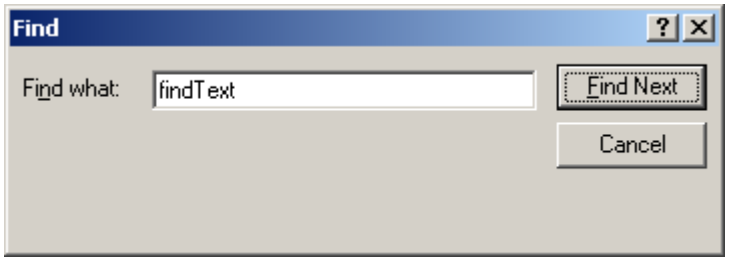
Programming Windows

Terry Marris Feb 2013

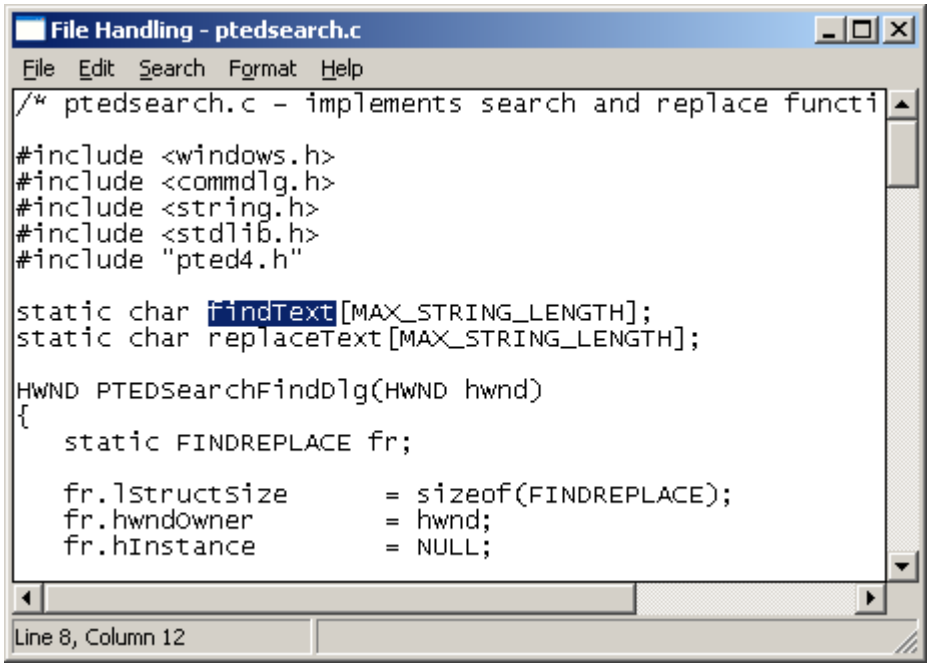
10 Searching

We see how to implement the *Search* menu options *Find*, *Find Next* and *Replace*.

We pop up a modeless *Find* dialog box



And, after supplying the text to be searched for, and clicking *Find Next*, the corresponding text in the document is found and highlighted.



The search starts from the top of the document, and each occurrence of the search-for text is highlighted for each click of *Find Next* until the end of the document is reached.

Options to specify the direction of search, and to ignore case, require their own logic and are not considered here.

A *modeless* dialog box allows the user to put the box to one side and continue using other parts of the program. So, the user can make changes to the text in the edit window while the *Find* dialog box is displayed.

