Appendix A

WiniEdit Code Listing

Following is the complete listing of WiniEdit, the Windows version of the example Macintosh program, MiniEdit, from my *Macintosh Revealed* series. Although the program uses C++-style comments and a few other syntactic conveniences, it makes no use of the object-oriented features of C++ and is essentially written in straight C.

Here is the program's header file, **`WiniEdit.h`**:

```
//
//                                        WiniEdit.h
//                                Example Windows application program
//                               S. Chernicoff         15 January 1995
//

//                        Global header file for WiniEdit example application program


//-------------------------------------------------------------------------------------------------------------
--



// Global Constants and Variables


   const INT  ClassNameMax = 32;                    // Maximum length for program name
   CHAR       ProgName[ClassNameMax];               // Name of program

   HANDLE  ThisInstance;                          // Handle to this instance of program
   HANDLE  PrevInstance;                          // Handle to previous instance of program

   HWND    TheWindow;                             // Handle to main window
   HWND    TheEditor;                             // Handle to edit control

   HACCEL  AccelTable;                            // Handle to accelerator table

   const INT   TextMargin = 4;                    // Inset from window to text rectangle
   CHOOSEFONT  FontParams;                           // Parameters for font dialog
```

```
   LOGFONT     TextFormat;                         // Current text formatting characteristics
   HFONT       TheFont;                            // Handle to current font
   HFONT       DeviceFont;                         // Handle to standard font for device

   CHOOSECOLOR  ColorParams;                       // Parameters for color dialog
   COLORREF     UserColors[16];                    // Array of user's custom colors
   COLORREF     TextColor;                         // Color for displaying text
   COLORREF     BackgroundColor;                   // Color for window background
   HBRUSH       BackgroundBrush;                   // Brush for painting window background

   const INT  DocMax = 32767;                      // Maximum length of text document
   const INT  MsgMax =   256;                      // Maximum length of on-screen messages

   HANDLE  TheFile = NULL;                         // Handle to current file

   const INT  PathMax = MAX_PATH;                 // Maximum length for file path names
   CHAR       PathName[PathMax];                   // Full path name of current file
   CHAR       NewPath [PathMax];                   // Full path name of new file

   const INT  TitleMax = 64;                       // Maximum length for title strings
   CHAR       FileTitle  [TitleMax];              // Local file name of current file
   CHAR       NewTitle   [TitleMax];              // Local file name of new file
   CHAR       NoNameTitle[TitleMax];              // Default title for window with no associated file

   const INT  FileExtMax = 4;                      // Maximum length for file extension
   CHAR       FileExt[FileExtMax];                 // Default file extension

   const INT  FilterMax = 256;                    // Maximum length for file filter string
   CHAR       FilterString[FilterMax];             // Filter string for Open and Save dialogs

   OPENFILENAME FileParams;                       // Parameters for file dialogs

   BOOL  ContinueFlag = TRUE;                      // Not time to quit yet?
   BOOL  ErrorFlag    = FALSE;                    // I/O error flag

//-----------------------------------------------------------------------------------------------------
--



// Function Prototypes


INT CALLBACK WinMain (HINSTANCE instHandle, HINSTANCE prevInst, LPSTR commandLine, INT showMode);
     // Main function
   VOID PASCAL Initialize (HANDLE thisInstance, HANDLE prevInstance, LPSTR commandLine, INT showMode);
        // Do one-time-only initialization.
     VOID InitProgram (VOID);
          // Do global program initialization.
     VOID InitInstance (LPSTR commandLine, INT showMode);
          // Initialize this instance of program.
       VOID InitWindow (LPSTR commandLine, INT showMode);
            // Initialize main window.
```

```
VOID InitFileParams (VOID);
     // Initialize parameters for file dialogs.
```

```
      VOID InitTextFormat (VOID);
            // Initialize text format.
      VOID InitColorParams (VOID);
            // Initialize parameters for color dialog.
      VOID InitAccelerators (VOID);
            // Initialize accelerator table.
VOID MainLoop (VOID);
     // Execute one pass of main program loop.
LONG CALLBACK DoMessage (HWND thisWindow, UINT msgCode, WPARAM wParam, LPARAM lParam);
     // Get and process one message.
   VOID DoCreate (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_CREATE message.
   VOID DoSize (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_SIZE message.
   LONG DoCtlColorEdit (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_CTLCOLOREDIT message.
   VOID DoSysColorChange (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_SYSCOLORCHANGE message.
   VOID DoSetFocus (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_SETFOCUS message.
   VOID DoInitMenuPopup (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_INITMENUPOPUP message.
      VOID FixSystemMenu (VOID);
         // Enable/disable system menu commands.
      VOID FixFileMenu (HMENU theMenu);
         // Enable/disable File menu commands.
      VOID FixEditMenu (HMENU theMenu);
         // Enable/disable Edit menu commands.
      VOID FixFormatMenu (HMENU theMenu);
         // Enable/disable Format menu commands.
      VOID FixHelpMenu (HMENU theMenu);
         // Enable/disable Help menu commands.
   VOID DoCommand (HWND thisWindow, WPARAM wParam, LPARAM lParam);
         // Handle WM_COMMAND message.
      VOID DoMenuCommand (UINT itemID);
            // Handle menu command.
         VOID DoNew (VOID);
               // Handle New command.
         VOID DoOpen (VOID);
               // Handle Open... command.
            VOID ReadDoc (LPSTR newPath);
                  // Read document into window.
         VOID DoClose (VOID);
               // Handle Close command.
            BOOL CloseDoc (VOID);
                  // Close document displayed in window.
         VOID DoSave (VOID);
               // Handle Save command.
         VOID DoSaveAs (VOID);
               // Handle Save As... command.
            VOID WriteDoc (VOID);
                  // Write window contents to a file.
         VOID DoRevert (VOID);
```

```
// Handle Revert to Saved... command.
```

```
        VOID DoSetup (VOID);
             // Handle Page Setup... command.
        VOID DoPrint (VOID);
             // Handle Print... command.
        VOID DoExit (VOID);
             // Handle Exit command.
        VOID DoUndo (VOID);
             // Handle Undo command.
        VOID DoCut (VOID);
             // Handle Cut command.
        VOID DoCopy (VOID);
             // Handle Copy command.
        VOID DoPaste (VOID);
             // Handle Paste command.
        VOID DoDelete (VOID);
             // Handle Delete command.
        VOID DoSelectAll (VOID);
             // Handle Select All command.
        VOID DoFormat (VOID);
             // Handle Format... command.
        VOID DoDefaultFormat (VOID);
             // Handle Default Format command.
          VOID SetFormat (VOID);
               // Set new text format.
        VOID DoBackground (VOID);
             // Handle Background Color... command.
          VOID SetBackground (VOID);
               // Set new background color.
        VOID DoHelp (VOID);
             // Handle Help command.
        VOID DoAbout (VOID);
             // Handle About WiniEdit... command.
          BOOL CALLBACK AboutProc (HWND thisWindow, UINT msgCode, WPARAM wParam, LPARAM lParam);
               // Dialog procedure for About WiniEdit... dialog.
