第十一章　对话方块

如果有很多输入超出了功能表可以处理的程度，那么我们可以使用对话方块来取得输入资讯。程式写作者可以通过在某选项後面加上省略号（¡）来表示该功能表项将启动一个对话方块。

对话方块的一般形式是包含多种子视窗控制项的弹出式视窗，这些控制项的大小和位置在程式资源描述档的「对话方块模板」中指定。虽然程式写作者能够「手工」定义对话方块模板，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，然後由 Developer Studio 建立对话方块模板。

当程式呼叫依据模板建立的对话方块时，Microsoft Windows 98 负责建立弹出式对话方块视窗和子视窗控制项，并提供处理对话方块讯息（包括所有键盘和滑鼠输入）的视窗讯息处理程式。有时候称呼完成这些功能的 Windows 内部程式码为「对话方块管理器」。

Windows 的内部对话方块视窗讯息处理程式所处理的许多讯息也传递给您自己程式中的函式，这个函式即是所谓的「对话方块程序」或者「对话程序」。对话程序与普通的视窗讯息处理程式类似，但是也存在著一些重要区别。一般来说，除了在建立对话方块时初始化子视窗控制项，处理来自子视窗控制项的讯息以及结束对话方块之外，程式写作者不需要再给对话方块程序增加其他功能。对话程序通常不处理 WM_PAINT 讯息，也不直接处理键盘和滑鼠输入。

对话方块这个主题的含义太广了，因为它还包含子视窗控制项的使用。不过，我们已经在第九章研究了子视窗控制项。当您在对话方块中使用子视窗控制项时，第九章所提到的许多工作都可以由 Windows 的对话方块管理器来完成。尤其是，在程式 COLORS1 中遇到在卷动列之间切换输入焦点的问题也不会在对话方块中出现。Windows 会处理对话方块中的控制项之间切换输入焦点所必需完成的全部工作。

不过，在程式中添加对话方块要比添加图示或者功能表更麻烦一些。我们将从一个简单的对话方块开始，让您对各部分之间的相互联系有所了解。

模态对话方块

对话方块分为两类：「模态的」和「非模态的」，其中模态对话方块最为普遍。当您的程式显示一个模态对话方块时，使用者不能在对话方块与同一个程式中的另一个视窗之间进行切换，使用者必须主动结束该对话方块，这藉由通过按一下「OK」或者「Cancel」键来完成。不过，在显示模态对话方块时，

使用者通常可以从目前的程式切换到另一个程式。而有些对话方块（称为「系统模态」）甚至连这样的切换程式操作也不允许。在 Windows 中，显示了系统模态对话方块之後，要完成其他任何工作，都必须先结束该对话方块。

建立「About」对话方块

Windows 程式即使不需要接收使用者输入，也通常具有由功能表上的「About」选项启动的对话方块，该对话方块用来显示程式的名字、图示、版权旗标和标记为「OK」的按键，也许还会有其他资讯（例如技术支援的电话号码）。我们将要看到的第一个程式除了显示一个「About」对话方块外，别无它用。这个 ABOUT1 程式如程式 11-1 所示：

程式 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) ;
```

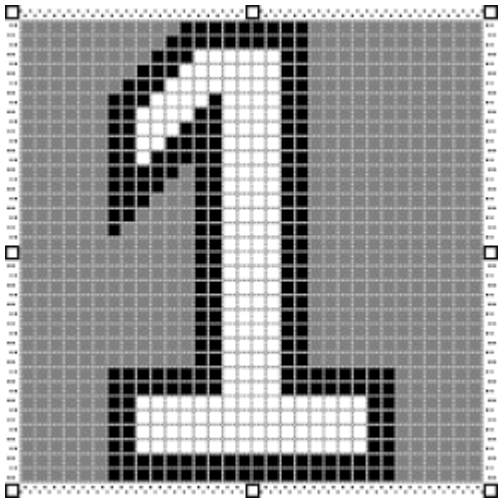
```
}

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



藉由後面章节中介绍的方法，您还可以在程式中建立图示和功能表。图示和功能表的 ID 名均为「About1」。功能表有一个选项，它产生一条 ID 名为 IDM_APP_ABOUT 的 WM_COMMAND 讯息。这使得程式显示的图 11-1 所示的对话方块。

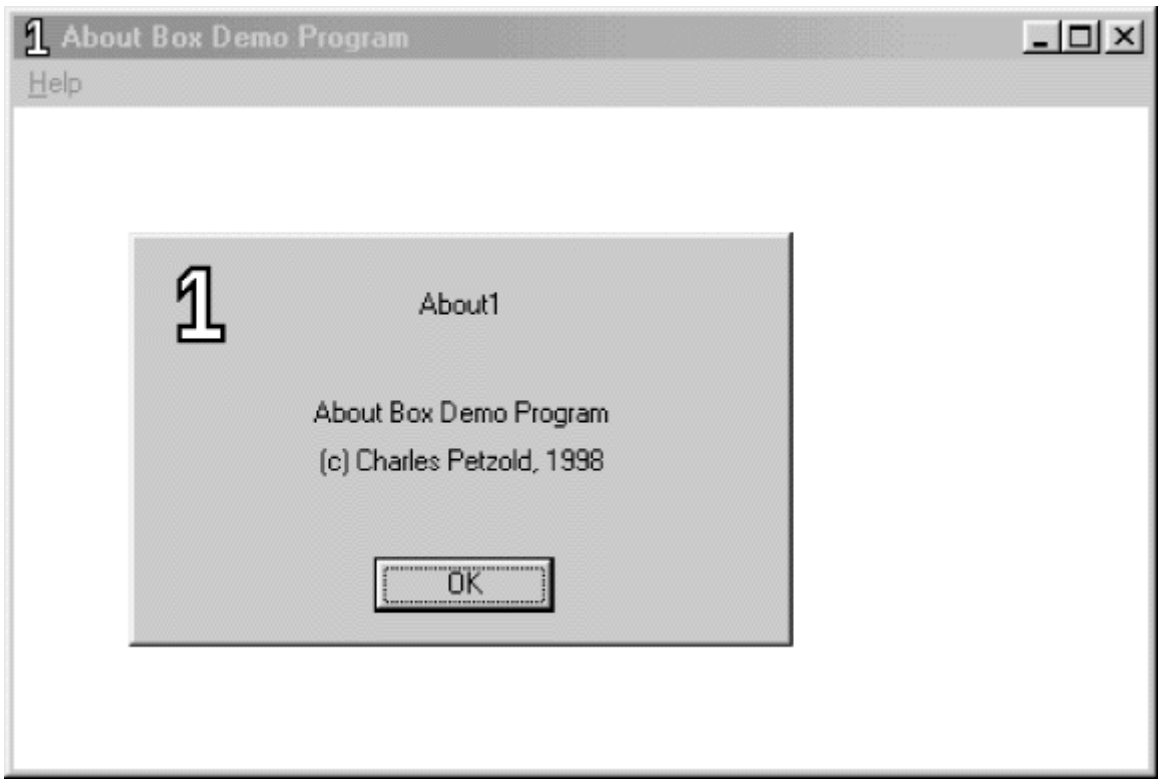


图 11-1 程式 ABOUT1 的对话方块

对话方块及其模板

要把一个对话方块添加到 Visual C++ Developer Studio 会有的应用程式上，可以先从 Insert 功能表中选择 Resource ，然後选择 Dialog Box 。现在一个对话方块出现在您的眼前，该对话方块带有标题列、标题（Dialog）以及 OK 和 Cancel 按钮。 Controls 工具列允许您在对话方块中插入不同的控制项。

Developer Studio 将对话方块的 ID 设为标准的 IDD_DIALOG1。您可以在此名称上（或者在对话方块本身）单击右键，然後从功能表中选择 Properties 。在本程式中，将 ID 改为「AboutBox」（带有引号）。为了与我建立的对话方块保持一致，请将 X Pos 和 Y Pos 栏位改为 32。这表示对话方块相对於程式视窗显示区域左上角的显示位置待会会有关於对话方块座标的详细讨论）。

现在，继续在 Properties 对话方块中选择 Styles 页面标签。因为此对话方块没有标题列，所以不要选取 Title Bar 核取方块。然後请单击 Properties 对话方块的 关闭 按钮。

现在可以设计对话方块了。因为不需要 Cancel 按钮，所以先单击该按钮，然後按下键盘上的 Delete 键。接著单击 OK 按钮，将其移动到对话方块的底部。在 Developer Studio 视窗下面的工具列上有一个小点阵图，它可使控制项在视窗内水平居中对齐，请按下此钮。

如果您要让程式的图示出现在对话方块中，可以这样做：先在浮动的 Controls 工具列中按下「 Pictures 」按钮。将滑鼠移动到对话方块的表面，按下左键，然後拉出一个矩形。这就是图示将出现的位置。然後在次矩形上按下滑鼠右键，从功能表中选择 Properties 。保持 ID 为 IDC_STATIC 。此识别字在 RESOURCE.H 中定义为-1，用於程式中不使用的所有 ID。将 Type 改为 Icon 。您可以在 Image 栏位输入程式图示的名称，或者，如果您已经建立了一个图示，那么您也可以从下拉式清单方块中选择一个名称（About1）。

对於对话方块中的三个静态字串，可以从 Controls 工具列中选择 Static Text ，然後确定文字在对话方块中的位置。右键单击控制项，然後从功能表中选择 Properties 。在 Properties 框的 Caption 栏位中输入要显示的文字。选择 Styles 页面标签，从 Align Text 栏位选择 Center 。

在添加这些字串的时候，若希望对话方块可以更大一些，请先选中对话方块，然後拖曳边框。您也可以选择并缩放控制项。通常用键盘上的游标移动键完成此操作会更容易些。箭头键本身移动控制项，按下 Shift 键後按箭头键，可以改变控制项的大小。所选控制项的座标和大小显示在 Developer Studio 视窗的右下角。

如果您建立了一个应用程式，那么以後在查看资源描述档 ABOUT1.RC 时，您将发现 Developer Studio 建立的模板。我所设计的对话方块模板如下：

```
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
```

第一行给出了对话方块的名称（这里为 ABOUTBOX）。如同其他资源，您也可以使用数字作为对话方块的名称。名称後面是关键字 DIALOG 和 DISCARDABLE 以及四个数字。前两个数字是对话方块左上角的 x、y 座标，该座标在程式呼叫对话方块时，是相对於父视窗显示区域的。後两个数字是对话方块的宽度和高度。

这些座标和大小的单位都不是图素。它们实际上依据一种特殊的座标系统，该系统只用於对话方块模板。数字依据对话方块使用字体的大小而定（这里是 8 点的 MS Sans Serif 字体）：x 座标和宽度的单位是字元平均宽度的 1/4；y 座

标和高度的单位是字元高度的 1/8。因此，对这个对话方块来说，对话方块左上角距离主视窗显示区域的左边是 5 个字元，距离顶边是 2-1/2 个字元。对话方块本身宽 40 个字元，高 10 个字元。

这样的座标系使得程式写作者可以使用座标和大小来大致勾勒对话方块的尺寸和外观，而不管视讯显示器的解析度是多少。由於系统字体字元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 叙述类似於 CreateWindow 呼叫中的 style 栏位。对於模态对话方块，通常使用 WS_POPUP 和 DS_MODALFRAME，我们将在稍後介绍其他的选项。