10.1 Search Text

We start by initialising the *Find* dialog box.

```
#define MAX_STRING_LENGTH 256
...
static char findText[MAX_STRING_LENGTH];
...
HWND PTEDSearchFindDlg(HWND hwnd)
{
   static FINDREPLACE fr;

   fr.lStructSize      = sizeof(FINDREPLACE);
   fr.hwndOwner        = hwnd;
   fr.hInstance        = NULL;
   fr.Flags            = FR_HIDEUPDOWN | FR_HIDEMATCHCASE |
                         FR_HIDEWHOLEWORD;
   fr.lpstrFindWhat    = findText;
   fr.lpstrReplaceWith = NULL;
   fr.wFindWhatLen     = MAX_STRING_LENGTH;
   fr.wReplaceWithLen  = 0;
   fr.lCustData        = 0;
   fr.lpfnHook         = NULL;
   fr.lpTemplateName   = NULL;

   return FindText(&fr);
}
```

MAX_STRING_LENGTH is defined in *pted.h*.

We confine our interest to ASCII and so *findText* is defined to be an array of *char*.

The *FINDREPLACE* structure contains information that is used by the *FindText* (and *ReplaceText*) functions to initialise the *Find* (or *Replace*) dialog boxes. The *FINDREPLACE* structure must be static since a modeless dialog box function returns after the box is displayed (rather than after the box is destroyed).

We would set the *Flags* to *0* if we were to implement the logic for searching in both directions in a file, for matching case and for searching for text within words.

FindText creates a modeless *Find* dialog box with the information supplied in its *FINDREPLACE* argument. *NULL* is returned if the function fails. If the function succeeds a handle to the dialog box is returned.

A handle to a modeless dialog box is defined as a global variable because it is used over several functions in *pted4.c*

```
static HWND hdlgModeless = 0;
```

Even though static variables are said to be automatically initialised to zero, it does no harm to explicitly do so, and may even be good practice.

The *IsDlgMessage* must be called in the message loop when a modeless dialog box is active.

```
while ((msgLoopReport = GetMessage(&msg, NULL, 0, 0)) != 0) {
   if (msgLoopReport == -1)
      report(REP_FATAL, "WinMain", "%s", "Message loop malfunction");

   if (!IsWindow(hdlgModeless) ||
       !IsDialogMessage(hdlgModeless, &msg)) {
      if (!TranslateAccelerator(hwnd, accel, &msg)) {
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
   }
}
```

Before we leave *WinMain* and pass on to *WndProc* we should mention the *ES_NOHIDESEL* flag used when the edit window is created.

```
hwndEdit = CreateWindow("EDIT", NULL, WS_CHILD |
                        WS_VISIBLE | WS_HSCROLL |
                        WS_VSCROLL | WS_BORDER |
                        ES_LEFT | ES_MULTILINE |
                        ES_AUTOHSCROLL | ES_AUTOVSCROLL |
                        ES_NOHIDESEL,
                        0, 0, 0, 0,
                        hwnd, (HMENU)IDE_EDITABLE,
                        hInstance, NULL);
```

This flag ensures that text found in the edit window is highlighted even if the edit window does not have the focus.

```
static UINT messageFindReplace;
...
messageFindReplace = RegisterWindowMessage(FINDMSGSTRING);
```

When the *Find* dialog is displayed, it communicates with the owner window via a unique message number returned by *RegisterWindowMessage*. This message number can be used for sending or posting messages. The function returns zero if an error occurs.

A *Find* (or *Replace*) dialog box sends the registered *FINDMSGSTRING* message to the window procedure of its owner window whenever the *Find Next* (or *Replace* or *Replace All*) buttons are clicked by the user.

The *Search* menu is initialised with

```
case WM_INITMENUPOPUP:
   switch(lParam) {
   ...

   case 2:  /* Search menu */
      /* enables Find, Next & Replace if modeless dialogs */
      /* are not already active */
```

```
         enable = hdlgModeless == NULL ?
            MF_ENABLED : MF_GRAYED;

         EnableMenuItem((HMENU)wParam, IDM_SEARCH_FIND, enable);
         EnableMenuItem((HMENU)wParam, IDM_SEARCH_NEXT, enable);
         EnableMenuItem((HMENU)wParam, IDM_SEARCH_REPLACE, enable);

         break;
```

hdlgModeless is initialised by a call to *PTEDSearchFindDlg* if the *Search Find* option is chosen by the user.

```
      case IDM_SEARCH_FIND:
         SendMessage(hwndEdit, EM_GETSEL, 0, (LPARAM)&offset);
         hdlgModeless = PTEDSearchFindDlg(hwnd);
         return 0;
```

