Programming Windows

Terry Marris Jan 2013

7 Files

We take a look at implementing some file menu options: *New*, *Open*, *Save*, *Save As* and *Exit*.

7.1 Header File

The header file is largely unchanged from chapter 6, Menus. The report constants are now included, as is a constant for text unchanged and a list of file handling function prototypes.

```
#define REP_FATAL 1
#define REP_WARNING 2
#define REP_DIAGNOSTIC 3

#define EDT_TEXTNOTCHANGED 0

OPENFILENAME PTEDFileInit(HWND);
BOOL PTEDFileOpenDlg(HWND, LPSTR, LPSTR, OPENFILENAME *);
BOOL PTEDFileSaveDlg(HWND, LPSTR, LPSTR, OPENFILENAME *);
BOOL PTEDFileRead(HWND, LPSTR);
BOOL PTEDFileWrite(HWND, LPSTR);
```

Here is the complete header file, *pted1.h.*

```
/* pted1.h */

#ifndef PTED1_H
#define PTED1_H

#define IDE_EDITABLE 101
#define IDB_STATUS_BAR 102

#define IDM_FILE_NEW 200
#define IDM_FILE_OPEN 201
#define IDM_FILE_SAVE 202
#define IDM_FILE_SAVE_AS 203
#define IDM_FILE_PRINT 204
#define IDM_APP_EXIT 205

#define IDM_EDIT_UNDO 210
#define IDM_EDIT_CUT 211
#define IDM_EDIT_COPY 212
#define IDM_EDIT_PASTE 213
#define IDM_EDIT_CLEAR 214
#define IDM_EDIT_SELECT_ALL 215
```

```
#define IDM_SEARCH_FIND 220
#define IDM_SEARCH_NEXT 221
#define IDM_SEARCH_REPLACE 222

#define IDM_FORMAT_FONT 230

#define IDM_HELP 240
#define IDM_APP_ABOUT 241

#define REP_FATAL 1
#define REP_WARNING 2
#define REP_DIAGNOSTIC 3

#define EDT_TEXTNOTCHANGED 0

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK EditProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK AboutDlgProc(HWND, UINT, WPARAM, LPARAM);
int CALLBACK report(int, PSTR, PSTR, ...);

OPENFILENAME PTEDFileInit(HWND);
BOOL PTEDFileOpenDlg(HWND, LPSTR, LPSTR, OPENFILENAME *);
BOOL PTEDFileSaveDlg(HWND, LPSTR, LPSTR, OPENFILENAME *);
BOOL PTEDFileRead(HWND, LPSTR);
BOOL PTEDFileWrite(HWND, LPSTR);

#endif
```

7.2 Resource File

The resource file is unchanged from chapter 6, Menus.

```
/* ptedres.rc - primitive text editor resources */
#include <windows.h>
#include <afxres.h>
#include "pted1.h"

ABOUTBOX DIALOG 32, 28, 180, 75
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
FONT 8, "Tahoma"
CAPTION "About PTED"
{
   DEFPUSHBUTTON "OK", IDOK, 66, 48, 50, 14
   CTEXT "PTED", IDC_STATIC, 40, 8, 100, 8
   CTEXT "Primitive Text Editor", IDC_STATIC, 7, 20, 166, 8
   CTEXT "Terry Marris Jan 2013", IDC_STATIC, 7, 32, 166, 8
}
```

```
PTED MENU
{
   POPUP "File" {
      MENUITEM "New", IDM_FILE_NEW
      MENUITEM "Open", IDM_FILE_OPEN
      MENUITEM "Save", IDM_FILE_SAVE
      MENUITEM "Save As", IDM_FILE_SAVE_AS
      MENUITEM SEPARATOR
      MENUITEM "Print", IDM_FILE_PRINT
      MENUITEM "Exit", IDM_APP_EXIT
   }

   POPUP "Edit" {
      MENUITEM "Undo", IDM_EDIT_UNDO
      MENUITEM SEPARATOR
      MENUITEM "Cut", IDM_EDIT_CUT
      MENUITEM "Copy", IDM_EDIT_COPY
      MENUITEM "Paste", IDM_EDIT_PASTE
      MENUITEM "Delete", IDM_EDIT_CLEAR
      MENUITEM SEPARATOR
      MENUITEM "Select All", IDM_EDIT_SELECT_ALL
   }

   POPUP "Search" {
      MENUITEM "Find", IDM_SEARCH_FIND
      MENUITEM "Find Next", IDM_SEARCH_NEXT
      MENUITEM "Replace", IDM_SEARCH_REPLACE
   }

   POPUP "Format" {
      MENUITEM "Font", IDM_FORMAT_FONT
   }

   POPUP "Help" {
      MENUITEM "Help Topics", IDM_HELP
      MENUITEM "About PTED", IDM_APP_ABOUT
   }
}
```