在 BEGIN 和 END 叙述（或者是左右大括弧，手工设计对话方块模板时，您可能会使用）之间，定义出现在对话方块中的子视窗控制项。这个对话方块使用了三种型态的子视窗控制项，它们分别是 DEFPUSHBUTTON（内定按键）、ICON（图示）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，後面的 iStyle 项是可选的，它使用 Windows 表头档案中定义的识别字来指定其他视窗样式。

DEFPUSHBUTTON、ICON 和 CTEXT 等识别字只可以在对话方块中使用，它们是某种特定视窗类别和视窗样式的缩写。例如，CTEXT 指示这个子视窗控制项类别是「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 识别字，但是在第九章的 COLORS1 程式中已经出现过 WS_CHILD、SS_CENTER 和 WS_VISIBLE 视窗样式，我们在建立静态子视窗文字控制项时已经用到了它们。

对於图示，文字栏位是程式的图示资源名称，它也在 ABOUT1 资源描述档中定义。对於按键，文字栏位是出现在按键里的文字，这个文字相同於在程式中建立子视窗控制项时呼叫 CreateWindow 所指定的第二个参数。

id 栏位是子视窗在向其父视窗发送讯息（通常为 WM_COMMMAND 讯息）时用来标示它自身的值。这些子视窗控制项的父视窗就是对话方块本身，它将这些讯息发送给 Windows 的一个视窗讯息处理程式。不过，这个视窗讯息处理程式也将这些讯息发送给您在程式中给出的对话方块程序。ID 值相同於我们在第九章建立子视窗时，在 CreateWindow 函式中使用的子视窗 ID。由於文字和图示控制项不向父视窗回送讯息，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视窗的位置（相对於对话方块显示区域的左上角）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。

对於 ICON 叙述，宽度和高度将被忽略。

对话方块模板中的 DEFPUSHBUTTON 叙述，除了包含 DEFPUSHBUTTON 关键字所隐含的视窗样式，还包含视窗样式 WS_GROUP。稍後讨论该程式的第二个版本 ABOUT2 时，还会详细说明 WS_GROUP（以及相关的 WS_TABSTOP 样式）。

对话方块程序

您程式内的对话方块程序处理传送给对话方块的讯息。尽管看起来很像是视窗讯息处理程式，但是它并不是真实的视窗讯息处理程式。对话方块的视窗讯息处理程式在 Windows 内部定义，这个视窗程序呼叫您编写的对话方块程序，把它所接收到的许多讯息作为参数。下面是 ABOUT1 的对话方块程序：

```
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 ;
}
```

该函式的参数与常规视窗讯息处理程式的参数相同，与视窗讯息处理程式类似，对话方块程序都必须定义为一个 CALLBACK（callback）函式。尽管我使用了 hDlg 作为对话方块视窗的代号，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式与视窗讯息处理程式的区别：

- 视窗讯息处理程式传回一个 LRESULT。对话方块传回一个 BOOL，它在 Windows 表头档案中定义为 int 型态。
- 如果视窗讯息处理程式不处理某个特定的讯息，那么它将呼叫 DefWindowProc。如果对话方块程序处理一个讯息，那么它传回 TRUE（非 0），如果不处理，则传回 FALSE（0）。
- 对话方块程序不需要处理 WM_PAINT 或 WM_DESTROY 讯息。对话方块程序

不接收 WM_CREAT 讯息，而是在特殊的 WM_INITDIALOG 讯息处理期间，对话方块程序执行初始化操作。

WM_INITDIALOG 讯息是对话方块接收到的第一个讯息，这个讯息只发送给对话方块程序。如果对话方块程序传回 TRUE，那么 Windows 将输入焦点设定给对话方块中第一个具有 WS_TABSTOP 样式（我们将在 ABOUT2 的讨论中加以解释）的子视窗控制项。在这个对话方块中，第一个具有 WS_TABSTOP 样式的子视窗控制项是按键。另外，对话方块程序也可以在处理 WM_INITDIALOG 时使用 SetFocus 来将输入焦点设定为对话方块中的某个子视窗控制项，然後传回 FALSE。

此外，对话方块程序只处理 WM_COMMAND 讯息。这是当按键被滑鼠点中，或者在按钮具有输入焦点的情况下按下空白键时，按键控制项发送给其父视窗的讯息。这个控制项的 ID（我们在对话方块模板中将其设定为 IDOK）在 wParam 的低字组中。对於这个讯息，对话方块程序呼叫 EndDialog，它告诉 Windows 清除对话方块。对於所有其他讯息，对话方块程序传回 FALSE，并告诉 Windows 内部的对话方块视窗讯息处理程式：我们的对话方块程序不处理这些讯息。

模态对话方块的讯息不通过您程式的讯息伫列，所以不必担心对话方块中键盘加速键的影响。

启动对话方块

在 WndProc 中处理 WM_CREATE 讯息时，ABOUT1 取得程式的执行实体代号并将它放在静态变数中：

```
hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;
```

ABOUT1 检查 WM_COMMAND 讯息，以确保讯息 wParam 的低位元字等於 IDM_APP_ABOUT。当它获得这样一个讯息时，程式呼叫 DialogBox：

```
DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;
```

该函式需要执行实体代号（在处理 WM_CREATE 时储存的）、对话方块名称（在资源描述档中定义的）、对话方块的父视窗（也是程式的主视窗）和对话方块程序的位址。如果您使用一个数字而不是对话方块模板名称，那么可以用 MAKEINTRESOURCE 巨集将它转换为一个字串。

从功能表中选择「About About1」，将显示图 11-2 所示的对话方块。您可以使用滑鼠单击「OK」按钮、按空白键或者按 Enter 键来结束这个对话方块。对任何包含内定按钮的对话方块，在按下 Enter 键或空白键之後，Windows 发送一个 WM_COMMAND 讯息给对话方块，并令 wParam 的低字组等於内定按键的 ID，此时的 ID 为 IDOK。按下 Escape 键也可以关闭对话方块，这时 Windows 将发送一个 WM_COMMAND 讯息，并令 ID 等於 IDCANCEL。

直到对话方块结束之後，用来显示对话方块的 DialogBox 才将控制权传回

给 WndProc。DialogBox 的传回值是对话方块程序内部呼叫的 EndDialog 函式的第二个参数(这个值未在 ABOUT1 中使用，但会在 ABOUT2 中使用)。然後，WndProc 可以将控制权传回给 Windows。

即使在显示对话方块时，WndProc 也可以继续接收讯息。实际上，您可以从对话方块程序内部给 WndProc 发送讯息。ABOUT1 的主视窗是弹出式对话方块视窗的父视窗，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 中的对话方块编辑器和其他资源编辑器，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。尤其对於对话方块模板来说，知道了语法，您就可以近一步了解对话方块的范围和限制。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方块模板（就像本章後面的 HEXCALC 程式）。资源编译器和资源描述语法的文件位於/Platform SDK/Windows Programming Guidelines/Platform SDK Tools/Compiling/Using the Resource Compiler。

在 Developer Studio 的「Properties」对话方块中指定了对话方块的视窗样式，它・译成对话方块模板中的 STYLE 叙述。对於 ABOUT1，我们使用模态对话方块最常用的样式；

```
STYLE WS_POPUP | DS_MODALFRAME
```

然而，您也可以尝试其他样式。有些对话方块有标题列，标题列用於指出对话方块的用途，并允许使用者通过滑鼠在显示幕上移动对话方块。此样式为 WS_CAPTION。如果您使用 WS_CAPTION，那么 DIALOG 叙述中所指定的 x 座标和 y 座标是对话方块显示区域的座标，并相对於父视窗显示区域的左上角。标题列将在 y 座标之上显示。

如果使用了标题列，那么您可以用 CAPTION 叙述将文字放入标题中。在对话方块模板中，CAPTION 叙述在 STYLE 叙述的後面：

```
CAPTION "Dialog Box Caption"
```

另外，在对话方块程序处理 WM_INITDIALOG 讯息处理期间，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 样式的系统功能表按钮。此样式允许使用者从系统功能表中选择 Move 或 Close 。

从 Properties 对话方块的 Border 清单方块中选择 Resizing （相同於样式 WS_THICKFRAME），允许使用者缩放对话方块，仅管此操作并不常用。如果

您不介意更特殊一点的话，还可以著为此对话方块样式添加最大化方块。

您甚至可以给对话方块添加一个功能表。这时对话方块模板将包括下面的叙述：

```
MENU menu-name
```

其参数不是功能表的名称，就是资源描述中的功能表号。模态对话方块很少使用功能表。如果使用了功能表，那么您必须确保功能表和对话方块控制项中的所有 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，以供对话方块文字使用。这在过去的对话方块中不常用，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方块都选用 8 点的 MS Sans Serif 字体。一个 Windows 程式能把自己外观打点得非常与众不同，这只需为程式的对话方块及其他文字输出单独准备一种字体即可。

尽管对话方块视窗讯息处理程式通常位於 Windows 内部，但是您也可以使用自己编写的视窗讯息处理程式来处理对话方块讯息。要这样做，您必须在对话方块模板中指定一个视窗类别名：

```
CLASS "class-name"
```

这种用法很少见，但是在本章後面所示的 HEXCALC 程式中我们将用到它。

当您使用对话方块模板的名称来呼叫 DialogBox 时，Windows 通过呼叫普通的 CreateWindow 函式来完成建立弹出式视窗所需要完成的一切操作。Windows 从对话方块模板中取得视窗的座标、大小、视窗样式、标题和功能表，从 DialogBox 的参数中获得执行实体代号和父视窗代号。它所需要的唯一其他资讯是一个视窗类别（假设对话方块模板不指定视窗类别的话）。Windows 为对话方块注册一个专用的视窗类别，这个视窗类别的视窗讯息处理程式可以存取对话方块程序位址（该位址是您在 DialogBox 呼叫中指定的），所以它可以使程式获得该弹出式视窗所接收的讯息。当然，您可以通过自己建立弹出式视窗来建立和维护自己的对话方块。不过，使用 DialogBox 则更简单。

也许您希望受益於 Windows 对话方块管理器，但不希望（或者能够）在资源描述中定义对话方块模板，也可能您希望程式在执行时可以动态地建立对话方块。这时可以完成这种功能的函式是 DialogBoxIndirect，此函式用资料结构来定义模板。

在 ABOUT1.RC 的对话方块模板中，我们使用缩写 CTEXT、ICON 和 DEFPUSHBUTTON 来定义对话方块所需要的三种型态的子视窗控制项。您还可以使用其他型态，每种型态都隐含一个特定的预先定义视窗类别和一种视窗样式。下表显示了与一些控制项型态相同的视窗类别和视窗样式：

表 11-1

控制项型态	视窗类别	视窗样式
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	静态图示	SS_ICON
EDITTEXT	编辑框	ES_LEFT | WS_BORDER | WS_TABSTOP
SCROLLBAR	卷动列	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式。除了表中所示的视窗样式外，每个控制项还具有下面的样式：

```
WS_CHILD | WS_VISIBLE
```

对於这些控制项型态，除了 EDITTEXT、SCROLLBAR、LISTBOX 和 COMBOBOX 之外，控制项叙述的格式为：

```
control-type "text", id, xPos, yPos, xWidth, yHeight, iStyle
```

