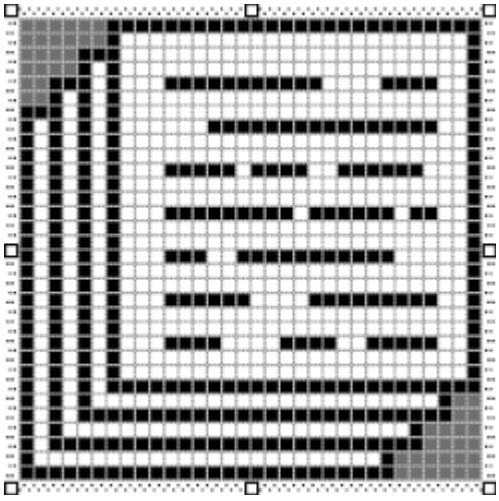
```
      VK_BACK,                IDM_EDIT_UNDO,          VIRTKEY, ALT, NOINVERT
      VK_DELETE,              IDM_EDIT_CLEAR,               VIRTKEY, NOINVERT
      VK_DELETE,              IDM_EDIT_CUT,           VIRTKEY, SHIFT, NOINVERT
      VK_F1,                  IDM_HELP_HELP,          VIRTKEY, 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
```

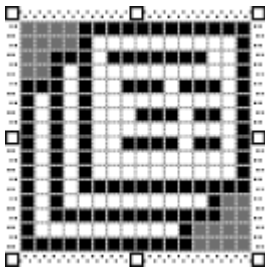
RESOURCE.H （摘录）

```
// 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 资源描述档包含功能表和加速键。您将注意到，所有加速键都表示在跳位字元（\t）後的「Edit」突现式功能表的字串中。

启用功能表项

视窗讯息处理程式的工作包括启用和无效化「Edit」功能表中的选项，这项工作在处理 WM_INITMENUPOPUP 时完成。首先，程式检查是否要显示「Edit」突现式功能表。因为功能表里「Edit」的位置索引（「File」从 0 开始）是 1，因此如果即将显示「Edit」突现式功能表，那么 lParam 应该等於 1。

为了确定是否启用「Undo」选项，POPPAD2 给编辑控制项发送一条 EM_CANUNDO 讯息。如果编辑控制项能够执行「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 识别字通过 IsClipboardFormatAvailable 呼叫来确定这一点：

```
EnableMenuItem (wParam, IDM_PASTE,
      IsClipboardFormatAvailable (CF_TEXT) ? MF_ENABLED : MF_GRAYED) ;
```

只有选择了编辑控制项中的文字，「Cut」、「Copy」和「Delete」选项才能够被启用。给编辑控制项发送一条 EM_GETSEL 讯息，并传回包含此资讯的整数：

```
iSelect = SendMessage (hwndEdit, EM_GETSEL, 0, 0) ;
```

iSelect 的低位元字是第一个被选中字元的位置，iSelect 的高字组是下一个被选中字元的位置。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 ;
```

然後可以将 iEnable 的值用於「Cut」、「Copy」和「Delete」选项：

```
EnableMenuItem (wParam, IDM_CUT,  iEnable) ;
EnableMenuItem (wParam, IDM_COPY, iEnable) ;
EnableMenuItem (wParam, IDM_DEL,  iEnable) ;
```

处理功能表项

当然，如果 POPPAD2 程式不使用子视窗编辑控制项，那么我们将面临一些问题，这涉及如何完成「Edit」功能表中的「Undo」、「Cut」、「Copy」、「Paste」、「Clear」和「Select All」选项。正是编辑控制项使得这种处理变得容易，因为对於每一个选项我们只需向编辑控制项发送一个讯息即可：

```
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 等等的值等於相对应的视窗讯息 WM_UNDO、WM_CUT 的值。

「File」突现式功能表上的「About」选项启动一个简单的讯息方块：

```
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方块。当您从功能表中选择「Help」选项或者按下 F1 加速键时，同样可以启动一个讯息方块。

「Exit」选项向视窗讯息处理程式发送一个 WM_CLOSE 讯息：

```
case  IDM_EXIT :
      SendMessage (hwnd, WM_CLOSE, 0, 0) ;
      return 0 ;
```

这正是 DefWindowProc 收到一个 wParam 等於 SC_CLOSE 的 WM_SYSCOMMAND 讯息时所完成的工作。

在前面的那些程式中，我们没有在视窗讯息处理程式中处理 WM_CLOSE 讯息，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 函式。可以不把 WM_CLOSE 讯息送给 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 中的一个函式，它显示一个请求确认关闭程式的讯息方块：

```
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，讯息方块（以及 AskConfirmation 函式）将传回 IDYES。只有这样，程式才会呼叫 DestroyWindow，否则，程式不会结束。

如果要在程式结束之前确认使用者真的要结束程式，那么您还必须处理 WM_QUERYENDSESSION 讯息。当使用者要关闭 Windows 时，Windows 开始向每个视窗讯息处理程式发送一个 WM_QUERYENDSESSION 讯息。如果有任何一个视窗讯息处理程式处理这个讯息後传回 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 ;
```

如果要在程式结束之前要求使用者的确认，必须处理 WM_CLOSE 和 WM_QUERYENDSESSION 这两个讯息，这就是为什么我们使 POPPAD2 中的「Exit」功能表选项只向视窗讯息处理程式发送一个 WM_CLOSE 讯息的原因。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 讯息，那么您也许还会对 WM_ENDSESSION 讯息感兴趣。Windows 把这个讯息发送给先前收到 WM_QUERYENDSESSION 讯息的每个视窗讯息处理程式。如果由於另一个程式从 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 讯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」功能表中我加上了常见的「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，我们需要使用对话方块。现在是讨论对话方块的时机，也是您准备学习它们的时候了。