      VOID DoNotification (UINT itemID, HWND theControl, UINT notifyCode);
           // Handle control notification.
   BOOL DoQuery (HWND thisWindow, UINT wParam, LONG lParam);
        // Handle WM_QUERYENDSESSION message.
   VOID DoDestroy (HWND thisWindow, UINT wParam, LONG lParam);
        // Handle WM_DESTROY message.
VOID Finalize (VOID);
     // Do one-time-only finalization.
   INT ShowUserMessage (UINT msgID, UINT iconStyle, UINT buttonStyle, LPSTR msgTitle, LPSTR argString);
        // Display user message on screen.
   VOID IOCheck (VOID);
        // Check for I/O error.
```

This is the main code file, **WiniEdit.cpp**, containing the C++ source code for the WiniEdit program:

```
//
//                                          WiniEdit
//                                  Example Windows application program
//                                 S. Chernicoff         15 January 1995
//

//                       Sample program to display and manipulate a window on the screen


   #include <windows.h>
   #include "WiniEdit.h"
   #include "WiniEdit Resources.h"

//---------------------------------------------------------------------------------------------------------------
--

INT CALLBACK WinMain (HINSTANCE instHandle, HINSTANCE prevHandle, LPSTR commandLine, INT showMode)

   // Main function

   {

      Initialize (instHandle, prevHandle, commandLine, showMode);  // Do one-time-only initialization


      do
         MainLoop ();                              // Execute one pass of main loop
      while ( ContinueFlag );                         // Continue until time to quit


      Finalize ();                                 // Do one-time-only finalization


      return NO_ERROR;                             // Signal successful completion


   }  /* end WinMain */

//---------------------------------------------------------------------------------------------------------------
--

VOID PASCAL Initialize (HANDLE instHandle, HANDLE prevHandle, LPSTR commandLine, INT showMode)

   // Do one-time-only initialization.

   {
      ThisInstance = instHandle;                     // Save global instance handle
```

```
      PrevInstance = prevHandle;                    // Save previous instance handle

      InitProgram ();                              // Do global program initialization
      InitInstance (commandLine, showMode);         // Initialize this instance of program

   }  /* end Initialize */

//-----------------------------------------------------------------------------------------------------------
--

VOID InitProgram (VOID)

   // Do global program initialization.

   {
      WNDCLASS  windowClass;                              // Window class
      LPSTR     resourceID;                               // Resource ID in string form
      HICON     progIcon;                                 // Program's screen icon
      HCURSOR   arrowCursor;                              // Default cursor
      HBRUSH    bkBrush;                                  // Brush for painting window background


      if ( PrevInstance == NULL )                         // Is this the first instance of program?
         {
            LoadString (ThisInstance, ProgName_Str,       // Get program name from resource
                        ProgName, ClassNameMax);
            windowClass.lpszClassName = ProgName;         // Use as class name

            windowClass.lpfnWndProc = WNDPROC(DoMessage);  // Set window procedure
            windowClass.hInstance   = ThisInstance;       // Current instance is the owner

            windowClass.style = CS_HREDRAW | CS_VREDRAW;  // Redraw on horizontal or vertical resize

            resourceID = MAKEINTRESOURCE(Main_Menu);      // Convert resource ID
            windowClass.lpszMenuName = resourceID;        // Set menu

            resourceID = MAKEINTRESOURCE(ProgIcon_ID);    // Convert resource ID
            progIcon   = LoadIcon(ThisInstance, resourceID);  // Load icon
            windowClass.hIcon = progIcon;                 // Set icon

            arrowCursor = LoadCursor(NULL, IDC_ARROW);    // Load standard arrow cursor
            windowClass.hCursor = arrowCursor;            // Set cursor

            bkBrush = HBRUSH(COLOR_WINDOW + 1);           // Create brush for system background color
            windowClass.hbrBackground = bkBrush;          // Set window background brush

            windowClass.cbClsExtra = 0;                   // No extra class data
            windowClass.cbWndExtra = 0;                   // No extra window data

            RegisterClass (&windowClass);                 // Register the class

         }  /* end if ( PrevInstance == NULL ) */
```

```
}  /* end InitProgram */
```

```
//-----------------------------------------------------------------------------------------------------------
--

VOID InitInstance (LPSTR commandLine, INT showMode)

   // Initialize this instance of program.

   {
      InitWindow (commandLine, showMode);            // Initialize main window

      InitFileParams  ();                          // Initialize parameters for file dialogs
      InitTextFormat  ();                          // Initialize text format
      InitColorParams ();                          // Initialize parameters for color dialog

      InitAccelerators ();                           // Initialize accelerator table

   }  /* end InitInstance */

//-----------------------------------------------------------------------------------------------------------
--

VOID InitWindow (LPSTR commandLine, INT showMode)

   // Initialize main window.

   {
      DWORD  windowStyle;                          // Style options for main window


      LoadString (ThisInstance, NoTitle_Str, NoNameTitle, TitleMax);  // Get default title from resource


      windowStyle = WS_OVERLAPPEDWINDOW |            // Use standard window style
                    WS_CLIPCHILDREN;               // Don't overwrite children

      TheWindow = CreateWindow (ProgName,            // Use program name for class name
                                NoNameTitle,         // Use default title for window with no associated file
                                windowStyle,         // Standard window with children clipped out
                                CW_USEDEFAULT,       // Let Windows choose initial x
                                CW_USEDEFAULT,       //   and y position
                                CW_USEDEFAULT,       // Let Windows choose initial width
                                CW_USEDEFAULT,       //   and height
                                NULL,                // No parent window
                                NULL,                // Use menu from window class
                                ThisInstance,        // Current program instance is the owner
                                NULL);               // No special creation parameters


      if ( commandLine[0] != '\0' )                  // Is there an initial file to open?
         ReadDoc (commandLine);                    // Read it into the window


      ShowWindow   (TheWindow, showMode);            // Display window on screen
```

```
UpdateWindow (TheWindow);                    // Force update of client area
```

```
   }  /* end InitWindow */

//-----------------------------------------------------------------------------------------------------------------
--

VOID InitFileParams (VOID)

   // Initialize parameters for file dialogs.