对於 EDITTEXT、SCROLLBAR、LISTBOX 和 COMBOBOX，其格式为：

```
control-type id, xPos, yPos, xWidth, yHeight, iStyle
```

其中没有文字栏位。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视窗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：对话方块模板中指定大小的单位为平均字元宽度的 1/4，及平均字元高度的 1/8。

控制项叙述的 style 栏位是可选的。它允许您包含其他视窗样式识别字。例如，如果您想建立在正方形框左边包含文字的核取方块，那么可以使用：

```
CHECKBOX "text", id, xPos, yPos, xWidth, yHeight, BS_LEFTTEXT
```

注意，控制项型态 EDITTEXT 会自动添加一个边框。如果您想建立一个没有边框的子视窗编辑控制项，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制项叙述：

```
CONTROL "text", id, "class", iStyle, xPos, yPos, xWidth, yHeight
```

此叙述允许您通过指定视窗类别和完整的视窗样式，来建立任意型态的子视窗控制项。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
```

当编译资源描述档时，这两条叙述在.RES 和.EXE 档案中的编码是相同的。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 程式中，我向您展示了如何用此选项建立一个控制项，且在您的程式中已定义了该控制项的视窗类别。

当您在对话方块模板中使用 CONTROL 叙述时，不必包含 WS_CHILD 和 WS_VISIBLE 样式。在建立子视窗时，Windows 已经包含了这些视窗样式。CONTROL 叙述的格式也说明 Windows 对话方块管理器在建立对话方块时就完成了此项操作。首先，就像我前面所讨论的，它建立一个弹出式视窗，其父视窗代号在 DialogBox 函式中提供。然後，对话方块管理器为对话方块模板中的每个控制项建立一个子视窗。所有这些控制项的父视窗均是这个弹出式对话方块。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 是系统字体字元的宽度和高度，以图素为单位。hDlg 参数是从建立该对话方块视窗的 CreateWindow 呼叫传回的值；hInstance 参数是从 DialogBox 呼叫获得的。

更复杂的对话方块

ABOUT1 中的简单对话方块展示了设计和执行一个对话方块的要点，现在让我们来看一个稍微复杂的例子。程式 11-2 给出的 ABOUT2 程式展示了如何在对话方块程序中管理控制项（这里用单选按钮）以及如何在对话方块的显示区域中绘图。

程式 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
```

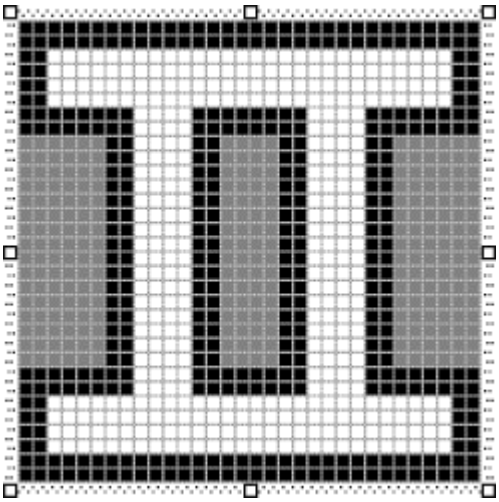
```
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
FONT 8, "MS Sans Serif"
BEGIN
      ICON
      "ABOUT2",IDC_STATIC,7,7,20,20
      CTEXT        "About2",IDC_STATIC,57,12,86,8
      CTEXT        "About Box Demo Program",IDC_STATIC,7,40,186,8
      LTEXT        "",IDC_PAINT,114,67,74,72
      GROUPBOX                          "&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方块内，其内部以目前选择的颜色著色。使用者按下「OK」按钮後，对话方块会终止，程式的视窗讯息处理程式在它自己的显示区域内绘出所选图形。如果您按下「Cancel」，则主视窗的显示区域会保持原样。对话方块如图 11-2 所示。尽管 ABOUT2 使用预先定义的识别字 IDOK 和 IDCANCEL 作为两个按键，但是每个单选按钮均有自己的识别字，它们以字首 IDC 开头（用於控制项的 ID）。这些识别字在 RESOURCE.H 中定义。

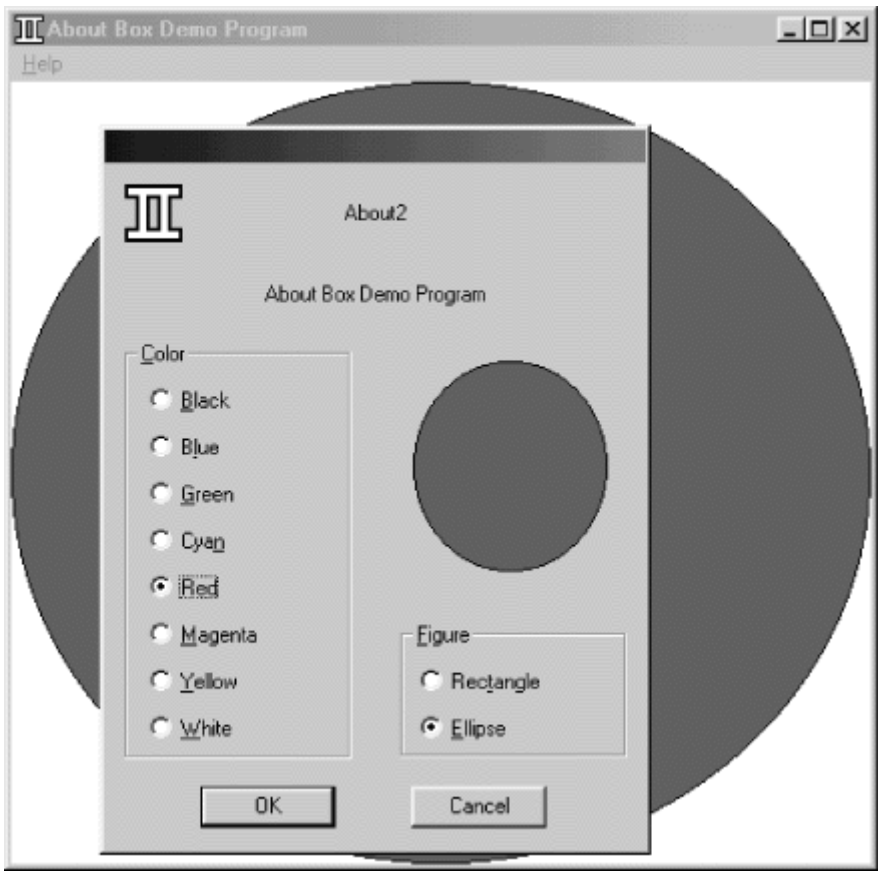


图 11-2 ABOUT2 程式的对话方块

当您在 ABOUT2 对话方块中建立单选按钮时，请按显示顺序建立。这能保证 Developer Studio 依照顺序定义识别字的值，程式将使用这些值。另外，每个单选按钮都不要选中「Auto」选项。「Auto Radio Button」需要的程式码较少，但基本上处理起来更深奥些。然後请依照 ABOUT2.RC 中的定义来设定它们的识别字。

选中「Properties」对话方块中下列物件的「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」核取方块。

当您有全部控制项在对话方块中的近似位置和大小时，就可以从「Layout」功能表选择「Tab Order」选项。按 ABOUT2.RC 资源描述中显示的顺序单击每一个控制项。

使用对话方块控制项

在第九章中，您会发现大多数子视窗控制项发送 WM_COMMAND 讯息给其父视窗（唯一例外的是卷动列控制项）。您还看到，经由发送讯息给子视窗控制项，父视窗可以改变子视窗控制项的状态（例如，选择或不选择单选按钮、核取方块）。您也可以用类似方法在对话方块程序中改变控制项。例如，如果您设计了一系列单选按钮，就可以发送讯息给它们，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方块控制项的简单办法。我们来看一看对话方块程序与子视窗控制项相互通信的方式。

ABOUT2 的对话方块模板显示在程式 11-2 的 ABOUT2.RC 资源描述档中。GROUPBOX 控制项只是一个带标题（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。

当用滑鼠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白键），子视窗向其父视窗发送一个 WM_COMMAND 讯息，讯息的 wParam 的低字组被设为控制项的 ID，wParam 的高字组是一个通知码，lParam 值是控制项的视窗代号。对於单选按钮，这个通知码是 BN_CLICKED 或者 0。然後 Windows 中的对话方块视窗讯息处理程式将这个 WM_COMMAND 讯息发送给 ABOUT2.C 内的对话方块程序。当对话方块程序收到一个单选按钮的 WM_COMMAND 讯息时，它为此按钮设定选中标记，并为组中其他按钮清除选中标记。

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视窗控制项发送 BM_CHECK 讯息。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 参数是子视窗按钮控制项的视窗代号。

但是在对话方块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视窗代号，只是从您获得的讯息中知道其中一个代号。幸运的是，Windows 为您提供了一个函式，可以用对话方块代号和控制项 ID 来取得一个对话方块控制项的视窗代号：

```
hwndCtrl = GetDlgItem (hDlg, id) ;
```

（您也可以使用如下函式，从视窗代号中取得控制项的 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，在程式 11-2 所示的表头档案 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 讯息时将会很有用。在第一次尝试选中或者不选中单选按钮时，您可能会在对话方块程序中编写如下的程式：

```
static int iColor ;
其他行程式
case  WM_COMMAND:
      switch (LOWORD (wParam))
      {
      其他行程式
```

```
      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 ;
   其他行程式
```

这种方法能让人满意地执行。您将新的颜色值储存在 iColor 中，并且还建立了一个回圈，轮流使用所有八种颜色的 ID 值。您取得每个单选按钮控制项的视窗代号，并用 SendMessage 给每个代号发送一条 BM_SETCHECK 讯息。只有对於向对话方块视窗讯息处理程式发送 WM_COMMAND 讯息的按钮，这个讯息的 wParam 值才被设定为 1。

第一种简化的方法是使用专门的对话方块程序 SendDlgItemMessage：

```
SendDlgItemMessage (hDlg, id, iMsg, wParam, lParam) ;
```

它相同於：

```
SendMessage (GetDlgItem (hDlg, id), id, wParam, lParam) ;
```

现在，回圈将变成这样：

```
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 函式时才会出现：

```
CheckRadioButton (hDlg, idFirst, idLast, idCheck) ;
```

这个函式将 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回圈，并使用：

```
CheckRadioButton (hDlg, IDC_BLACK, IDC_WHITE, LOWORD (wParam)) ;
```

这正是 ABOUT2 对话方块程序所采用的方法。

在使用核取方块时，也提供了类似的简化函式。如果您建立了一个「CHECKBOX」对话方块视窗控制项，那么可以使用如下的函式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方块中某个核取方块的状态：

```
iCheck = IsDlgButtonChecked (hDlg, idCheckbox) ;
```

在对话方块程序中，您既可以将选中标记的目前状态储存在一个静态变数中，又可以在收到一个 WM_COMMAND 讯息後，使用如下方法触发按钮：

```
CheckDlgButton (hDlg, idCheckbox,
    !IsDlgButtonChecked (hDlg, idCheckbox)) ;
```

