Programming Windows

Terry Marris Jan 2013

6 Menus

Three programs are presented in this chapter, each one building on the preceding program. In the first, the beginnings of a primitive text editor are implemented. In the second, a menu system is introduced. In the third, subclassing is used to trace the location of the cursor in an edit window.

6.1 Editable

We see how you can enter and edit lines of text in a window.

The editable window is created in *WndProc*.

```
static HINSTANCE hInstance;
static HWND hwndEdit;
...
case WM_CREATE:
   hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
   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);
```

hInstance = ((LPCREATESTRUCT)lParam)->hInstance obtains the program's instance handle and stores it in the static variable, *hInstance*.

"EDIT" is a pre-defined window class, a rectangle that can contain editable text.

NULL means the window has no caption or title.

But the window does have a set of styles. *WS_CHILD* means the window is a child window, without a menu bar. *WS_HSCROLL* and *WS_VSCROLL* means that the window has both horizontal and vertical scroll bars. *WS_BORDER* specifies that the window has a thin line border. *ES_LEFT* says that the text is left justified within the window. *ES_MULTILINE* means you can have many lines (rather than just one, which is the default).

ES_AUTOHSCROLL and *ES_AUTOVSCROLL* are both edit control styles. *ES_AUTOHSCROLL* automatically scrolls text to the right by ten characters when the user types a character at the end of a line, or scrolls to the beginning of the next line down if the user presses the enter key. *ES_AUTOVSCROLL* automatically scrolls the text up one page when the user presses the enter key on the last line at the bottom of the window.

The initial co-ordinates are set to 0, 0. The width and height are also set to 0, 0. But the edit window is set to the dimensions of the main window by a call to *MoveWindow* when *WndProc* receives a *WM_SIZE* message.

```
   case WM_SIZE:
      MoveWindow(hwndEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
      return 0;
```

IDE_EDITABLE is defined in *menu.h*

The default font is pretty ugly. We set it to something more pleasing.

```
   static HFONT hFont;
   static HDC hdc;
   static 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));
```

You may remember that a commentary on this code was given in chapter four, Buttons.

A *TEXTMETRIC* structure contains the information about a physical font. *tmExternaLeading* is the space between rows.

A *LOGFONT* structure defines the attributes of a font, characteristics such as height, width and face name e.g. Arial, Tahoma and Times New Roman.

ZeroMemory clears a block of memory and fills it with zeros. The MSDN documentation says it would be safer to use *SecureZeroMemory* instead.

MulDiv multiplies the first two arguments, and then divides the result with the third argument. The result is rounded up to the nearest integer if the result is positive, rounded down if the result is negative.

CreateFontIndirect creates a logical font with the given characteristics.

The pre-defined multiline edit control is limited to about 32Kb of text. We warn the user when this limit is reached.

```
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");
   return 0;
```

You may remember that *IDE_EDITABLE* is defined in *menu.h* *EN_ERRSPACE* signals the application is out of memory. *EN_MAXTEXT* is sent when the you try to enter more characters than can be accommodated in the window, any additional text is truncated. But the edit class comes complete with text editing features.

The edit control may well be limited in the amount of text it can handle. But is does come with some text editing features. For example, you can enter, delete and insert text with an edit control.

Shown below is the header file and the entire program.

```
/* editable.h */

#define IDE_EDITABLE 101
```

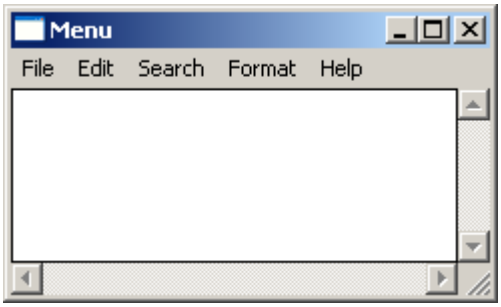
```
/* editable.c - provides an editable text area */

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

#include "editable.h"

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

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

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[] = "editable";
   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  = NULL;
   wc.lpszClassName = appName;

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

   if (NULL == (hwnd = CreateWindow(appName,
                       "Editable",
                       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 ;
}


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

   static HFONT hFont;
   static HDC hdc;
   static LOGFONT logFont;
   int pointSize = 10;
   char *faceName = "Lucida Console";

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      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);
```

```
      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));
      return 0;

   case WM_SETFOCUS:
      SetFocus(hwndEdit);
      return 0;

   case WM_SIZE:
      MoveWindow(hwndEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
      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");
      return 0;

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

6.2 Menu

We see how to display a menu in the classic Windows style.



We begin by defining a constant for each menu option. These constants will be used by both the resource file, which defines the menu, and the program, which uses the menu.