   {
      INT   filterLength;                       // Length of filter string
      INT   charIndex;                          // Index into filter string
      CHAR  dummyChar;                          // Character used as placeholder in filter string


      FileParams.lStructSize = sizeof(OPENFILENAME); // Size of data structure

      FileParams.hwndOwner = TheWindow;         // Owning window
      FileParams.hInstance = ThisInstance;      // Owning instance of program

      FileParams.lpstrTitle = NULL;                // Use standard title for dialog window
      FileParams.Flags      = OFN_CREATEPROMPT |   // Prompt if file doesn't exist on open
                              OFN_OVERWRITEPROMPT; //    or already exists on save

      filterLength = LoadString(ThisInstance, FileFilter_Str,          // Get filter string
                                FilterString, FilterMax);
      dummyChar = FilterString[filterLength - 1];                      // Last character is placeholder
      for ( (charIndex = 0); (charIndex < filterLength); (charIndex++) ) // Step through filter string
         if ( FilterString[charIndex] == dummyChar )                   // Find placeholders and
            FilterString[charIndex] = '\0';                            //    replace them with nulls

      FileParams.lpstrFilter  = FilterString;      // Point to filter string
      FileParams.nFilterIndex = 1;                 // Select first filter in string

      FileParams.lpstrInitialDir = NULL;           // Start with current directory

      LoadString (ThisInstance, FileExt_Str,       // Get default extension from resource
                  FileExt, FileExtMax);
      FileParams.lpstrDefExt = FileExt;         // Set default extension

      NewPath[0] = '\0';                        // Start with null path name
      FileParams.lpstrFile = NewPath;              // Pointer to path buffer
      FileParams.nMaxFile  = PathMax;              // Length of path buffer

      NewTitle[0] = '\0';                       // Start with null file title
      FileParams.lpstrFileTitle = NewTitle;        // Pointer to title buffer
      FileParams.nMaxFileTitle  = TitleMax;        // Length of title buffer

      FileParams.nFileOffset    = 0;            // File and extension offsets
      FileParams.nFileExtension = 0;            //    will be returned at time of call

      FileParams.lpTemplateName    = NULL;      // No custom dialog template
      FileParams.lpfnHook          = NULL;      // No custom hook function
```

```
FileParams.lCustData         = 0;           // Data for custom hook not needed
FileParams.lpstrCustomFilter = NULL;        // No custom filter buffer
```

```
      FileParams.nMaxCustFilter    = 0;           // Bufferlength is zero

   }  /* end InitFileParams */

//-----------------------------------------------------------------------------------------------------------
--

VOID InitTextFormat (VOID)

   // Initialize text format.

   {

      DeviceFont = GetStockObject(DEVICE_DEFAULT_FONT);       // Get standard device font
      GetObject (DeviceFont, sizeof(TextFormat), &TextFormat);  // Initialize logical font characteristics
      TheFont = CreateFontIndirect(&TextFormat);              // Save as current font


      FontParams.lStructSize = sizeof(CHOOSEFONT);            // Size of data structure

      FontParams.hwndOwner = TheWindow;                     // Owning window
      FontParams.hInstance = ThisInstance;                  // Owning instance of program

      FontParams.Flags = CF_INITTOLOGFONTSTRUCT |             // Initialize dialog from logical font
                         CF_SCREENFONTS |                     // List screen fonts only
                         CF_EFFECTS;                        // Enable effects

      FontParams.lpLogFont = &TextFormat;                     // Use for initial font
      FontParams.nFontType = 0;                               // Font type will be returned at time of call

      FontParams.iPointSize = 0;                              // Point size will be returned at time of call
      FontParams.nSizeMin   = 0;                              // No minimum point size
      FontParams.nSizeMax   = 0;                              // No maximum point size

      TextColor = 0;                                          // Default to system text color
      FontParams.rgbColors = GetSysColor(COLOR_WINDOWTEXT);     // Start with system text color

      FontParams.hDC            = NULL;                     // No printer context
      FontParams.lpszStyle      = NULL;                     // No style buffer
      FontParams.lpTemplateName = NULL;                     // No custom dialog template
      FontParams.lpfnHook       = NULL;                     // No custom hook function
      FontParams.lCustData      = 0;                        // Data for custom hook not needed


   }  /* end InitTextFormat */

//-----------------------------------------------------------------------------------------------------------
--

VOID InitColorParams (VOID)

   // Initialize parameters for color dialog.
```

```
{
   ColorParams.lStructSize = sizeof(CHOOSECOLOR); // Size of data structure
```

```
      ColorParams.hwndOwner = TheWindow;              // Owning window
      ColorParams.hInstance = ThisInstance;           // Owning instance of program

      ColorParams.Flags = CC_RGBINIT;                 // Use initial color setting

      BackgroundColor = 0;                                  // Default to system background color
      ColorParams.rgbResult    = GetSysColor(COLOR_WINDOW);   // Start with system background color
      ColorParams.lpCustColors = UserColors;                // Pointer to custom color array

      ColorParams.lpTemplateName = NULL;              // No custom dialog template
      ColorParams.lpfnHook       = NULL;              // No custom hook function
      ColorParams.lCustData      = 0;                 // Data for custom hook not needed

   }  /* end InitColorParams */

//-----------------------------------------------------------------------------------------------------------
--

VOID InitAccelerators (VOID)

   // Initialize accelerator table.

   {
      LPSTR resourceID;                                     // Resource ID in string form


      resourceID = MAKEINTRESOURCE(Accel_ID);               // Convert resource ID
      AccelTable = LoadAccelerators(ThisInstance, resourceID);  // Load accelerator table

   }  /* end InitAccelerators */

//-----------------------------------------------------------------------------------------------------------
--

VOID MainLoop (VOID)

   // Execute one pass of main program loop.

   {
      MSG   theMessage;                             // Next message to process
      BOOL  translated;                             // Was message translated as a keyboard accelerator?


      ContinueFlag = GetMessage(&theMessage, NULL, 0, 0);  // Get next message

      translated = TranslateAccelerator (TheWindow, AccelTable, &theMessage); // Check for keyboard
accelerator

      if ( !translated )                          // Was the message an accelerator?
         {
            TranslateMessage (&theMessage);           // Convert virtual keys to characters
            DispatchMessage  (&theMessage);           // Send message to window procedure
```

```
    }  /* end if ( !translated ) */

}  /* end MainLoop */
```

```
//-----------------------------------------------------------------------------------------------------------
--

LONG CALLBACK DoMessage (HWND thisWindow, UINT msgCode, WPARAM wParam, LPARAM lParam)

   // Get and process one message.

   {
      LONG  result = 0;                                    // Function result


      ErrorFlag = FALSE;                                  // Clear I/O error flag


      switch ( msgCode )                                  // Dispatch on message code
         {
            case WM_CREATE:
               DoCreate (thisWindow, wParam, lParam);          // Handle WM_CREATE message
               break;

            case WM_SIZE:
               DoSize (thisWindow, wParam, lParam);         // Handle WM_SIZE message
               break;

            case WM_CTLCOLOREDIT:
               result = DoCtlColorEdit (thisWindow, wParam, lParam);     // Handle WM_CTLCOLOREDIT message
               break;

            case WM_SYSCOLORCHANGE:
               DoSysColorChange (thisWindow, wParam, lParam); // Handle WM_SYSCOLORCHANGE message
               break;

            case WM_SETFOCUS:
               DoSetFocus (thisWindow, wParam, lParam);       // Handle WM_SETFOCUS message
               break;

            case WM_INITMENUPOPUP:
               DoInitMenuPopup (thisWindow, wParam, lParam);  // Handle WM_INITMENUPOPUP message
               break;

            case WM_COMMAND:
               DoCommand (thisWindow, wParam, lParam);        // Handle WM_COMMAND message
               break;

            case WM_QUERYENDSESSION:
               result = DoQuery (thisWindow, wParam, lParam); // Handle WM_QUERYENDSESSION message
               break;

            case WM_CLOSE:
               DoClose ();                                 // Handle WM_CLOSE message
               break;

            case WM_DESTROY:
```

```
  DoDestroy (thisWindow, wParam, lParam);        // Handle WM_DESTROY message
  break;
```

```
         default:
            result = DefWindowProc (thisWindow, msgCode,    // Pass message to Windows
                                    wParam, lParam);      //    for default processing
            break;

      }  /* end switch ( msgCode ) */


   return result;


}  /* end DoMessage */

//-------------------------------------------------------------------------------------------------------------
--

VOID DoCreate (HWND thisWindow, WPARAM wParam, LPARAM lParam)

   // Handle WM_CREATE message.