如果您定义了BS_AUTOCHECKBOX控制项,那么完全没有必要处理WM_COMMAND讯息。在终止对话方块之前，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，直到函式传回 TRUE。实际上，您还要拦截 WM_COMMAND 讯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 的对话方块模板中，「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
```

在对话方块中，通常都这样安排「OK」和「Cancel」按钮：将「OK」按钮作为内定按钮有助於用键盘介面终止对话。一般情况下，您通过单击两个滑鼠按键之一,或者当所期望的按钮具有输入焦点时按下 Spacebar 来终止对话方块。不过，如果使用者按下 Enter，对话方块视窗讯息处理程式也将产生一个 WM_COMMAND 讯息，而不管哪个控制项具有输入焦点。wParam 的低字组被设定为对话方块中内定按键的 ID 值，除非另一个按键拥有输入焦点。在後一种情况下，wParam 的低字组被设定为具有输入焦点之按键的 ID 值。如果对话方块中没有内定按键,那么 Windows 向对话方块程序发送一个 WM_COMMAND 讯息,讯息中 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 等於 IDCANCEL，并给对话方块程序发送一个 WM_COMMAND 讯息。所以，您不用在对话方块程序中加入单独的处理键盘操作，因为通常终止对话方块的按键会由 Windows 将这两个按键动作转换为 WM_COMMAND 讯息。

AboutDlgProc 函式通过呼叫 EndDialog 来处理这两种 WM_COMMAND 讯息：

```
switch (LWORD (wParam))
{
case IDOK:
    iCurrentColor  = iColor ;
    iCurrentFigure = iFigure ;
    EndDialog (hDlg, TRUE) ;
```

```
    return TRUE ;

case IDCANCEL :
    EndDialog (hDlg, FALSE) ;
    return TRUE ;
```

ABOUT2 的视窗讯息处理程式在程式的显示区域中绘制矩形或椭圆时，使用了整体变数 iCurrentColor 和 iCurrentFigure。AboutDlgProc 在对话方块中画图时使用了静态区域变数 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 函式的传回值传回的：

```
case  IDM_ABOUT:
      if (DialogBox (hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc))
                  InvalidateRect (hwnd, NULL, TRUE) ;
      return 0 ;
```

如果 DialogBox 传回 TRUE（非 0），则意味著按下了「OK」按钮，然後需要使用新的颜色来更新 WndProc 显示区域。当 AboutDlgProc 收到一个 WM_COMMAND 讯息并且讯息的 wParam 的低字组等於 IDOK 时，AboutDlgProc 将图形和颜色储存在整体变数 iCurrentColor 和 iCurrentFigure 中。如果 DialogBox 传回 FALSE，则主视窗继续使用 iCurrentColor 和 iCurrentFigure 的原始设定。

TRUE 和 FALSE 通常用於 EndDialog 呼叫中，以告知主视窗讯息处理程式使用者是用「OK」还是用「Cancel」来终止对话方块的。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 传回更多的资讯。

避免使用整体变数

在 ABOUT2 中使用整体变数可能会、也可能不会影响您。一些程式写作者（包括我自己）较喜欢少用整体变数。ABOUT2 中的整体变数 iCurrentColor 和 iCurrentFigure 看来使用得完全合法，因为它们必须同时在视窗讯息处理程式和对话方块程序中使用。不过，在一个有一大堆对话方块的程式中，每个对话方块都可能改变一堆变数的值，使整体变数的数量容易用得过多。

您可能更喜欢将程式中的对话方块与资料结构相联系，该资料结构含有对话方块可以改变的所有变数。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数：

```
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。呼叫对话方块时，使用 DialogBoxParam 而不用 DialogBox。此函式的第五个参数可以是任意的 32 位元值。一般来说，此值设定为指向一个结构的指标，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 的最後一个参数是作为 WM_INITDIALOG 讯息中的 lParam 传递给对话方块程序的。

对话方块程序有两个 ABOUTBOX_DATA 结构型态的静态变数（一个结构和一个指向结构的指标）：

```
static ABOUTBOX_DATA ad, * pad ;
```

在 AboutDlgProc 中，此定义代替了 iColor 和 iFigure 的定义。在 WM_INITDIALOG 讯息的开始部分，对话方块程序根据 lParam 设定了这两个变数的值：

```
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 中的区域结构的栏位对栏位内容复制。

现在，除了使用者按下「OK」按钮时所用的程式码以之外，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，我们利用视窗子类别化为 COLORS1 增加功能，使我们能够按下 Tab 键从一个卷动列转移到另一个卷动列。在对话方块中，视窗子类别化是不必要的，因为 Windows 完成了将输入焦点从一个控制项移动到另一个控制项的所有工作。尽管如此，您必须在对话方块模板中使用 WS_TABSTOP 和 WS_GROUP 视窗样式达到此目的。对於所有想要使用 Tab 键存取的控制项，都要在其视窗样

式中指定 WS_TABSTOP。

如果参阅表 11-1，您就会注意到许多控制项将 WS_TABSTOP 定义为内定样式，其他一些则没有将它作为内定样式。一般而言，不包含 WS_TABSTOP 样式的控制项（特别是静态控制项）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 讯息时您将输入焦点设定给一个特定的控制项，并从讯息中传回 FALSE。否则 Windows 将输入焦点设定为对话方块内第一个具有 WS_TABSTOP 样式的控制项。

Windows 给对话方块增加的第二个键盘介面包括游标移动键，这种介面对於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，就需要使用游标移动键，将输入焦点从该单选按钮移动到组内其他单选按钮上。使用 WS_GROUP 视窗样式即可获得这个功能。对於对话方块模板中的特定控制项序列，Windows 将使用游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 样式的控制项中。如果有必要，Windows 将从对话方块的最後一个控制项回圈到第一个控制项，以便找到分组的结尾。

在内定设定下，控制项 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 样式加到其他型态的控制项中。

让我们来看一看 ABOUT2.RC 中的对话方块模板。四个具有 WS_TABSTOP 样式的控制项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）。在第一次启动对话方块时，您可以使用 Tab 键在这四个控制项之间移动。

在每组单选按钮中，您可以使用游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。这意味著您可以用游标移动键将焦点从「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，所以您可以使用游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，以阻止游标移动键在按键具有输入焦点时起作用。

使用 ABOUT2 时，Windows 的对话方块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，处於单选按钮组内时，游标移动键切换输入焦点，并给对话方块程序发送 WM_COMMAND 讯息。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 切换到这一组後，Windows 将会把输入焦点设定为选中的单选按

钮。

文字栏位中的「&」将导致紧跟其後的字母以底线显示，这就增加了另一种键盘介面，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。

尽管程式写作者通常让对话方块管理器来完成这些工作，但是 Windows 提供了两个函式，以便程式写作者找寻下一个或者前一个 Tab 键停留项或者组项。这些函式为：

```
hwndCtrl = GetNextDlgTabItem (hDlg, hwndCtrl, bPrevious) ;
```

和

```
hwndCtrl = GetNextDlgGroupItem (hDlg, hwndCtrl, bPrevious) ;
```

如果 bPrevious 为 TRUE，那么函式传回前一个 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。

在对话方块上画图

ABOUT2 还完成了一些相对说来很特别的事情，亦即在对话方块上画图。让我们来看一看它是怎样做的。在 ABOUT2.RC 的对话方块模板内，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制项：

```
LTEXT  ""  IDC_PAINT, 114, 67, 72, 72
```

这个区域为 18 个字元宽和 9 个字元高。由於这个控制项没有文字，所以视窗讯息处理程式为「静态」类别所做的工作，只是在必须重绘这个子视窗控制项时清除其背景。

在目前颜色或图形选择发生改变，或者对话方块自身获得一个 WM_PAINT 讯息时，对话方块程序呼叫 PaintTheBlock，这个函式在 ABOUT2.C 中：

```
PaintTheBlock (hCtrlBlock, iColor, iFigure) ;
```

在 AboutDlgProc 中，视窗代号 hCtrlBlock 已经在处理 WM_INITDIALOG 讯息时被设定：

```
hCtrlBlock = GetDlgItem (hDlg, IDD_PAINT) ;
```

下面是 PaintTheBlock 函式：

```
void PaintTheBlock (HWND hCtrl, int iColor, int iFigure)
{
     InvalidateRect (hCtrl, NULL, TRUE) ;
     UpdateWindow (hCtrl) ;
     PaintWindow (hCtrl, iColor, iFigure) ;
}
```

这个函式使得子视窗控制项无效，并为控制项视窗讯息处理程式产生一个 WM_PAINT 讯息，然後呼叫 ABOUT2 中的另一个函式 PaintWindow 。

PaintWindow 函式取得一个装置内容代号，并将其放到 hCtrl 中，画出所选图形，根据所选颜色用一个著色画刷填入图形。子视窗控制项的大小从 GetClientRect 获得。尽管对话方块模板以字元为单位定义了控制项的大小，但 GetClientRect 取得以图素为单位的尺寸。您也可以使用函式 MapDialogRect 将对话方块中的字元座标转换为显示区域中的图素座标。

我们并非真的绘制了对话方块的显示区域，实际绘制的是子视窗控制项的显示区域。每当对话方块得到一个 WM_PAINT 讯息时，就令子视窗控制项的显示区域失效，并更新它，使它确信现在其显示区域又有效了，然後在其上画图。

将其他函式用於对话方块

大多数可以用在子视窗的函式也可以用於对话方块中的控制项。例如，如果您想捣乱的话，那么可以使用 MoveWindow 在对话方块内移动控制项，强迫使用者用滑鼠来追踪它们。

有时，您需要根据其他控制项的设定，动态地启用或者禁用某些控制项，这需要呼叫：

```
EnableWindow (hwndCtrl, bEnable) ;
```

当 bEnable 为 TRUE（非 0）时，它启用控制项；当 bEnable 为 FALSE（0）时，它禁用控制项。在控制项被禁用时，它不再接收键盘或者滑鼠输入。您不能禁用一个拥有输入焦点的控制项。

定义自己的控制项

尽管 Windows 承揽了许多维护对话方块和子视窗控制项的工作，它同时也为您提供了各种加入程式码的方法。前面我们已经看到了在对话方块上绘图的方法。您也可以使用第九章中讨论的视窗子类别化来改变子视窗控制项的操作。

您还可以定义自己的子视窗控制项，并将它们用到对话方块中。例如，假定您特别不喜欢普通的矩形按键，而倾向於建立椭圆形按键，那么您可以通过注册一个视窗类别，并使用自己编写的视窗讯息处理程式处理来自您所建立视窗的讯息，从而建立椭圆形按键。在 Developer Studio 中，您可以在与自订控制项相联系的「Properties」对话方块中指定这个视窗类别，这将转换成对话方块模板中的 CONTROL 叙述。程式 11-3 所示的 ABOUT3 程式正是这样做的。

程式 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) ;
}
```

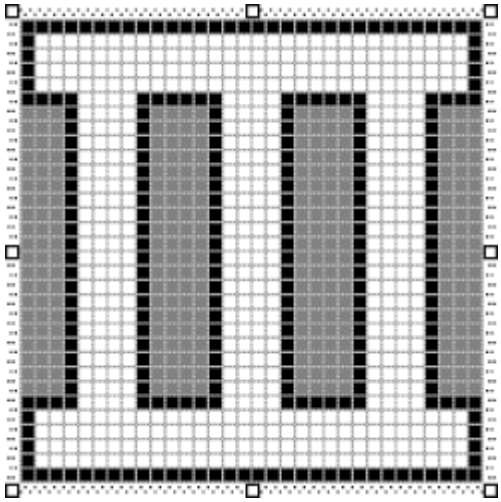
ABOUT3.RC （摘录）

```
//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
```

