第十三章　使用印表机

为了处理文字和图形而使用视讯显示器时，装置无关的概念看来非常完美，但对於印表机，装置无关的概念又怎样呢？

总的说来，效果也很好。在 Windows 程式中，用於视讯显示器的 GDI 函式一样可以在印表纸上列印文字和图形，在以前讨论的与装置无关的许多问题（多数都与平面显示的尺寸、解析度以及颜色数有关）都可以用相同的方法解决。当然，一台印表机不像使用阴极射线管的显示器那么简单，它们使用的是印表纸。它们之间有一些比较大的差异。例如，我们从来不必考虑视讯显示器没有与显示卡连结好，或者显示器出现「萤幕空间不够」的错误，但印表机 off line 和缺纸却是经常会遇到的问题。

我们也不必担心显示卡不能执行某些图形操作，更不用担心显示卡能否处理图形，因为，如果它不能处理图形，就根本不能使用 Windows。但有些印表机不能列印图形（尽管它们能在 Windows 环境中使用）。绘图机尽管可以列印向量图形，却存在位元图块的传输问题。

以下是其他一些需要考虑的问题：

- 印表机比视讯显示器慢。尽管我们没有机会将程式性能调整到最佳状态，却不必担心视讯显示器更新所需的时间。然而，没有人想在做其他工作前一直等待印表机完成列印任务。
- 程式可以用新的输出覆盖原有的显示输出，以重新使用视讯显示器表面。这对印表机是不可能的，印表机只能用完一整页纸，然後在新一页的纸上列印新的内容。
- 在视讯显示器上，不同的应用程式都被视窗化。而对於印表机，不同应用程式的输出必须分成不同的文件或列印作业。

为了在 GDI 的其余部分中加入印表机支援功能，Windows 提供几个只用於印表机的函式。这些限用在印表机上的函式（StartDoc、EndDoc、StartPage 和 EndPage）负责将印表机的输出组织列印到纸页上。而一个程式呼叫普通的 GDI 函式在一张纸上显示文字和图形，和在萤幕上显示的方式一样。

在第十五、十七和十八章有列印点阵图、格式化的文字以及 metafile 的其他资讯。

列印入门

当您在 Windows 下使用印表机时，实际上启动了一个包含 GDI32 动态连结

程式库模组、列印驱动程式动态连结模组（带.DRV 副档名）、Windows 幕後列印程式，以及有用到的其他相关模组。在写印表机列印程式之前，让我们先看一看这个程序是如何进行的。

列印和背景处理

当应用程式要使用印表机时，它首先使用 CreateDC 或 PrintDlg 来取得指向印表机装置内容的代号，於是使得印表机装置驱动程式动态连结程式库模组被载入到记忆体（如果还没有载入记忆体的话）并自己进行初始化。然後，程式呼叫 StartDoc 函式，通知说一个新文件开始了。StartDoc 函式是由 GDI 模组来处理的，GDI 模组呼叫印表机装置驱动程式中的 Control 函式告诉装置驱动程式准备进行列印。

列印一个文件的程序以 StartDoc 呼叫开始，以 EndDoc 呼叫结束。这两个呼叫对於在文件页面上书写文字或者绘制图形的 GDI 命令来说，其作用就像分隔页面的书挡一样。每页本身是这样来划清界限的：呼叫 StartPage 来开始一页，呼叫 EndPage 来结束该页。

例如，如果应用程式想在一页纸上画出一个椭圆，它首先呼叫 StartDoc 开始列印任务，然後再呼叫 StartPage 通知这是新的一页，接著呼叫 Ellipse，正如同在萤幕上画一个椭圆一样。GDI 模组通常将程式对印表机装置内容做出的 GDI 呼叫储存在磁片上的 metafile 中，该档案名以字串~EMF（代表「增强型 metafile」）开始，且以.TMP 为副档名。然而，我在这里应该指出，印表机驱动程式可能会跳过这一步骤。

当绘制第一页的 GDI 呼叫结束时，应用程式呼叫 EndPage。现在，真正的工作开始了。印表机驱动程式必须把存放在 metafile 中的各种绘图命令翻译成印表机输出资料。绘制一页图形所需的印表机输出资料量可能非常大，特别是当印表机没有高级页面制作语言时，更是如此。例如，一台每英寸 600 点且使用 8.5 11 英寸印表纸的雷射印表机，如果要定义一个图形页，可能需要 4 百万以上位元组的资料。

为此，印表机驱动程式经常使用一种称作「列印分带」的技术将一页分成若干称为「输出带」的矩形。GDI 模组从印表机驱动程式取得每个输出带的大小，然後设定一个与目前要处理的输出带相等的剪裁区，并为 metafile 中的每个绘图函式呼叫印表机装置驱动程式的 Output 函式，这个程序叫做「将 metafile 输出到装置驱动程式」。对装置驱动程式所定义的页面上的每个输出带，GDI 模组必须将整个 metafile「输出到」装置驱动程式。这个程序完成以後，该 metafile 就可以删除了。

对每个输出带，装置驱动程式将这些绘图函式转换为在印表机上列印这些图形所需要的输出资料。这种输出资料的格式是依照印表机的特性而异的。对点阵印表机，它将是包括图形序列在内的一系列控制命令序列的集合（印表机驱动程式也能呼叫在 GDI 模组中的各种「helper」辅助常式，用来协助这种输出的构造）。对於带有高阶页面制作语言（如 PostScript）的雷射印表机，印表机将用这种语言进行输出。

列印驱动程式将列印输出的每个输出带传送到 GDI 模组。随後，GDI 模组将该列印输出存入另一个暂存档案中，该暂存档案名以字串~SPL 开始，带有.TMP 副档名。当处理好整页之後，GDI 模组对幕後列印程式进行一个程序间呼叫，通知它一个新的列印页已经准备好了。然後，应用程式就转向处理下一页。当应用程式处理完所有要列印的输出页後，它就呼叫 EndDoc 发出一个信号，表示列印作业已经完成。图 13-1 显示了应用程式、GDI 模组和列印驱动程式的交互作用程序。

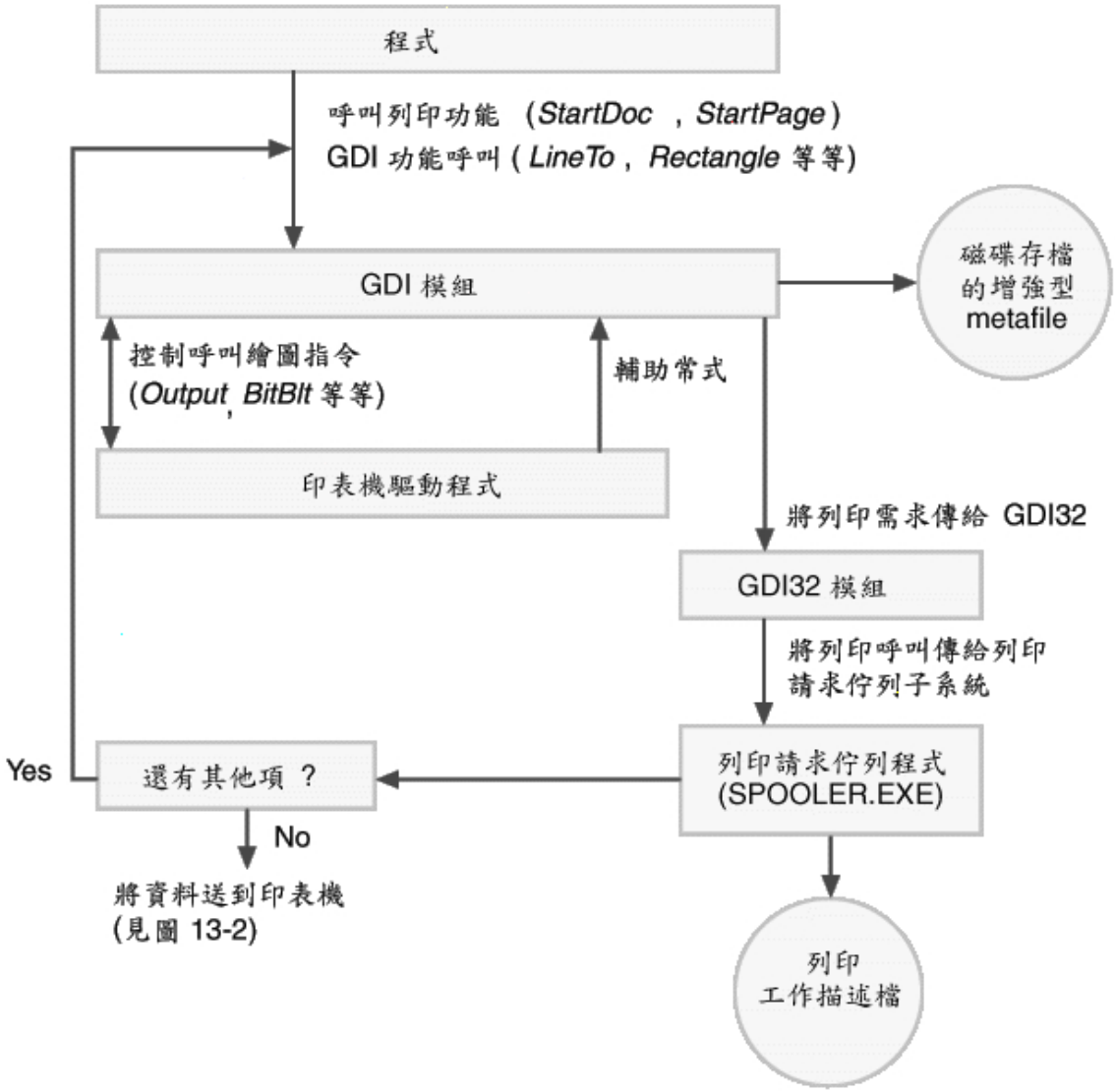

图 13-1 应用程式、GDI 模组、列印驱动程式和列印伫列程式的交互作用过程

Windows 幕後列印程式实际上是几个元件的一种组合（见表 13-1）。

表 13-1

列印伫列程式元件	说明
列印请求伫列程式	将资料流程传递给列印功能提供者
本地列印功能提供者	为本地印表机建立背景档案
网路列印功能提供者	为网路印表机建立背景档案
列印处理程式	将列印伫列中与装置无关的资料转换为针对目的印表机的格式
列印埠监视程式	控制项连结印表机的埠
列印语言监视程式	控制项可以双向通讯的印表机，设定装置设定并检测印表机状态

列印伫列程式可以减轻应用程式的列印负担。 Windows 在启动时就载入列印伫列程式，因此，当应用程式开始列印时，它已经是活动的了。当程式列印一个档案时，GDI 模组会建立包含列印输出资料的档案。幕後列印程式的任务是将这些档案发往印表机。GDI 模组发出一个讯息来通知它一个新的列印作业开始，然後它开始读档案并将档案直接传送到印表机。为了传送这些档案，列印伫列程式依照印表机所连结的并列埠或串列埠使用各种不同的通信函式。在列印伫列程式向印表机发送档案的操作完成後，它就将包含输出资料的暂存档案删除。这个交互作用过程如图 13-2 所示。