7.3 File Operations

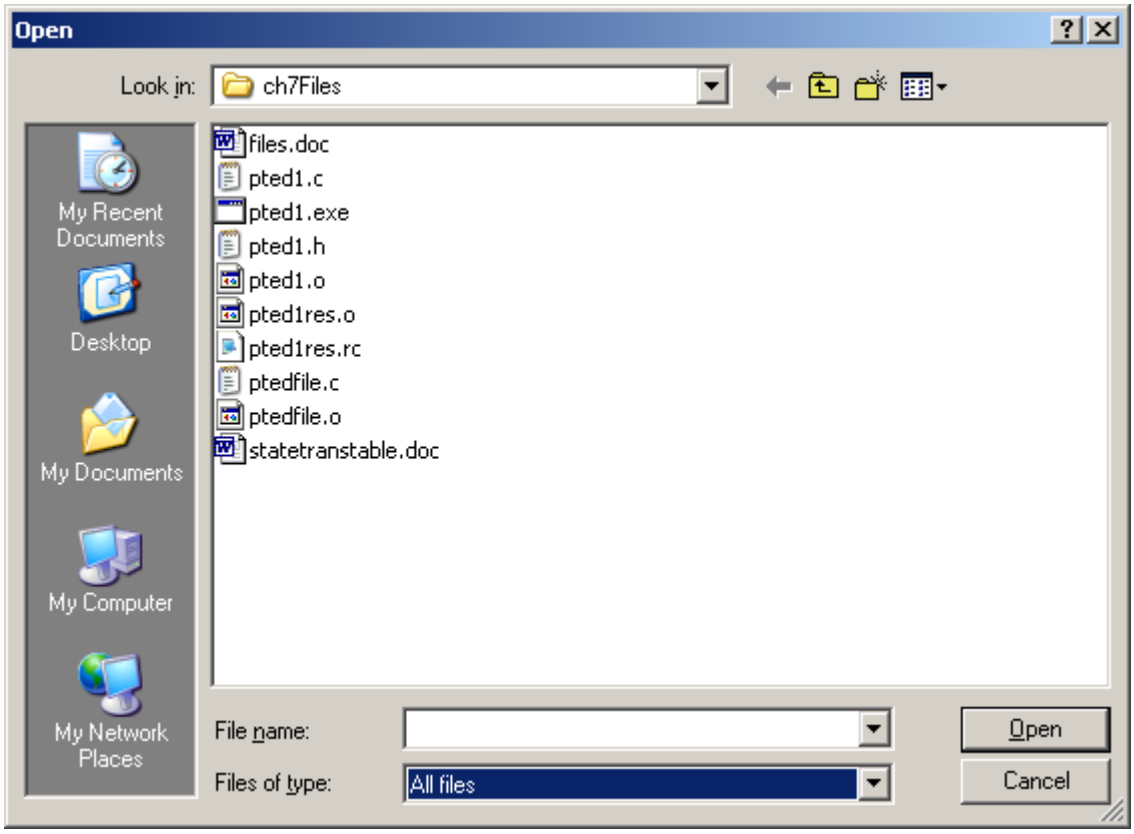
The Win32 API file handling functions such as *CreateFile*, *GetFileSize* and *ReadFile* are used.

OPENFILENAME is a structure used by the *GetOpenFileName* and *GetSaveFileName* to initialise the traditional (Windows 2000 and XP style) *Open* and *Save As* dialog boxes.

```
OPENFILENAME PTEDFileInit(HWND hwnd)
{
   static OPENFILENAME ofn;
   static char filter[] = "All files\0*.*\0\0";

   ofn.lStructSize       = sizeof(OPENFILENAME);
   ofn.hwndOwner         = hwnd;
   ofn.hInstance         = NULL;
   ofn.lpstrFilter       = filter;
   ofn.lpstrCustomFilter = NULL;
   ofn.nMaxCustFilter    = 0;
   ofn.nFilterIndex      = 0;
   ofn.lpstrFile         = NULL;     /* Set in Open and Save dialogs */
   ofn.nMaxFile          = MAX_PATH;
   ofn.lpstrFileTitle    = NULL;     /* Set in Open and Save dialogs */
   ofn.nMaxFileTitle     = MAX_PATH;
   ofn.lpstrInitialDir   = NULL;
   ofn.lpstrTitle        = NULL;
   ofn.Flags             = 0;       /* Set in Open and Save dialogs */
   ofn.nFileOffset       = 0;
   ofn.nFileExtension    = 0;
   ofn.lpstrDefExt       = NULL;
   ofn.lCustData         = 0L;
   ofn.lpfnHook          = NULL;
   ofn.lpTemplateName    = NULL;
   return ofn;
}
```

The *filter* represents the string displayed on the *File Types* combo box.



