Designed for Microsoft® Windows NT® Windows 98

CD-ROM Included

Microsoft® 程序设计系列

完全修订和更新！

Windows® 程序设计

（第5版）下册

Win32 API 的权威指导

北京大学出版社

Microsoft Press

Windows 程序设计

（第 5 版）

下　册

[美]Charles Petzold　著

北京博彦科技发展有限公司　译

北京大学出版社

·北　京·

第二部分　其他图形知识

第十三章

使用打印机

当使用视频显示器显示文本和图形时，设备无关的概念非常完美，但对于打印机，设备无关的概念又怎样呢？

总的说来，效果也很好。在 Windows 程序中，您可以使用同样的用于视频显示的 GDI 函数，在打印纸上打印文本和图形，在以前讨论的与设备无关的许多问题（多数都与显示表面的尺寸和分辨率，以及颜色容量有关）都可以用相同的方法解决。当然，一台打印机不像阴极射线管那么简单，它们使用的是打印纸。它们之间有一些比较大的差异。例如，我们从来不必考虑视频显示器没有与视频适配器连结好，或者出现“屏幕空间不够”的错误，但打印机脱机和缺纸则是经常会遇到的问题。

我们也不必担心视频适配器不能执行某些图形操作，更不用担心视频适配器能否处理图形，因为如果它不能处理图形，就根本不能使用 Windows。但有些打印机不能打印图形（尽管它们能在 Windows 环境中使用）；绘图仪尽管可以打印矢量图形，却存在与位图有关的实际问题。

以下是其他若干需要考虑的问题：

- 打印机比视频显示器慢。尽管我们没有机会将程序性能调整到最佳状态，却不必担心视频显示器刷新所需的时间。然而，没有人想在做其他工作前一直等打印机完成打印任务。
- 程序可以用新的输出覆盖原有的显示输出，以重新使用视频显示器表面。这对打印机是不可能的，打印机只能走完一整页纸，然后在新的一页纸上打印新的内容。
- 在视频显示器上，不同的应用程序都被窗口化。而对于打印机，不同应用程序的输出必须分成不同的文档或打印作业。

为了给 GDI 的其余部分添加打印机支持功能，Windows 提供了几个只用于打印机

的函数。这些打印机专用的函数(StartDoc、EndDoc、StartPage 和 EndPage)负责将打印机的输出组织到纸张页面上。一个程序调用常用的 GDI 函数在一张纸上"显示"文本和图形,与在屏幕上显示的方式一样。

在第十五、十七和十八章有打印位图、格式化的文本,以及元文件的其他信息。

13.1　打印基础

当您在 Windows 下使用打印机时,实际上启动了一个包含 GDI32 库模块、打印机驱动程序库模块(带.DRV 扩展名)、Windows 后台打印程序,以及起作用的其他一些模块。在为打印机编写程序前,让我们先看一看这个过程是如何进行的。

13.1.1　打印和后台处理

当应用程序要使用打印机时,它首先使用 CreateDC 或 PrintDlg 来获取指向打印机设备描述表的一个句柄,这就使得打印机设备驱动程序库模块被加载到内存(如果还没有加载到内存中的话),并自己进行初始化。然后,程序调用 StartDoc 函数,通知一个新文档开始了。StartDoc 函数是由 GDI 模块来处理的,GDI 模块调用打印机设备驱动程序中的 Control 函数告诉设备驱动程序准备进行打印。

打印一个文档的过程以 StartDoc 调用开始,以 EndDoc 调用结束。这两个调用对于在文档页面上书写文本或者绘制图形的 GDI 命令来说,其作用就像书挡一样。每页本身是这样来划清界限的:调用 StartPage 来开始一页,调用 EndPage 来结束该页。

例如,如果应用程序想在一页纸上画一个椭圆,它首先调用 StartDoc 开始打印任务,然后再调用 StartPage 通知这是新的一页,接着调用 Ellipse,就像在屏幕上画一个椭圆一样。GDI 模块通常将程序对打印机设备描述表做出的 GDI 调用存储在磁盘上的元文件中,该文件名以字符串 ~ EMF(代表"增强型元文件")开始,且以.TMP 为扩展名。然而,打印机驱动程序可能会跳过这一步。