ABOUT3.ICO



我们所注册的视窗类别叫做「EllipPush」（椭圆形按键）。在 Developer Studio 的对话方块编辑器中，删除「Cancel」和「OK」按钮。要添加依据此视窗类别的控制项，请从「 Controls 」工具列选择「 Custom Control 」。在此控制项的「 Properties 」对话方块的「 Class 」栏位输入「 EllipPush 」。在对话方块模板中我们没有使用 DEFPUSHBUTTON 叙述，而是用 CONTROL 叙述来指定此视窗类别：

```
CONTROL "OK" IDOK, "EllipPush", TABGRP, 64, 60, 32, 14
```

当在对话方块中建立子视窗控制项时，对话方块管理器把这个视窗类别用於 CreateWindow 呼叫中。

ABOUT3.C 程式在 WinMain 中注册了 EllipPush 视窗类别：

```
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) ;
```

该视窗类别指定视窗讯息处理程式为 EllipPushWndProc，在 ABOUT3.C 中正是这样。

EllipPushWndProc 视窗讯息处理程式只处理三种讯息：WM_PAINT、WM_KEYUP 和 WM_LBUTTONUP。在处理 WM_PAINT 讯息时，它从 GetClientRect 中取得视窗的大小，从 GetWindowText 中取得显示在按键上的文字，用 Windows 函式 Ellipse 和 DrawText 来输出椭圆和文字。

WM_KEYUP 和 WM_LBUTTONUP 讯息的处理非常简单：

```
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 ;
```

视窗讯息处理程式使用 GetParent 来取得其父视窗（即对话方块）的代号，并发送一个 WM_COMMAND 讯息，讯息的 wParam 等於控制项的 ID，这个 ID 是用 GetWindowLong 取得的。然後，对话方块视窗讯息处理程式将这个讯息传给 ABOUT3 内的对话方块程序，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他自订对话方块控制项。

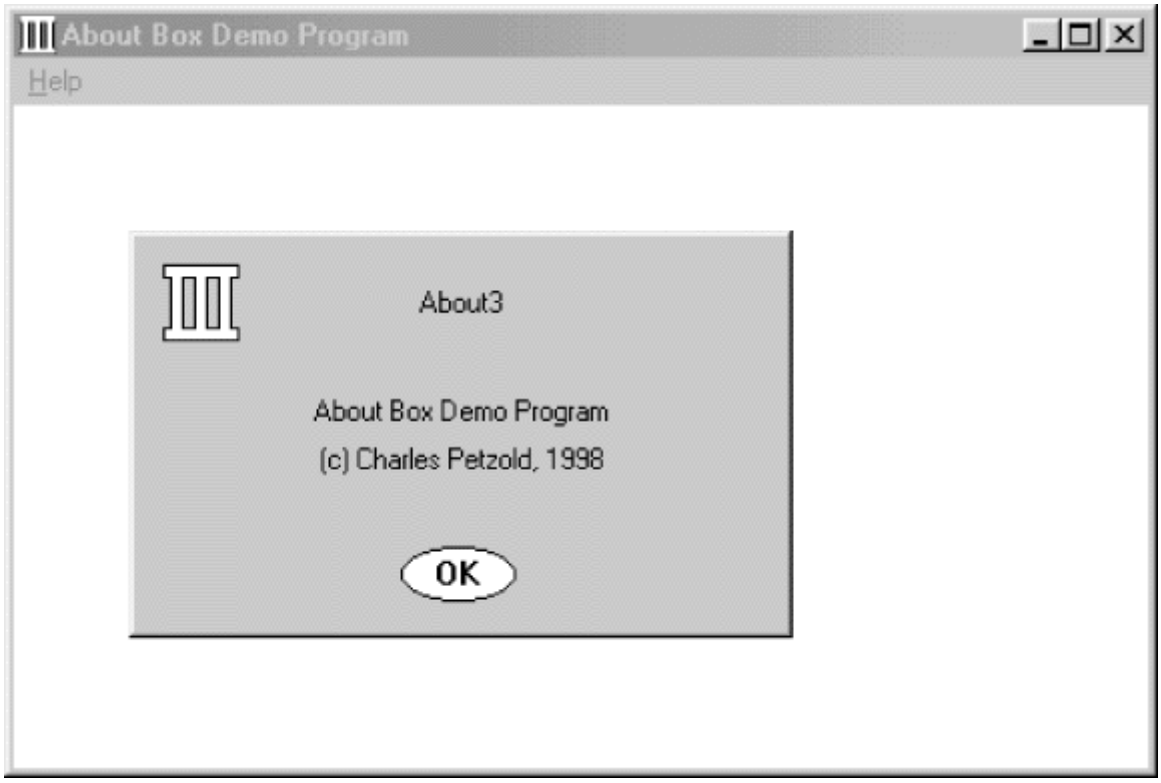


图 11-3 ABOUT3 建立的自订按键

这就是全部要做的吗？其实不然。通常，对於维护子视窗控制项所需要的处理而言，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•转按键内的颜色，视窗讯息处理程式必须处理 WM_KEYDOWN（来自空白键）和 WM_LBUTTONDOWN 讯息。视窗讯息处理程式还必须在收到

WM_LBUTTONDOWN 讯息时拦截滑鼠，并且，如果当按钮还处於按下状态，而滑鼠移到了子视窗的显示区域之外，那么得要释放滑鼠拦截（并将按钮的内部颜色回复为正常状态）。只有在滑鼠被拦截时松开该按钮，子视窗才会给其父视窗送回一个 WM_COMMAND 讯息。

EllipPushWndProc 也不处理 WM_ENABLE 讯息。如上所述，对话方块程序可以使用 EnableWindow 函式来禁用某视窗。於是，子视窗将显示灰色文字，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讯息了。

如果子视窗控制项的视窗讯息处理程式需要为所建立的每个视窗存放各自不同的资料，那么它可以通过使用视窗类别结构中的 cbWndExtra 值来做到。这样就在内部视窗结构中保留了空间，并可以用 SetWindowLong 和 GetWindowLong 来存取该资料。

非模态对话方块

在本章的开始，我曾经说过对话方块分为「模态的」和「非模态的」两种。现在我们已经研究过这两种对话方块中最常见的一种——模态对话方块。模态对话方块（不包括系统模态对话方块）。允许使用者在对话方块与其他程式之间进行切换。但是，使用者不能切换到同一程式的另一个视窗，直到模态对话方块被清除为止。非模态对话方块允许使用者在对话方块与其他程式之间进行切换，又可以在对话方块与建立对话方块的视窗之间进行切换。因此，非模态对话方块与使用者程式常见的普通弹出式视窗可能更为相似。

当使用者觉得让对话方块保留片刻会更加方便时，使用非模态对话方块是合适的。例如，文书处理程式经常使用非模态对话方块来进行「Find」和「Change」操作。如果「Find」对话方块是模态的，那么使用者必须从功能表中选择「Find」，然後输入要寻找的字串，结束对话方块，传回到档案中，接著再重复整个程序来寻找同一字串的另一次出现。允许使用者在档案与对话方块之间进行切换则会方便得多。

您已经看到，模态对话方块是用 DialogBox 来建立的。只有在清除对话方块之後，函式才会传回值。在对话方块程序内使用 EndDialog 呼叫来终止对话方块，DialogBox 传回的是该呼叫的第二个参数的值。非模态对话方块是使用 CreateDialog 来建立的，该函式所使用的参数与 DialogBox 相同。

```
hDlgModeless = CreateDialog (     hInstance, szTemplate,
                              hwndParent, DialogProc) ;
```

区别是 CreateDialog 函式立即传回对话方块的视窗代号，并通常将这个视窗代号存放到整体变数中。

尽管将 DialogBox 这一名字用於模态对话方块而 CreateDialog 用於非模态

对话方块是随意的，但是您可以通过非模态对话方块与普通视窗类似这一点来记住这两个函式的区别。CreateDialog 可以令人想起 CreateWindow 函式来，而後者建立的是普通视窗。

模态对话方块与非模态对话方块的区别

使用非模态对话方块与使用模态对话方块相似，但是也有一些重要的区别：

首先，非模态对话方块通常包含一个标题列和一个系统功能表按钮。当您在 Developer Studio 中建立对话方块时，这些是内定选项。用於非模态对话方块的对话方块模板中的 STYLE 叙述形如：

STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | WS_VISIBLE

标题列和系统功能表允许使用者，使用滑鼠或者键盘将非模态对话方块移动到另一个显示区域。对於模态对话方块，您通常无须提供标题列和系统功能表，因为使用者不能在其下面的视窗中做任何其他的事情。

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 」对话方块的「 More Styles 」页面标签中选择此选项。如果省略了 WS_VISIBLE，那么您必须在 CreateDialog 呼叫之後呼叫 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，那么非模态对话方块将不会被显示。如果忽略这个事实，那么习惯於模态对话方块的程式写作者在第一次试图建立非模态对话方块时，经常会出现问题。

第三项区别：与模态对话方块和讯息方块的讯息不同，非模态对话方块的讯息要经过程序式的讯息伫列。要将这些讯息传送给对话方块视窗讯息处理程式，则必须改变讯息伫列。方法如下：当您使用 CreateDialog 建立非模态对话方块时，应该将从呼叫中传回的对话方块代号储存在一个整体变数（如 hDlgModeless）中，并将讯息回圈改变为：

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

如果讯息是发送给非模态对话方块的，那么 IsDialogMessage 将它发送给对话方块中视窗讯息处理程式，并传回 TRUE（非 0）；否则，它将传回 FALSE

（0）。只有 hDlgModeless 为 0 或者讯息不是该对话方块的讯息时，才必须呼叫 TranslateMessage 和 DispatchMessage 函式。如果您将键盘加速键用於您的程式视窗，那么讯息回圈将如下所示：

```
while (GetMessage (&msg, NULL, 0, 0))
{
      if (hDlgModeless == 0 || !IsDialogMessage (hDlgModeless, &msg))
      {
                  if (!TranslateAccelerator (hwnd, hAccel, &msg))
                  {
                                    TranslateMessage (&msg) ;
                                    DispatchMessage  (&msg) ;
                  }
      }
}
```

由於整体变数被初始化为 0，所以 hDlgModeless 将为 0，直到建立对话方块为止，从而保证不会使用无效的视窗代号来呼叫 IsDialogMessage。在清除非模态对话方块时，您也必须注意这一点，正如最後一点所说明的。

hDlgModeless 变数也可以由程式的其他部分使用，以便对非模态对话方块是否存在加以验证。例如，程式中的其他视窗可以在 hDlgModeless 不等於 0 时给对话方块发送讯息。

最後一项重要的区别：使用 DestroyWindow 而不是 EndDialog 来结束非模态对话方块。当您呼叫 DestroyWindow 後，将 hDlgModeless 整体变数设定为 0。