The *filter* is a set of pairs of strings. The first item in a pair is the descriptive text e.g. *Text Files*. The second item in a pair represents the filter pattern e.g. **.txt*. Here, the pair *All files\0*.** is used. Two NULL characters must terminate the filter string.

lpstrFile contains the drive, path, file name and extension of the selected file.

lpstrFileTitle contains just the file name with its extension.

flag contains a set of bit flags that initialises the *Save As* and *Open* dialog boxes. Values include

OFN_CREATEPROMPT - prompts the user to create the file if it does not exist
OFN_OVERWRITEPROMPT - asks if the user would like to replace an existing file

Some fields of the *OPENFILENAME* structure are updated by *PTEDFileOpenDlg* function shown below.

```
BOOL PTEDFileOpenDlg(HWND hwnd, LPSTR fileName, LPSTR fileTitle,
                     OPENFILENAME *ofn)
{
   ofn -> hwndOwner      = hwnd;
   ofn -> lpstrFile      = fileName;
   ofn -> lpstrFileTitle = fileTitle;
   ofn -> Flags          = OFN_HIDEREADONLY | OFN_CREATEPROMPT;
   return GetOpenFileName(ofn);
}
```

GetOpenFileName creates an *Open* file dialog box to obtain the required file from the user. It returns zero if the user cancels or closes the dialog box, or if an error occurs.

OPENFILENAME fields are also set by *PTEDFileSaveDlg*.

```
BOOL PTEDFileSaveDlg(HWND hwnd, LPSTR fileName, LPSTR fileTitle,
                     OPENFILENAME *ofn)
{
   ofn -> hwndOwner      = hwnd;
   ofn -> lpstrFile      = fileName;
   ofn -> lpstrFileTitle = fileTitle;
   ofn -> Flags          = OFN_OVERWRITEPROMPT;
   return GetSaveFileName(ofn);
}
```

GetSaveFileName creates a *Save* dialog box. The user specifies the name of the file to save, as well as its path.

PTEDFileRead reads the contents of a given file into memory. (Hope that memory is large enough!)

```
BOOL PTEDFileRead(HWND hwndEdit, LPSTR fileName)
{
   HANDLE file;
   int fileLength;
   LPSTR pBuffer;
   DWORD bytesRead;

   if (INVALID_HANDLE_VALUE ==
      (file = CreateFile(fileName, GENERIC_READ, FILE_SHARE_READ,
                         NULL, OPEN_EXISTING, 0, NULL)));
```

```
    fileLength = GetFileSize(file, NULL);
    pBuffer = (LPSTR)malloc(fileLength + 2);

    ReadFile(file, pBuffer, fileLength, &bytesRead, NULL);
    CloseHandle(file);
    pBuffer[fileLength] = '\0';
    pBuffer[fileLength + 1] = '\0';

    SetWindowText(hwndEdit, pBuffer);
    free(pBuffer);
    pBuffer = NULL;
    return TRUE;
}
```

CreateFile creates or opens a file, and returns a handle to the file. It returns *INVALID_HANDLE_VALUE* on failure. Its format is

CreateFile(fileName, access, share, security, creation, flags, template)

fileName is the name of the file to be created or opened. Its size is limited to *MAX_PATH* i.e. 256 characters.

Values for desired *access* include *GENERIC_READ* and *GENERIC_WRITE*.

Values for *share* mode include *FILE_SHARE_READ* and *FILE_SHARE_WRITE*. These allow subsequent requests for read or write access to the file.

NULL means there are no *security* attributes.

Creation types include:

- *CREATE_ALWAYS* always creates a new file. If a file with the given name already exists, it is overwritten.
- *CREATE_NEW* creates a new file but only if it does not already exist.
- *OPEN_ALWAYS* opens the given file. If it does not already exist, it is created.
- *OPEN_EXISTING* opens a file, but only if it already exists.
- *TRUNCATE_EXISTING* deletes the contents of the given file.

flags 0 is specified here. Apparently, the usual, most commonly used flag is *FILE_ATTRIBUTE_NORMAL.*

template NULL says no template is specified. *template* may be a handle to a template file with *GENERIC_READ* access.

GetFileSize retrieves the size of the given file in bytes. The *NULL* argument means the application does not want the high order double-word. A double-word is made up of two groups of 16 bits. The first 16 bits is called the high order word.