   {
      DWORD     editStyle;                        // Style options for edit control
      COLORREF  bkColor;                        // Color for window background


      editStyle = WS_CHILD |                     // Child window
                  WS_VISIBLE |                   //    visible on screen
                  WS_VSCROLL |                   //    vertical scroll bar
                  ES_AUTOVSCROLL |               //    vertical autoscroll
                  ES_MULTILINE |                    //    multiple lines of text
                  ES_LEFT;                       //    flush-left alignment

      TheEditor = CreateWindow ("EDIT",          // Standard edit control
                                NULL,            // No title
                                editStyle,       // Style options as above
                                0, 0, 0, 0,      // Position and size will be set later
                                thisWindow,      // Main window is the parent
                                HMENU(Edit_Control),// Child identifier for notification messages
                                ThisInstance,    // Current program instance is the owner
                                NULL);           // No special creation parameters


      lstrcpy (FileTitle, NoNameTitle);          // Set default file title


      SendMessage (TheEditor, WM_SETFONT,           // Set to current text font
                   WPARAM(TheFont), 0);

      if ( BackgroundColor != 0 )                // Is there an explicit background color?
         bkColor = BackgroundColor;                 // Use it
      else
         bkColor = GetSysColor(COLOR_WINDOW);    // Otherwise use system background color
```

```
BackgroundBrush = CreateSolidBrush(bkColor);    // Create background brush
```

```
   }  /* end DoCreate */

//-------------------------------------------------------------------------------------------------------------
--

VOID DoSize (HWND thisWindow, WPARAM wParam, LPARAM lParam)

   // Handle WM_SIZE message.

   {
      INT     newWidth;                                  // New width of client area
      INT     newHeight;                                 // New height of client area
      RECT    textRect;                                  // Formatting rectangle for wrapping text
      LPARAM  rectParam;                                // Pointer to rectangle as long-word parameter


      newWidth  = LOWORD(lParam);                       // Extract new dimensions
      newHeight = HIWORD(lParam);                       //    from message parameter

      MoveWindow (TheEditor,                            // Resize edit control to fit
                  0, 0,
                  newWidth, newHeight,
                  TRUE);

      rectParam = LPARAM(&textRect);                    // Convert to long integer
      SendMessage (TheEditor, EM_GETRECT, 0, rectParam);   // Get formatting rectangle
      InflateRect (&textRect, -TextMargin, -TextMargin);   // Inset by text margin
      SendMessage (TheEditor, EM_SETRECT, 0, rectParam);   // Set new rectangle

   }  /* end DoSize */

//-------------------------------------------------------------------------------------------------------------
--

LONG DoCtlColorEdit (HWND thisWindow, WPARAM wParam, LPARAM lParam)

   // Handle WM_CTLCOLOREDIT message.

   {
      HDC       theContext;                          // Handle to device context
      COLORREF  txColor;                             // Color for displaying text
      COLORREF  bkColor;                             // Color for window background


      theContext = HDC(wParam);                        // Get device context from parameter


      if ( TextColor != 0 )                          // Is there an explicit text color?
         txColor = TextColor;                          // Use it
      else
         txColor = GetSysColor(COLOR_WINDOWTEXT);     // Otherwise use system text color
```

```
if ( BackgroundColor != 0 )                    // Is there an explicit background color?
   bkColor = BackgroundColor;                     // Use it
else
```

```
      bkColor = GetSysColor(COLOR_WINDOW);      // Otherwise use system background color


   SetTextColor (theContext, txColor);            // Set text color
   SetBkColor   (theContext, bkColor);            // Set background color


   return LONG(BackgroundBrush);                  // Return window's background brush


} /* end DoCtlColorEdit */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoSysColorChange (HWND thisWindow, WPARAM wParam, LPARAM lParam)

   // Handle WM_SYSCOLORCHANGE message.

   {
      COLORREF  txColor;                          // New system text color
      COLORREF  bkColor;                          // New system background color


      txColor = GetSysColor(COLOR_WINDOWTEXT);       // Get new text color
      bkColor = GetSysColor(COLOR_WINDOW);        // Get new background color


      if ( TextColor == 0 )                       // Is text defaulted to system color?
         FontParams.rgbColors = txColor;             // Set to new system text color
      else if ( txColor == TextColor )               // Does new color match existing setting?
         TextColor = 0;                              // Default back to system color


      if ( BackgroundColor == 0 )                 // Is background defaulted to system color?
         {
            ColorParams.rgbResult = bkColor;         // Set to new system background color

            DeleteObject (BackgroundBrush);             // Destroy old background brush
            BackgroundBrush = CreateSolidBrush(bkColor); // Create new brush

         } /* end if ( BackgroundColor == 0 ) */
      else if ( bkColor == BackgroundColor )         // Does new color match existing setting?
         BackgroundColor = 0;                        // Default back to system color


      InvalidateRect (TheWindow, NULL, TRUE);        // Force repaint


   } /* end DoSysColorChange */

//-----------------------------------------------------------------------------------------------------------
--
```

```
VOID DoSetFocus (HWND thisWindow, WPARAM wParam, LPARAM lParam)
```

```
   // Handle WM_SETFOCUS message.

   {
      SetFocus (TheEditor);                        // Pass focus to the edit control

   }  /* end DoSetFocus */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoInitMenuPopup (HWND thisWindow, WPARAM wParam, LPARAM lParam)

   // Handle WM_INITMENUPOPUP message.

   {
      HMENU  theMenu   = HMENU(wParam);            // Handle to menu to be adjusted
      UINT   menuIndex = LOWORD(lParam);              // Relative position of menu in menu bar
      BOOL   isSystem  = HIWORD(lParam);              // Is it the system menu?


      if ( isSystem )                              // Is it the system menu?
         FixSystemMenu ();                            // Enable/disable system menu commands
      else
         switch ( menuIndex )
            {
               case File_Menu:
                  FixFileMenu (theMenu);           // Enable/disable File menu commands
                  break;

               case Edit_Menu:
                  FixEditMenu (theMenu);           // Enable/disable Edit menu commands
                  break;

               case Format_Menu:
                  FixFormatMenu (theMenu);         // Enable/disable Format menu commands
                  break;

               case Help_Menu:
                  FixHelpMenu (theMenu);           // Enable/disable Help menu commands
                  break;

            }  /* end switch ( menuIndex ) */

   }  /* end DoInitMenuPopup */

//-----------------------------------------------------------------------------------------------------------
--

VOID FixSystemMenu (VOID)

   // Enable/disable system menu commands.

