Programming Windows

Terry Marris Jan 2013

8 Printing

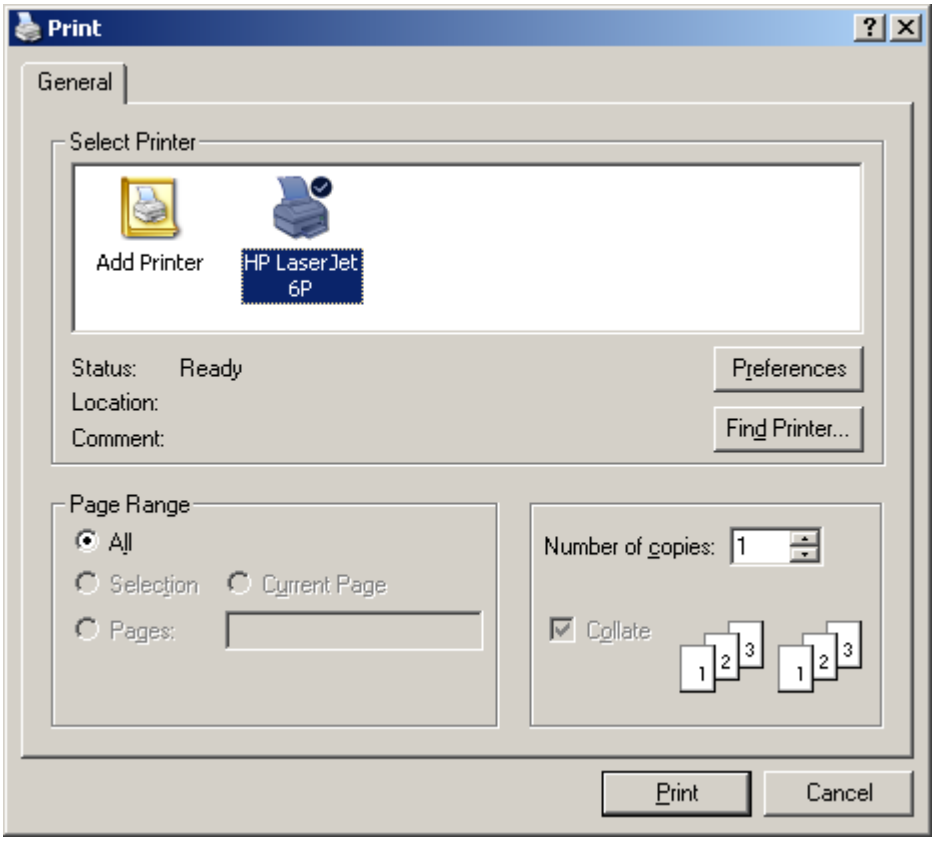
We continue looking at the *File* menu by considering the *Print* option.

In essence, to print we

- display a print dialog box
- set the printer font
- determine printer metrics
- print the document

8.1 Displaying a Print dialog Box

A print dialog box might look like this:



First, a print dialog structure, *PRNTDLG*, is initialised.

```
PRINTDLG initPrintDialogStruct(HWND hwnd)
{
   PRINTDLG pd;

   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_NOPAGENUMS | PD_HIDEPRINTTOFILE |
                            PD_DISABLEPRINTTOFILE;
   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;
   return pd;
}
```

PD_ALLPAGES indicates that *Page Range All* radio button is initially selected.

PD_COLLATE puts a tick in the *Collate* check box

PD_RETURNDC causes the *PrintDlg* function (see later) to return a device context that matches the user's selections. The device context is returned in the *hDC* member.

PD_NOSELECTION disables the *Selection* button.

PD_NOPAGENUMS disables the *Pages* radio button and causes the *Collate* check box to appear.

PD_HIDEPRINTTOFILE hides the *Print to File* check box.

PD_DISABLEPRINTTOFILE disables the *Print to File* check box.

The structure is used as the argument to *PrintDlg*.

```
BOOL PTEDFilePrint(HINSTANCE hInst, HWND hwnd,
                   HWND hwndEdit, LPSTR fileTitle)
{
   static PRINTDLG pd;
      ...
   pd = initPrintDialogStruct(hwnd);
   if (!PrintDlg(&pd))
      return TRUE;  /* error occurred or user cancelled */
```

The *PrintDlg* function displays a print dialog box. The function returns zero (*FALSE*) if an error occurred or if the user cancelled the operation, non-zero (*TRUE*) otherwise.