PTEDFileWrite writes the contents of the edit window into the given file.

```
BOOL PTEDFileWrite(HWND hwndEdit, LPSTR fileName)
{
   DWORD bytesWritten;
   HANDLE file;
   int length;
   LPSTR pBuffer;

   if (INVALID_HANDLE_VALUE ==
      (file = CreateFile(fileName, GENERIC_WRITE, 0, NULL,
                         CREATE_ALWAYS, 0, NULL)))
      return FALSE;

   length = GetWindowTextLength(hwndEdit);
   pBuffer = (LPSTR)malloc((length + 1) * sizeof(char));

   if (pBuffer == NULL) {
      CloseHandle(file);
      return FALSE;
   }

   GetWindowText(hwndEdit, pBuffer, length + 1);
   WriteFile(file, pBuffer, length * sizeof(char), &bytesWritten,
             NULL);

   if ((length * sizeof(char)) != (int)bytesWritten) {
      CloseHandle(file);
      free(pBuffer);
      pBuffer = NULL;
      return FALSE;
   }

   CloseHandle(file);
   free(pBuffer);
   pBuffer = NULL;
   return TRUE;
}
```

PTEDFileWrite returns *FALSE* if an error occurs.

Here is the complete file, *ptedfile.c*.

```
/* ptedfile.c - file ops */

#include <windows.h>
#include <commdlg.h>
#include "pted1.h"

OPENFILENAME ofn;

OPENFILENAME PTEDFileInit(HWND hwnd)
{
   static OPENFILENAME ofn;
   static char filter[] = "All files\0*.*\0\0";
```

```
   ofn.lStructSize       = sizeof(OPENFILENAME);
   ofn.hwndOwner         = hwnd;
   ofn.hInstance         = NULL;
   ofn.lpstrFilter       = filter;
   ofn.lpstrCustomFilter = NULL;
   ofn.nMaxCustFilter    = 0;
   ofn.nFilterIndex      = 0;
   ofn.lpstrFile         = NULL;     /* Set in Open and Save dialogs */
   ofn.nMaxFile          = MAX_PATH;
   ofn.lpstrFileTitle    = NULL;     /* Set in Open and Save dialogs */
   ofn.nMaxFileTitle     = MAX_PATH;
   ofn.lpstrInitialDir   = NULL;
   ofn.lpstrTitle        = NULL;
   ofn.Flags             = 0;       /* Set in Open and Save dialogs */
   ofn.nFileOffset       = 0;
   ofn.nFileExtension    = 0;
   ofn.lpstrDefExt       = NULL;
   ofn.lCustData         = 0L;
   ofn.lpfnHook          = NULL;
   ofn.lpTemplateName    = NULL;
   return ofn;
}


BOOL PTEDFileOpenDlg(HWND hwnd, LPSTR fileName, LPSTR fileTitle,
                     OPENFILENAME *ofn)
{
   ofn -> hwndOwner      = hwnd;
   ofn -> lpstrFile      = fileName;
   ofn -> lpstrFileTitle = fileTitle;
   ofn -> Flags          = OFN_HIDEREADONLY | OFN_CREATEPROMPT;
   return GetOpenFileName(ofn);
}


BOOL PTEDFileSaveDlg(HWND hwnd, LPSTR fileName, LPSTR fileTitle,
                     OPENFILENAME *ofn)
{
   ofn -> hwndOwner      = hwnd;
   ofn -> lpstrFile      = fileName;
   ofn -> lpstrFileTitle = fileTitle;
   ofn -> Flags          = OFN_OVERWRITEPROMPT;
   return GetSaveFileName(ofn);
}


BOOL PTEDFileRead(HWND hwndEdit, LPSTR fileName)
{
   HANDLE file;
   int fileLength;
   LPSTR pBuffer;
   DWORD bytesRead;

   if (INVALID_HANDLE_VALUE ==
      (file = CreateFile(fileName, GENERIC_READ, FILE_SHARE_READ,
                         NULL, OPEN_EXISTING, 0, NULL)));

   fileLength = GetFileSize(file, NULL);
   pBuffer = (LPSTR)malloc(fileLength + 2);
```

```
   ReadFile(file, pBuffer, fileLength, &bytesRead, NULL);
   CloseHandle(file);
   pBuffer[fileLength] = '\0';
   pBuffer[fileLength + 1] = '\0';

   SetWindowText(hwndEdit, pBuffer);
   free(pBuffer);
   pBuffer = NULL;
   return TRUE;
}


BOOL PTEDFileWrite(HWND hwndEdit, LPSTR fileName)
{
   DWORD bytesWritten;
   HANDLE file;
   int length;
   LPSTR pBuffer;

   if (INVALID_HANDLE_VALUE ==
      (file = CreateFile(fileName, GENERIC_WRITE, 0, NULL,
                         CREATE_ALWAYS, 0, NULL)))
      return FALSE;

   length = GetWindowTextLength(hwndEdit);
   pBuffer = (LPSTR)malloc((length + 1) * sizeof(char));

   if (pBuffer == NULL) {
      CloseHandle(file);
      return FALSE;
   }

   GetWindowText(hwndEdit, pBuffer, length + 1);
   WriteFile(file, pBuffer, length * sizeof(char), &bytesWritten,
             NULL);

   if ((length * sizeof(char)) != (int)bytesWritten) {
      CloseHandle(file);
      free(pBuffer);
      pBuffer = NULL;
      return FALSE;
   }

   CloseHandle(file);
   free(pBuffer);
   pBuffer = NULL;
   return TRUE;
}
```