Do you remember setting *messageFindReplace* with a unique value returned by a call to *RegisterWindowMessage*? Well, it is tested for in the default leg of the switch-with-case message-processing construct.

```
   default:  /* process find-replace messages */
      if (msg == messageFindReplace) {
         pfr = (LPFINDREPLACE)lParam;

         if (pfr->Flags & FR_DIALOGTERM)
            hdlgModeless = NULL;
```

FR_DIALOGTERM means the dialog box is closing and its handle is no longer valid. We set the handle to *NULL*.

10.2 PTEDSearch

The entire PTEDSearch.c file is shown below.

```
/* ptedsearch.c - implements search and replace functions */

#include <windows.h>
#include <commdlg.h>
#include <string.h>
#include <stdlib.h>
#include "pted4.h"

static char findText[MAX_STRING_LENGTH];
static char replaceText[MAX_STRING_LENGTH];

HWND PTEDSearchFindDlg(HWND hwnd)
{
   static FINDREPLACE fr;

   fr.lStructSize      = sizeof(FINDREPLACE);
   fr.hwndOwner        = hwnd;
   fr.hInstance        = NULL;
   fr.Flags            = FR_HIDEUPDOWN | FR_HIDEMATCHCASE |
                         FR_HIDEWHOLEWORD;
```

```
   fr.lpstrFindWhat    = findText;
   fr.lpstrReplaceWith = NULL;
   fr.wFindWhatLen     = MAX_STRING_LENGTH;
   fr.wReplaceWithLen  = 0;
   fr.lCustData        = 0;
   fr.lpfnHook         = NULL;
   fr.lpTemplateName   = NULL;

   return FindText(&fr);
}


HWND PTEDSearchReplaceDlg(HWND hwnd)
{
   static FINDREPLACE fr;

   fr.lStructSize      = sizeof(FINDREPLACE);
   fr.hwndOwner        = hwnd;
   fr.hInstance        = NULL;
   fr.Flags            = FR_HIDEUPDOWN | FR_HIDEMATCHCASE |
                         FR_HIDEWHOLEWORD;
   fr.lpstrFindWhat    = findText;
   fr.lpstrReplaceWith = replaceText;
   fr.wFindWhatLen     = MAX_STRING_LENGTH;
   fr.wReplaceWithLen  = MAX_STRING_LENGTH;
   fr.lCustData        = 0;
   fr.lpfnHook         = NULL;
   fr.lpTemplateName   = NULL;

   return ReplaceText(&fr);
}


BOOL PTEDSearchFindText(HWND hwndEdit, int *pSearchOffset,
                        LPFINDREPLACE pfr)
{
   int length, pos;
   LPSTR pstrDoc, pstrPos;

   /* retrieves edit document */
   length = GetWindowTextLength(hwndEdit);

   if (NULL == (pstrDoc = malloc((length + 1) * sizeof(char))))
      return FALSE;

   GetWindowText(hwndEdit, pstrDoc, length + 1);

   /* searches document for given string */
   pstrPos = strstr(pstrDoc + *pSearchOffset, pfr->lpstrFindWhat);

   if (pstrPos == NULL)
      return FALSE;

   /* finds the position in the document & the new start offset */
   pos = pstrPos - pstrDoc;
   *pSearchOffset = pos + strlen(pfr->lpstrFindWhat);
```

```
   /* selects the found text */
   SendMessage(hwndEdit, EM_SETSEL, pos, *pSearchOffset);
   SendMessage(hwndEdit, EM_SCROLLCARET, 0, 0);

   free(pstrDoc);
   pstrDoc = NULL;

   return TRUE;
}


BOOL PTEDSearchNextText(HWND hwndEdit, int *pSearchOffset)
{
   FINDREPLACE fr;

   fr.lpstrFindWhat = findText;
   return PTEDSearchFindText(hwndEdit, pSearchOffset, &fr);
}


BOOL PTEDSearchReplaceText(HWND hwndEdit, int *pSearchOffset,
                           LPFINDREPLACE pfr)
{
   /* finds the text */
   if (!PTEDSearchFindText(hwndEdit, pSearchOffset, pfr))
      return FALSE;

   /* and replaces it */
   SendMessage(hwndEdit, EM_REPLACESEL, 0, (LPARAM)pfr->lpstrReplaceWith);
   return TRUE;
}


BOOL PTEDSearchValidFind()
{
   return *findText != '\0';
}
```

