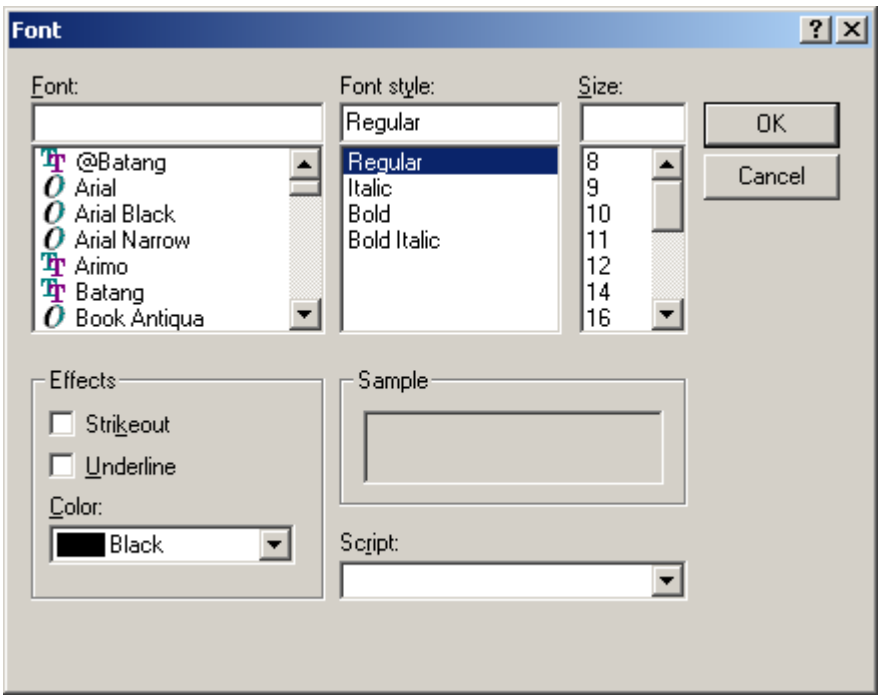
Programming Windows

Terry Marris Feb 2013

11 Choose Font

We see how to implement the *Font* option on the *Format* menu.

We pop up a regular *Font Dialog Box* for the user to select the required font.



The user's choice of font is reflected in the edit window text.

11.1 Choosing a Font

We initialise a *CHOOSEFONT* structure and call *ChooseFont*.

```
static LOGFONT logfont;
...
BOOL chooseFont(HWND hwnd)
{
   CHOOSEFONT cf ;

   cf.lStructSize    = sizeof(CHOOSEFONT);
   cf.hwndOwner      = hwnd;
   cf.hDC            = NULL;
   cf.lpLogFont      = &logfont;
   cf.iPointSize     = 0;
   cf.Flags          = CF_INITTOLOGFONTSTRUCT |
                       CF_SCREENFONTS | CF_EFFECTS;
   cf.rgbColors      = 0;
```

```
   cf.lCustData      = 0;
   cf.lpfnHook       = NULL;
   cf.lpTemplateName = NULL;
   cf.hInstance      = NULL;
   cf.lpszStyle      = NULL;
   cf.nFontType      = 0;  /* Returned from ChooseFont */
   cf.nSizeMin       = 0;
   cf.nSizeMax       = 0;

   return ChooseFont(&cf);
}
```

CF_INITLOGFONTSTRUCT says the *ChooseFont* API function should use the structure pointed to by the *lpLogFont* member to initialise the dialog box controls.

CF_SCREENFONTS causes the dialog box to list only the screen fonts supported by the system.

CF_EFFECTS allows the user to choose effects such as strikeout, underline and text colours.

ChooseFont creates the *Font Dialog Box*. *FALSE* is returned if the user cancels or closes the dialog box, or if an error occurs.

The chosen font is set with

```
void setFont(HWND hwndEdit)
{
   HFONT hFontNew;
   RECT  rect;

   hFontNew = CreateFontIndirect(&logfont);
   SendMessage(hwndEdit, WM_SETFONT, (WPARAM)hFontNew, 0);
   DeleteObject(hFont);
   hFont = hFontNew;
   GetClientRect(hwndEdit, &rect);
   InvalidateRect(hwndEdit, &rect, TRUE);
}
```

The edit window is redrawn with

```
   InvalidateRect(hwndEdit, &rect, TRUE);
```

The *TRUE* argument specifies that the background is to be re-drawn.

Here is the entire file.

```
/* ptedfont.c - implements choose font functions */

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

static LOGFONT logfont;
static HFONT hFont;
```

```
void setInitialFont(HWND hwndEdit)
{
   HFONT hFont;
   HDC hdc;
   LOGFONT logFont;
   int pointSize = 10;
   char *faceName = "Lucida Console";
   HWND hwnd;

   hwnd = GetParent(hwndEdit);
   hdc = GetDC(hwnd);
   ZeroMemory(&logFont, sizeof(logFont));
   logFont.lfHeight = -MulDiv(pointSize,
                              GetDeviceCaps(hdc, LOGPIXELSY), 72);
   strcpy(logFont.lfFaceName, faceName);
   hFont = CreateFontIndirect(&logFont);
   SendMessage(hwndEdit, WM_SETFONT, (WPARAM)hFont, MAKELPARAM(TRUE,0));
}


BOOL chooseFont(HWND hwnd)
{
   CHOOSEFONT cf ;

   cf.lStructSize    = sizeof(CHOOSEFONT);
   cf.hwndOwner      = hwnd;
   cf.hDC            = NULL;
   cf.lpLogFont      = &logfont;
   cf.iPointSize     = 0;
   cf.Flags          = CF_INITTOLOGFONTSTRUCT |
                       CF_SCREENFONTS | CF_EFFECTS;
   cf.rgbColors      = 0;
   cf.lCustData      = 0;
   cf.lpfnHook       = NULL;
   cf.lpTemplateName = NULL;
   cf.hInstance      = NULL;
   cf.lpszStyle      = NULL;
   cf.nFontType      = 0;  /* Returned from ChooseFont */
   cf.nSizeMin       = 0;
   cf.nSizeMax       = 0;

   return ChooseFont(&cf);
}


void setFont(HWND hwndEdit)
{
   HFONT hFontNew;
   RECT  rect;

   hFontNew = CreateFontIndirect(&logfont);
   SendMessage(hwndEdit, WM_SETFONT, (WPARAM)hFontNew, 0);
   DeleteObject(hFont);
   hFont = hFontNew;
   GetClientRect(hwndEdit, &rect);
   InvalidateRect(hwndEdit, &rect, TRUE);
}
```

The *setInitialFont* function has been moved from *WndProc*.

The *Font* menu choice is implemented with

```
case IDM_FORMAT_FONT:
   if (chooseFont(hwnd))
      setFont(hwndEdit) ;
   return 0;
```

One problem we have not addressed is setting the printer font based on the chosen edit window font. We probably need to revisit *PTEDPrint.c* and derive the printer font from the current edit screen font. Might get round to it one day. But next we look at how to word-wrap text so that is does not extend beyond the confines of its window.