使用者习惯於从系统功能表中选择「Close」来结束非模态对话方块。尽管启用了「Close」选项，Windows 内的对话方块视窗讯息处理程式并不处理 WM_CLOSE 讯息。您必须自己在对话方块程序中处理它：

```
case  WM_CLOSE :
      DestroyWindow (hDlg) ;
      hDlgModeless = NULL ;
      break ;
```

注意这两个视窗代号之间的区别：DestroyWindow 的 hDlg 参数是传递给对话方块程序的参数；hDlgModeless 是从 CreateDialog 传回的整体变数，程式在讯息回圈内检验它。

您也可以允许使用者使用按键来关闭非模态对话方块，处理方式与处理 WM_CLOSE 讯息一样。对话方块必须传回给建立它的视窗之任何资料都可以储存在整体变数中。如果不喜欢使用整体变数，那么您也可以用 CreateDialogParam 来建立非模态对话方块，并按前面介绍的方法让它储存一个结构指标。

新的 COLORS 程式

第九章中所描述的 COLORS1 程式建立了九个子视窗，以便显示三个卷动列和六个文字项。那时候，这个程式还是我们所写过的程式中相当复杂的一个。如果将 COLORS1 转换为使用非模态对话方块则会使程式——特别是 WndProc 函式——变得令人难以置信的简单，修正後的 COLORS2 程式如程式 11-4 所示。

程式 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 程式所显示的卷动列大小是依据视窗大小决定的，而新程式在非模态对话方块内以固定的尺寸来显示它们，如图 11-4 所示。

当您建立对话方块模板时，直接将三个卷动列的 ID 分别设为 10、11 和 12，将显示卷动列目前值的三个静态文字栏位的 ID 分别设为 13、14 和 15。将每个卷动列都设定为 Tab Stop 样式，而从所有的六个静态文字栏位中删除 Group 样式。

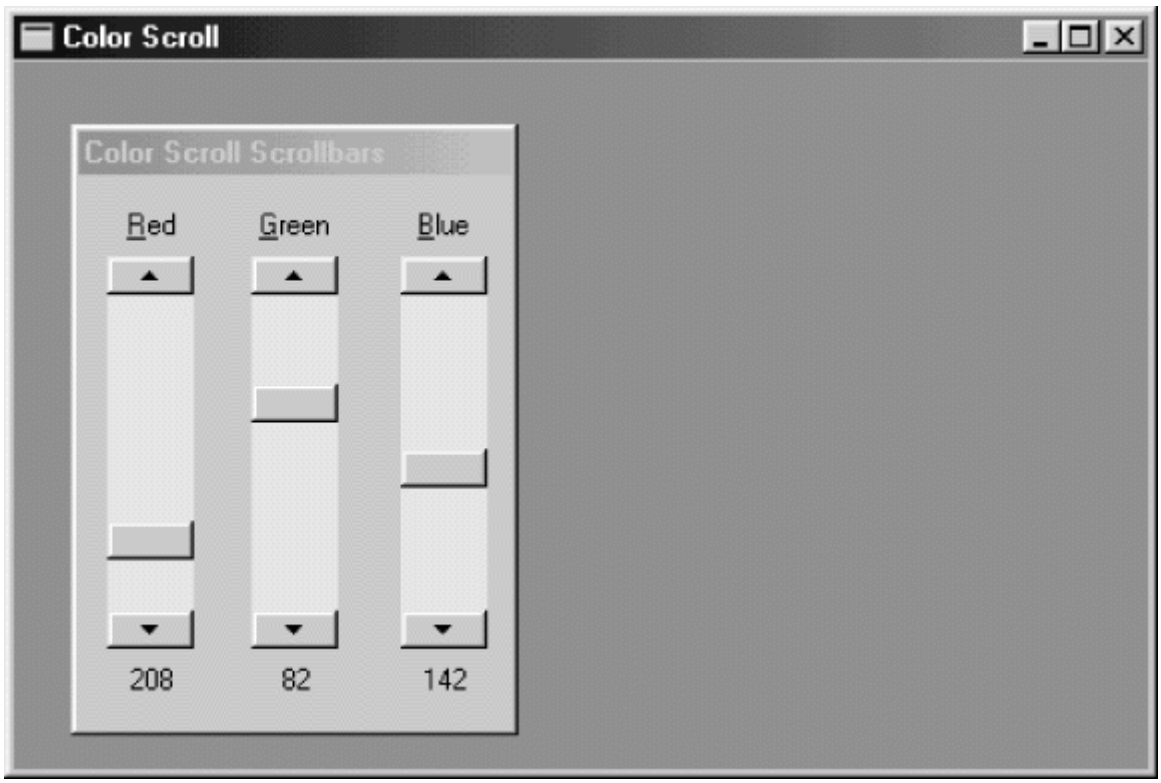


图 11-4 COLORS2 的萤幕显示

在 COLORS2 中，非模态对话方块是在 WinMain 函式里建立的，紧跟在程式主视窗的 ShowWindow 呼叫之後。注意，主视窗的视窗样式包含 WS_CLIPCHILDREN，这允许程式无须擦除对话方块就能够重画主视窗。

如上所述，从 CreateDialog 传回的对话方块视窗代号存放在整体变数 hDlgModeless 中，并在讯息回圈中被测试。不过，在这个程式中，不需要将代号存放在整体变数中，也不需要在呼叫 IsDialogMessage 之前测试这个值。讯息回圈可以编写如下：

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

由於对话方块是在程式进入讯息回圈前建立，并且直到程式结束时才会被清除，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方块的视窗讯息处理程式中加入一段清除对话方块的程式码：

```
case  WM_CLOSE :
      DestroyWindow (hDlg) ;
      hDlgModeless = NULL ;
      break ;
```

在原来的 COLORS1 程式中，SetWindowText 在使用 wsprintf 将三个数值标

签转换为文字之後才设定它们的值。叙述为：

```
wsprintf (szBuffer, TEXT ("%i"), color[i]) ;
SetWindowText (hwndValue[i], szBuffer) ;
```

i 的值为目前处理的卷动列的 ID，hwndValue 是一个阵列，它包含颜色数值的三个静态文字子视窗的视窗代号。

新版本使用 SetDlgItemInt 为每个子视窗的每个文字栏位设定一个号码：

```
SetDlgItemInt (hDlg, iCtrlID + 3, color [iCtrlID], FALSE) ;
```

尽管 SetDlgItemInt 和与其对应的 GetDlgItemInt 在编辑控制项中用得最多，它们也可以用来设定其他控制项的文字栏位，如静态文字控制项等。iCtrlID 变数是卷动列的 ID，给 ID 加上 3 使之变成对应数字标签的 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，对於这个程式，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，请看下一个程式。

HEXCALC: 视窗还是对话方块?

HEXCALC 程式可能是写程式偷懒的经典之作，如程式 11-5 所示。这个程式完全不呼叫 CreateWindow，也不处理 WM_PAINT 讯息，不取得装置内容，也不处理滑鼠讯息。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滑鼠介面以及 10 种运算的十六进位计算机。计算机如图 11-5 所示。

程式 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

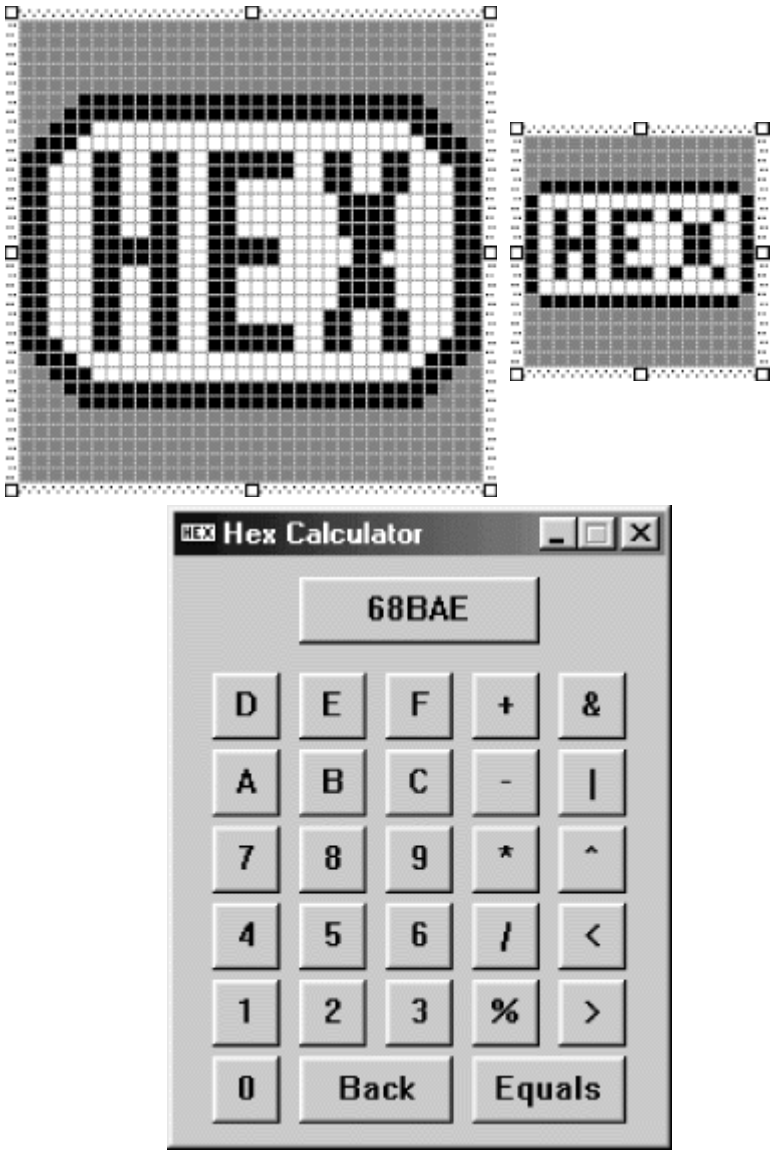


图 11-5 HEXCALC 的萤幕显示

HEXCALC 是一个普通的中序运算式计算机，使用 C 语言的符号表示方式进行计算。它对无正负号 32 位元整数作加、减、乘、除和取余数运算，位元 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 中既可以使用滑鼠又可以使用键盘。您从按键点入」或者输入第一个数（最多 8 位元十六进位数位）开始，然後输入运算子，然後是第二个数。接著，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，萤幕上显示的视窗似乎是普通的重叠式视窗与非模态对话方块的混合体。一方面，HEXCALC 的所有讯息都在函式的 WndProc 中处理，这个函式与通常的视窗讯息处理程式相似，该函式传回一个长整数，它处理 WM_DESTROY 讯息，呼叫 DefWindowProc，就像普通的视窗讯息处理程式一样。另一方面，视窗是在 WinMain 中呼叫 CreateDialog 并使用 HEXCALC.DLG 中

的对话方块模板建立的。那么，HEXCALC 到底是一个普通的可重叠视窗，还是一个非模态对话方块呢？

简单的回答是，对话方块就是视窗。通常，Windows 使用它自己内部的视窗讯息处理程式处理对话方块视窗的讯息，然後，Windows 将这些讯息传送给建立对话方块的程式内的对话方块程序。在 HEXCALC 中，我们让 Windows 使用对话方块模板建立一个视窗，但是自己写程式处理这个视窗的讯息。