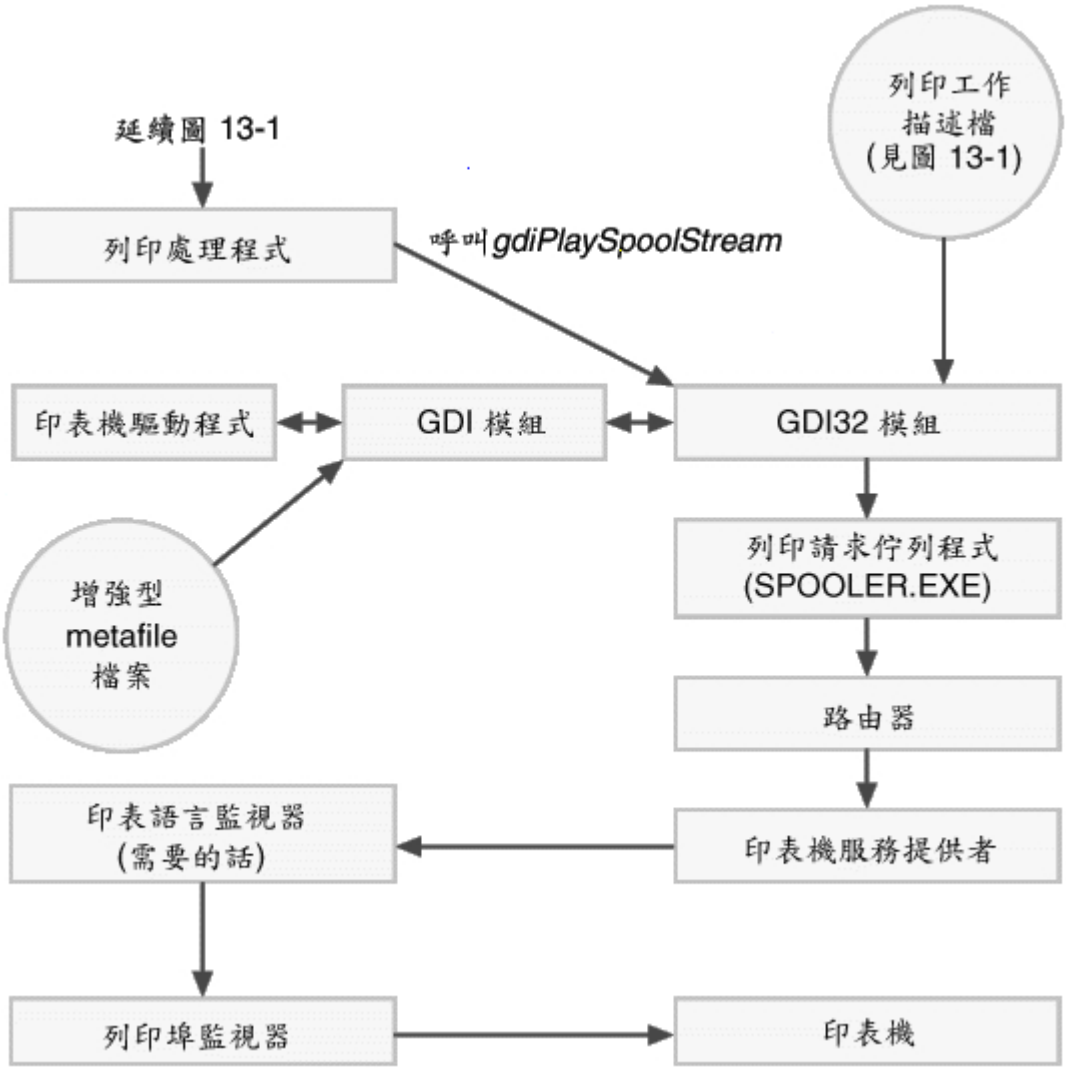

图 13-2 幕後列印程式的操作程序

这个程序的大部分对应用程式来说是透明的。从应用程式的角度来看，「列印」只发生在 GDI 模组将所有列印输出资料储存到磁片档案中的时候，在这之後（如果列印是由第二个执行绪来操作的，甚至可以在这之前）应用程式可以自由地进行其他操作。真正的档案列印操作成了幕後列印程式的任务，而不是应用程式的任务。通过印表机档案夹，使用者可以暂停列印作业、改变作业的优先顺序或取消列印作业。这种管理方式使应用程式能更快地将列印资料以即时方式列印，况且这样必须等到列印完一页後才能处理下一页。

我们已经描述了一般的列印原理，但还有一些例外情况。其中之一是 Windows 程式要使用印表机时，并非一定需要幕後列印程式。使用者可以在印表机属性表格的详细资料属性页中关闭印表机的背景操作。

为什么使用者希望不使用背景操作呢？因为使用者可能使用了比 Windows 列印伫列程式更快的硬体或软体幕後列印程式，也可能是印表机在一个自身带有列印伫列器的网路上使用。一般的规则是，使用一个列印伫列程式比使用两个列印伫列程式更快。去掉 Windows 幕後列印程式可以加快列印速度，因为列

印输出资料不必储存在硬碟上，而可以直接输出到印表机，并被外部的硬体列印伫列器或软体的幕後列印程式所接收。

如果没有启用 Windows 列印伫列程式，GDI 模组就不把来自装置驱动程式的列印输出资料存入档案中，而是将这些输出资料直接输出到列印输出埠。与列印伫列程式进行的列印不同，GDI 进行的列印一定会让应用程式暂停执行一段时间（特别是进行列印中的程式）直到列印完成。

还有另一个例外。通常，GDI 模组将定义一页所需的所有函式存入一个增强型 metafile 中，然後替驱动程式定义的每个列印输出带输出一遍该 metafile 到列印驱动程式中。然而，如果列印驱动程式不需要列印分带的话，就不会建立这个 metafile；GDI 只需简单地将绘图函式直接送往驱动程式。进一步的变化是，应用程式也可能得承担起对列印输出资料进行列印分带的责任，这就使得应用程式中的列印程式码更加复杂了，但却免去了 GDI 模组建立 metafile 的麻烦。这样，GDI 只需简单地为每个输出带将函式传到列印驱动程式。

或许您现在已经发现了从一个 Windows 应用程式进行列印操作要比使用视讯显示器的负担更大，这样可能出现一些问题——特别是，如果 GDI 模组在建立 metafile 或列印输出档案时耗尽了磁碟空间。您可以更关切这些问题，并尝试著处理这些问题并告知使用者，或者您当然也可以置之不理。

对於一个应用程式，列印文件的第一步就是如何取得印表机装置的内容。

印表机装置内容

正如在视讯显示器上绘图前需要得到装置内容代号一样，在列印之前，使用者必须取得一个印表机装置内容代号。一旦有了这个代号（并为建立一个新文件呼叫了 StartDoc 以及呼叫 StartPage 开始一页），就可以用与使用视讯显示装置内容代号相同的方法来使用印表机装置内容代号，该代号即为各种 GDI 呼叫的第一个参数。

大多数应用程式经由呼叫 PrintDlg 函式打开一个标准的列印对话方块（本章後面会展示该函式的用法）。这个函式还为使用者提供了一个在列印之前改变印表机或者指定其他特性的机会。然後，它将印表机装置内容代号交给应用程式。该函式能够省下应用程式的一些工作。然而，某些应用程式（例如 Notepad）仅需要取得印表机装置内容，而不需要那个对话方块。要做到这一点，需要呼叫 CreateDC 函式。

在第五章中，您已知道如何通过如下的呼叫来为整个视讯显示器取得指向装置内容的代号：

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

您也可以使用该函式来取得印表机装置内容代号。然而，对印表机装置内容，CreateDC 的一般语法为：

```
hdc = CreateDC (NULL, szDeviceName, NULL, pInitializationData) ;
```

pInitializationData 参数一般被设为 NULL。szDeviceName 参数指向一个字串，以告诉 Windows 印表机设备的名称。在设定设备名称之前，您必须知道有哪些印表机可用。

一个系统可能有不只一台连结著的印表机，甚至可以有其他程式，如传真软体，将自己伪装成印表机。不论连结的印表机有多少台，都只能有一台被认为是「目前的印表机」或者「内定印表机」，这是使用者最近一次选择的印表机。许多小型的 Windows 程式只使用内定印表机来进行列印。

取得内定印表机装置内容的方式不断在改变。目前，标准的方法是使用 EnumPrinters 函式来获得。该函式填入一个包含每个连结著的印表机资讯的阵列结构。根据所需的细节层次，您还可以选择几种结构之一作为该函式的参数。这些结构的名称为 PRINTER_INFO_x，x 是一个数字。

不幸的是，所使用的函式还取决於您的程式是在 Windows 98 上执行还是在 Windows NT 上执行。程式 13-1 展示了 GetPrinterDC 函式在两种作业系统上工作的用法。

程式 13-1　GETPRNDC

```
GETPRNDC.C
/*--------------------------------------------------------------------------
      GETPRNDC.C -- GetPrinterDC function
--------------------------------------------------------------------------*/

#include <windows.h>
HDC GetPrinterDC (void)
{
      DWORD                                          dwNeeded, dwReturned ;
      HDC                                            hdc ;
      PRINTER_INFO_4 *       pinfo4 ;
      PRINTER_INFO_5 *       pinfo5 ;

      if (GetVersion () & 0x80000000)     // Windows 98
      {
                  EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, NULL,
            0, &dwNeeded, &dwReturned) ;
                  pinfo5 = malloc (dwNeeded) ;
                  EnumPrinters (PRINTER_ENUM_DEFAULT, NULL, 5, (PBYTE)
pinfo5,
            dwNeeded, &dwNeeded, &dwReturned) ;
                  hdc = CreateDC (NULL, pinfo5->pPrinterName, NULL, NULL) ;
                  free (pinfo5) ;
```

```
      }
      else
//Windows NT
    {
                        EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, NULL,
                0, &dwNeeded, &dwReturned) ;
                        pinfo4 = malloc (dwNeeded) ;
                        EnumPrinters  (PRINTER_ENUM_LOCAL,  NULL,  4,  (PBYTE)
pinfo4,
                dwNeeded, &dwNeeded, &dwReturned) ;
                        hdc = CreateDC (NULL, pinfo4->pPrinterName, NULL, NULL) ;
                        free (pinfo4) ;
      }
      return hdc ;
}
```

这些函式使用 GetVersion 函式来确定程式是执行在 Windows 98 上还是 Windows NT 上。不管是什么作业系统，函式呼叫 EnumPrinters 两次：一次取得它所需结构的大小，一次填入结构。在 Windows 98 上，函式使用 PRINTER_INFO_5 结构；在 Windows NT 上，函式使用 PRINTER_INFO_4 结构。这些结构在 EnumPrinters 文件（/Platform SDK/Graphics and Multimedia Services/GDI/Printing and Print Spooler/Printing and Print Spooler Reference/Printing and Print Spooler Functions/EnumPrinters，范例小节的前面）中有说明，它们是「容易而快速」的。

修改後的 DEVCAPS 程式

第五章的 DEVCAPS1 程式只显示了从 GetDeviceCaps 函式获得的关於视讯显示的基本资讯。程式 13-2 所示的新版本显示了关於视讯显示和连结到系统之所有印表机的更多资讯。