10.3 PTED4

The entire code for *pted4.c* is shown below.

```
/* pted4.c - primitive text editor */

#include <windows.h>
#include <commctrl.h>
#include <commdlg.h>
#include <stdarg.h>
#include <stdio.h>
#include "pted4.h"

WNDPROC oldEdit;
static HWND hdlgModeless = 0;
```

```
int report(int repType, PSTR caption, PSTR format, ...)
{
   CHAR string[MAX_STRING_LENGTH];
   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[] = "PTED";
   HWND hwnd;
   MSG msg;
   WNDCLASS wc;
   HACCEL accel;
   BOOL msgLoopReport;

   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,
                       "PTED4",
                       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);

  accel = LoadAccelerators(hInst, appName);
```

```
   while ((msgLoopReport = GetMessage(&msg, NULL, 0, 0)) != 0) {
      if (msgLoopReport == -1)
         report(REP_FATAL, "WinMain", "%s", "Message loop malfunction");

      if (!IsWindow(hdlgModeless) ||
          !IsDialogMessage(hdlgModeless, &msg)) {
         if (!TranslateAccelerator(hwnd, accel, &msg)) {
            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 |
                           ES_NOHIDESEL,
                           0, 0, 0, 0,
                           hwnd, (HMENU)IDE_EDITABLE,
                           hInstance, NULL);
   SendMessage(hwndEdit, EM_LIMITTEXT, 32000, 0L);
   return hwndEdit;
}


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));
}
```

```
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;
}


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];

   static int offset;
   static UINT messageFindReplace;
   int selBegin, selEnd, enable;
   LPFINDREPLACE pfr;

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

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

      messageFindReplace = RegisterWindowMessage(FINDMSGSTRING);
      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_INITMENUPOPUP:
   switch(lParam) {
   case 1:  /* Edit menu */
      /* enables Undo if edit control can do it */

      EnableMenuItem((HMENU)wParam, IDM_EDIT_UNDO,
         SendMessage(hwndEdit, EM_CANUNDO, 0, 0L) ?
            MF_ENABLED : MF_GRAYED);

      /* enables Paste if text is in the clipboard */

      EnableMenuItem((HMENU)wParam, IDM_EDIT_PASTE,
         IsClipboardFormatAvailable(CF_TEXT) ?
            MF_ENABLED : MF_GRAYED);

      /* enables Cut, Copy, Del if text is selected */

      SendMessage(hwndEdit, EM_GETSEL, (WPARAM)&selBegin,
                                       (LPARAM)&selEnd);

      enable = selBegin != selEnd ? MF_ENABLED : MF_GRAYED;

      EnableMenuItem((HMENU)wParam, IDM_EDIT_CUT, enable);
      EnableMenuItem((HMENU)wParam, IDM_EDIT_COPY, enable);
      EnableMenuItem((HMENU)wParam, IDM_EDIT_CLEAR, enable);

      break;

   case 2:  /* Search menu */
      /* enables Find, Next & Replace if modeless dialogs */
      /* are not already active */

      enable = hdlgModeless == NULL ?
         MF_ENABLED : MF_GRAYED;