7.4 File Menu Options

If *File*, *New* is selected from the menu, then a call is made to *fileNew*.

```
case WM_COMMAND:
...
   switch (LOWORD(wParam)) {
   /* File Menu Messages */
   case IDM_FILE_NEW:
      return fileNew(hwndEdit, fileName, fileTitle);
```

If there is no text in the edit window, there is nothing to do but return.

```
if (GetWindowTextLength(hwndEdit) == 0) {
   printFileTitle(hwnd, "");
   return 0;
}
```

If there is some text and it has not changed since being placed in the edit window, again there is nothing to do but return.

```
if (EDT_TEXTNOTCHANGED == (SendMessage(hwndEdit, EM_GETMODIFY,0,0))) {
   SetWindowText(hwndEdit, '\0');
   SendMessage(hwndEdit, EM_SETMODIFY, FALSE, 0);
   fileName[0] = '\0';
   fileTitle[0] = '\0';
   return 0;
}
```

EDT_TEXTNOTCHANGED is defined in *pted1.h*

EM_GETMODIFY gets the state of the edit control's modification flag. If the contents of the edit control have been changed, non-zero is returned; otherwise, zero is returned.

EM_SETMODIFY sets or clears the modification flag. The value of *wParam* is the new value of the flag. *FALSE* indicates the text has not been modified. *TRUE* indicates the text has been modified.

The *confirmSave* dialog returns either *IDCANCEL*, *IDYES* or *IDNO*. If *Cancel* is clicked, we just return. If *Yes* is clicked, we save the file with a call to *PTEDFileWrite*. If *No* is clicked, we just empty the edit control of any text.

```
int result;
...
result = confirmSave(hwnd, fileTitle);
if (result == IDCANCEL)
   return 0;

if (result == IDYES)
   PTEDFileWrite(hwndEdit, fileName);

SetWindowText(hwndEdit, '\0');
fileName[0] = '\0';
fileTitle[0] = '\0';
printFileTitle(hwnd, fileTitle);
```

If *File*, *Open* is selected, then a call is made to *fileOpen*.

```
      case IDM_FILE_OPEN:
         return fileOpen(hwndEdit, &ofn, fileName, fileTitle);
```

We make a call to the *Open* file dialog. and retrieve the contents of the file chosen by the user.

```
int fileOpen(HWND hwndEdit, OPENFILENAME *pofn,
                     LPSTR fileName, LPSTR fileTitle)
{
   HWND hwnd;

   hwnd = GetParent(hwndEdit);

   fileName[0] = '\0';
   fileTitle[0] = '\0';

   if (PTEDFileOpenDlg(hwnd, fileName, fileTitle, pofn)) {
      if (!PTEDFileRead(hwndEdit, fileName)) {
         report(REP_WARNING, "WndProc", "Unable to read from %s",
                     fileName);
      }
   }
   printFileTitle(hwnd, fileTitle);
   return 0;
}
```

If *File*, *Save* is chosen by the user, we make a call to *PTEDFileWrite*, but only if a file is open in the first place.

```
      case IDM_FILE_SAVE:
         if (fileName[0]) {
            PTEDFileWrite(hwndEdit, fileName);
            return 1;
         }
         /* fall through */

      case IDM_FILE_SAVE_AS:
         return fileSaveAs(hwndEdit, &ofn, fileName, fileTitle);
```

If there is no file already opened, we fall through to the *Save As* case, where *fileSaveAs* is called.

```
int fileSaveAs(HWND hwndEdit, OPENFILENAME *pofn,
                     LPSTR fileName, LPSTR fileTitle)
{
   HWND hwnd;

   hwnd = GetParent(hwndEdit);
   fileName[0] = '\0';
   fileTitle[0] = '\0';
```

```
   if (PTEDFileSaveDlg(hwnd, fileName, fileTitle, pofn)) {
      printFileTitle(hwnd, fileTitle);
      if (!PTEDFileWrite(hwndEdit, fileName)) {
         report(REP_WARNING, "WndProc", "Unable to save %s",
                fileName);
         return 0;
      }
      return 1;
   }
   return 0;
}
```

Calls to the file *Save As* dialog and to the *PTEDfileWrite* function are made.