程式 13-2　DEVCAPS2

```
DEVCAPS2.C
/*------------------------------------------------------------------------
      DEVCAPS2.C --          Displays Device Capability Information (Version 2)
                                                  (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>
#include "resource.h"

LRESULT CALLBACK WndProc           (HWND, UINT, WPARAM, LPARAM) ;
void DoBasicInfo                        (HDC, HDC, int, int) ;
void DoOtherInfo                        (HDC, HDC, int, int) ;
```

```
void DoBitCodedCaps                    (HDC, HDC, int, int, int) ;

typedef struct
{
     int                   iMask ;
     TCHAR *          szDesc ;
}
BITS ;
#define IDM_DEVMODE     1000
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                  PSTR    szCmdLine,    int
iCmdShow)
{
     static TCHAR           szAppName[] = TEXT ("DevCaps2") ;
     HWND                                hwnd ;
     MSG                            msg ;
     WNDCLASS                       wndclass ;

     wndclass.style                 = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc           = WndProc ;
     wndclass.cbClsExtra            = 0 ;
     wndclass.cbWndExtra            = 0 ;
     wndclass.hInstance             = hInstance ;
     wndclass.hIcon                 = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor               = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName       = szAppName ;
     wndclass.lpszClassName      = szAppName ;

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

     hwnd = CreateWindow (szAppName, NULL,
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

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

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

```
     {
                    TranslateMessage (&msg) ;
                    DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

LRESULT CALLBACK WndProc (  HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
   static TCHAR                  szDevice[32], szWindowText[64] ;
     static int                  cxChar, cyChar,  nCurrentDevice        =
IDM_SCREEN,
      nCurrentInfo          = IDM_BASIC ;
     static DWORD                        dwNeeded, dwReturned ;
     static PRINTER_INFO_4 * pinfo4 ;
     static PRINTER_INFO_5 * pinfo5 ;
     DWORD               i ;
     HDC                 hdc, hdcInfo ;
     HMENU               hMenu ;
     HANDLE              hPrint ;
     PAINTSTRUCT          ps ;
     TEXTMETRIC           tm ;

     switch (message)
     {
     case  WM_CREATE :
                    hdc = GetDC (hwnd) ;
               SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;
               GetTextMetrics (hdc, &tm) ;
                    cxChar = tm.tmAveCharWidth ;
                    cyChar = tm.tmHeight + tm.tmExternalLeading ;
                    ReleaseDC (hwnd, hdc) ;
// fall through
     case  WM_SETTINGCHANGE:
                    hMenu = GetSubMenu (GetMenu (hwnd), 0) ;

                    while (GetMenuItemCount (hMenu) > 1)
                         DeleteMenu (hMenu, 1, MF_BYPOSITION) ;

                         // Get a list of all local and remote printers
                         //
                         // First, find out how large an array we need; this
                         //   call will fail, leaving the required size in
dwNeeded
                         //
                         // Next, allocate space for the info array and fill
it
```

```
                              //
                              // Put the printer names on the menu

            if (GetVersion () & 0x80000000)                              //
Windows 98
                  {
                              EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, NULL,
                     0, &dwNeeded, &dwReturned) ;

                              pinfo5 = malloc (dwNeeded) ;

                              EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 5, (PBYTE)
pinfo5,
                     dwNeeded, &dwNeeded, &dwReturned) ;

                              for (i = 0 ; i < dwReturned ; i++)
                              {
                                    AppendMenu  (hMenu,  (i+1)  %  16  ?  0  :
MF_MENUBARBREAK, i + 1,

      pinfo5[i].pPrinterName) ;
                              }
                              free (pinfo5) ;
            }
                  else
// Windows NT
         {
                              EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, NULL,
                     0, &dwNeeded, &dwReturned) ;
                              pinfo4 = malloc (dwNeeded) ;
                              EnumPrinters (PRINTER_ENUM_LOCAL, NULL, 4, (PBYTE)
pinfo4,
                     dwNeeded, &dwNeeded, &dwReturned) ;
                              for (i = 0 ; i < dwReturned ; i++)
                                    {
                              AppendMenu (hMenu, (i+1) % 16 ? 0 : MF_MENUBARBREAK,
i + 1,
                                    pinfo4[i].pPrinterName) ;
             }
                                    free (pinfo4) ;
                  }

                  AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
                  AppendMenu (hMenu, 0, IDM_DEVMODE, TEXT ("Properties")) ;

                  wParam = IDM_SCREEN ;
// fall through
      case  WM_COMMAND :
```

```
                        hMenu = GetMenu (hwnd) ;

                        if (  LOWORD (wParam) == IDM_SCREEN ||   // IDM_SCREEN &
Printers
                                    LOWORD (wParam) < IDM_DEVMODE)
                        {
                        CheckMenuItem (hMenu, nCurrentDevice, MF_UNCHECKED) ;
                        nCurrentDevice = LOWORD (wParam) ;
                        CheckMenuItem (hMenu, nCurrentDevice, MF_CHECKED) ;
                        }
                        else if (LOWORD (wParam) == IDM_DEVMODE)      //
Properties selection
                        {
                                        GetMenuString (hMenu, nCurrentDevice,
szDevice,
                              sizeof (szDevice) / sizeof (TCHAR), MF_BYCOMMAND);

                                        if  (OpenPrinter  (szDevice,  &hPrint,
NULL))
                                        {
                                        PrinterProperties (hwnd, hPrint) ;
                                        ClosePrinter (hPrint) ;
                                        }
                        }
                        else
// info menu items
                {
                                    CheckMenuItem     (hMenu,     nCurrentInfo,
MF_UNCHECKED) ;
                                    nCurrentInfo = LOWORD (wParam) ;
                                    CheckMenuItem     (hMenu,     nCurrentInfo,
MF_CHECKED) ;
                        }
                        InvalidateRect (hwnd, NULL, TRUE) ;
                        return 0 ;

      case  WM_INITMENUPOPUP :
                        if (lParam == 0)
                                        EnableMenuItem    (GetMenu    (hwnd),
IDM_DEVMODE,
                                                nCurrentDevice               ==
IDM_SCREEMF_GRAYED : MF_ENABLED) ;
                        return 0 ;

      case  WM_PAINT :
                  lstrcpy (szWindowText, TEXT ("Device Capabilities: ")) ;

                        if (nCurrentDevice == IDM_SCREEN)
```

```
                    {
                              lstrcpy (szDevice, TEXT ("DISPLAY")) ;
                              hdcInfo = CreateIC (szDevice, NULL, NULL,
NULL) ;
                    }
                    else
                    {
                              hMenu = GetMenu (hwnd) ;
                              GetMenuString    (hMenu,    nCurrentDevice,
szDevice,
                     sizeof (szDevice), MF_BYCOMMAND) ;
                              hdcInfo = CreateIC (NULL, szDevice, NULL,
NULL) ;
                    }

                    lstrcat (szWindowText, szDevice) ;
                    SetWindowText (hwnd, szWindowText) ;

                    hdc = BeginPaint (hwnd, &ps) ;
                    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT)) ;

                    if (hdcInfo)
                    {
                              switch (nCurrentInfo)
                                   {
                              case  IDM_BASIC :
                    DoBasicInfo (hdc, hdcInfo, cxChar, cyChar) ;
                                        break ;

                         case  IDM_OTHER :
                    DoOtherInfo (hdc, hdcInfo, cxChar, cyChar) ;
                                   break ;

                         case  IDM_CURVE :
                         case  IDM_LINE :
                         case  IDM_POLY :
                         case  IDM_TEXT :
                    DoBitCodedCaps (hdc, hdcInfo, cxChar, cyChar,
                              nCurrentInfo - IDM_CURVE) ;
                                   break ;
                              }
                         DeleteDC (hdcInfo) ;
               }

               EndPaint (hwnd, &ps) ;
               return 0 ;

     case  WM_DESTROY :
```

```
                    PostQuitMessage (0) ;
                    return 0 ;
     }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}

void DoBasicInfo (HDC hdc, HDC hdcInfo, int cxChar, int cyChar)
{
     static struct
     {
               int    nIndex ;
               TCHAR * szDesc ;
     }
     info[] =
     {
               HORZSIZE,              TEXT ("HORZSIZE              Width in
millimeters:"),
               VERTSIZE,              TEXT ("VERTSIZE              Height
in millimeters:"),
               HORZRES,               TEXT ("HORZRES               Width in
pixels:"),
               VERTRES,               TEXT ("VERTRES               Height
in raster lines:"),
               BITSPIXEL,             TEXT ("BITSPIXEL             Color
bits per pixel:"),
               PLANES,                TEXT ("PLANES
     Number of color planes:"),
               NUMBRUSHES,            TEXT ("NUMBRUSHES            Number
of device brushes:"),
               NUMPENS,               TEXT ("NUMPENS
     Number of device pens:"),
               NUMMARKERS,            TEXT ("NUMMARKERS            Number
of device markers:"),
               NUMFONTS,              TEXT  ("NUMFONTS
     Number of device fonts:"),
               NUMCOLORS,             TEXT  ("NUMCOLORS
     Number of device colors:"),
               PDEVICESIZE, TEXT("PDEVICESIZE    Size       of       device
structure:"),
ASPECTX,      TEXT("ASPECTX Relative width of pixel:"),
ASPECTY,      TEXT("ASPECTY Relative height of pixel:"),
ASPECTXY,     TEXT("ASPECTXY Relative diagonal of pixel:"),
LOGPIXELSX,   TEXT("LOGPIXELSX Horizontal dots per inch:"),
LOGPIXELSY,   TEXT("LOGPIXELSY Vertical dots per inch:"),
SIZEPALETTE,  TEXT("SIZEPALETTE Number of palette entries:"),
NUMRESERVED,  TEXT("NUMRESERVED Reserved palette entries:"),
COLORRES,     TEXT("COLORRES Actual color resolution:"),
PHYSICALWIDTH, TEXT("PHYSICALWIDTH Printer page pixel width:"),
```

```
PHYSICALHEIGHT,TEXT("PHYSICALHEIGHT Printer page pixel height:"),
PHYSICALOFFSETX,TEXT("PHYSICALOFFSETX Printer page x offset:"),
PHYSICALOFFSETY,TEXT("PHYSICALOFFSETY Printer page y offset:")
      } ;
      int   i ;
      TCHAR szBuffer[80] ;

      for (i = 0 ; i < sizeof (info) / sizeof (info[0]) ; i++)
              TextOut (hdc, cxChar, (i + 1) * cyChar, szBuffer,
             wsprintf (szBuffer, TEXT ("%-45s%8d"), info[i].szDesc,
                 GetDeviceCaps (hdcInfo, info[i].nIndex))) ;
}

void DoOtherInfo (HDC hdc, HDC hdcInfo, int cxChar, int cyChar)
{
      static BITS clip[] =
      {
        CP_RECTANGLE, TEXT ("CP_RECTANGLE  Can Clip To Rectangle:")
      } ;

      static BITS raster[] =
     {
       RC_BITBLT,    TEXT ("RC_BITBLT  Capable of simple BitBlt:"),
            RC_BANDING,   TEXT ("RC_BANDING Requires banding support:"),
       RC_SCALING,   TEXT ("RC_SCALING Requires scaling support:"),
       RC_BITMAP64,  TEXT ("RC_BITMAP64  Supports bitmaps >64K:"),
       RC_GDI20_OUTPUT, TEXT ("RC_GDI20_OUTPUT Has 2.0 output calls:"),
       RC_DI_BITMAP, TEXT ("RC_DI_BITMAP  Supports DIB to memory:"),
            RC_PALETTE,   TEXT ("RC_PALETTE    Supports a palette:"),
            RC_DIBTODEV,  TEXT ("RC_DIBTODEV Supports bitmap conversion:"),
            RC_BIGFONT,   TEXT ("RC_BIGFONT  Supports fonts >64K:"),
            RC_STRETCHBLT,TEXT ("RC_STRETCHBLT Supports StretchBlt:"),
            RC_FLOODFILL, TEXT ("RC_FLOODFILL  Supports FloodFill:"),
            RC_STRETCHDIB,TEXT ("RC_STRETCHDIB Supports StretchDIBits:")
      } ;

      static TCHAR * szTech[]=     {     TEXT ("DT_PLOTTER (Vector plotter)"),
                     TEXT ("DT_RASDISPLAY (Raster display)"),
                     TEXT ("DT_RASPRINTER (Raster printer)"),
                     TEXT ("DT_RASCAMERA (Raster camera)"),
                     TEXT ("DT_CHARSTREAM (Character stream)"),
                     TEXT ("DT_METAFILE (Metafile)"),
                     TEXT ("DT_DISPFILE (Display file)") } ;
      int                                i ;
      TCHAR                              szBuffer[80] ;