不幸的是，在 Developer Studio 的 Dialog Editor 中，对话方块模板需要一些我们不能添加的东西。因此，对话方块模板包含在 HEXCALC.DLG 档案中，而且需要手工输入。依照下面的方法，您可以将一个文字档案添加到任何专案中：从「 File 」功能表选择「 New 」，再选择「 Files 」页面标签，然後从档案型态列表中选择「 Text File 」。像这样的档案——包含附加资源定义——需要包含在资源描述中。从「 View 」功能表选择「 Resource Includes 」。这显示一个对话方块。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 档案中的对话方块模板，您将发现 HEXCALC 如何为对话方块使用它自己的视窗讯息处理程式。对话方块模板的上方如下：

```
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 等识别字，我们可以将它们用在 CreateWindow 呼叫中以建立普通的视窗。CLASS 叙述是这个对话方块与曾经建立过的对话方块之间最重要的区别（而且它也是 Developer Studio 中的 Dialog Editor 不允许我们指定的）。当对话方块模板省略了这个叙述时，Windows 为对话方块注册一个视窗类别，并使用它自己的视窗讯息处理程式处理对话方块讯息。这里，包含 CLASS 叙述就告诉 Windows 将讯息发送到其他的地方——具体的说，就是发送到在 HexCalc 视窗类别中指定的视窗讯息处理程式。

HexCalc 视窗类别是在 HEXCALC 的 WinMain 函式中注册的，就像普通视窗的视窗类别一样。但是，请注意有个十分重要的区别：WNDCLASS 结构的 cbWndExtra 栏位设定为 DLGWINDOWEXTRA。对於您自己注册的对话方块程序，这是必需的。

在注册视窗类别之後，WinMain 呼叫 CreateDialog：

```
hwnd = CreateDialog (hInstance, szAppName, 0, NULL) ;
```

第二个参数（字串「HexCaEc」）是对话方块模板的名字。第三个参数通常是父视窗的视窗代号，这里设定为 0，因为视窗没有父视窗。最後一个参数，通常是对话方块程序的位址，这里不需要。因为 Windows 不会处理这些讯息，因

而也不会将讯息发送给对话方块程序。

这个 CreateDialog 呼叫与对话方块模板一起，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 变数分别是系统字体字元的宽度和高度。

我们通过让 Windows 来进行 CreateWindow 呼叫而收获甚丰：Windows 不会在建立弹出式视窗 1 後就停止，它还会为对话方块模板中定义的其他 29 个子视窗按键控制项呼叫 CreateWindow。所有这些控制项都给父视窗的视窗讯息处理程式发送 WM_COMMAND 讯息，该程序正是 WndProc。对於建立一个包含许多子视窗的视窗来说，这是一个很好的技巧。

下面是使 HEXCALC 的程式码量下降到最少的另一种方法：或许您会注意到 HEXCALC 没有表头档案，表头档案中通常包含对话方块模板中，需要为所有子视窗控制项定义的识别字。我们之所以可以不要这个档案，是因为每个按键控制项的 ID 设定为出现在控制项上的文字的 ASCII 码。这意味著，WndProc 可以完全相同地对待 WM_COMMAND 讯息和 WM_CHAR 讯息。在每种情况下，wParam 的低字组都是按钮的 ASCII 码。

当然，对键盘讯息进行一些处理是必要的。WndProc 拦截 WM_KEYDOWN 讯息，将左箭头键转换为 Backspace 键。在处理 WM_CHAR 讯息时，WndProc 将字元代码转换为大写，Enter 键转换为等号键的 ASCII 码。

WM_CHAR 讯息的有效性是通过呼叫 GetDlgItem 来检验的。如果 GetDlgItem 函式传回 0，那么键盘字元不是对话方块模板中定义的 ID 之一。如果字元是 ID 之一，则通过给相应的按钮发送一对 BM_SETSTATE 讯息，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 的键盘介面增色不少。Sleep 函式将程式暂停 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 讯息时，它总是将输入焦点设定给父视窗：

```
case  WM_COMMAND :
     SetFocus (hwnd) ;
```

否则，一旦使用滑鼠单击某按钮，输入焦点就会切换到该按钮上。

通用对话方块

Windows 的一个主要目的是推动标准的使用者介面。对许多常用的功能表项来说，这推行得很快，几乎所有软体厂商都采用 Alt-File-Open 选择来打开一个档案。然而，实际的档案开启对话方块却经常各不相同。

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「通用对话方块程式库」的增强。这个程式库由几个函式组成，这些函式启动标准对话方块来进行打开和储存档案、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）以及列印（我将在第十三章讨论）。

为了使用这些函式，您基本上都要初始化某一结构的各个栏位，并将该结构的指标传送给通用对话方块程式库的某个函式，该函式会建立并显示对话方块。当使用者关闭对话方块时，被呼叫的函式将控制权传回给程式，您可以从传送给它的结构中获得资讯。

在使用通用对话方块程式库的任何 C 原始码档案时，您都需要含入 COMMDLG.H 表头档案。通用对话方块的文件在/Platform SDK/User Interface Services/User Input/Common Dialog Box Library 中。

增强 POPPAD

当我们往第十章的 POPPAD 中增加功能表时，还有几个标准功能表项没有实作。现在我们已经准备好在 POPPAD 中加入打开档案、读入档案以及在磁片上储存编辑过档案的功能。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 程式的档案如程式 11-6 所示。

程式 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 ;
}
```