8.2 Setting the Printer Font

The print dialog member, *hDC*, is updated by the value returned by *setPrinterFont*.

```
   pd.hDC = setPrinterFont(pd.hDC);
```

Usually we would like the printer font to be the same as (or similar to) the edit screen font.

```
HDC setPrinterFont(HDC pdhDC)
{
   HDC printerDC = NULL;
   LOGFONT printerLogFont;
   int printerPointSize = 10;
   char *printerFontFaceName = "Lucida Console";
   HFONT printerFont;

   printerDC = pdhDC;
   ZeroMemory(&printerLogFont, sizeof(printerLogFont));
   printerLogFont.lfHeight = -MulDiv(printerPointSize,
                          GetDeviceCaps(printerDC, LOGPIXELSY), 72);
   strcpy(printerLogFont.lfFaceName, printerFontFaceName);
   printerFont = CreateFontIndirect(&printerLogFont);
   SelectObject(printerDC, printerFont);
   return printerDC;
}
```

As you can see, the code is almost identical to the code for setting an edit window font.

Yes. The font is hard-wired into the function. Perhaps, at a later date, we shall modify the function so that the user can choose the font.

8.3 Printer Metrics

Just like with an edit window, we get the printer text metrics

```
   TEXTMETRIC tm;
      ...
   GetTextMetrics(pd.hDC, &tm);
```

calculate the height of a character

```
   int yChar, ...
      ...
   yChar = tm.tmHeight + tm.tmExternalLeading;
```

then calculate the number of characters per line, the number of lines per page, and the total number of pages.

```
int ..., charsPerLine, linesPerPage, totalLines,
    totalPages, ...;
LPSTR buff;
   ...
charsPerLine = GetDeviceCaps(pd.hDC, HORZRES) / tm.tmAveCharWidth;
linesPerPage = GetDeviceCaps(pd.hDC, VERTRES) / yChar;
linesPerPage -= PMAR_LINES;
totalPages = (totalLines + linesPerPage - 1) / linesPerPage;

/* allocates buffer for each line of text */
buff = malloc(sizeof(char) * (charsPerLine + 1));
```

8.4 Printing the Document

The code for printing the document is unashamedly lifted from PETZOLD C Programming Windows 95 page 824.

The main window title is retrieved and stored as the document name.

```
static DOCINFO di = { sizeof(DOCINFO) };
   ...
GetWindowText(hwnd, job, sizeof(job));
di.lpszDocName = job;
```

StartDoc starts a print job. If the function fails, the return value is less than or equal to zero.

```
if (StartDoc(pd.hDC, &di) > 0) {
```

The logic for printing contains three *for..* loops. The outer loop deals with the case where the user has selected collated copies. The middle loop deals with pagination. The inner loop is exercised when the user has selected non-collated copies.

```
#define PMAR_LINES 8       /* Print Margins */
#define PMAR_PIXELS 450

BOOL isAborted;      /* GLOBAL VARIABLE */
...

WORD colCopy, noColCopy;  /* collated or notCollated */
                          /* WORD unsigned short int */
BOOL isOk;
int totalPages, page, line, lineNum;
   ...

   for (colCopy = 0;  /* colCopy - collated copies */
        colCopy < ((WORD)pd.Flags & PD_COLLATE? pd.nCopies : 1);
        colCopy++) {
      for (page = 0; page < totalPages; page++) {
         for (noColCopy = 0;  /* noColCopy - none collated copies */
              noColCopy < (pd.Flags & PD_COLLATE? 1 : pd.nCopies);
              noColCopy++) {

            /* starts the page */
            if (StartPage(pd.hDC) < 0) {
               isOk = FALSE;
               break;
            }
```

```
                /* prints the lines for each page */
                for (line = 0; line < linesPerPage; line++) {
                   lineNum = linesPerPage * page + line;

                   if (lineNum > totalLines)
                      break;

                   *(int *)buff = charsPerLine;

                   TextOut(pd.hDC, PMAR_PIXELS,
                           (yChar * line) + PMAR_PIXELS, buff,
                           (int)SendMessage(hwndEdit,
                           EM_GETLINE,
                           (WPARAM)lineNum, (LPARAM)buff));
                }

                if (EndPage(pd.hDC) < 0) {
                  isOk = FALSE;
                   break;
                }
                if (isAborted)
                   break;
             }
             if (!isOk || isAborted)
                break;
          }
          if (!isOk || isAborted)
             break;
       }
    }
    else
       isOk = FALSE;

    if (isOk)
       EndDoc(pd.hDC);

    if (!isAborted) {
       EnableWindow(hwnd, TRUE);
       DestroyWindow(hdlgPrint);
    }
```