   {
```

```
   /* Nothing to do */

}  /* end FixSystemMenu */
```

```
//-----------------------------------------------------------------------------------------------------------
--

VOID FixFileMenu (HMENU theMenu)

   // Enable/disable File menu commands.

   {
      BOOL  dirty;                                  // Have window's contents been changed since last save?


      EnableMenuItem (theMenu, New_Item,    MF_ENABLED);           // These commands are always available
      EnableMenuItem (theMenu, Open_Item,   MF_ENABLED);
      EnableMenuItem (theMenu, Close_Item,  MF_ENABLED);
      EnableMenuItem (theMenu, SaveAs_Item, MF_ENABLED);
      EnableMenuItem (theMenu, Exit_Item,   MF_ENABLED);

      EnableMenuItem (theMenu, Setup_Item, MF_GRAYED);          // Printing not implemented
      EnableMenuItem (theMenu, Print_Item, MF_GRAYED);


      dirty = SendMessage(TheEditor, EM_GETMODIFY, 0, 0);       // Ask if text has been edited
      if ( dirty )                                             // Has it been?
         {
            EnableMenuItem (theMenu, Save_Item, MF_ENABLED);    // Enable Save command

            if ( TheFile != NULL )                             // Is window associated with a file?
               EnableMenuItem (theMenu, Revert_Item, MF_ENABLED);   // Enable Revert to Saved... command
            else
               EnableMenuItem (theMenu, Revert_Item, MF_GRAYED); // If no file, gray out Revert

         }  /* end if ( dirty ) */
      else
         {
            EnableMenuItem (theMenu, Save_Item,   MF_GRAYED);    // If text not dirty,
            EnableMenuItem (theMenu, Revert_Item, MF_GRAYED);    //    gray out Save and Revert

         }  /* end else */


   }  /* end FixFileMenu */

//-----------------------------------------------------------------------------------------------------------
--

VOID FixEditMenu (HMENU theMenu)

   // Enable/disable Edit menu commands.

   {
      BOOL  canUndo;                                  // Can editor support Undo command?
      LONG  selStart;                                // Character position at start of selection
```

```
LONG  selEnd;                    // Character position at end of selection
BOOL  canPaste;                  // Is there text on the clipboard?
LONG  textLength;                   // Number of characters in document
```

```
   canUndo = SendMessage(TheEditor, EM_CANUNDO, 0, 0);     // Is Undo available?
   if ( canUndo )
      EnableMenuItem (theMenu, Undo_Item, MF_ENABLED);     // Enable Undo command
   else
      EnableMenuItem (theMenu, Undo_Item, MF_GRAYED);      // Gray out Undo command


   SendMessage ( TheEditor, EM_GETSEL,                     // Get selection range
                 WPARAM(&selStart), LPARAM(&selEnd) );
   if ( selStart != selEnd )                               // Is there a selection?
      {
         EnableMenuItem (theMenu, Cut_Item,    MF_ENABLED);   // Enable Cut command
         EnableMenuItem (theMenu, Copy_Item,   MF_ENABLED);   // Enable Copy command
         EnableMenuItem (theMenu, Delete_Item, MF_ENABLED);   // Enable Delete command

      }  /* end if ( selStart == selEnd ) */
   else
      {
         EnableMenuItem (theMenu, Cut_Item,    MF_GRAYED); // Gray out Cut command
         EnableMenuItem (theMenu, Copy_Item,   MF_GRAYED); // Gray out Copy command
         EnableMenuItem (theMenu, Delete_Item, MF_GRAYED); // Gray out Delete command

      }  /* end else */


   canPaste = IsClipboardFormatAvailable(CF_TEXT);         // Does clipboard contain text?
   if ( canPaste )
      EnableMenuItem (theMenu, Paste_Item, MF_ENABLED);    // Enable Paste command
   else
      EnableMenuItem (theMenu, Paste_Item, MF_GRAYED);     // Gray out Paste command


   textLength = GetWindowTextLength(TheEditor);            // Get length of text
   if ( textLength > 0 )                                 // Any text in document?
      EnableMenuItem (theMenu, SelectAll_Item, MF_ENABLED);   // Enable Select All command
   else
      EnableMenuItem (theMenu, SelectAll_Item, MF_GRAYED); // Gray out Select All command


}  /* end FixEditMenu */

//-----------------------------------------------------------------------------------------------------------
--

VOID FixFormatMenu (HMENU theMenu)

   // Enable/disable Format menu commands.

   {
      EnableMenuItem (theMenu, Format_Item,     MF_ENABLED);  // Commands always enabled
      EnableMenuItem (theMenu, Default_Item,    MF_ENABLED);
```

```
EnableMenuItem (theMenu, Background_Item, MF_ENABLED);
```

```
   }  /* end FixFormatMenu */

//-----------------------------------------------------------------------------------------------------------------
--

VOID FixHelpMenu (HMENU theMenu)

   // Enable/disable Help menu commands.

   {
      EnableMenuItem (theMenu, Help_Item,  MF_GRAYED);  // Gray out Help... command
      EnableMenuItem (theMenu, About_Item, MF_ENABLED); // Enable About WiniEdit... command

   }  /* end FixHelpMenu */

//-----------------------------------------------------------------------------------------------------------------
--

VOID DoCommand (HWND thisWindow, WPARAM wParam, LPARAM lParam)

   // Handle WM_COMMAND message.

   {
      UINT  notifyCode;                              // Notification code from child control
      UINT  itemID;                                 // Item ID of message originator
      HWND  theControl;                              // Handle to control sending notification


      notifyCode = HIWORD(wParam);                   // Extract notification code
      itemID     = LOWORD(wParam);                   // Extract item ID
      theControl = HWND(lParam);                     // Get control handle

      switch ( notifyCode )
         {
            case 0:
            case 1:
               DoMenuCommand (itemID);                          // Handle menu command
               break;

            default:
               DoNotification (itemID, theControl, notifyCode);  // Handle control notification
               break;

         }  /* end switch ( notifyCode ) */

   }  /* end DoCommand */

//-----------------------------------------------------------------------------------------------------------------
--

VOID DoMenuCommand (UINT itemID)

   // Handle menu command.