Finally, if *File*, *Exit* is chosen by the user,

```
      case IDM_APP_EXIT:
         fileSaveOnExit(hwndEdit, fileName, fileTitle);
         SendMessage(hwnd, WM_CLOSE, 0, 0);
```

fileSaveOnExit is called.

```
int fileSaveOnExit(HWND hwndEdit, LPSTR fileName, LPSTR fileTitle)
{
   HWND hwnd;

   hwnd = GetParent(hwndEdit);

   if (EDT_TEXTNOTCHANGED == (SendMessage(hwndEdit, EM_GETMODIFY,0,0))) {
      SetWindowText(hwndEdit, '\0');
      SendMessage(hwndEdit, EM_SETMODIFY, FALSE, 0);
      fileName[0] = '\0';
      fileTitle[0] = '\0';
      return 0;
   }

   if (IDYES == (confirmSave(hwnd, fileTitle)))
      PTEDFileWrite(hwndEdit, fileName);

   SetWindowText(hwndEdit, '\0');
   fileName[0] = '\0';
   fileTitle[0] = '\0';
   printFileTitle(hwnd, fileTitle);
   return 0;
}
```

The entire program, *pted1.c*, is shown below.

```
/* pted1.c - primitive text editor.  Implements file handling. */

#include <windows.h>
#include <commctrl.h>
#include <stdarg.h>
#include <stdio.h>

#include "pted1.h"

WNDPROC oldEdit;
```

```
int report(int repType, PSTR caption, PSTR format, ...)
{
   CHAR string[256];
   va_list args;

   va_start(args, format);
   vsprintf(string, format, args);
   va_end(args);

   MessageBox(NULL, string, caption, MB_OK);

   if (repType == REP_FATAL)
     PostQuitMessage(0);
   return 0;
}


int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   PSTR cmdLine, int cmdShow)
{
   static char appName[] = "filehandling";
   HWND hwnd;
   MSG msg;
   WNDCLASS wc;
   BOOL ret;

   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInst;
   wc.hIcon         = NULL;
   wc.hCursor       = LoadImage(NULL, IDC_ARROW, IMAGE_CURSOR,
                        0, 0, LR_SHARED);
   wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName  = "PTED";
   wc.lpszClassName = appName;

   if (0 == (RegisterClass(&wc)))
      report(REP_FATAL, "WinMain", "%s", "RegisterClass failed");

   if (NULL == (hwnd = CreateWindow(appName,
                       "File Handling",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,
                       NULL,
                       hInst,
                       NULL)))
      report(REP_FATAL, "WinMain", "%s", "CreateWindow failed");

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

```
   while ((ret = GetMessage(&msg, NULL, 0, 0)) != 0) {
      if (ret < 0)
         report(REP_FATAL, "WinMain", "%s", "GetMessage failed");
      else {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
   return msg.wParam ;
}


HWND newEditWindow(HWND hwnd, HINSTANCE hInstance)
{
   HWND hwndEdit;

   hwndEdit = CreateWindow("EDIT", NULL, WS_CHILD |
                           WS_VISIBLE | WS_HSCROLL |
                           WS_VSCROLL | WS_BORDER |
                           ES_LEFT | ES_MULTILINE |
                           ES_AUTOHSCROLL | ES_AUTOVSCROLL,
                           0, 0, 0, 0,
                           hwnd, (HMENU)IDE_EDITABLE,
                           hInstance, NULL);
   SendMessage(hwndEdit, EM_LIMITTEXT, 32000, 0L);
   return hwndEdit;
}


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

   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));
}


HWND newStatusBar(HWND hwnd, HINSTANCE hInstance)
{
   HWND hwndStatusBar;
   int parts[] = { 150, -1 };

   hwndStatusBar = CreateWindow(STATUSCLASSNAME, NULL,
                           SBARS_SIZEGRIP | WS_CHILD | WS_VISIBLE,
                           0, 0, 0, 0,
                           hwnd, (HMENU)IDB_STATUS_BAR,
                           hInstance, NULL);
```

```
   SendMessage(hwndStatusBar, SB_SETPARTS,
               (WPARAM)sizeof(parts) / sizeof(int),
               (LPARAM)&parts);