8.5 File Print

Shown below is the entire *ptedprint.c* file.

```
/* ptedprint.c - file print */

#include <windows.h>
#include <commdlg.h>
#include "pted2.h"

#define PMAR_LINES 8
#define PMAR_PIXELS 450

BOOL isAborted;
HWND hdlgPrint;
```

```
BOOL CALLBACK PrintDlgProc(HWND hdlg, UINT msg,
                           WPARAM wParam, LPARAM lParam)
{
   HMENU systemMenu;
   HWND hwndParent;

   switch(msg) {
   case WM_INITDIALOG:
      systemMenu = GetSystemMenu(hdlg, FALSE);
      EnableMenuItem(systemMenu, SC_CLOSE, MF_GRAYED);
      return TRUE;

   case WM_COMMAND:
      isAborted = TRUE;
      hwndParent = GetParent(hdlg);
      EnableWindow(hwndParent, TRUE);
      DestroyWindow(hdlg);
      hdlgPrint = NULL;
      return TRUE;
   }
   return FALSE;
}


BOOL CALLBACK AbortProc(HDC printerDC, int code)
{
   MSG msg;

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


PRINTDLG initPrintDialogStruct(HWND hwnd)
{
   PRINTDLG pd;

   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_NOPAGENUMS | PD_HIDEPRINTTOFILE |
                            PD_DISABLEPRINTTOFILE;
   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;
   return pd;
}


HDC setPrinterFont(HDC pdhDC)
{
   HDC printerDC = NULL;
   LOGFONT printerLogFont;
   int printerPointSize = 10;
   char *printerFontFaceName = "Lucida Console";
   HFONT printerFont;

   printerDC = pdhDC;
   ZeroMemory(&printerLogFont, sizeof(printerLogFont));
   printerLogFont.lfHeight = -MulDiv(printerPointSize,
                           GetDeviceCaps(printerDC, LOGPIXELSY), 72);
   strcpy(printerLogFont.lfFaceName, printerFontFaceName);
   printerFont = CreateFontIndirect(&printerLogFont);
   SelectObject(printerDC, printerFont);
   return printerDC;
}


BOOL PTEDFilePrint(HINSTANCE hInst, HWND hwnd,
                   HWND hwndEdit, LPSTR fileTitle)
{
   static DOCINFO di = { sizeof(DOCINFO) };
   static PRINTDLG pd;

   BOOL isOk;
   int yChar, charsPerLine, linesPerPage, totalLines,
       totalPages, page, line, lineNum;

   LPSTR buff;
   char job[64 + MAX_PATH];
   TEXTMETRIC tm;
   WORD colCopy, noColCopy;  /* collated or notCollated */
                             /* WORD unsigned short int */

   /* invokes common dialog box */
   pd = initPrintDialogStruct(hwnd);
   if (!PrintDlg(&pd))
      return TRUE;  /* error occurred or user cancelled */

   if (0 == (totalLines = SendMessage(hwndEdit,
                                      EM_GETLINECOUNT, 0, 0L)))
      return TRUE;  /* nothing to print */

   pd.hDC = setPrinterFont(pd.hDC);
```

```
/* calculates printer metrics */
GetTextMetrics(pd.hDC, &tm);
yChar = tm.tmHeight + tm.tmExternalLeading;
charsPerLine = GetDeviceCaps(pd.hDC, HORZRES) / tm.tmAveCharWidth;
linesPerPage = GetDeviceCaps(pd.hDC, VERTRES) / yChar;
linesPerPage -= PMAR_LINES;
totalPages = (totalLines + linesPerPage - 1) / linesPerPage;

/* allocates buffer for each line of text */
buff = malloc(sizeof(char) * (charsPerLine + 1));

/* displays printing dialog box */
EnableWindow(hwnd, FALSE);  /* FALSE disables mouse & kbd input */

isOk = TRUE;
isAborted = FALSE;

hdlgPrint = CreateDialog(hInst, "PrintDlgBox",
                         hwnd, PrintDlgProc);

SetDlgItemText(hdlgPrint, IDC_FILENAME, fileTitle);
SetAbortProc(pd.hDC, AbortProc);

/* starts printing the document */
GetWindowText(hwnd, job, sizeof(job));
di.lpszDocName = job;

if (StartDoc(pd.hDC, &di) > 0) {
   /* collation requires this loop and noColCopy */

   for (colCopy = 0;  /* colCopy - collated copies */
        colCopy < ((WORD)pd.Flags & PD_COLLATE? pd.nCopies : 1);
        colCopy++) {
      for (page = 0; page < totalPages; page++) {
         for (noColCopy = 0;  /* noColCopy - none collated copies */
              noColCopy < (pd.Flags & PD_COLLATE? 1 : pd.nCopies);
              noColCopy++) {

            /* starts the page */
            if (StartPage(pd.hDC) < 0) {
               isOk = FALSE;
               break;
            }

            /* prints the lines for each page */
            for (line = 0; line < linesPerPage; line++) {
               lineNum = linesPerPage * page + line;

               if (lineNum > totalLines)
                  break;

               *(int *)buff = charsPerLine;

               TextOut(pd.hDC, PMAR_PIXELS,
                       (yChar * line) + PMAR_PIXELS, buff,
                       (int)SendMessage(hwndEdit,
                       EM_GETLINE,
                       (WPARAM)lineNum, (LPARAM)buff));
            }
```

```
               if (EndPage(pd.hDC) < 0) {
                 isOk = FALSE;
                  break;
               }
               if (isAborted)
                  break;
            }
            if (!isOk || isAborted)
               break;
         }
         if (!isOk || isAborted)
            break;
      }
   }
   else
      isOk = FALSE;

   if (isOk)
      EndDoc(pd.hDC);

   if (!isAborted) {
      EnableWindow(hwnd, TRUE);
      DestroyWindow(hdlgPrint);
   }

   free(buff);
   buff = NULL;

   return isOk && !isAborted;
}
```