```

```
{
   switch ( itemID )
      {
```

```
/* File menu */

case New_Item:
   DoNew ();                          // Handle New command
   break;

case Open_Item:
   DoOpen ();                         // Handle Open... command
   break;

case Close_Item:
   DoClose ();                        // Handle Close command
   break;

case Save_Item:
   DoSave ();                         // Handle Save command
   break;

case SaveAs_Item:
   DoSaveAs ();                       // Handle Save As... command
   break;

case Revert_Item:
   DoRevert ();                       // Handle Revert to Saved... command
   break;

case Setup_Item:
   DoSetup ();                        // Handle Page Setup... command
   break;

case Print_Item:
   DoPrint ();                        // Handle Print... command
   break;

case Exit_Item:
   DoExit ();                         // Handle Exit command
   break;


/* Edit menu */

case Undo_Item:
   DoUndo ();                         // Handle Undo command
   break;

case Cut_Item:
   DoCut ();                          // Handle Cut command
   break;

case Copy_Item:
   DoCopy ();                         // Handle Copy command
   break;
```

```
        case Paste_Item:
           DoPaste ();                          // Handle Paste command
           break;

        case Delete_Item:
           DoDelete ();                         // Handle Delete command
           break;

        case SelectAll_Item:
           DoSelectAll ();                      // Handle Select All command
           break;


        /* Format menu */

        case Format_Item:
           DoFormat ();                         // Handle Text Format... command
           break;

        case Default_Item:
           DoDefaultFormat ();                  // Handle Default Format command
           break;

        case Background_Item:
           DoBackground ();                     // Handle Background Color... command
           break;


        /* Help menu */

        case Help_Item:
           DoHelp ();                           // Handle Help command
           break;

        case About_Item:
           DoAbout ();                          // Handle About WiniEdit... command
           break;



        default:
           MessageBeep (MB_OK);                   // Error: control should never reach this point
           break;


      }  /* end switch ( itemID ) */


   }  /* end DoMenuCommand */

//-------------------------------------------------------------------------------------------------------
--
```

```
VOID DoNew (VOID)
```

```
   // Handle New command.

   {
      BOOL  confirmed;                          // Did user confirm operation?


      confirmed = CloseDoc ();                  // Allow user to save document if necessary
      if ( confirmed )                          // Did user confirm?
         {
            TheFile     = NULL;                 // Clear current file
            PathName[0] = '\0';                 // Clear path name
            lstrcpy (FileTitle, NoNameTitle);   // Set default file title

            SetWindowText (TheWindow, NoNameTitle);   // Set window title to default

         }  /* end if ( confirmed ) */

   }  /* end DoNew */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoOpen (VOID)

   // Handle Open... command.

   {
      BOOL    confirmed;                        // Did user confirm operation?


      confirmed = GetOpenFileName (&FileParams);     // Get file name from user

      if ( confirmed )                          // Did user confirm file selection?
         ReadDoc (NewPath);                     // Open file and read into window

   }  /* end DoOpen */

//-----------------------------------------------------------------------------------------------------------
--

VOID ReadDoc (LPSTR *newPath)

   // Read document into window.

   {
      BOOL    confirmed;                        // Did user confirm operation?
      HANDLE  newFile;                          // Handle to new file


      confirmed = CloseDoc ();                  // Allow user to save previous document if necessary
      if ( confirmed )                          // Did user confirm?
         {
            newFile = CreateFile (newPath,                    // Open new file
```

```
  GENERIC_READ | GENERIC_WRITE,  // Read-write access
FILE_SHARE_READ,                // Share for reading only
NULL,                         // No security attributes
```

```
                        OPEN_ALWAYS,                  // Open existing file or create a new one
                        FILE_ATTRIBUTE_NORMAL,           // No special attributes
                        NULL);                        // No template file

         if ( newFile == INVALID_HANDLE_VALUE )           // Did operation fail?
            IOCheck ();                               // Handle error
         else
            SetLastError (NO_ERROR);                  // Clear possible "already exists" error

         if ( ErrorFlag )                             // Was there an error?
            {
               TheFile      = NULL;                     // Window is left with no file:
               PathName[0]  = '\0';                     //    clear global file info
               FileTitle[0] = '\0';

               SetWindowText (TheWindow, NoNameTitle);  // Set window title to default

            }  /* end if ( ErrorFlag ) */
         else
            {
               TheFile = newFile;                     // Make new file current
               lstrcpy (PathName,  newPath);             // Save path name
               GetFileTitle (PathName, FileTitle, TitleMax);  // Extract local file name

               SetWindowText (TheWindow, FileTitle);     // Local file name becomes window title
               DoRevert ();                           // Read file into window

            }  /* end else */

      }  /* end if ( confirmed ) */


   }  /* end ReadDoc */

//---------------------------------------------------------------------------------------------------------
--

VOID DoClose (VOID)

   // Handle Close command.

   {
      BOOL  confirmed;                        // Did user confirm operation?


      confirmed = CloseDoc ();                // Allow user to save document if necessary
      if ( confirmed )                        // Did user confirm?
         DestroyWindow (TheWindow);              // Destroy the window

   }  /* end DoClose */

//---------------------------------------------------------------------------------------------------------
--
```

```
BOOL CloseDoc (VOID)
```

```
// Close document displayed in window.

{
   BOOL  dirty;                              // Have window's contents been changed since last save?
   INT   msgResult;                          // Result value returned by message box
   BOOL  confirmed;                          // Did user confirm operation?


   dirty = SendMessage(TheEditor, EM_GETMODIFY, 0, 0);  // Ask if text has been edited
   if ( dirty )
      {
         msgResult = ShowUserMessage (Save_Msg,         // Ask user whether to save window contents
                                      MB_ICONQUESTION,  // Question-mark icon
                                      MB_YESNOCANCEL,   // Yes, No, and Cancel buttons
                                    "???",           // Title for message box
                                      FileTitle);       // Merge file title into message

         switch ( msgResult )                     // Dispatch on message result
            {
               case IDYES:
                  DoSave ();                    // Save window contents to disk
                  confirmed = !ErrorFlag;          // Confirm if no error
                  break;

               case IDNO:
                  confirmed = TRUE;                // Confirm without saving
                  break;

               case IDCANCEL:
                  confirmed = FALSE;            // Cancel operation
                  break;

            }  /* end switch ( msgResult ) */

      }  /* end if ( dirty ) */

   else
      confirmed = TRUE;                           // Confirm if not dirty


   if ( confirmed )                             // Did user confirm operation?
      {
         if ( (TheFile != NULL) )               // Is window associated with a file?
            {
               CloseHandle (TheFile);           // Close file
               IOCheck ();                      // Check for I/O error

            }  /* end if ( (TheFile != NULL ) */

         SetWindowText (TheEditor, "");         // Clear edit control's text

      }  /* end if ( confirmed ) */
```

```
      return confirmed;                              // Report confirmation or cancellation


   }  /* end CloseDoc */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoSave (VOID)

   // Handle Save command.

   {
      if ( TheFile == NULL )                        // Is window associated with a file?
         DoSaveAs ();                               // Get file name from user
      else
         WriteDoc ();                               // Write to window's file

   }  /* end DoSave */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoSaveAs (VOID)

   // Handle Save As... command.

   {
      BOOL    confirmed;                            // Did user confirm dialog?
      HANDLE  newFile;                              // Handle to new file


      lstrcpy (NewPath,  PathName);                    // Start with current file name
      lstrcpy (NewTitle, FileTitle);


      confirmed = GetSaveFileName (&FileParams);     // Get file name from user
      if ( !confirmed )                                // Did user cancel?
         {
            ErrorFlag = TRUE;                          // Force exit to main message loop
            return;                                 // Skip rest of operation

         }  /* end if ( !confirmed ) */


      if ( lstrcmp(PathName, NewPath) != 0 )                   // Changing to a different file?