当绘制第一页的 GDI 调用结束时,应用程序调用 EndPage。现在,真正的工作开始了。打印机驱动程序必须把存放在元文件中的各种绘图命令翻译成打印机输出数据。绘制一页图形所需的打印机的输出数据量可能非常大,特别是当打印机没有高级页面制作语言时更是如此。例如,一台每英寸 600 点且使用 8.5 英寸×11 英寸打印纸的激光打印机,如果要定义一个图形页,可能需要 4 兆以上字节的数据。

为此,打印机驱动程序经常使用一种被称作"打印分带"的技术将一页分成若干个被称为"带"的矩形。GDI 模块从打印机驱动程序获取每个带的大小,然后设置一个与当前要处理的带相等的剪裁区,并为从元文件获得的每个绘图函数调用打印机设备驱动程序的 Output 函数,这个过程叫做"将元文件输出到设备驱动程序"。对设备驱动程序定义的页面上的每个带,GDI 模块必须将整个元文件"输出到"设备驱动程序。这个

过程完成以后,就可以删除该元文件。

对每个带,设备驱动程序都会将这些绘图函数转换为在打印机上打印这些图形所需要的输出数据。这种输出数据的格式是特定于打印机的。对点阵打印机,它将是包括图形序列在内的一系列控制序列的集合(打印机驱动程序也能调用在 GDI 模块中的各种"助手"例程,用来协助这种输出的构造)。对于带有高级页面制作语言(如 PostScript)的激光打印机,打印机将用这种语言进行输出。

打印机驱动程序将打印输出的每个带传送到 GDI 模块,随后,GDI 模块将该打印输出存入另一个临时文件中,该临时文件名以字符串 ~SPL 开始,带有.TMP 扩展名。当处理好整个页之后,GDI 模块对后台打印程序进行一个进程间调用,通知它一个新的打印页已经准备好了。然后,应用程序就转向处理下一页。当应用程序处理完所有要打印的页后,它就调用 EndDoc 发出一个信号,表明打印作业已经完成。图 13-1 显示了应用程序、GDI 模块和打印机驱动程序的交互作用过程。

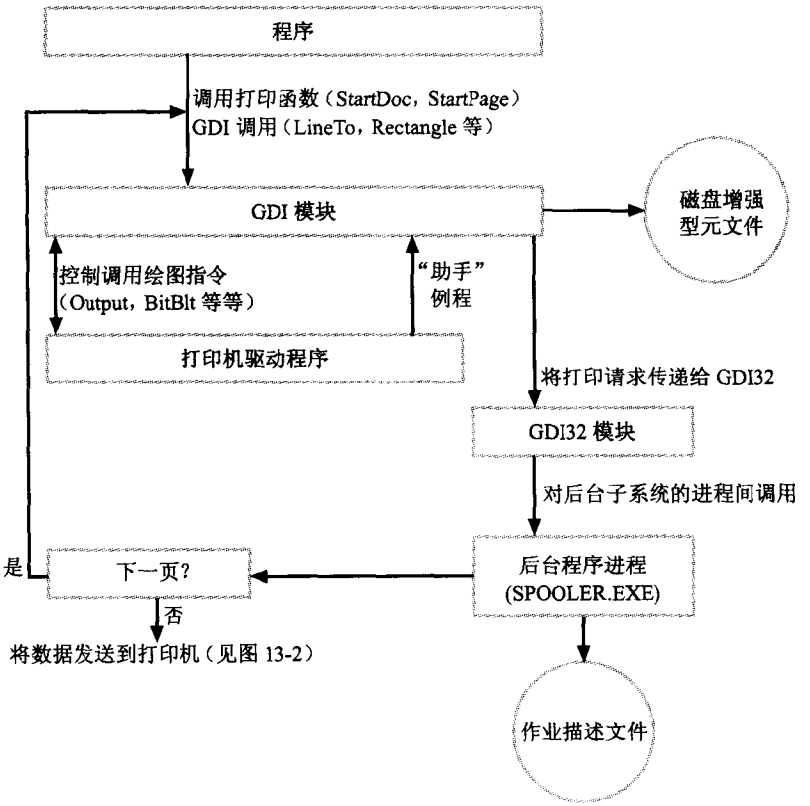

图 13-1 应用程序、GDI 模块、打印机驱动程序和后台程序的交互过程

Windows 后台打印程序实际上是几个部件的一种组合(见表 13-1)。

表 13-1

后台程序部件	说明
打印请求后台程序	将数据流传递给打印提供者
本地打印提供者	为本地打印机创建后台文件
网络打印提供者	为网络打印机创建后台文件
打印处理程序	将后台的设备无关数据转换为特定于目标打印机的格式
端口监视程序	控制连接打印机的端口
语言监视程序	控制可以双向通讯的打印机,设置设备配置并检测打印机状态