      TextOut (hdc, cxChar, cyChar, szBuffer,
        wsprintf (szBuffer, TEXT ("%-24s%04XH"), TEXT ("DRIVERVERSION:"),
```

```
            GetDeviceCaps (hdcInfo, DRIVERVERSION))) ;
     TextOut (hdc, cxChar, 2 * cyChar, szBuffer,
                      wsprintf   (szBuffer,   TEXT   ("%-24s%-40s"),   TEXT
("TECHNOLOGY:"),
                                 szTech[GetDeviceCaps             (hdcInfo,
TECHNOLOGY)])) ;
     TextOut (hdc, cxChar, 4 * cyChar, szBuffer,
                      wsprintf   (szBuffer,   TEXT   ("CLIPCAPS   (Clipping
capabilities)"))) ;
     for (i = 0 ; i < sizeof (clip) / sizeof (clip[0]) ; i++)
                      TextOut (hdc, 9 * cxChar, (i + 6) * cyChar, szBuffer,
                                       wsprintf (szBuffer, TEXT ("%-45s %3s"),
clip[i].szDesc,
                                            GetDeviceCaps (hdcInfo, CLIPCAPS)
& clip[i].iMask ?
                                       TEXT ("Yes") : TEXT ("No"))) ;
     TextOut (hdc, cxChar, 8 * cyChar, szBuffer,
                  wsprintf    (szBuffer,    TEXT    ("RASTERCAPS    (Raster
capabilities)"))) ;
     for (i = 0 ; i < sizeof (raster) / sizeof (raster[0]) ; i++)
     TextOut (hdc, 9 * cxChar, (i + 10) * cyChar, szBuffer,
                                       wsprintf (szBuffer, TEXT ("%-45s %3s"),
raster[i].szDesc,
                                             GetDeviceCaps         (hdcInfo,
RASTERCAPS) & raster[i].iMask ?
                            TEXT ("Yes") : TEXT ("No"))) ;
}

void DoBitCodedCaps (  HDC hdc, HDC hdcInfo, int cxChar, int cyChar,int iType)
{
     static BITS curves[] =
     {
                CC_CIRCLES,      TEXT ("CC_CIRCLES      Can do circles:"),
                CC_PIE,          TEXT ("CC_PIE          Can do pie wedges:"),
                CC_CHORD,        TEXT ("CC_CHORD        Can do chord arcs:"),
                CC_ELLIPSES,     TEXT ("CC_ELLIPSES     Can do ellipses:"),
                CC_WIDE,         TEXT ("CC_WIDE         Can     do     wide
borders:"),
                CC_STYLED,       TEXT ("CC_STYLED       Can    do    styled
borders:"),
                CC_WIDESTYLED,   TEXT  ("CC_WIDESTYLED Can do wide and styled
borders:"),
                CC_INTERIORS,    TEXT ("CC_INTERIORS    Can do interiors:")
     } ;

     static BITS lines[] =
     {
                LC_POLYLINE,     TEXT  ("LC_POLYLINE Can do polyline:"),
```

```
                LC_MARKER,        TEXT  ("LC_MARKER Can do markers:"),
                LC_POLYMARKER,    TEXT  ("LC_POLYMARKER Can do polymarkers"),
                LC_WIDE,          TEXT  ("LC_WIDE Can do wide lines:"),
                LC_STYLED,        TEXT  ("LC_STYLED Can do styled lines:"),
                LC_WIDESTYLED,    TEXT  ("LC_WIDESTYLED  Can do wide and styled
lines:"),
                LC_INTERIORS,     TEXT ("LC_INTERIORS  Can do interiors:")
     } ;

     static BITS poly[] =
     {
                PC_POLYGON,
                          TEXT ("PC_POLYGON      Can   do   alternate   fill
polygon:"),
                PC_RECTANGLE,     TEXT  ("PC_RECTANGLE Can do rectangle:"),
           PC_WINDPOLYGON,
                          TEXT ("PC_WINDPOLYGON Can do winding number fill
polygon:"),
                PC_SCANLINE,      TEXT ("PC_SCANLINE   Can do scanlines:"),
                PC_WIDE,          TEXT ("PC_WIDE       Can do wide borders:"),
                PC_STYLED,        TEXT  ("PC_STYLED          Can  do  styled
borders:"),
                PC_WIDESTYLED,
                          TEXT  ("PC_WIDESTYLED   Can  do  wide  and  styled
borders:"),
                PC_INTERIORS,     TEXT ("PC_INTERIORS    Can do interiors:")
     } ;

     static BITS text[] =
     {
                TC_OP_CHARACTER, TEXT ("TC_OP_CHARACTER Can   do   character
output precision:"),
                TC_OP_STROKE,    TEXT ("TC_OP_STROKE     Can do stroke output
precision:"),
                TC_CP_STROKE,    TEXT ("TC_CP_STROKE     Can  do  stroke  clip
precision:"),
                TC_CR_90,        TEXT ("TC_CP_90      Can do 90 degree character
rotation:"),
                TC_CR_ANY,        TEXT  ("TC_CR_ANY      Can do any character
rotation:"),
                TC_SF_X_YINDEP,    TEXT  ("TC_SF_X_YINDEP    Can  do  scaling
independent of X and Y:"),
                TC_SA_DOUBLE,    EXT ("TC_SA_DOUBLE    Can do doubled character
for scaling:"),
                TC_SA_INTEGER,     TEXT  ("TC_SA_INTEGER     Can  do  integer
multiples for scaling:"),
                TC_SA_CONTIN,    TEXT ("TC_SA_CONTIN     Can do any multiples
for exact scaling:"),
```

```
                TC_EA_DOUBLE,    TEXT ("TC_EA_DOUBLE   Can do double weight
characters:"),
                TC_IA_ABLE,      TEXT ("TC_IA_ABLE     Can do italicizing:"),
                TC_UA_ABLE,      TEXT ("TC_UA_ABLE     Can do underlining:"),
                TC_SO_ABLE,      TEXT ("TC_SO_ABLE     Can do strikeouts:"),
                TC_RA_ABLE,      TEXT ("TC_RA_ABLE     Can do raster fonts:"),
                TC_VA_ABLE,      TEXT ("TC_VA_ABLE     Can do vector fonts:")
     } ;

     static struct
     {
          int                    iIndex ;
                TCHAR *          szTitle ;
                BITS                   (*pbits)[] ;
                int                    iSize ;
     }
     bitinfo[] =
     {
                CURVECAPS,       TEXT ("CURVCAPS (Curve Capabilities)"),
                           (BITS (*)[]) curves, sizeof (curves) / sizeof
(curves[0]),
                LINECAPS,        TEXT ("LINECAPS (Line Capabilities)"),
                           (BITS (*)[]) lines, sizeof (lines) / sizeof
(lines[0]),
                POLYGONALCAPS,     TEXT     ("POLYGONALCAPS     (Polygonal
Capabilities)"),
                           (BITS (*)[]) poly, sizeof (poly) / sizeof (poly[0]),
                TEXTCAPS,        TEXT ("TEXTCAPS (Text Capabilities)"),
                           (BITS (*)[]) text, sizeof (text) / sizeof (text[0])
     } ;

     static TCHAR szBuffer[80] ;
     BITS                  (*pbits)[] = bitinfo[iType].pbits ;
     int              i,     iDevCaps     =     GetDeviceCaps     (hdcInfo,
bitinfo[iType].iIndex) ;

     TextOut (hdc, cxChar, cyChar, bitinfo[iType].szTitle,
                                       lstrlen (bitinfo[iType].szTitle)) ;
     for (i = 0 ; i < bitinfo[iType].iSize ; i++)
            extOut (hdc, cxChar, (i + 3) * cyChar, szBuffer,
          wsprintf (szBuffer, TEXT ("%-55s %3s"), (*pbits)[i].szDesc,
          iDevCaps & (*pbits)[i].iMask ? TEXT ("Yes") : TEXT ("No")));
}
```

DEVCAPS2.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
```

```
//////////////////////////////////////////////////////////////////////////
/
// Menu
DEVCAPS2 MENU DISCARDABLE
BEGIN
      POPUP "&Device"
      BEGIN
                        MENUITEM "&Screen",IDM_SCREEN, CHECKED
      END
      POPUP "&Capabilities"
      BEGIN
                  MENUITEM "&Basic Information",IDM_BASIC
                        MENUITEM "&Other Information",IDM_OTHER
                        MENUITEM "&Curve Capabilities",IDM_CURVE
                        MENUITEM "&Line Capabilities",IDM_LINE
                        MENUITEM "&Polygonal Capabilities",IDM_POLY
                        MENUITEM "&Text Capabilities",IDM_TEXT
      END
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by DevCaps2.rc

#define IDM_SCREEN       40001
#define IDM_BASIC        40002
#define IDM_OTHER        40003
#define IDM_CURVE        40004
#define IDM_LINE         40005
#define IDM_POLY         40006
#define IDM_TEXT         40007
```

因为 DEVCAPS2 只取得印表机的资讯内容，使用者仍然可以从 DEVCAPS2 的功能表中选择所需印表机。如果使用者想比较不同印表机的功能，可以先用印表机档案夹增加各种列印驱动程式。

PrinterProperties 呼叫

DEVCAPS2 的「Device」功能表中上还有一个称为「Properties」的选项。要使用这个选项，首先得从 Device 功能表中选择一个印表机，然後再选择 Properties ，这时弹出一个对话方块。对话方块从何而来呢？它由印表机驱动程式呼叫，而且至少还让使用者选择纸的尺寸。大多数印表机驱动也可以让使用者在「直印（portrait）」或「横印（landscape）」模式中进行选择。在直印模式（一般为内定模式）下，纸的短边是顶部。在横印模式下，纸的长边是顶部。如果改变该模式，则所作的改变将在 DEVCAPS2 程式从 GetDeviceCaps 函式取得的资讯中反应出来：水平尺寸和解析度将与垂直尺寸和解析度交换。

彩色绘图机的「Properties」对话方块内容十分广泛，它们要求使用者输入安装在绘图机上之画笔的颜色和使用之绘图纸（或透明胶片）的型号。

所有印表机驱动程式都包含一个称为 ExtDeviceMode 的输出函式，它呼叫对话方块并储存使用者输入的资讯。有些印表机驱动程式也将这些资讯储存在系统登录的自己拥有的部分中，有些则不然。那些储存资讯的印表机驱动程式在下次执行 Windows 时将存取该资讯。