         {

            if ( TheFile != NULL )                           // Was there a previous file?
               {
                  CloseHandle (TheFile);                     // Close old file
                  IOCheck ();                                // Check for error
                  if (ErrorFlag) return;                     // On error, exit to main message loop
```

```
  }  /* end if ( TheFile != NULL ) */
```

```
      newFile = CreateFile (NewPath,                    // Open new file
                            GENERIC_READ | GENERIC_WRITE,  // Read-write access
                           FILE_SHARE_READ,               // Share for reading only
                           NULL,                          // No security attributes
                           CREATE_ALWAYS,                 // Overwrite existing file or create a new one
                           FILE_ATTRIBUTE_NORMAL,         // No special attributes
                           NULL);                         // No template file

      if ( newFile == INVALID_HANDLE_VALUE )              // Did operation fail?
         IOCheck ();                                      // Handle error
      else
         SetLastError (NO_ERROR);                         // Clear possible "already exists" error


      if ( ErrorFlag )                                    // Was there an error?
         {
            TheFile      = NULL;                          // Window is left with no file:
            PathName[0]  = '\0';                          //    clear global file info
            FileTitle[0] = '\0';

            SetWindowText (TheWindow, NoNameTitle);       // Set window title to default

            return;                                       // Exit to main message loop

         }  /* end if ( ErrorFlag ) */
      else
         {
            TheFile = newFile;                            // Make new file current
            lstrcpy (PathName,  NewPath);
            lstrcpy (FileTitle, NewTitle);

            SetWindowText (TheWindow, NewTitle);          // Local file name becomes window title

         }  /* end else */


   }  /* end if ( lstrcmp(PathName, NewPath) != 0 ) */


   WriteDoc ();                                           // Write window's contents to file


}  /* end DoSaveAs */


//-------------------------------------------------------------------------------------------------
--

VOID WriteDoc (VOID)

   // Write window contents to a file.
```

```
{
   CHAR   textBuffer[DocMax];                    // Buffer to hold text of document
```

```
      INT    textLength;                          // Length of text in bytes
      LONG   textAddr;                            // Address of text buffer
      ULONG  bytesWritten;                           // Number of bytes written to file


      textLength = GetWindowTextLength(TheEditor);    // Get length of text
      textLength++;                               // Adjust for terminating null character
      textAddr = LPARAM(textBuffer);              // Convert to long parameter
      SendMessage (TheEditor, WM_GETTEXT, textLength, textAddr);// Get text from edit control

      SetFilePointer (TheFile, 0, NULL, FILE_BEGIN); // Reset file pointer to beginning
      IOCheck ();                                 // Check for error
      if (ErrorFlag) return;                      // On error, exit to main message loop

      WriteFile (TheFile, textBuffer, --textLength,  // Write text to file
                 &bytesWritten, NULL);
      IOCheck ();                                 // Check for error
      if (ErrorFlag) return;                      // On error, exit to main message loop

      SetEndOfFile (TheFile);                        // Set length of file
      IOCheck ();                                 // Check for error
      if (ErrorFlag) return;                      // On error, exit to main message loop

      FlushFileBuffers (TheFile);                 // Flush buffer to disk
      IOCheck ();                                 // Check for error
      if (ErrorFlag) return;                      // On error, exit to main message loop

      SendMessage(TheEditor, EM_SETMODIFY, FALSE, 0);// Mark text as clean

   }  /* end WriteDoc */

//-------------------------------------------------------------------------------------------------------------------
--

VOID DoRevert (VOID)

   // Handle Revert to Saved... command.

   {
      BOOL   dirty;                               // Have window's contents been changed since last save?
      INT    msgResult;                              // Result value returned by message box
      CHAR   textBuffer[DocMax];                     // Buffer to hold text of document
      LONG   textLength;                          // Length of text in bytes
      LONG   textAddr;                            // Address of text buffer
      ULONG  bytesRead;                              // Number of bytes read from file


      dirty = SendMessage(TheEditor, EM_GETMODIFY, 0, 0);  // Ask if text has been edited
      if ( dirty )                                          // Has it been?
         {
            msgResult = ShowUserMessage (Revert_Msg,       // Display revert message
                                         MB_ICONQUESTION,  // Question-mark icon
                                         MB_OKCANCEL,      // OK and Cancel buttons
```

```
"???",           // Title for message box
  FileTitle);       // Merge file title into message
```

```
         if ( msgResult == IDCANCEL )                  // Did user cancel?
            return;                                   // Skip rest of operation

      }  /* end if ( dirty ) */

   textLength = GetFileSize(TheFile, NULL);       // Get length of file
   if ( textLength > DocMax )                     // File too long?
      {
         ShowUserMessage (TooLong_Msg,            // Display "file too long" message
                          MB_ICONEXCLAMATION,   // Exclamation-point icon
                         MB_OK,                 // OK button
                         NULL,                  // Use default title for message box
                          FileTitle);           // Merge file title into message

         CloseHandle (TheFile);                 // Close file
         ErrorFlag = TRUE;                         // Force exit

      }  /* end if ( textLength > DocMax ) */
   else
      IOCheck ();                               // Check for I/O error
   if (ErrorFlag) return;                       // On error, exit to main message loop

   SetFilePointer (TheFile, 0, NULL, FILE_BEGIN); // Move file pointer to beginning
   IOCheck ();                                  // Check for error
   if (ErrorFlag) return;                       // On error, exit to main message loop

   ReadFile (TheFile, textBuffer, textLength,     // Read text of file into block
             &bytesRead, NULL);
   textBuffer[textLength] = '\0';               // Add terminating null character
   IOCheck ();                                  // Check for error
   if (ErrorFlag) return;                       // On error, exit to main message loop

   textAddr = LPARAM(textBuffer);                  // Convert to long parameter
   SendMessage (TheEditor, WM_SETTEXT, 0, textAddr); // Copy file contents into edit control

}  /* end DoRevert */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoSetup (VOID)

   // Handle Page Setup... command.

   {
      MessageBeep (MB_OK);                          // Printing not implemented

   }  /* end DoSetup */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoPrint (VOID)
```

```
// Handle Print... command.
```

```
   {
      MessageBeep (MB_OK);                          // Printing not implemented

   }  /* end DoPrint */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoExit (VOID)

   // Handle Exit command.

   {
      SendMessage (TheWindow, WM_CLOSE, 0, 0);      // Close the window

   }  /* end DoExit */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoUndo (VOID)

   // Handle Undo command.

   {
      SendMessage (TheEditor, WM_UNDO, 0, 0);       // Relay operation to edit control

   }  /* end DoUndo */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoCut (VOID)

   // Handle Cut command.

   {
      SendMessage (TheEditor, WM_CUT, 0, 0);        // Relay operation to edit control

   }  /* end DoCut */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoCopy (VOID)

   // Handle Copy command.

   {
      SendMessage (TheEditor, WM_COPY, 0, 0);       // Relay operation to edit control

   }  /* end DoCopy */
```

```
//-----------------------------------------------------------------------------------------------------------
--

VOID DoPaste (VOID)

   // Handle Paste command.