   SendMessage(hwndStatusBar, SB_SETTEXT, 0,
               (LPARAM)"Line 0, Column 0");
   return hwndStatusBar;
}


int printFileTitle(HWND hwnd, char *fileTitle)
{
   char title[64 + MAX_PATH];

   sprintf(title, "%s - %s", "File Handling", fileTitle);
   SetWindowText(hwnd, title);
   return 0;
}


int confirmSave(HWND hwnd, LPSTR fileTitle)
{
   char buf[64 + MAX_PATH];

   sprintf(buf, "Save changes in %s?",
           fileTitle[0]? fileTitle : "UNAMED");

   return MessageBox(hwnd, buf, "File Handling", MB_YESNOCANCEL);
}


int fileNew(HWND hwndEdit, LPSTR fileName, LPSTR fileTitle)
{
   int result;
   HWND hwnd;

   hwnd = GetParent(hwndEdit);

   if (GetWindowTextLength(hwndEdit) == 0) {
      printFileTitle(hwnd, "");
      return 0;
   }

   if (EDT_TEXTNOTCHANGED == (SendMessage(hwndEdit, EM_GETMODIFY,0,0))) {
      SetWindowText(hwndEdit, '\0');
      SendMessage(hwndEdit, EM_SETMODIFY, FALSE, 0);
      fileName[0] = '\0';
      fileTitle[0] = '\0';
      return 0;
   }

   result = confirmSave(hwnd, fileTitle);
   if (result == IDCANCEL)
      return 0;

   if (result == IDYES)
      PTEDFileWrite(hwndEdit, fileName);
```

```
   SetWindowText(hwndEdit, '\0');
   fileName[0] = '\0';
   fileTitle[0] = '\0';
   printFileTitle(hwnd, fileTitle);
   return 0;
}


int fileOpen(HWND hwndEdit, OPENFILENAME *pofn,
                     LPSTR fileName, LPSTR fileTitle)
{
   HWND hwnd;

   hwnd = GetParent(hwndEdit);

   fileName[0] = '\0';
   fileTitle[0] = '\0';

   if (PTEDFileOpenDlg(hwnd, fileName, fileTitle, pofn)) {
      if (!PTEDFileRead(hwndEdit, fileName)) {
         report(REP_WARNING, "WndProc", "Unable to read from %s",
                     fileName);
      }
   }
   printFileTitle(hwnd, fileTitle);
   return 0;
}


int fileSaveAs(HWND hwndEdit, OPENFILENAME *pofn,
                     LPSTR fileName, LPSTR fileTitle)
{
   HWND hwnd;

   hwnd = GetParent(hwndEdit);
   fileName[0] = '\0';
   fileTitle[0] = '\0';

   if (PTEDFileSaveDlg(hwnd, fileName, fileTitle, pofn)) {
      printFileTitle(hwnd, fileTitle);
      if (!PTEDFileWrite(hwndEdit, fileName)) {
         report(REP_WARNING, "WndProc", "Unable to save %s",
                fileName);
         return 0;
      }
      return 1;
   }
   return 0;
}


int fileSaveOnExit(HWND hwndEdit, LPSTR fileName, LPSTR fileTitle)
{
   HWND hwnd;

   hwnd = GetParent(hwndEdit);
```

```
   if (EDT_TEXTNOTCHANGED == (SendMessage(hwndEdit, EM_GETMODIFY,0,0))) {
      SetWindowText(hwndEdit, '\0');
      SendMessage(hwndEdit, EM_SETMODIFY, FALSE, 0);
      fileName[0] = '\0';
      fileTitle[0] = '\0';
      return 0;
   }

   if (IDYES == (confirmSave(hwnd, fileTitle)))
      PTEDFileWrite(hwndEdit, fileName);

   SetWindowText(hwndEdit, '\0');
   fileName[0] = '\0';
   fileTitle[0] = '\0';
   printFileTitle(hwnd, fileTitle);
   return 0;
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                  WPARAM wParam, LPARAM lParam)
{
   static HINSTANCE hInstance;
   static HWND hwndEdit;
   static HWND hwndStatusBar;
   int statusBarHeight = 20;

   static OPENFILENAME ofn;
   static char fileName[MAX_PATH];
   static char fileTitle[MAX_PATH];