允许使用者选择印表机的 Windows 程式通常只呼叫 PrintDlg（本章後面我会展示用法）。这个有用的函式在准备列印时负责和使用者之间所有的通讯工作，并负责处理使用者要求的所有改变。当使用者单击「Properties」按钮时，PrintDlg 还会启动属性表格对话方块。

程式还可以通过直接呼叫印表机驱动程式的 ExtDeviceMode 或 ExtDeveModePropSheet 函式，来显示印表机的属性对话方块，然而，我不鼓励您这样做。像 DEVCAPS2 那样，透过呼叫 PrinterProperties 来启动对话方块会好得多。

PrinterProperties 要求印表机物件的代号，您可以通过 OpenPrinter 函式来得到。当使用者取消属性表格对话方块时，PrinterProperties 传回，然後使用者通过呼叫 ClosePrinter，释放印表机代号。DEVCAPS2 就是这样做到这一点的。

程式首先取得刚刚在 Device 功能表中选择的印表机名称，并将其存入一个名为 szDevice 的字元阵列中。

```
GetMenuString (  hMenu, nCurrentDevice, szDevice,
                                        sizeof (szDevice) / sizeof (TCHAR),
MF_BYCOMMAND) ;
```

然後，使用 OpenPrinter 获得该设备的代号。如果呼叫成功，那么程式接著呼叫 PrinterProperties 启动对话方块，然後呼叫 ClosePrinter 释放设备代号：

```
if (OpenPrinter (szDevice, &hPrint, NULL))
{
      PrinterProperties (hwnd, hPrint) ;
      ClosePrinter (hPrint) ;
}
```

检查 BitBlt 支援

您可以用 GetDeviceCaps 函式来取得页中可列印区的尺寸和解析度（通常，该区域不会与整张纸的大小相同）。如果使用者想自己进行缩放操作，也可以获得相对的图素宽度和高度。

印表机能力的大多数资讯是用於 GDI 而不是应用程式的。通常，在印表机不能做某件事时，GDI 会模拟出那项功能。然而，这是应用程式应该事先检查的。

以 RASTERCAPS（「位元映射支援」）参数呼叫 GetDeviceCaps，它传回的 RC_BITBLT 位元包含了另一个重要的印表机特性，该位元标示设备是否能进行位元块传送。大多数点阵印表机、雷射印表机和喷墨印表机都能进行位元块传送，而大多数绘图机却不能。不能处理位元块传送的设备不支援下列 GDI 函式：CreateCompatibleDC、CreateCompatibleBitmap、PatBlt、BitBlt、StretchBlt、GrayString、DrawIcon、SetPixel、GetPixel、FloodFill、ExtFloodFill、FillRgn、FrameRgn、InvertRgn、PaintRgn、FillRect、FrameRect 和 InvertRect。这是在视讯显示器上使用 GDI 函式与在印表机上使用它们的唯一重要区别。

最简单的列印程式

现在可以开始列印了，我们尽可能简单地开始。事实上，我们的第一个程式只是让印表机送纸而已。程式 13-3 的 FORMFEED 程式，展示了列印所需的最小需求。

程式 13-3　FORMFEED

```
FORMFEED.C
/*------------------------------------------------------------------------
      FORMFEED.C --     Advances printer to next page
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------*/

#include <windows.h>
HDC GetPrinterDC (void) ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                    LPSTR lpszCmdLine, int iCmdShow)
{
      static DOCINFO di = { sizeof (DOCINFO), TEXT ("FormFeed") } ;
      HDC                                   hdcPrint = GetPrinterDC () ;

      if (hdcPrint != NULL)
      {
            if (StartDoc (hdcPrint, &di) > 0)
                                    if  (StartPage  (hdcPrint)  >  0  &&  EndPage
(hdcPrint) > 0)
                                                EndDoc (hdcPrint) ;
                  DeleteDC (hdcPrint) ;
      }
      return 0 ;
}
```

这个程式也需要前面程式 13-1 中的 GETPRNDC.C 档案。

除了取得印表机装置内容（然後再删除它）外，程式只呼叫了我们在本章前面讨论过的四个列印函式。FORMFEED 首先呼叫 StartDoc 开始一个新的档案，它测试从 StartDoc 传回的值，只有传回值是正数时，才继续下去：

```
if (StartDoc (hdcPrint, &di) > 0)
```

StartDoc 的第二个参数是指向 DOCINFO 结构的指标。该结构在第一个栏位包含了结构的大小，在第二个栏位包含了字串「FormFeed」。当档案正在被列印或者在等待列印时，这个字串将出现在印表机任务伫列中的「Document Name」列中。通常，该字串包含进行列印的应用程式名称和被列印的档案名称。

如果 StartDoc 成功（由一个正的传回值表示），那么 FORMFEED 呼叫 StartPage，紧接著立即呼叫 EndPage。这一程序将印表机推进到新的一页，再次对传回值进行测试：

```
if (StartPage (hdcPrint) > 0 && EndPage (hdcPrint) > 0)
```

最後，如果不出错，文件就结束：

```
EndDoc (hdcPrint) ;
```

要注意的是，只有当没出错时，才呼叫 EndDoc 函式。如果其他列印函式中的某一个传回错误代码，那么 GDI 实际上已经中断了文件的列印。如果印表机目前未列印，这种错误代码通常会使印表机重新设定。测试列印函式的传回值是检测错误的最简单方法。如果您想向使用者报告错误，就必须呼叫 GetLastError 来确定错误。

如果您写过 MS-DOS 下的简单利用印表机送纸的程式，就应该知道，对於大多数印表机，ASCII 码 12 启动送纸。为什么不简单地使用 C 的程式库函式 open，然後用 write 输出 ASCII 码 12 呢？当然，您完全可以这么做，但是必须确定印表机连结的是串列埠还是并列埠。然後您还要确定另外的程式（例如，列印伫列程式）是不是正在使用印表机。您并不希望在文件列印到一半时被别的程式把正在列印的那张纸送出印表机，对不对？最後，您还必须确定 ASCII 码 12 是不是所连结印表机的送纸字元，因为并非所有印表机的送纸字元都是 12。事实上，在 PostScript 中的送纸命令便不是 12，而是单字 showpage。

简单地说，不要试图直接绕过 Windows；而应该坚持在列印中使用 Windows 函式。

列印图形和文字

在一个 Windows 程式中，列印所需的额外负担通常比 FORMFEED 程式高得多，而且还要用 GDI 函式来实际列印一些东西。我们来写个列印一页文字和图形的程式，采用 FORMFEED 程式中的方法，并加入一些新的东西。该程式将有三个版本 PRINT1、PRINT2 和 PRINT3。为避免程式码重复，每个程式都用前面所示的

GETPRNDC.C 档案和 PRINT.C 档案中的函式，如程式 13-4 所示。

程式 13-4 PRINT

```
PRINT.C
/*--------------------------------------------------------------------------
      PRINT.C -- Common routines for Print1, Print2, and Print3
--------------------------------------------------------------------------*/

#include <windows.h>
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL PrintMyPage (HWND) ;

extern HINSTANCE hInst ;
extern TCHAR                szAppName[] ;
extern TCHAR                szCaption[] ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
      HWND              hwnd ;
      MSG               msg ;
      WNDCLASS          wndclass ;

      wndclass.style                    = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc              = WndProc ;
      wndclass.cbClsExtra               = 0 ;
      wndclass.cbWndExtra               = 0 ;
      wndclass.hInstance                = hInstance ;
      wndclass.hIcon                    = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor                  = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground      = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName       = NULL ;
      wndclass.lpszClassName      = szAppName ;

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

      hInst = hInstance ;
      hwnd = CreateWindow (szAppName, szCaption,
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL) ;
```

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

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

void PageGDICalls (HDC hdcPrn, int cxPage, int cyPage)
{
      static TCHAR szTextStr[] = TEXT ("Hello, Printer!") ;
      Rectangle (hdcPrn, 0, 0, cxPage, cyPage) ;
      MoveToEx (hdcPrn, 0, 0, NULL) ;
      LineTo   (hdcPrn, cxPage, cyPage) ;
      MoveToEx (hdcPrn, cxPage, 0, NULL) ;
      LineTo   (hdcPrn, 0, cyPage) ;

      SaveDC (hdcPrn) ;

      SetMapMode             (hdcPrn, MM_ISOTROPIC) ;
      SetWindowExtEx         (hdcPrn, 1000, 1000, NULL) ;
      SetViewportExtEx       (hdcPrn, cxPage / 2, -cyPage / 2, NULL) ;
      SetViewportOrgEx       (hdcPrn, cxPage / 2,  cyPage / 2, NULL) ;

      Ellipse (hdcPrn, -500, 500, 500, -500) ;
      SetTextAlign (hdcPrn, TA_BASELINE | TA_CENTER) ;
      TextOut (hdcPrn, 0, 0, szTextStr, lstrlen (szTextStr)) ;

      RestoreDC (hdcPrn, -1) ;
}

LRESULT CALLBACK WndProc (   HWND  hwnd,  UINT  message,  WPARAM  wParam,LPARAM
lParam)
{
      static int             cxClient, cyClient ;
      HDC              hdc ;
      HMENU                  hMenu ;
      PAINTSTRUCT            ps ;

      switch (message)
      {
      case  WM_CREATE:
                        hMenu = GetSystemMenu (hwnd, FALSE) ;
                        AppendMenu (hMenu, MF_SEPARATOR, 0, NULL) ;
```

```
                    AppendMenu (hMenu, 0, 1, TEXT ("&Print")) ;
                    return 0 ;

     case  WM_SIZE:
                    cxClient = LOWORD (lParam) ;
                    cyClient = HIWORD (lParam) ;
                    return 0 ;

     case  WM_SYSCOMMAND:
                    if (wParam == 1)
               {
                              if (!PrintMyPage (hwnd))
                                   MessageBox(hwnd,TEXT("Couldnotprint
page!"),
                              szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                              return 0 ;
               }
               break ;

     case  WM_PAINT :
               hdc = BeginPaint (hwnd, &ps) ;

                    PageGDICalls (hdc, cxClient, cyClient) ;

                    EndPaint (hwnd, &ps) ;
                    return 0 ;

     case  WM_DESTROY :
                    PostQuitMessage (0) ;
                    return 0 ;
     }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

PRINT.C 包括函式 WinMain、WndProc 以及一个称为 PageGDICalls 的函式。PageGDICalls 函式接收印表机装置内容代号和两个包含列印页面宽度及高度的变数。这个函式还负责画一个包围整个页面的矩形，有两条对角线，页中间有一个椭圆（其直径是印表机高度和宽度中较小的那个的一半），文字「Hello, Printer!」位於椭圆的中间。

处理 WM_CREATE 讯息时，WndProc 将一个「Print」选项加到系统功能表上。选择该选项将呼叫 PrintMyPage，此函式的功能在程式的三个版本中将不断增强。当列印成功时，PrintMyPage 传回 TRUE 值，如果遇到错误时则传回 FALSE。如果 PrintMyPage 传回 FALSE，WndProc 就会显示一个讯息方块以告知使用者发生了错误。

列印的基本程序

列印程式的第一个版本是 PRINT1，见程式 13-5。经编译後即可执行此程式，然後从系统功能表中选择「Print」。接著，GDI 将必要的印表机输出储存在一个暂存档案中，然後列印伫列程式将它发送给印表机。