```
/* menu.h */

#define IDE_EDITABLE 101

#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
```

Now we turn to the resource file.

```
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
   }
```

PTED, for Primitive Text Editor, names the menu. This identifier will also be used in the program that displays and uses the menu. *MENU* is a keyword. It introduces a menu specification. *POPUP* introduces a main menu. Here, the main menu is named *File*. *MENUITEM* defines an option in the menu. Here, menu options include *New, Open, Save,* ... Alongside each menu option is an identifier, defined in the header file *menu.h* shown above. These identifiers will also be used in the program.

Shown below is the entire resource file.

```
/* menures.rc - specifies menu structure and content */

#include <windows.h>
#include <afxres.h>
#include "menu.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
   }
}
```

The menu is shown by initialising the *lpszMenuName* field of the *WNDCLASS* variable in *WinMain*.

```
   WNDCLASS wc;
   ...
   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;
```

The menu option identifiers are passed in *LOWORD(wParam)*.

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                  WPARAM wParam, LPARAM lParam)
{
   ...
   switch (msg) {
   ...
   case WM_COMMAND:
   ...
      switch (LOWORD(wParam)) {
      case IDM_APP_EXIT:
         SendMessage(hwnd, WM_CLOSE, 0, 0);
         return 0;

      case IDM_FILE_NEW:
      case IDM_FILE_OPEN:
      case IDM_FILE_SAVE:
      case IDM_FILE_SAVE_AS:
      case IDM_FILE_PRINT:
        report(REP_DIAGNOSTIC,"WndProc","%s","File menu not implemented");
         return 0;
      ...
```

So, for example, if the user selects the *Exit* option of the *File* menu, identified by I*DM_APP_EXIT*, a *WM_CLOSE* message is sent to the main window.

Here is the entire program.

```
/* menu.c - provides the basis of a menu system */

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

#include "menu.h"

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

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK AboutDlgProc(HWND, UINT, WPARAM, LPARAM);
```

```
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[] = "menu";
   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,
                       "Menu",
                       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 CALLBACK 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));
}


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

   switch (msg) {
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      hwndEdit = newEditWindow(hwnd, hInstance);
      setInitialFont(hwnd, hwndEdit);
      return 0;
```

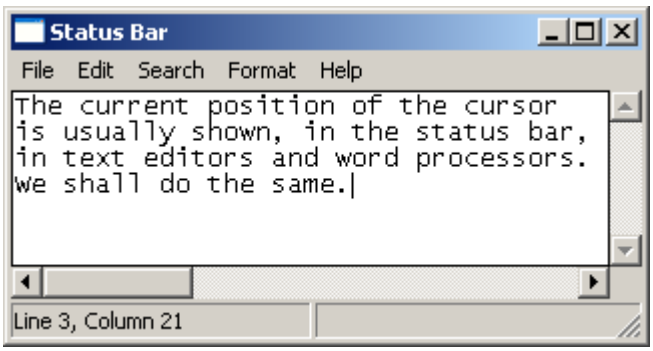
```
case WM_SETFOCUS:
   SetFocus(hwndEdit);
   return 0;

case WM_SIZE:
   MoveWindow(hwndEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
   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)) {
   case IDM_APP_EXIT:
      SendMessage(hwnd, WM_CLOSE, 0, 0);
      return 0;

   case IDM_FILE_NEW:
   case IDM_FILE_OPEN:
   case IDM_FILE_SAVE:
   case IDM_FILE_SAVE_AS:
   case IDM_FILE_PRINT:
      report(REP_DIAGNOSTIC, "WndProc", "%s",
            "File menu not implemented");
      return 0;

   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;

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

   case IDM_FORMAT_FONT:
      report(REP_DIAGNOSTIC, "WndProc", "%s",
            "Format menu not implemented");
      return 0;

   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;
}
```

Look at the *AboutDlgProc* procedure shown above. Where is *IDOK* defined? *IDOK* (and *IDCANCEL*) are both pre-defined identifiers for *OK* (and *Cancel*) buttons.

6.3 Statusbar

The current position of the cursor is usually shown, in the status bar, in text editors and word processors. We shall do the same.



Unfortunately, the edit control window does not respond to keyboard messages such as *WM_KEYUP*. We need to extend the functionality of the edit control window. We do this with a technique known as *subclassing*.

First, we define a *WNDPROC* variable and create the edit window in the normal way.

```
WNDPROC oldEdit;
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);
```

Then we obtain the address the *EditProc* procedure. This procedure contains the code that extends the functionality of the *hwndEdit* window. Here, *SetWindowLong* with the *GWL_WNDPROC* argument makes the connection between *hwndEdit* and *EditProc*.

```
oldEdit = (WNDPROC)SetWindowLong(hwndEdit, GWL_WNDPROC, (LONG)EditProc);
```

Finally, we implement *EditProc* to extend the functionality of *hwndEdit* by responding to *WM_KEYUP* messages.

```
/* 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:
      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);
}
```

Ok. We go through the code line by line.