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      hwndEdit = newEditWindow(hwnd, hInstance);
      setInitialFont(hwnd, hwndEdit);
      hwndStatusBar = newStatusBar(hwnd, hInstance);

      oldEdit = (WNDPROC)SetWindowLong(hwndEdit, GWL_WNDPROC,
                                          (LONG)EditProc);
      ofn = PTEDFileInit(hwnd);
      return 0;

   case WM_SETFOCUS:
      SetFocus(hwndEdit);
      return 0;

   case WM_SIZE:
      MoveWindow(hwndEdit, 0, 0, LOWORD(lParam),
                 HIWORD(lParam) - statusBarHeight, TRUE);
      SendMessage(hwndStatusBar, WM_SIZE, 0, 0);
      return 0;

   case WM_COMMAND:
      if (LOWORD(wParam) == IDE_EDITABLE)
         if (HIWORD(wParam) == EN_ERRSPACE ||
             HIWORD(wParam) == EN_MAXTEXT)
            report(REP_FATAL, "WndProc", "%s",
                   "Edit control out of space");
```

```
switch (LOWORD(wParam)) {
/* File Menu Messages */
case IDM_FILE_NEW:
   return fileNew(hwndEdit, fileName, fileTitle);

case IDM_FILE_OPEN:
   return fileOpen(hwndEdit, &ofn, fileName, fileTitle);

case IDM_FILE_SAVE:
   if (fileName[0]) {
      PTEDFileWrite(hwndEdit, fileName);
      return 1;
   }
   /* fall through */

case IDM_FILE_SAVE_AS:
   return fileSaveAs(hwndEdit, &ofn, fileName, fileTitle);

case IDM_APP_EXIT:
   fileSaveOnExit(hwndEdit, fileName, fileTitle);
   SendMessage(hwnd, WM_CLOSE, 0, 0);
   return 0;

case IDM_FILE_PRINT:
   report(REP_DIAGNOSTIC, "WndProc", "%s",
         "Print option not implemented");
   return 0;

/* Edit Menu Messages */
case IDM_EDIT_UNDO:
case IDM_EDIT_CUT:
case IDM_EDIT_COPY:
case IDM_EDIT_PASTE:
case IDM_EDIT_CLEAR:
case IDM_EDIT_SELECT_ALL:
   report(REP_DIAGNOSTIC, "WndProc", "%s",
         "Edit menu not implemented");
   return 0;

/* Search Menu Messages */
case IDM_SEARCH_FIND:
case IDM_SEARCH_NEXT:
case IDM_SEARCH_REPLACE:
   report(REP_DIAGNOSTIC, "WndProc", "%s",
         "Search menu not implemented");
   return 0;

/* Font Menu Messages */
case IDM_FORMAT_FONT:
   report(REP_DIAGNOSTIC, "WndProc", "%s",
          "Format menu not implemented");
   return 0;

/* Help Menu Messages */
case IDM_HELP:
   report(REP_DIAGNOSTIC, "WndProc", "%s",
         "Help menu not implemented");
   return 0;
```

```
      case IDM_APP_ABOUT:
         DialogBox(hInstance, "ABOUTBOX" , hwnd,
                   (DLGPROC)AboutDlgProc);
         return 0;
      }
      return 0;

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


LRESULT CALLBACK AboutDlgProc(HWND hDlg, UINT msg,
                              WPARAM wParam, LPARAM lParam)
{
   switch (msg) {
   case WM_INITDIALOG:
      return TRUE;

   case WM_COMMAND:
      switch (LOWORD(wParam)) {
      case IDOK:
         EndDialog(hDlg, 0);
         return TRUE;
      }
      break;
   }
   return FALSE;
}


/* EDitProc - extends hwndEdit */
LRESULT CALLBACK EditProc(HWND hwnd, UINT msg, WPARAM wParam,
                          LPARAM lParam)
{
   static long line, column;
   static long selection, start, lineIndex;
   static char buf[256];

   HWND hwndParent, hwndStatusBar;

   switch (msg) {
   case WM_KEYUP:
   case WM_LBUTTONUP:
      line = SendMessage(hwnd, EM_LINEFROMCHAR, -1, 0);

      selection = SendMessage(hwnd, EM_GETSEL, 0, 0);
      start = LOWORD(selection);
      lineIndex = SendMessage(hwnd, EM_LINEINDEX, -1, 0);
      column = start - lineIndex;

      sprintf(buf, "Line %ld, Column %ld", line, column);
```

```
      hwndParent = GetParent(hwnd);
      hwndStatusBar = GetDlgItem(hwndParent, IDB_STATUS_BAR);
      SendMessage(hwndStatusBar, SB_SETTEXT, 0, (LPARAM)buf);
      break;

   }
   return CallWindowProc(oldEdit, hwnd, msg, wParam, lParam);
}
```

Phew. As you can see, Windows programs quickly get rather large. Notice that we have largely removed as much code as possible from the message processing in *WndProc* into functions to help us manage size and complexity and to minimise clutter.

Next, we look at printing.