程式 13-5 PRINT1

```
PRINT1.C
/*--------------------------------------------------------------------
      PRINT1.C -- Bare Bones Printing
                                                (c) Charles Petzold, 1998
--------------------------------------------------------------------*/

#include <windows.h>
HDC   GetPrinterDC (void) ;                    // in GETPRNDC.C
void  PageGDICalls (HDC, int, int) ;           // in PRINT.C

HINSTANCE hInst ;
TCHAR            szAppName[] = TEXT ("Print1") ;
TCHAR            szCaption[] = TEXT ("Print Program 1") ;

BOOL PrintMyPage (HWND hwnd)
{
      static DOCINFO di = {  sizeof (DOCINFO), TEXT ("Print1: Printing") } ;
      BOOL                        bSuccess = TRUE ;
      HDC                   hdcPrn ;
      int                   xPage, yPage ;

      if    (NULL == (hdcPrn = GetPrinterDC ()))
                  return FALSE ;
      xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
      yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

      if (StartDoc (hdcPrn, &di) > 0)
      {
                  if (StartPage (hdcPrn) > 0)
                  {
                              PageGDICalls (hdcPrn, xPage, yPage) ;

                              if (EndPage (hdcPrn) > 0)
                                          EndDoc (hdcPrn) ;
                              else
                                    bSuccess = FALSE ;
                  }
      }
      else
                  bSuccess = FALSE ;
```

```
      DeleteDC (hdcPrn) ;
      return bSuccess ;
}
```

我们来看看 PRINT1.C 中的程式码。如果 PrintMyPage 不能取得印表机的装置内容代号，它就传回 FALSE，并且 WndProc 显示讯息方块指出错误。如果函式成功取得了装置内容代号，它就通过呼叫 GetDeviceCaps 来确定页面的水平和垂直大小（以图素为单位）。

```
xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
yPage = GetDeviceCaps (hdcPrn, VERTRES) ;
```

这不是纸的全部大小，只是纸的可列印区域。呼叫後，除了 PRINT1 在 StartPage 和 EndPage 呼叫之间呼叫 PageGDICalls，PRINT1 的 PrintMyPage 函式中的程式码在结构上与 FORMFEED 中的程式码相同。仅当呼叫 StartDoc、StartPage 和 EndPage 都成功时，PRINT1 才呼叫 EndDoc 列印函式。

使用放弃程序来取消列印

对於大型文件，程式应该提供使用者在应用程式列印期间取消列印任务的便利性。也许使用者只要列印文件中的一页，而不是列印全部的 537 页。应该要能在印完全部的 537 页之前纠正这个错误。

在一个程式内取消一个列印任务需要一种被称为「放弃程序」的技术。放弃程序在程式中只是个较小的输出函式，使用者可以使用 SetAbortProc 函式将该函式的位址传给 Windows。然後 GDI 在列印时，重复呼叫该程序，不断地问：「我是否应该继续列印？」

我们看看将放弃程序加到列印处理程式中去需要些什么，然後检查一些旁枝末节。放弃程序一般命名为 AbortProc，其形式为：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
      //其他行程式
}
```

列印前，您必须通过呼叫 SetAbortProc 来登记放弃程序：

```
SetAbortProc (hdcPrn, AbortProc) ;
```

在呼叫 StartDoc 前呼叫上面的函式，列印完成後不必清除放弃程序。

在处理 EndPage 呼叫时（亦即，在将 metafile 放入装置驱动程式并建立临时列印档案时），GDI 常常呼叫放弃程序。参数 hdcPrn 是印表机装置内容代号。如果一切正常，iCode 参数是 0，如果 GDI 模组在生成暂存档案时耗尽了磁碟空间，iCode 就是 SP_OUTOFDISK。

如果列印作业继续，那么 AbortProc 必须传回 TRUE（非零）；如果列印作

业异常结束，就传回 FALSE（零）。放弃程序可以被简化为如下所示的形式：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
      MSG   msg ;

      while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
      {
                  TranslateMessage (&msg) ;
                  DispatchMessage (&msg) ;
      }
      return TRUE ;
}
```

这个函式看起来有点特殊，其实它看起来像是讯息回圈。使用者会注意到，这个「讯息回圈」呼叫 PeekMessage 而不是 GetMessage。我在第五章的 RANDRECT 程式中讨论过 PeekMessage。应该还记得，PeekMessage 将会控制权返回给程式，而不管程式的讯息伫列中是否有讯息存在。

只要 PeekMessage 传回 TRUE，那么 AbortProc 函式中的讯息回圈就重复呼叫 PeekMessage。TRUE 值表示 PeekMessage 已经找到一个讯息，该讯息可以通过 TranslateMessage 和 DispatchMessage 发送到程式的视窗讯息处理程式。若程式的讯息伫列中没有讯息，则 PeekMessage 的传回值为 FALSE，因此 AbortProc 将控制权返回给 Windows。

Windows 如何使用 AbortProc

当程式进行列印时，大部分工作发生在要呼叫 EndPage 时。呼叫 EndPage 前，程式每呼叫一次 GDI 绘图函式，GDI 模组只是简单地将另一个记录加到磁片上的 metafile 中。当 GDI 得到 EndPage 後，对列印页中由装置驱动程式定义的每个输出带，GDI 都将该 metafile 送入装置驱动程式中。然後，GDI 将印表机驱动程式建立的列印输出储存到一个档案中。如果没有启用幕後列印，那么 GDI 模组必须自动将该列印输出写入印表机。

在 EndPage 呼叫期间，GDI 模组呼叫您设定的放弃程序。通常 iCode 参数为 0，但如果由於存在未列印的其他暂存档案，而造成 GDI 执行时磁碟空间不够，iCode 参数就为 SP_OUTOFDISK（通常您不会检查这个值，但是如果愿意，您可以进行检查）。放弃程序随後进入 PeekMessage 回圈从自己的讯息伫列中找寻讯息。

如果在程式的讯息伫列中没有讯息，PeekMessage 会传回 FALSE，然後放弃程序跳出它的讯息回圈并给 GDI 模组传回一个 TRUE 值，指示列印应该继续进行。然後 GDI 模组继续处理 EndPage 呼叫。

如果有错误发生，那么 GDI 将中止列印程序，这样，放弃程序的主要目的是允许使用者取消列印。为此，我们还需要一个显示「Cancel」按钮的对话方块，让我们采用两个独立的步骤。首先，我们在建立 PRINT2 程式时增加一个放弃程序，然後在 PRINT3 中增加一个带有「Cancel」按钮的对话方块，使放弃程序可用。

实作放弃程序

现在快速复习一下放弃程序的机制。可以定义一个如下所示的放弃程序：

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
      MSG  msg ;
      while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
      {
                  TranslateMessage (&msg) ;
                  DispatchMessage (&msg) ;
      }
      return TRUE ;
}
```

当您想列印什么时，使用下面的呼叫将指向放弃程序的指标传给 Windows：

```
SetAbortProc (hdcPrn, AbortProc) ;
```

在呼叫 StartDoc 之前进行这个呼叫就行了。

不过，事情没有这么简单。我们忽视了 AbortProc 程序中 PeekMessage 回圈这个问题，它是个很大的问题。只有在程式处於列印程序时，AbortProc 程序才会被呼叫。如果在 AbortProc 中找到一个讯息并把它传送给视窗讯息处理程式，就会发生一些非常令人讨厌的事情：使用者可以从功能表中再次选择「Print」，但程式已经处於列印常式之中。程式在列印前一个档案的同时，使用者也可以把一个新档案载入到程式里。使用者甚至可以退出程式！如果这种情况发生了，所有使用者程式的视窗都将被清除。当列印常式执行结束时，除了退到不再有效的视窗常式之外，您无处可去。

这种东西会把人搞得晕头转向，而我们的程式对此并未做任何准备。正是由於这个原因，当设定放弃程序时，首先应禁止程式的视窗接受输入，使它不能接受键盘和滑鼠输入。可以用以下的函式完成这项工作：

```
EnableWindow (hwnd, FALSE) ;
```

它可以禁止键盘和滑鼠的输入进入讯息伫列。因此在列印程序中，使用者不能对程式做任何工作。当列印完成时，应重新允许视窗接受输入：

```
EnableWindow (hwnd, TRUE) ;
```

您可能要问，既然没有键盘或滑鼠讯息进入讯息伫列，为什么我们还要进行 AbortProc 中的 TranslateMessage 和 DispatchMessage 呼叫呢？实际上并不

一定非得需要 TranslateMessage，但是，我们必须使用 DispatchMessage，处理 WM_PAINT 讯息进入讯息伫列中的情况。如果 WM_PAINT 讯息没有得到视窗讯息处理程式中的 BeginPaint 和 EndPaint 的适当处理，由於 PeekMessage 不再传回 FALSE，该讯息就会滞留在伫列中并且妨碍工作。

当列印期间阻止视窗处理输入讯息时，您的程式不会进行显示输出。但使用者可以切换到其他程式，并在那里进行其他工作，而幕後列印程式则能继续将输出档案送到印表机。

程式 13-6 所示的 PRINT2 程式在 PRINT1 中增加了一个放弃程序和必要的支援——呼叫 AbortProc 函式并呼叫 EnableWindow 两次（第一次阻止视窗接受输入讯息，第二次启用视窗）。

程式 13-6　PRINT2

```
PRINT2.C
/*--------------------------------------------------------------------------
      PRINT2.C -- Printing with Abort Procedure
                                                (c) Charles Petzold, 1998
--------------------------------------------------------------------------*/

#include <windows.h>
HDC   GetPrinterDC (void) ;                   // in GETPRNDC.C
void PageGDICalls (HDC, int, int) ;           // in PRINT.C

HINSTANCE hInst ;
TCHAR            szAppName[] = TEXT ("Print2") ;
TCHAR            szCaption[] = TEXT ("Print Program 2 (Abort Procedure)") ;

BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
            MSG msg ;
            while (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
      {
                        TranslateMessage (&msg) ;
                        DispatchMessage (&msg) ;
      }
      return TRUE ;
}

BOOL PrintMyPage (HWND hwnd)
{
      static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print2: Printing") } ;
      BOOL                              bSuccess = TRUE ;
      HDC                               hdcPrn ;
      short                             xPage, yPage ;

      if (NULL == (hdcPrn = GetPrinterDC ()))
```

```
                    return FALSE ;
     xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
     yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