      EnableMenuItem((HMENU)wParam, IDM_SEARCH_FIND, enable);
      EnableMenuItem((HMENU)wParam, IDM_SEARCH_NEXT, enable);
      EnableMenuItem((HMENU)wParam, IDM_SEARCH_REPLACE, enable);

      break;
   }
   return 0;

case WM_COMMAND:
   if (lParam && LOWORD(wParam) == IDE_EDITABLE) {
      switch (HIWORD(wParam)) {
      case EN_ERRSPACE:
      case EN_MAXTEXT:
         report(REP_FATAL, "WndProc", "%s",
               "Edit control out of space");
         return 0;
      }
      break;
   }
```

```
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]) {
      fileWrite(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:
   if (!PTEDFilePrint(hInstance, hwnd, hwndEdit, fileTitle))
      report(REP_WARNING, "WndProc", "%s could not be printed",
             fileTitle);
   return 0;

/* Edit Menu Messages */
case IDM_EDIT_UNDO:
   SendMessage(hwndEdit, WM_UNDO, 0, 0);
   return 0;

case IDM_EDIT_CUT:
   SendMessage(hwndEdit, WM_CUT, 0, 0);
   return 0;

case IDM_EDIT_COPY:
   SendMessage(hwndEdit, WM_COPY, 0, 0);
   return 0;

case IDM_EDIT_PASTE:
   SendMessage(hwndEdit, WM_PASTE, 0, 0);
   return 0;

case IDM_EDIT_CLEAR:
   SendMessage(hwndEdit, WM_CLEAR, 0, 0);
   return 0;

case IDM_EDIT_SELECT_ALL:
   SendMessage(hwndEdit, EM_SETSEL, 0, -1);
   return 0;

/* Search Menu Messages */
case IDM_SEARCH_FIND:
   SendMessage(hwndEdit, EM_GETSEL, 0, (LPARAM)&offset);
   hdlgModeless = PTEDSearchFindDlg(hwnd);
   return 0;
```

```
      case IDM_SEARCH_NEXT:
         SendMessage(hwndEdit, EM_GETSEL, 0, (LPARAM)&offset);

         if (PTEDSearchValidFind())
            PTEDSearchNextText(hwndEdit, &offset);
         else
            hdlgModeless = PTEDSearchFindDlg(hwnd);
         return 0;

      case IDM_SEARCH_REPLACE:
         SendMessage(hwndEdit, EM_GETSEL, 0, (LPARAM)&offset);
         hdlgModeless = PTEDSearchReplaceDlg(hwnd);
         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_CLOSE:
      fileSaveOnExit(hwndEdit, fileName, fileTitle);
      DestroyWindow(hwnd);
      return 0;

   case WM_DESTROY:
       PostQuitMessage(0);
       return 0;

   default:  /* process find-replace messages */
      if (msg == messageFindReplace) {
         pfr = (LPFINDREPLACE)lParam;

         if (pfr->Flags & FR_DIALOGTERM)
            hdlgModeless = NULL;

         if (pfr->Flags & FR_FINDNEXT)
            if (!PTEDSearchFindText(hwndEdit, &offset, pfr))
               report(REP_WARNING, "hwndEdit",
                     "Cannot find %s", pfr->lpstrFindWhat);

         if (pfr->Flags & FR_REPLACE || pfr->Flags & FR_REPLACEALL)
            if (!PTEDSearchReplaceText(hwndEdit, &offset, pfr))
               report(REP_WARNING, "hwndEdit",
                     "Cannot find %s", pfr->lpstrFindWhat);
```

```
         if (pfr->Flags & FR_REPLACEALL)
            while (PTEDSearchReplaceText(hwndEdit, &offset, pfr))
               ;

         return 0;
      }
      break;
   }
   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);
}
```

Next, we look at the *Format* menu options.