后台程序可以减轻应用程序的打印负担。Windows 在启动时就加载后台打印程序,因此,当应用程序开始打印时,它已经是活动的了。当程序打印一个文件时,GDI 模块创建包含打印输出数据的文件。后台打印程序的任务是将这些文件发往打印机。GDI 模块发出一个消息来通知它开始一个新的打印作业,然后它开始读文件,并将文件直接传送到打印机。为了传送这些文件,后台程序为打印机所连接的并口和串口使用各种通讯函数。在后台程序向打印机发送文件的操作结束后,它就删除包含输出数据的临时文件。这个过程如图 13-2 所示。

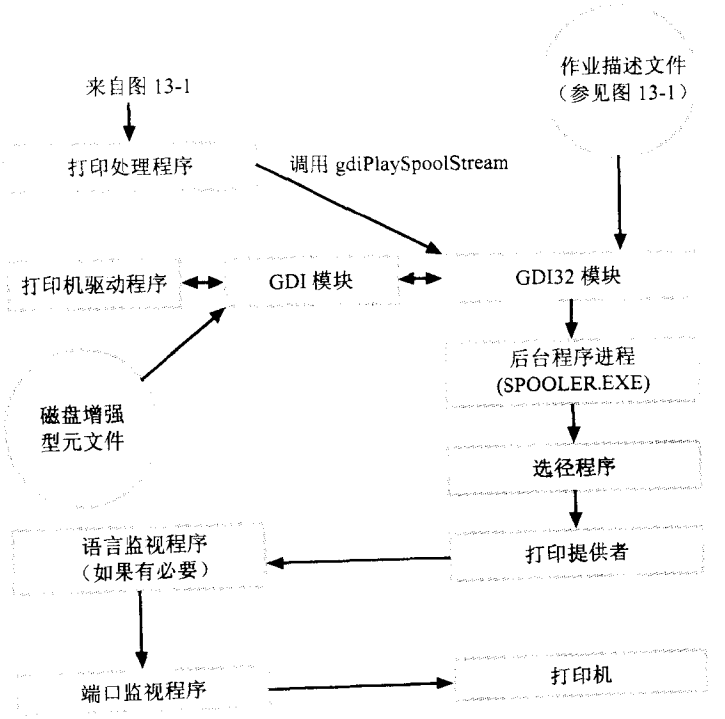

图 13-2　后台打印程序的操作过程

这个过程的大部分对应用程序是透明的。从应用程序角度来看,“打印”只发生在GDI模块将所有打印输出数据保存到磁盘文件中的时候,这以后(如果打印是由第二个线程来操作的,甚至可以在这之前)应用程序可以自由地进行其他操作。真正的文件打印操作成了后台打印程序的任务,而不是应用程序的任务。用户可以暂停打印作业、改变作业的优先级,或取消打印作业。这种管理方式使得应用程序可能比下面这种情况“打印”得更快,即作业以实时方式打印,且必须等到打印完一页后才能处理下一页。

我们已经描述了一般的打印原理,但还有一些例外情况。其中之一是Windows程序要使用打印机时,并非一定需要后台打印程序。用户可以在打印机属性表中关闭打印机的后台操作。

为什么有时用户不希望使用后台操作呢?因为用户可能使用了比Windows后台程序更快的硬件或软件后台打印程序,也可能是打印机在一个自身带有后台程序的网络上使用。一般的规律是,使用一个后台程序比使用两个后台程序更快。去掉Windows后台打印程序可以加快打印速度,因为打印输出数据不必保存在硬盘上,而可以直接输出到打印机,并被外部硬件或软件后台打印程序所接收。

如果没有启用Windows后台程序,GDI模块就不把来自设备驱动程序的打印输出数据存入文件中,而是将这些输出数据直接输出到并行或串行打印端口。与后台程序进行的打印不同,GDI进行的打印潜在地挂起应用程序(特别地,就是指正在进行打印的程序)的操作,直到打印完成。

