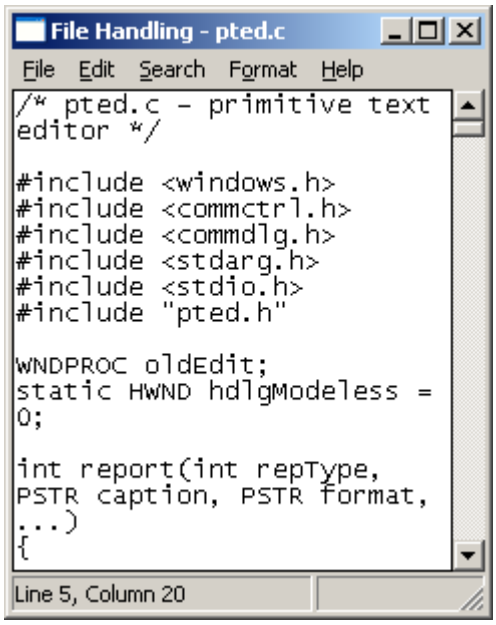
Programming Windows

Terry Marris Feb 2013

12 Word Wrap

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



12.1 Word Wrap

We want a tick to appear next to the *Word Wrap* option when word wrap is switched on, and to disappear when word wrap is switched off. We achieve this with a toggle.

```
static BOOL wordWrapIsOn = FALSE;
...
if (!wordWrapIsOn) {
   CheckMenuItem(GetMenu(hwnd), IDM_FORMAT_WORDWRAP,
                 MF_CHECKED);
   wordWrapIsOn = TRUE;
}
else if (wordWrapIsOn) {
   CheckMenuItem(GetMenu(hwnd), IDM_FORMAT_WORDWRAP,
                 MF_UNCHECKED);
   wordWrapIsOn = FALSE;
}
```

The essence of the implementation is to discard the current edit window and create a new window with (or without) the *WS_HSCROLL* and *ES_AUTOHSCROLL* window and edit styles.

First, we establish the owner of the edit window along with the owner's size.

```
   HWND hwnd;
   RECT rect;
   ...
   hwnd = GetParent(hwndEdit);
   GetClientRect(hwnd, &rect);
```

Before discarding the old edit window we preserve the current text and font.

```
   UINT textSize;
   LPSTR text;
   HFONT currentFont;
   ...
   textSize = GetWindowTextLength(hwndEdit) + 1;
   text = (LPSTR) malloc (sizeof(char) * textSize + 1);
   GetWindowText(hwndEdit, text, textSize);
   currentFont = (HFONT)SendMessage(hwndEdit, WM_GETFONT, 0, 0);
   DestroyWindow(hwndEdit);
```

Then we make a call to *newEditWindow*, along with setting its status bar.

```
   hwndEdit = newEditWindow(hwnd, hInstance, wordWrapIsOn,
                            rect.right, rect.bottom - STATUS_BAR_HEIGHT);
   oldEdit = (WNDPROC)SetWindowLong(hwndEdit, GWL_WNDPROC,
                                    (LONG)EditProc);
```

STATUS_BAR_HEIGHT is defined as 20 in *pted.h*.

Then we copy the preserved text into the new edit window and set its font.

```
   SetWindowText(hwndEdit, (LPSTR)text);
   SendMessage(hwndEdit, WM_SETFONT, (WPARAM)currentFont,
               MAKELPARAM(TRUE,0));
```

Here is the entire function.

```
HWND wordWrap(HWND hwndEdit, HINSTANCE hInstance)
{
   static BOOL wordWrapIsOn = FALSE;
   UINT textSize;
   LPSTR text;
   HFONT currentFont;
   HWND hwnd;
   RECT rect;

   hwnd = GetParent(hwndEdit);
   GetClientRect(hwnd, &rect);

   if (!wordWrapIsOn) {
      CheckMenuItem(GetMenu(hwnd), IDM_FORMAT_WORDWRAP,
                    MF_CHECKED);
      wordWrapIsOn = TRUE;
   }
   else if (wordWrapIsOn) {
      CheckMenuItem(GetMenu(hwnd), IDM_FORMAT_WORDWRAP,
                    MF_UNCHECKED);
      wordWrapIsOn = FALSE;
   }

   textSize = GetWindowTextLength(hwndEdit) + 1;
```

```
   text = (LPSTR) malloc (sizeof(char) * textSize + 1);
   GetWindowText(hwndEdit, text, textSize);

   currentFont = (HFONT)SendMessage(hwndEdit, WM_GETFONT, 0, 0);
   DestroyWindow(hwndEdit);

   hwndEdit = newEditWindow(hwnd, hInstance, wordWrapIsOn,
                            rect.right, rect.bottom - STATUS_BAR_HEIGHT);
   oldEdit = (WNDPROC)SetWindowLong(hwndEdit, GWL_WNDPROC,
                                    (LONG)EditProc);
   SetWindowText(hwndEdit, (LPSTR)text);
   SendMessage(hwndEdit, WM_SETFONT, (WPARAM)currentFont,
               MAKELPARAM(TRUE,0));

   ShowWindow(hwndEdit, SW_SHOW);
   SetFocus(hwndEdit);

   free(text);
   text = NULL;
   return hwndEdit;
}
```