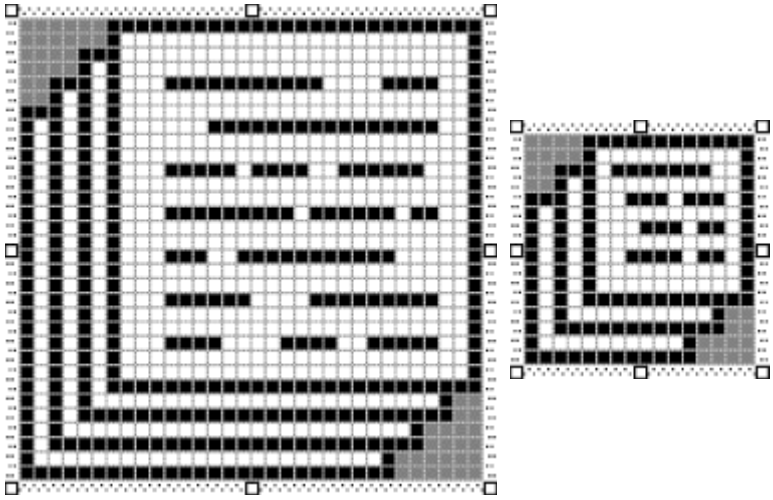
POPPAD.RC （摘录）

```
//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 的功能表中加入了列印专案和一些其他的支援。

POPPAD.C 包含了程式中所有的基本原始码。POPFILE.C 具有启动 File Open 和 File Save 对话方块的程式码，它还包含档案 I/O 常式。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。POPPRNTO.C 不完成什么工作：在第十三章中将使用 POPPRNT.C 替换 POPPRNTO.C 以建立最终的 POPPAD 程式。

让我们先来看一看 POPPAD.C。POPPAD.C 含有两个档案名字串：第一个，储存在 WndProc，名称为 szFileName，含有详细的驱动器名称、路径名称和档案名称；第二个，储存为 szTitleName，是程式本身的档案名称。它用在 POPPAD3 的DoCaption函式中,以便将档案名称显示在视窗的标题列上;也用在OKMessage 函式和 AskAboutSave 函式中，以便向使用者显示讯息方块。

POPFILE.C 包含了几个显示「File Open」和「File Save」对话方块以及实际执行档案 I/O 的函式。对话方块是使用函式 GetOpenFileName 和 GetSaveFileName 来显示的。这两个函式都使用一个型态为 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数，取名为 ofn。ofn 的大多数栏位在 PopFileInitialize 函式中被初始化，POPPAD.C 在 WndProc 中处理 WM_CREATE 讯息时呼叫该函式。

将 ofn 作为静态整体结构变数会比较方便，因为 GetOpenFileName 和 GetSaveFileName 给该结构传回的一些资讯，并将在以後呼叫这些函式时用到。

尽管通用对话方块具有许多选项——包括设定自己的对话方块模板，以及为对话方块程序增加「挂勾（hook）」——POPFILE.C 中使用的「File Open」和「File Save」对话方块是最基本的。OPENFILENAME 结构中被设定的栏位只有 lStructSize（结构的长度）、hwndOwner（对话方块拥有者）、lpstrFilter（下面将简要讨论）、lpstrFile 和 nMaxFile（指向接收完整档案名称的缓冲区指标和该缓冲区的大小）、lpstrFileTitle 和 nMaxFileTitle（档案名称缓冲区及其大小）、Flags（设定对话方块的选项）和 lpstrDefExt（如果使用者在对话方块中输入档案名时不指定档案副档名，那么它就是内定的档案副档名）。

当使用者在「File」功能表中选择「Open」时，POPPAD3 呼叫 POPFILE 的 PopFileOpenDlg 函式，将视窗代号、一个指向档案名称缓冲区的指标和一个指向档案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 栏位，将 Flags 设定为 OFN_CREATEPROMPT，然後呼叫 GetOpenFileName，显示如图 11-6 所示的普通对话方块。

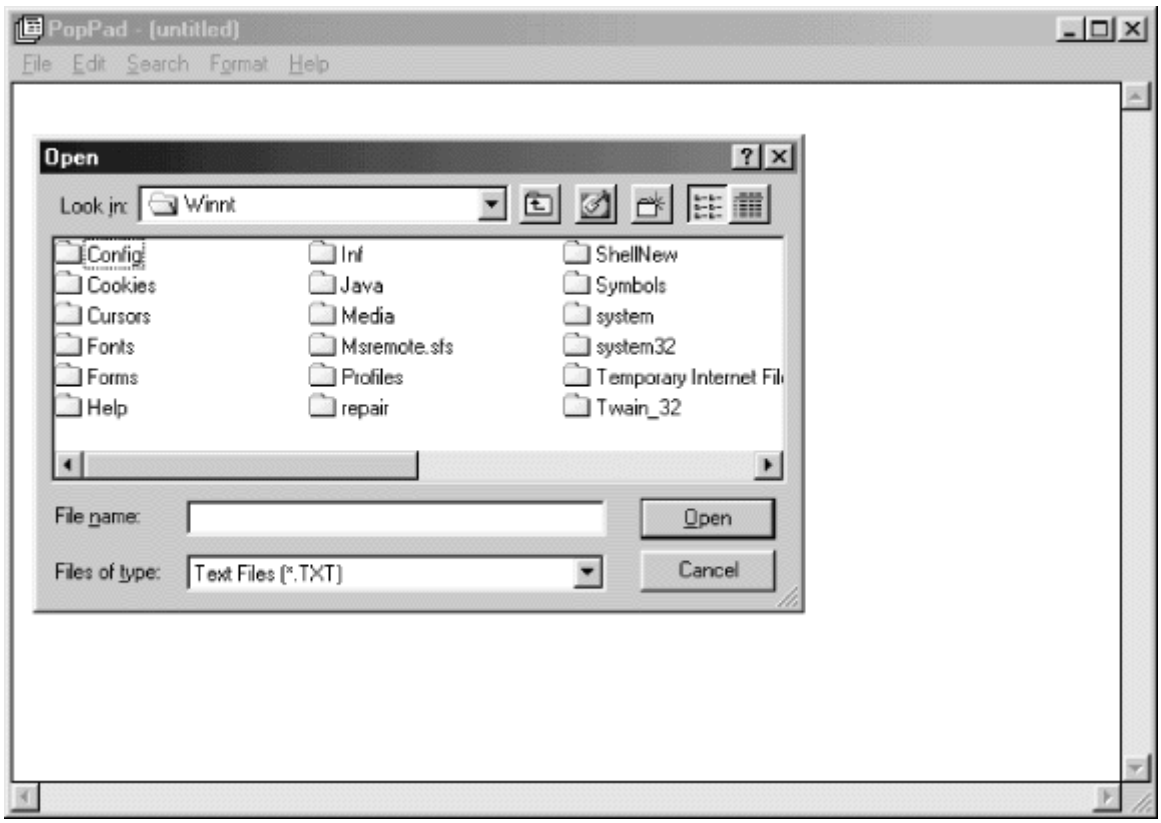


图 11-6 「File Open」对话方块

当使用者结束这个对话方块时，GetOpenFileName 函式传回。OFN_CREATEPROMPT 旗标指示 GetOpenFileName 显示一个讯息方块，询问使用者如果所选档案不存在，是否要建立该档案。

左下角的下拉式清单方块列出了将要显示在档案列表中的档案型态，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档案型态，来改变筛选条件。在 POPFILE.C 的 PopFileInitialize 函式中，我在变数 szFilter（一个字串阵列）中为三种型态的档案定义了一个筛检清单：带有.TXT 副档名的文字档案、带有.ASC 副档名的 ASCII 档案和所有档案。OPENFILENAME 结构的 lpstrFilter 栏位储存指向此阵列第一个字串的指标。

如果使用者在对话方块处於活动状态时改变了筛选条件，那么 OPENFILENAME 的 nFilterIndex 栏位反映出使用者的选择。由於该结构是静态变数，下次启动对话方块时，筛选条件将被设定为选中的档案型态。

POPFILE.C 中的 PopFileSaveDlg 函式与此类似，它将 Flags 参数设定为 OFN_OVERWRITEPROMPT，并呼叫 GetSaveFileName 启动「File Save」对话方块。OFN_OVERWRITEPROMPT 旗标导致显示一个讯息方块，如果被选档案已经存在，那么将询问使用者是否覆盖该档案。

Unicode 档案 I/O

对於本书中的大多数程式，您都不必注意 Unicode 和非 Unicode 版的区别。

例如，在 POPPAD3 的 Unicode 中，编辑控制项将保留 Unicode 文字和使用 Unicode 字串的所有通用对话方块。例如，当程式需要搜索和替换时，所有的操作都会处理 Unicode 字串，而不需要转换。

不过，POPPAD3 得处理档案 I/O，也就是说，程式不能闭门造车。如果 Unicode 版的 POPPAD3 获得了编辑缓冲区的内容并将其写入磁片，档案将是使用 Unicode 存放的。如果非 Unicode 版的 POPPAD3 读取了该档案，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 版储存的档案时也会这样。

解决的办法在於辨别和转换。首先，在 POPFILE.C 的 PopFileWrite 函式中，您将看到 Unicode 版的程式将在档案的开始位置写入 0xFEFF。这定义为位元组顺序标记，以表示文字档案含有 Unicode 文字。

其次，在 PopFileRead 函式中，程式用 IsTextUnicode 函式来决定档案是否含有位元组顺序标记。此函式甚至检测位元组顺序标记是否反向了，亦即 Unicode 文字档案在 Macintosh 或者其他使用与 Intel 处理器相反的位元组顺序的机器上建立的。这时，位元组的顺序都经过・转。如果档案是 Unicode 版，但是被非 Unicode 版的 POPPAD3 读取，这时，文字将被 WideCharToMultiChar 转换。WideCharToMultiChar 实际上是一个宽字元 ANSI 函式（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，如果档案是非 Unicode 文字档案，而执行的是 Unicode 版的程式，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方块函式来选择字体。

在 WM_CREATE 讯息处理期间，POPFONT.C 中的 POPPAD 呼叫 PopFontInitialize。这个函式取得一个依据系统字体建立的 LOGFONT 结构，由此建立一种字体，并向编辑控制项发送一个 WM_SETFONT 讯息来设定一种新的字体（内定编辑控制项字体是系统字体，而 PopFontInitialize 为编辑控制项建立一种新的字体，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 收到来自程式的字体选项的 WM_COMMAND 讯息时，它呼叫 PopFontChooseFont。这个函式初始化一个 CHOOSEFONT 结构，然後呼叫 ChooseFont 显示字体选择对话方块。如果使用者按下「OK」按钮，那么 ChooseFont 将传回 TRUE。随後，POPPAD 呼叫 PopFontSetFont 来设定编辑控制项中的新字体，旧字体将被删除。

最後，在 WM_DESTROY 讯息处理期间，POPPAD 呼叫 PopFontDeinitialize 来

删除最近一次由 PopFontSetFont 建立的字体。

搜寻与替换

通用对话方块程式库也提供两个用於文字搜寻和替换函式的对话方块，这两个函式（FindText 和 ReplaceText）使用一个型态为 FINDREPLACE 的结构。图 10-11 中所示的 POPFIND.C 档案有两个常式（PopFindFindDlg 和 PopFindReplaceDlg）呼叫这些函式，还有两个函式在编辑控制项中搜寻和替换文字。

使用搜寻和替换函式有一些考虑。首先，它们启动的对话方块是非模态对话方块，这意味著必须改写讯息回圈，以便在对话方块活动时呼叫 IsDialogMessage。第二，传送给 FindText 和 ReplaceText 的 FINDREPLACE 结构必须是一个静态变数，因为对话方块是模态的，函式在对话方块显示之後传回，而不是在对话方块结束之後传回；而对话方块程序必须仍然能够存取该结构。

第三，在显示 FindText 和 ReplaceText 对话方块时，它们通过一条特殊讯息与拥有者视窗联络，讯息编号可以通过以 FINDMSGSTRING 为参数呼叫 RegisterWindowMessage 函式来获得。这是在 WndProc 中处理 WM_CREATE 讯息时完成的，讯息号存放在静态变数中。

在处理内定讯息时，WndProc 将讯息变数与 RegisterWindowMessage 传回的值相比较。lParam 讯息参数是一个指向 FINDREPLACE 结构的指标，Flags 栏位指示使用者使用对话方块是为了搜寻文字还是替换文字，以及是否要终止对话方块。POPPAD3 是呼叫 POPFIND.C 中的 PopFindFindText 和 PopFindReplaceText 函式来执行搜寻和替换功能的。

只呼叫一个函式的 Windows 程式

到现在为止，我们已经说明了两个程式，让您浏览选择颜色，这两个程式分别是第九章中的 COLORS1 和本章中的 COLORS2。现在是讲解 COLORS3 的时候了，这个程式只有一个 Windows 函式呼叫。COLORS3 的原始码如程式 11-7 所示。

COLORS3 所呼叫的唯一 Windows 函式是 ChooseColor，这也是通用对话方块程式库中的函式，它显示如图 11-7 所示的对话方块。颜色选择类似於 COLORS1 和 COLORS2，但是它与使用者交谈互动能力更强。

程式 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) ;
}
```

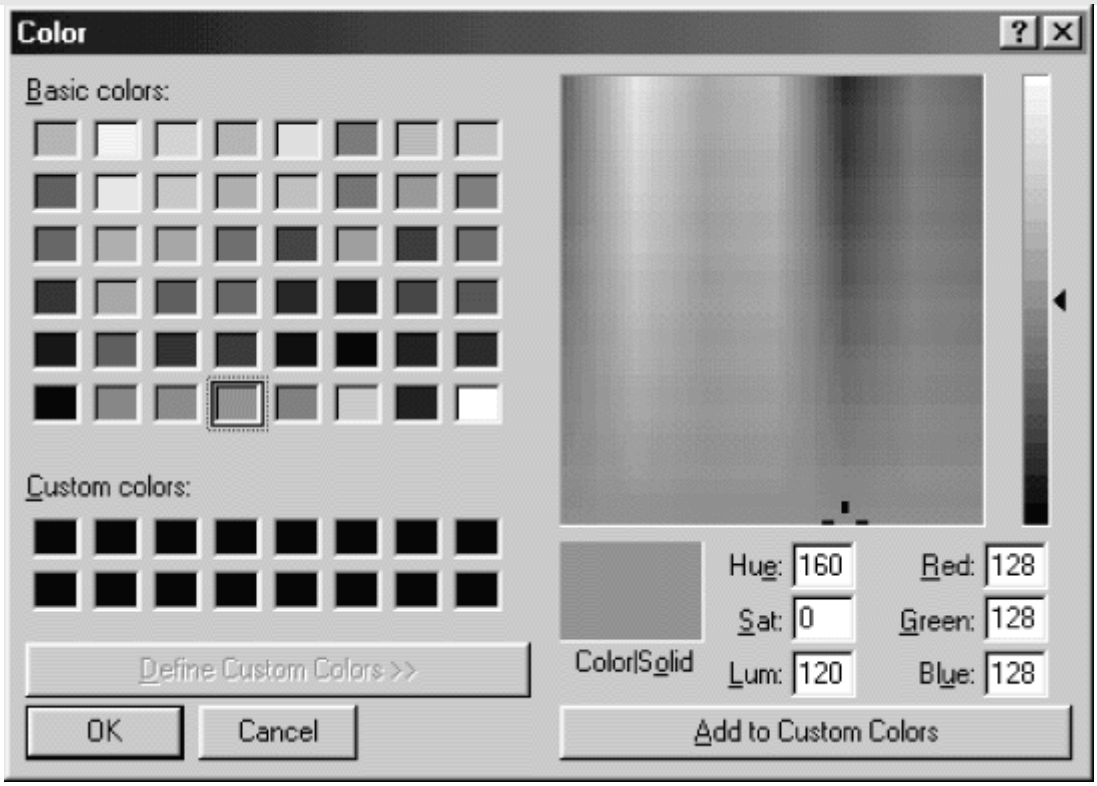


图 11-7 COLORS3 的萤幕显示

ChooseColor 函式使用一个 CHOOSECOLOR 型态的结构和含有 16 个 DWORD 的阵列来存放常用颜色，使用者将从对话方块中选择这些颜色之一。rgbResult 栏

位可以初始化为一个颜色值，如果 Flags 栏位的 CC_RGBINIT 旗标被设立，则显示该颜色。通常在使用这个函式时，rgbResult 将被设定为使用者选择的颜色。

请注意，Color 对话方块的 hwndOwner 栏位被设定为 NULL。在 ChooseColor 函式呼叫 DialogBox 以显示对话方块时，DialogBox 的第三个参数也被设定为 NULL。这是完全合法的，其含义是对话方块不为另一个视窗所拥有。对话方块的标题将显示在工作列中，而对话方块就像一个普通的视窗那样执行。

您也可以在自己程式的对话方块中使用这种技巧。使 Windows 程式只建立对话方块，其他事情都在对话方块程序中完成，这是可能的。