     EnableWindow (hwnd, FALSE) ;
     SetAbortProc (hdcPrn, AbortProc) ;
     if (StartDoc (hdcPrn, &di) > 0)
     {
                    if (StartPage (hdcPrn) > 0)
                    {
                              PageGDICalls (hdcPrn, xPage, yPage) ;
                              if (EndPage (hdcPrn) > 0)
                                        EndDoc (hdcPrn) ;
                              else
                                        bSuccess = FALSE ;
                    }
     }
     else
                    bSuccess = FALSE ;
     EnableWindow (hwnd, TRUE) ;
     DeleteDC (hdcPrn) ;
     return bSuccess ;
}
```

增加列印对话方块

PRINT2 还不能令人十分满意。首先，这个程式没有直接指示出何时开始列印和何时结束列印。只有将滑鼠指向程式并且发现它没有反应时，才能断定它仍然在处理 PrintMyPage 常式。PRINT2 在进行背景处理时也没有给使用者提供取消列印作业的机会。

您可能注意到，大多数 Windows 程式都为使用者提供了一个取消目前正在进行列印操作的机会。一个小的对话方块出现在萤幕上，它包括一些文字和「Cancel」按键。在 GDI 将列印输出储存到磁片档案或（如果停用列印伫列程式）印表机正在列印的整个期间，程式都显示这个对话方块。它是一个非系统模态对话方块，您必须提供对话程序。

通常称这个对话方块为「放弃对话方块」，称这种对话程序为「放弃对话程序」。为了更清楚地把它和「放弃程序」区别开来，我们称这种对话程序为「列印对话程序」。放弃程序（名为 AbortProc）和列印对话程序（将命名为 PrintDlgProc）是两个不同的输出函式。如果想以一种专业的 Windows 式列印方式进行列印工作，就必须拥有这两个函式。

这两个函式的交互作用方式如下：AbortProc 中的 PeekMessage 回圈得被修改，以便将非系统模态对话方块的讯息发送给对话方块视窗讯息处理程式。

PrintDlgProc 必须处理 WM_COMMAND 讯息，以检查「Cancel」按钮的状态。如果「Cancel」钮被按下，就将一个叫做 bUserAbort 的整体变数设为 TRUE。AbortProc 传回的值正好和 bUserAbort 相反。您可能还记得，如果 AbortProc 传回 TRUE 会继续列印，传回 FALSE 则放弃列印。在 PRINT2 中，我们总是传回 TRUE。现在，使用者在列印对话方块中按下「Cancel」按钮时将传回 FALSE。程式 13-7 所示的 PRINT3 程式实作了这个处理方式。

程式 13-7　PRINT3

```
PRINT3.C
/*------------------------------------------------------------------
      PRINT3.C -- Printing with Dialog Box
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------*/

#include <windows.h>
HDC   GetPrinterDC (void) ;                    // in GETPRNDC.C
void  PageGDICalls (HDC, int, int) ;           // in PRINT.C

HINSTANCE hInst ;
TCHAR            szAppName[] = TEXT ("Print3") ;
TCHAR            szCaption[] = TEXT ("Print Program 3 (Dialog Box)") ;

BOOL bUserAbort ;
HWND hDlgPrint ;

BOOL CALLBACK PrintDlgProc (HWND hDlg, UINT message,
                        WPARAM wParam, LPARAM lParam)
{
      switch (message)
      {
      case  WM_INITDIALOG:
                        SetWindowText (hDlg, szAppName) ;
                        EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE,
MF_GRAYED) ;
                        return TRUE ;

      case  WM_COMMAND:
                        bUserAbort = TRUE ;
                        EnableWindow (GetParent (hDlg), TRUE) ;
                        DestroyWindow (hDlg) ;
                        hDlgPrint = NULL ;
                        return TRUE ;
      }
      return FALSE ;
}
```

```
BOOL CALLBACK AbortProc (HDC hdcPrn, int iCode)
{
      MSG msg ;
      while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
      {
                  if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
                  {
                              TranslateMessage (&msg) ;
                              DispatchMessage (&msg) ;
                  }
      }
      return !bUserAbort ;
}

BOOL PrintMyPage (HWND hwnd)
{
      static DOCINFO di = { sizeof (DOCINFO), TEXT ("Print3: Printing") } ;
      BOOL                        bSuccess = TRUE ;
      HDC                         hdcPrn ;
      int                         xPage, yPage ;

      if (NULL == (hdcPrn = GetPrinterDC ()))
                        return FALSE ;
      xPage = GetDeviceCaps (hdcPrn, HORZRES) ;
      yPage = GetDeviceCaps (hdcPrn, VERTRES) ;

      EnableWindow (hwnd, FALSE) ;
      bUserAbort = FALSE ;
      hDlgPrint = CreateDialog (hInst, TEXT ("PrintDlgBox"),
                                                            hwnd,
PrintDlgProc) ;
      SetAbortProc (hdcPrn, AbortProc) ;
      if (StartDoc (hdcPrn, &di) > 0)
      {
                  if (StartPage (hdcPrn) > 0)
                  {
                              PageGDICalls (hdcPrn, xPage, yPage) ;
                                    if (EndPage (hdcPrn) > 0)
                                                EndDoc (hdcPrn) ;
                              else
                                    bSuccess = FALSE ;
                  }
      }
      else
                              bSuccess = FALSE ;
      if (!bUserAbort)
     {
                        EnableWindow (hwnd, TRUE) ;
```

```
                        DestroyWindow (hDlgPrint) ;
     }

       DeleteDC (hdcPrn) ;
       return bSuccess && !bUserAbort ;
}
```

PRINT.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Dialog
PRINTDLGBOX DIALOG DISCARDABLE  20, 20, 186, 63
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
       PUSHBUTTON                    "Cancel",IDCANCEL,67,42,50,14
       CTEXT                                    "Cancel
Printing",IDC_STATIC,7,21,172,8
END
```

如果您使用 PRINT3，那么最好临时暂停使用幕後列印；否则，只有在列印伫列程式从 PRINT3 中接收资料时才可见到的「Cancel」按钮可能会很快消失，让您根本没有机会去按它。如果您按「Cancel」按钮时列印并不立即终止（特别是在一个慢速印表机上），不要惊讶。印表机有一个内部缓冲区，在印表机停止之前其中的资料必须全部送出，按「Cancel」只是告诉 GDI 不要向印表机的缓冲区发送更多的资料而已。

PRINT3 增加了两个整体变数：一个是叫做 bUserAbort 的布林变数，另一个是叫做 hDlgPrint 的对话方块视窗代号。PrintMyPage 函式将 bUserAbort 初始化为 FALSE。与 PRINT2 一样，程式的主视窗是不接收输入讯息的。指向 AbortProc 的指标用於 SetAbortProc 呼叫中，而指向 PrintDlgProc 的指标用於 CreateDialog 呼叫中。CreateDialog 传回的视窗代号储存在 hDlgPrint 中。

现在，AbortProc 中的讯息回圈如下：

```
while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
{
       if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
       {
                  TranslateMessage (&msg) ;
                  DispatchMessage (&msg) ;
       }
}
return !bUserAbort ;
```

只有在 bUserAbort 为 FALSE，也就是使用者还没有终止列印工作时，这段程式码才会呼叫 PeekMessage。IsDialogMessage 函式用来将讯息发送给非系统模态对话方块。和普通的非系统模态对话方块一样，对话方块视窗的代号在这个呼叫之前受到检查。AbortProc 的传回值正好与 bUserAbort 相反。开始时，bUserAbort 为 FALSE，因此 AbortProc 传回 TRUE，表示继续进行列印；但是 bUserAbort 可能在列印对话程序中被设定为 TRUE。

PrintDlgProc 函式是相当简单的。处理 WM_INITDIALOG 时，该函式将视窗标题设定为程式名称，并且停用系统功能表上的「Close」选项。如果使用者按下了「Cancel」钮，PrintDlgProc 将收到 WM_COMMAND 讯息：

```
case  WM_COMMAND :
      bUserAbort = TRUE ;
      EnableWindow (GetParent (hDlg), TRUE) ;
      DestroyWindow (hDlg) ;
      hDlgPrint = NULL ;
      return TRUE ;
```

将 bUserAbort 设定为 TRUE，则说明使用者已经决定取消列印操作，主视窗被启动，而对话方块被清除（按顺序完成这两项活动是很重要的，否则，在 Windows 中执行其他程式之一将变成活动程式，而您的程式将消失到背景中）。与通常的情况一样，将 hDlgPrint 设定为 NULL，防止在讯息回圈中呼叫 IsDialogMessage。

只有在 AbortProc 用 PeekMessage 找到讯息，并用 IsDialogMessage 将它们传送给对话方块视窗讯息处理程式时，这个对话方块才接收讯息。只有在 GDI 模组处理 EndPage 函式时，才呼叫 AbortProc。如果 GDI 发现 AbortProc 的传回值是 FALSE，它将控制权从 EndPage 传回到 PrintMyPage。它不传回错误码。至此，PrintMyPage 认为列印页已经发完了，并呼叫 EndDoc 函式。但是，由於 GDI 模组还没有完成对 EndPage 呼叫的处理，所以不会列印出什么东西来。

有些清除工作尚待完成。如果使用者没在对话方块中取消列印作业，那么对话方块仍然会显示著。PrintMyPage 重新启用它的主视窗并清除对话方块：

```
if (!bUserAbort)
{
      EnableWindow (hwnd, TRUE) ;
      DestroyWindow (hDlgPrint) ;
}
```

两个变数会通知您发生了什么事：bUserAbort 可以告诉您使用者是否终止了列印作业，bSuccess 会告诉您是否出了故障，您可以用这些变数来完成想做的工作。PrintMyPage 只简单地对它们进行逻辑上的 AND 运算，然後把值传回给 WndProc：

```
    return bSuccess && !bUserAbort ;
```

为 POPPAD 增加列印功能

现在准备在 POPPAD 程式中增加列印功能，并且宣布 POPPAD 已告完毕。这需要第十一章中的各个 POPPAD 档案，此外，还需要程式 13-8 中的 POPPRNT.C 档案。