12.2 New Edit Window

The original *newEditWindow* is changed to help implement the word wrap function.

```
HWND newEditWindow(HWND hwnd, HINSTANCE hInstance, BOOL wordWrap,
                   int right, int bottom)
{
   HWND hwndEdit;

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

   SendMessage(hwndEdit, EM_LIMITTEXT, 32000, 0L);
   return hwndEdit;
}
```

First, three additional parameters, *wordWrap*, *right* and *bottom* are included. *right* and *bottom* define the width and height of the new edit window. *WS_HSCROLL* and *ES_AUTOHSCROLL* are included only if *wordWrap* is *FALSE*.

Of course, we are obliged to amend the call to *newEditWindow* when creating the initial edit window.

```
   case WM_CREATE:
      hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
      hwndEdit = newEditWindow(hwnd, hInstance, FALSE, 0, 0);
```

Here is the entire *pted.c* file.

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

#include <windows.h>
#include <commctrl.h>
#include <commdlg.h>
#include <stdarg.h>
#include <stdio.h>
#include "pted.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,
                       "Primitive Text Editor",
                       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 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;
}


HWND newEditWindow(HWND hwnd, HINSTANCE hInstance, BOOL wordWrap,
                   int right, int bottom)
{
   HWND hwndEdit;
```

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

   SendMessage(hwndEdit, EM_LIMITTEXT, 32000, 0L);
   return hwndEdit;
}


HWND wordWrap(HWND hwndEdit, HINSTANCE hInstance)
{
   static BOOL wordWrapIsOn = FALSE;
   UINT textSize;
   LPSTR text;
   HFONT currentFont;
   HWND hwnd;
   RECT rect;

   hwnd = GetParent(hwndEdit);
   GetClientRect(hwnd, &rect);

   if (!wordWrapIsOn) {
      CheckMenuItem(GetMenu(hwnd), IDM_FORMAT_WORDWRAP,
                    MF_CHECKED);
      wordWrapIsOn = TRUE;
   }
   else if (wordWrapIsOn) {
      CheckMenuItem(GetMenu(hwnd), IDM_FORMAT_WORDWRAP,
                    MF_UNCHECKED);
      wordWrapIsOn = FALSE;
   }

   textSize = GetWindowTextLength(hwndEdit) + 1;
   text = (LPSTR) malloc (sizeof(char) * textSize + 1);
   GetWindowText(hwndEdit, text, textSize);

   currentFont = (HFONT)SendMessage(hwndEdit, WM_GETFONT, 0, 0);
   DestroyWindow(hwndEdit);

   hwndEdit = newEditWindow(hwnd, hInstance, wordWrapIsOn,
                            rect.right, rect.bottom - STATUS_BAR_HEIGHT);
   oldEdit = (WNDPROC)SetWindowLong(hwndEdit, GWL_WNDPROC,
                                    (LONG)EditProc);
   SetWindowText(hwndEdit, (LPSTR)text);
   SendMessage(hwndEdit, WM_SETFONT, (WPARAM)currentFont,
               MAKELPARAM(TRUE,0));

   ShowWindow (hwndEdit, SW_SHOW);
   SetFocus (hwndEdit);

   free(text);
   text = NULL;
   return hwndEdit;
}
```

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

   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, FALSE, 0, 0);
      setInitialFont(hwndEdit);
      hwndStatusBar = newStatusBar(hwnd, hInstance);

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

      messageFindReplace = RegisterWindowMessage(FINDMSGSTRING);
      return 0;

   case WM_SETFOCUS:
      SetFocus(hwndEdit);
      return 0;

   case WM_SIZE:
      MoveWindow(hwndEdit, 0, 0, LOWORD(lParam),
                 HIWORD(lParam) - STATUS_BAR_HEIGHT, 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 (!filePrint(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 = findDlg(hwnd);
   return 0;

case IDM_SEARCH_NEXT:
   SendMessage(hwndEdit, EM_GETSEL, 0, (LPARAM)&offset);
   if (isValidFind())
      nextText(hwndEdit, &offset);
   else
      hdlgModeless = findDlg(hwnd);
   return 0;

case IDM_SEARCH_REPLACE:
   SendMessage(hwndEdit, EM_GETSEL, 0, (LPARAM)&offset);
   hdlgModeless = replaceDlg(hwnd);
   return 0;

/* Format Menu Messages */
case IDM_FORMAT_WORDWRAP:
   hwndEdit = wordWrap(hwndEdit, hInstance);
   return 0;

case IDM_FORMAT_FONT:
   if (chooseFont(hwnd))
      setFont(hwndEdit) ;
   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 (!findText(hwndEdit, &offset, pfr))
               report(REP_WARNING, "hwndEdit", "Cannot find %s",
                      pfr->lpstrFindWhat);

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

         if (pfr->Flags & FR_REPLACEALL)
            while (replaceText(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);
}
```

Well, that is just about as far as I intend to go with the functionality of the primitive text editor.