还有一个例外。通常,GDI模块将定义一页所需的所有函数存入一个增强型元文件中,然后为由驱动程序定义的每个打印分带输出一遍该元文件到打印机驱动程序中。然而,如果打机印驱动程序不需要打印分带的话,就不会创建这个元文件;GDI只需简单地将绘图函数直接送往驱动程序。进一步的变化是,应用程序也可能承担起对打印输出数据进行打印分带的责任,这就使得应用程序中的打印代码更加复杂了,但却免去了GDI模块创建元文件的麻烦。这样,GDI只需简单地为每个带将函数传到打印机驱动程序。

或许您现在已经发现了从一个Windows应用程序进行打印操作要比使用视频显示器的开销大,这样可能出现一些问题——特别是当创建元文件或打印输出文件时,如果GDI模块耗尽了磁盘空间。您可以更多地关心这些问题,并尝试着处理这些问题,也可以置之不理。

应用程序进行打印的第一步是获取打印机设备描述表。

13.1.2 打印机设备描述表

正如在视频显示器上绘图前需要得到设备描述表句柄一样,在打印之前,用户必须获取一个打印机设备描述表句柄。一旦有了这个句柄(并为创建一个新文档调用StartDoc,

以及调用 StartPage 开始一页),就可以用与使用视频显示设备描述表句柄相同的方法来使用打印机设备描述表句柄,该句柄即为各种 GDI 调用的第一个参数。

很多应用程序通过调用 PrintDlg 函数打开一个标准的打印对话框。(本章后面会演示该函数的用法)。这个函数还给用户提供了一个在打印之前更改打印机或者指定其他作业特性的机会。然后,它将打印机设备描述表句柄赋予该应用程序。该函数能够保存应用程序的某些工作。然而,某些应用程序(例如记录本)只获得打印机设备描述表,而不需要对话框。要做到这一点,需要调用 CreateDC 函数。

在第五章,您知道如何通过如下调用来为整个视频显示器获取指向设备描述表的句柄:

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

您也可以使用该函数来获取打印机设备描述表句柄。然而,对于打印机设备描述表,CreateDC 的一般语法为

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

pInitializationData 参数一般被置为 NULL,szDeviceName 参数指向一个字符串,以告诉 Windows 打印机设备的名称。在设置设备名称之前,您必须知道有哪些打印机可用。

一个系统可以连接多台打印机,甚至可以有其他程序,如传真软件等,将自己伪装成打印机。不论连接的打印机有多少,都只能有一个被认为是"当前打印机"或者"默认打印机",这是用户最近一次选择的打印机。许多小的 Windows 程序只使用这个打印机来进行打印。

获取默认打印机设备描述表的方式在不断地变化。目前,标准的方法是使用 EnumPrinters 函数来获得。该函数填充一个包含每个所连打印机信息的数组结构。根据所需的细节层次,您还可以选择几种结构之一作为该函数的参数。这些结构的名称为 PRINTER_INFO_x,x 是一个数字。

遗憾的是,使用的结构还要依赖于您的程序是运行在 Windows 98 上,还是运行在 Windows NT 上。程序 13-1 表明 GetPrinterDC 函数在这两种操作系统上都能工作。

程序 13-1　GETPRNDC.C 文件

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 ;
}
```

这些函数使用 GetVersion 函数来确定程序是运行在 Windows 98 上，还是运行在 Windows NT 上。不管是什么操作系统，函数调用 EnumPrinters 两次：一次获取它所需结构的大小；一次填充结构。在 Windows 98 上，函数使用 PRINTER_INFO_5 结构；在 Windows NT 上，函数使用 PRINTER_INFO_4 结构。这些结构在 EnumPrinters 文档（/Platform SDK/Graphics and Multimedia Services/GDI/Printing and Print Spooler/Printing and Print Spooler Reference/Printing and Print Spooler Functions/EnumPrinters，Examples 小节的前面）中说明，它们是“容易而快速”的。

13.1.3 修改后的 DEVCAPS 程序

第五章的 DEVCAPS1 程序只显示了视频显示器的 GetDeviceCaps 函数的一些可用的基本信息。程序 13-2 所示的新版本显示了关于视频显示器和所有连接到系统的打印机的详细信息。

程序 13-2　DEVCAPS2 程序

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_SCREEN ? MF_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,
```