程式 13-8　POPPRNT

```
POPPRNT.C
/*---------------------------------------------------------------------
      POPPRNT.C -- Popup Editor Printing Functions
---------------------------------------------------------------------*/

#include <windows.h>
#include <commdlg.h>
#include "resource.h"

BOOL bUserAbort ;
HWND hDlgPrint ;

BOOL CALLBACK PrintDlgProc ( HWND hDlg, UINT msg, WPARAM wParam,LPARAM lParam)
{
      switch (msg)
      {
      case  WM_INITDIALOG :
                        EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE,
MF_GRAYED) ;
                        return TRUE ;

      case  WM_COMMAND :
                        bUserAbort = TRUE ;
                        EnableWindow (GetParent (hDlg), TRUE) ;
                        DestroyWindow (hDlg) ;
                        hDlgPrint = NULL ;
                        return TRUE ;
      }
      return FALSE ;
}

BOOL CALLBACK AbortProc (HDC hPrinterDC, int iCode)
{
      MSG msg ;
      while (!bUserAbort && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
      {
                  if (!hDlgPrint || !IsDialogMessage (hDlgPrint, &msg))
                  {
```

```
                              TranslateMessage (&msg) ;
                              DispatchMessage (&msg) ;
                    }
          }
          return !bUserAbort ;
}

BOOL PopPrntPrintFile (HINSTANCE hInst, HWND hwnd, HWND hwndEdit,
                                                                  PTSTR
szTitleName)
{
          static DOCINFO   di = { sizeof (DOCINFO) } ;
          static PRINTDLG  pd ;
          BOOL             bSuccess ;
          int              yChar, iCharsPerLine, iLinesPerPage, iTotalLines,
                              iTotalPages, iPage, iLine, iLineNum ;
          PTSTR            pstrBuffer ;
          TCHAR            szJobName [64 + MAX_PATH] ;
          TEXTMETRIC       tm ;
          WORD             iColCopy, iNoiColCopy ;

                              // Invoke Print common dialog box

          pd.lStructSize         =     sizeof (PRINTDLG) ;
          pd.hwndOwner           =     hwnd ;
          pd.hDevMode            =     NULL ;
          pd.hDevNames           =     NULL ;
          pd.hDC                 =     NULL ;
          pd.Flags               =     PD_ALLPAGES | PD_COLLATE |
                                        PD_RETURNDC | PD_NOSELECTION ;
          pd.nFromPage           =     0 ;
          pd.nToPage             =     0 ;
          pd.nMinPage            =     0 ;
          pd.nMaxPage            =     0 ;
          pd.nCopies             =     1 ;
          pd.hInstance           =     NULL ;
          pd.lCustData           =     0L ;
          pd.lpfnPrintHook       =     NULL ;
          pd.lpfnSetupHook       =     NULL ;
          pd.lpPrintTemplateName =     NULL ;
          pd.lpSetupTemplateName =     NULL ;
          pd.hPrintTemplate      =     NULL ;
          pd.hSetupTemplate      =     NULL ;

          if    (!PrintDlg (&pd))
                              return TRUE ;

          if    (0 == (iTotalLines = SendMessage (hwndEdit, EM_GETLINECOUNT, 0, 0)))
```

```
                    return TRUE ;

                         // Calculate necessary metrics for file

     GetTextMetrics (pd.hDC, &tm) ;
     yChar = tm.tmHeight + tm.tmExternalLeading ;

     iCharsPerLine = GetDeviceCaps (pd.hDC, HORZRES) / tm.tmAveCharWidth ;
     iLinesPerPage = GetDeviceCaps (pd.hDC, VERTRES) / yChar ;
     iTotalPages      = (iTotalLines + iLinesPerPage - 1) / iLinesPerPage ;

                         // Allocate a buffer for each line of text

     pstrBuffer  = malloc (sizeof (TCHAR) * (iCharsPerLine + 1)) ;

                         // Display the printing dialog box

     EnableWindow (hwnd, FALSE) ;
     bSuccess            = TRUE ;
     bUserAbort          = FALSE ;

     hDlgPrint           = CreateDialog (hInst, TEXT ("PrintDlgBox"),
                                                          hwnd,
PrintDlgProc) ;

     SetDlgItemText (hDlgPrint, IDC_FILENAME, szTitleName) ;
     SetAbortProc (pd.hDC, AbortProc) ;

                         // Start the document
     GetWindowText (hwnd, szJobName, sizeof (szJobName)) ;
     di.lpszDocName = szJobName ;
     if (StartDoc (pd.hDC, &di) > 0)
     {
                              //  Collation  requires  this  loop  and
iNoiColCopy
               for (iColCopy = 0 ;
                              iColCopy < ((WORD) pd.Flags & PD_COLLATE ?
pd.nCopies : 1) ;
                              iColCopy++)
               {
                         for (iPage = 0 ; iPage < iTotalPages ; iPage++)
                         {
            for (iNoiColCopy = 0 ;
                iNoiColCopy < (pd.Flags & PD_COLLATE ? 1 : pd.nCopies);
                                                  iNoiColCopy++)
                              {
                                                       // Start the page
                                             if (StartPage (pd.hDC) < 0)
```

```
                                                {
                                                                    bSuccess     =
FALSE ;
                                                                    break ;
                                                }

             // For each page, print the lines
            for (iLine = 0 ; iLine < iLinesPerPage ; iLine++)
                                                {
             iLineNum = iLinesPerPage * iPage + iLine ;
             if (iLineNum > iTotalLines)
                                                            break ;
             *(int *) pstrBuffer = iCharsPerLine ;
             TextOut   (pd.hDC, 0, yChar * iLine, pstrBuffer,
             (int) SendMessage (hwndEdit, EM_GETLINE,
                (WPARAM) iLineNum, (LPARAM) pstrBuffer));
                                                }

              if (EndPage (pd.hDC) < 0)
                                                {
               bSuccess = FALSE ;
               break ;
                                                }

               if (bUserAbort)
               break ;
                                                }

               if (!bSuccess || bUserAbort)
               break ;
                                    }

                  if (!bSuccess || bUserAbort)
               break ;
                        }
     }
     else
                        bSuccess = FALSE ;
     if     (bSuccess)
                        EndDoc (pd.hDC) ;

     if     (!bUserAbort)
    {
                        EnableWindow (hwnd, TRUE) ;
                        DestroyWindow (hDlgPrint) ;
     }

     free (pstrBuffer) ;
```

```
     DeleteDC (pd.hDC) ;

     return bSuccess && !bUserAbort ;
}
```

与 POPPAD 尽量利用 Windows 高阶功能来简化程式的方针一致，POPPRNT.C 档案展示了使用 PrintDlg 函式的方法。这个函式包含在通用对话方块程式库（common dialog box library）中，使用一个 PRINTDLG 型态的结构。

通常，程式的「File」功能表中有个「Print」选项。当使用者选中「Print」选项时，程式可以初始化 PRINTDLG 结构的栏位，并呼叫 PrintDlg。

PrintDlg 显示一个对话方块，它允许使用者选择列印页的范围。因此，这个对话方块特别适用於像 POPPAD 这样能列印多页文件的程式。这种对话方块同时也给出了一个确定副本份数的编辑区和名为「Collate（逐份列印）」的核取方块。「逐份列印」影响著多个副本页的顺序。例如，如果文件是 3 页，使用者要求列印三份副本，则这个程式能以两种顺序之一列印它们。选择逐份列印後的副本的页码顺序为 1、2、3、1、2、3、1、2、3，未选择逐份列印的副本的页码顺序是 1、1、1、2、2、2、3、3、3。程式在这里应负起的责任就是以正确的顺序列印副本。

这个对话方块也允许使用者选择非内定印表机，它包括一个标记为「Properties」的按钮，可以启动设备模式对话方块。这样，至少允许使用者选择直印或横印。

从 PrintDlg 函式传回後，PRINTDLG 结构的栏位指明列印页的范围和是否对多个副本进行逐份列印。这个结构同时也给出了准备使用的印表机装置内容代号。

在 POPPRNT.C 中，PopPrntPrintFile 函式（当使用者在「File」功能表里选中「Print」选项时，它由 POPPAD 呼叫）呼叫 PrintDlg，然後开始列印档案。PopPrntPrintFile 完成某些计算，以确定一行能容纳多少字元和一页能容纳多少行。这个程序涉及到呼叫 GetDeviceCaps 来确定页的解析度，呼叫 GetTextMetrics 来确定字元的大小。

这个程式通过发送一条 EM_GETLINECOUNT 讯息给编辑控制项来取得文件中的总行数（在变数 iTotalLines 中）。储存各行内容的缓冲区配置在局部记忆体中。对每一行，缓冲区的第一个字被设定为该行中字元的数目。把 EM_GETLINE 讯息发送给编辑控制项可以把一行复制到缓冲区中，然後用 TextOut 把这一行送到印表机装置内容中（POPPRNT.C 还没有聪明到对超出列印宽度的文字换到下一行去处理。在第十七章我们会讨论这种文字绕行的技术）。

为了确定副本份数，应注意列印文字的处理方式包括两个 for 回圈。第一

个 for 回圈使用了一个叫作 iColCopy 的变数，当使用者指定将副本逐份列印时，它将会起作用。第二个 for 回圈使用了一个叫作 iNonColCopy 的变数，当不对副本进行逐份列印时，它将起作用。

如果 StartPage 或 EndPage 传回一个错误，或者如果 bUserAbort 为 TRUE，那么这个程式退出增加页号的那个 for 回圈。如果放弃程序的传回值是 FALSE，则 EndPage 不传回错误。正是由於这个原因，在下一页开始之前，要直接测试 bUserAbort。如果没有报告错误，则进行 EndDoc 呼叫：

```
if (!bError)
      EndDoc (hdcPrn) ;
```

您可能想通过列印多页档案来测试 POPPAD。您可以从列印任务视窗中监视列印进展情况。在 GDI 处理完第一个 EndPage 呼叫之後，首先列印的档案将显示在列印任务视窗中。此时，幕後列印程式开始把档案发送到印表机。然後，如果在 POPPAD 中取消列印作业，那么幕後列印程式将终止列印，这也就是放弃程序传回 FALSE 的结果。当档案出现在列印任务视窗中，您也可以透过从「Document」功能表中选择「Cancel Printing」来取消列印作业，在这种情况下， POPPAD 中的 EndPage 呼叫会传回一个错误。

Windows 的程式设计的新手经常会抱住 AbortDoc 函式不放，但实际上这个函式几乎不在列印中使用。像在 POPPAD 中看到的那样，使用者几乎随时可以取消列印作业，或者通过 POPPAD 的列印对话方块及通过列印任务视窗。这两种方法都不需要程式使用 AbortDoc 函式。 POPPAD 中允许 AbortDoc 的唯一时刻是在对 StartDoc 的呼叫和对 EndPage 的第一个呼叫之间，但是程式很快就会执行过去，以至不再需要 AbortDoc。

图 13-3 显示出正确列印多页文件之列印函式的呼叫顺序。检查 bUserAbort 的值是否为 TRUE 的最佳位置是在每个 EndPage 函式之後。只有当对先前的列印函式的呼叫没有产生错误时，才使用 EndDoc 函式。实际上，如果任何一个列印函式的呼叫出现错误，那么表演就结束了，同时您也可以回家了。

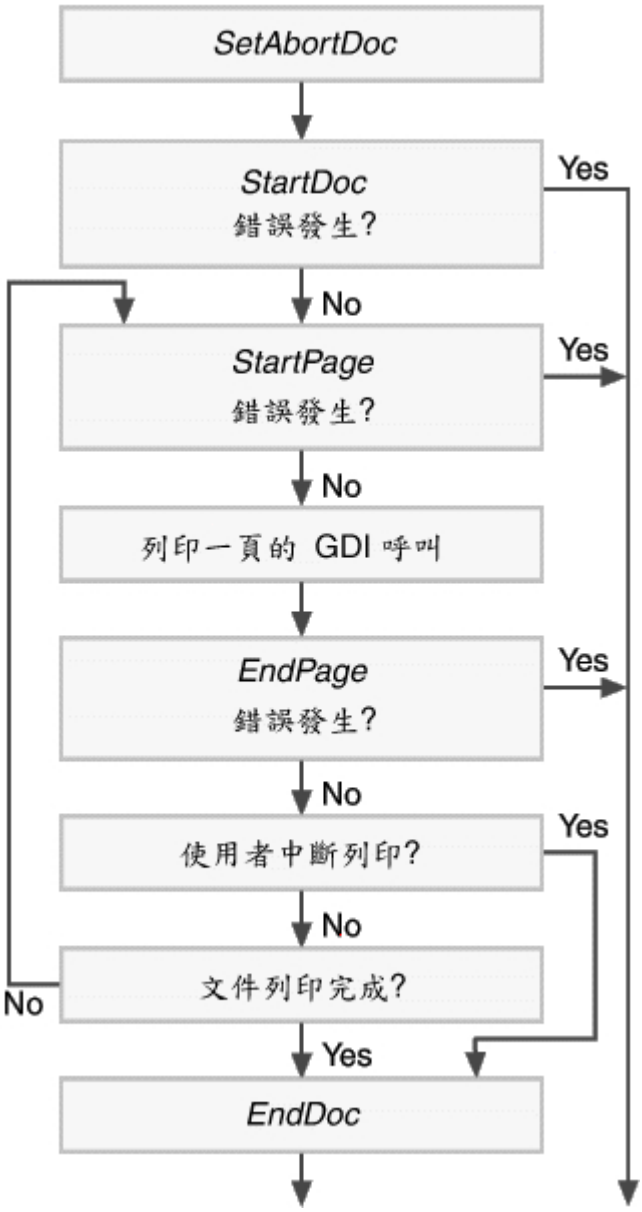

图 13-3 列印一个文件时的函式呼叫顺序