A limitation is that a line of text can go beyond the right hand page print margin and not word break or overflow onto the next line down. We might address this problem in a later chapter.

8.6 The AbortProc

Cancelling a print job requires an abort *CALLBACK* procedure. Such as procedure is shown below.

```
BOOL CALLBACK AbortProc(HDC printerDC, int code)
{
   MSG msg;

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

The abort procedure is registered before printing starts by a call to *SetAbortProc*.

```
BOOL PTEDFilePrint(HINSTANCE hInst, HWND hwnd,
                   HWND hwndEdit, LPSTR fileTitle)
{
      ...
   static PRINTDLG pd;
      ...
   pd = initPrintDialogStruct(hwnd);
         ...
   pd.hDC = setPrinterFont(pd.hDC);
      ...
   SetAbortProc(pd.hDC, AbortProc);
```

SetAbortProc is a Win32 API function that sets the application-defined abort function and allows a print job to be cancelled. The function returns *SP_ERROR* if it fails.

Before setting the abort procedure we disable the program's window to prevent keyboard and mouse messages entering the message queue and potentially interfering with the print process.

```
  EnableWindow(hwnd, FALSE);  /* FALSE disables mouse & kbd input */
```

When printing has finished we re-enable the program's window.

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

DispatchMessage must be called in case a *WM_PAINT* message gets into the queue and remains there because a *BeginPaint ... EndPaint* pair is not processed properly, thereby causing *PeekMessage* to never return *FALSE*.

PeekMessage despatches incoming sent messages, checks the message queue for a posted message, and retrieves the message - if any. *PeekMessage* has the format

PeekMessage(&message, hwnd, filterMin, filterMax, removeMessage)

where *removeMessage* values includes *PM_REMOVE* and *PM_NOREMOVE*. *PeekMessage* returns zero if no messages are available, and non-zero if a message is available.

hdlgPrint is defined as a global variable,

```
HWND hdlgPrint;
```

Its value is set by the value returned by *CreateDialog* in *PTEDFilePrint*

```
   hdlgPrint = CreateDialog(hInst, "PrintDlgBox",
                            hwnd, PrintDlgProc);
```

and it is used as an argument in both *SetDlgItemText* and *DestroyWindow*.

```
   SetDlgItemText(hdlgPrint, IDC_FILENAME, fileTitle);
      ...
   DestroyWindow(hdlgPrint);
```

IDC_FILENAME is *#define*d in *pted2.h*.

hdlgPrint is reset to *NULL* in *PrintDlgProc* shown below.

8.7 The PrintDlgProc

The *PrintDlgProc* receives a *WM_COMMAND* message if the user clicks the *Cancel* button. If it is pressed the global variable *isAborted* is set to *TRUE*. *FALSE* is returned to indicate that the print job has been cancelled.

```
BOOL CALLBACK PrintDlgProc(HWND hdlg, UINT msg,
                           WPARAM wParam, LPARAM lParam)
{
   HMENU systemMenu;
   HWND hwndParent;

   switch(msg) {
   case WM_INITDIALOG:
      systemMenu = GetSystemMenu(hdlg, FALSE);
      EnableMenuItem(systemMenu, SC_CLOSE, MF_GRAYED);
      return TRUE;

   case WM_COMMAND:
      isAborted = TRUE;
      hwndParent = GetParent(hdlg);
      EnableWindow(hwndParent, TRUE);
      DestroyWindow(hdlg);
      hdlgPrint = NULL;
      return TRUE;
   }
   return FALSE;
}
```

The function disables the *Close* option on the system menu, closes the dialog box and re-enables the main window.

Well, that is about as much as I am going to do with the *File* menu. Next, we look at implementing the *Edit* menu operations.