An *id* for the status bar is defined in *statbar.h*.

```
/* statbar.h */

#define IDE_EDITABLE 101
#define IDB_STATUS_BAR 102
...
```

There are no resources to define. So we move onto the source code.

```
#include <commctrl.h>
```

is required because that is where *STATUSCLASSNAME* and the *SB_* (for Status Bar) series of constants are defined.

The prototype for *EditProc* is defined.

```
LRESULT CALLBACK EditProc(HWND, UINT, WPARAM, LPARAM);
```

And a global variable named *oldEdit* is declared.

```
WNDPROC oldEdit;
```

WNDPROC is a pointer to a *CALLBACK* function type.

The edit window control is created in the usual way.

```
   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);
```

The status bar window is created in a similar way.

```
   HWND hwndStatusBar;

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

STATUSCLASSNAME creates a status window. It is a pre-defined window class.

SBARS_SIZEGRIP puts the little grey triangle at the bottom right hand corner of a window, and serves as an anchor point when you want to resize the window.

IDB_STATUS_BAR is defined in *statbar.h* shown above.

The status bar is divided into two parts.

```
   int parts[] = { 150, -1 };
```

150 represents the width of the left hand part. The *-1* signals no more parts.

The parts of the status bar is set by

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

wParam contains the number of elements. *lParam* is a pointer to the integer array.

The text on the status bar is set by

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

The *wParam* argument, 0, indicates the first part. (In this case it is the only part). The *lParam* is the text to be displayed.

The size and position of both the edit and status bar windows is set by

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

Now we come to the *EditProc* function.

```
/* 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:
      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);
}
```

Retrieving the current line number is straightforward. *EM_LINEFROMCHAR* along with *-1* for *wParam* retrieves the current line number in a multiline edit control.

But obtaining the current column requires a little more work.

```
selection = SendMessage(hwnd, EM_GETSEL, 0, 0);
start = LOWORD(selection);
```

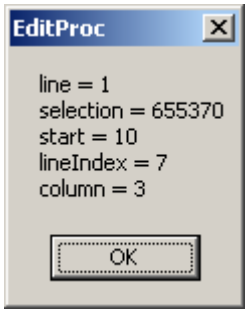
EM_GETSEL retrieves the starting and ending character positions of the current selection. These could be the same if there is no selection made (as is the case here). The starting position of the selection is returned in *LOWORD*.

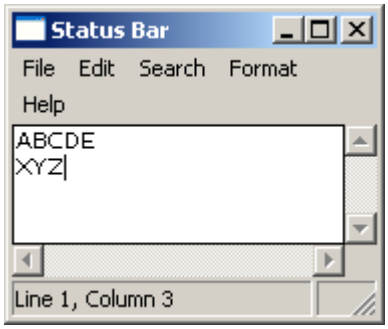
Now, *EM_LINEINDEX* retrieves the index of the first character of the given line. When -1 is specified, this is the current line.

```
lineIndex = SendMessage(hwnd, EM_LINEINDEX, -1, 0);
```

Then the current column is the difference between the start of the selection and the line index.

For example, this diagnostic was produced when *ABCDE XYZ* was entered over two lines.





Shown below are the complete listings for *statbar.h*, *statbarres.rc* and *statbar.c*.

```
/* statbar.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
```

```
/* statbarres.rc - specifies menu structure and content */

#include <windows.h>
#include <afxres.h>
#include "menu.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
   }
}
```

```
/* statbar.c - implements a status bar */

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

#include "statbar.h"

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

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK EditProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK AboutDlgProc(HWND, UINT, WPARAM, LPARAM);

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[] = "statusbar";
   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,
                       "Status Bar",
                       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 CALLBACK 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 CALLBACK 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;

   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);
      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)) {
   case IDM_APP_EXIT:
      SendMessage(hwnd, WM_CLOSE, 0, 0);
      return 0;

   case IDM_FILE_NEW:
   case IDM_FILE_OPEN:
   case IDM_FILE_SAVE:
   case IDM_FILE_SAVE_AS:
   case IDM_FILE_PRINT:
      report(REP_DIAGNOSTIC, "WndProc", "%s",
                             "File menu not implemented");
      return 0;

   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;

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

   case IDM_FORMAT_FONT:
      report(REP_DIAGNOSTIC, "WndProc", "%s",
                             "Format menu not implemented");
      return 0;

   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:
      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 take a look at implementing the *File* menu options.