```

```
   {
      SendMessage (TheEditor, WM_PASTE, 0, 0);       // Relay operation to edit control

   }  /* end DoPaste */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoDelete (VOID)

   // Handle Delete command.

   {
      SendMessage (TheEditor, WM_CLEAR, 0, 0);       // Relay operation to edit control

   }  /* end DoDelete */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoSelectAll (VOID)

   // Handle Select All command.

   {
      SendMessage (TheEditor, EM_SETSEL, 0, -1);     // Select entire document

   }  /* end DoSelectAll */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoFormat (VOID)

   // Handle Format... command.

   {
      BOOL  confirmed;                            // Did user confirm dialog?


      confirmed = ChooseFont (&FontParams);          // Get new format from user
      if ( confirmed )                            // Did user confirm?
         SetFormat ();                            // Set new text format

   }  /* end DoFormat */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoDefaultFormat (VOID)

   // Handle Default Format command.
```

```
{
   INT       msgResult;                    // Result value returned by message box
   COLORREF  txColor;                    // System text color
   COLORREF  bkColor;                    // System background color
```

```
   msgResult = ShowUserMessage (DefaultFormat_Msg,// Display default format message
                                MB_ICONQUESTION,  // Question-mark icon
                                MB_OKCANCEL,      // OK and Cancel buttons
                               "???",           // Title for message box
                                FileTitle);      // Merge file title into message


   if ( msgResult != IDCANCEL )                  // Did user confirm?
      {
         GetObject (DeviceFont, sizeof(TextFormat), &TextFormat);  // Get standard font characteristics

         txColor = GetSysColor(COLOR_WINDOWTEXT); // Get system text color
         bkColor = GetSysColor(COLOR_WINDOW);   // Get system background color

         FontParams.rgbColors  = txColor;         // Set text to system color
         ColorParams.rgbResult = bkColor;         // Set background to system color

         SetFormat ();                          // Set new text format
         SetBackground ();                         // Set new background color

      }  /* end if ( msgResult != IDCANCEL ) */


   }  /* end DoDefaultFormat */

//-------------------------------------------------------------------------------------------------------------
--

VOID SetFormat (VOID)

   // Set new text format.

   {
      RECT      textRect;                       // Formatting rectangle for wrapping text
      LPARAM    redrawFlag;                     // Redraw parameter for setting font
      COLORREF  newColor;                       // Requested text color
      COLORREF  sysColor;                       // System text color


      SendMessage (TheEditor, EM_GETRECT,          // Get formatting rectangle
                   0, LPARAM(&textRect));

         DeleteObject (TheFont);                       // Delete previous font
         TheFont    = CreateFontIndirect(&TextFormat);  // Create new font
         redrawFlag = MAKELPARAM(TRUE, 0);          // Set flag to redraw text
         SendMessage (TheEditor, WM_SETFONT,           // Set new font
                      WPARAM(TheFont), redrawFlag);

         newColor = FontParams.rgbColors;           // Get requested color
         sysColor = GetSysColor(COLOR_WINDOWTEXT);  // Get system text color
```

```
if ( newColor == sysColor )                // Did user choose system color?
   TextColor = 0;                             // Default back to system color
```

```
      else
         TextColor = newColor;                  // Save requested color

   SendMessage (TheEditor, EM_SETRECT,            // Restore formatting rectangle
                0, LPARAM(&textRect));

} /* end SetFormat */

//-------------------------------------------------------------------------------------------------------------
--

VOID DoBackground (VOID)

   // Handle Background Color... command.

   {
      BOOL  confirmed;                          // Did user confirm dialog?


      confirmed = ChooseColor (&ColorParams);      // Get new color from user
      if ( confirmed )                          // Did user confirm?
         SetBackground ();                         // Set new background color

   } /* end DoBackground */

//-------------------------------------------------------------------------------------------------------------
--

VOID SetBackground (VOID)

   // Set new background color.

   {
      COLORREF  newColor;                       // Requested background color
      COLORREF  sysColor;                       // System background color


      newColor = ColorParams.rgbResult;         // Get requested color
      sysColor = GetSysColor(COLOR_WINDOW);        // Get system background color

      if ( newColor == sysColor )               // Did user choose system color?
         BackgroundColor = 0;                      // Default back to system color
      else
         BackgroundColor = newColor;            // Save requested color

      DeleteObject (BackgroundBrush);              // Destroy old background brush
      BackgroundBrush = CreateSolidBrush(newColor);  // Create new brush

      InvalidateRect (TheEditor, NULL, TRUE);      // Force repaint

   } /* end SetBackground */

//-------------------------------------------------------------------------------------------------------------
```

```
--

VOID DoHelp (VOID)
```

```
   // Handle Help command.

   {
      MessageBeep (MB_OK);                          // Help not implemented

   }  /* end DoHelp */

//-----------------------------------------------------------------------------------------------------------------
--

VOID DoAbout (VOID)

   // Handle About WiniEdit... command.

   {
      LPSTR  resourceID;                           // Resource ID in string form


      resourceID = MAKEINTRESOURCE(About_Dialog);    // Convert resource ID
      DialogBox ( ThisInstance, resourceID,          // Invoke dialog
                  TheWindow, DLGPROC(AboutProc) );

   }  /* end DoAbout */

//-----------------------------------------------------------------------------------------------------------------
--

BOOL CALLBACK AboutProc (HWND theDialog, UINT msgCode, WPARAM wParam, LPARAM lParam)

   // Dialog procedure for About WiniEdit... dialog.

   {
      BOOL  result;                                // Function result


      switch ( msgCode )                           // Dispatch on message code
         {
            case WM_INITDIALOG:
               result = TRUE;                         // Tell Windows to set the input focus
               break;

            case WM_COMMAND:
               switch ( wParam )                      // Dispatch on item ID
                  {
                     case IDOK:                    // Continue button, space bar, or Enter key
                     case IDCANCEL:                   // Esc key
                        EndDialog (theDialog, TRUE);  // Dismiss the dialog
                        result = TRUE;                // Indicate message accepted
                        break;

                     default:
                        result = FALSE;           // Indicate message not accepted
                        break;
```

```
    }  /* end switch ( wParam ) */
  break;
```

```
            default:
               result = FALSE;                    // Indicate message not accepted
               break;

         }  /* end switch ( msgCode ) */

      return result;

   }  /* end AboutProc */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoNotification (UINT itemID, HWND theControl, UINT notifyCode)

   // Handle control notification.

   {
      /* Nothing to do */

   }  /* end DoNotification */

//-----------------------------------------------------------------------------------------------------------
--

BOOL DoQuery (HWND thisWindow, UINT wParam, LONG lParam)

   // Handle WM_QUERYENDSESSION message.

   {
      BOOL  confirmed;                           // Did user confirm operation?


      confirmed = CloseDoc ();                   // Allow user to save document if necessary
      return confirmed;                             // Report confirmation or cancellation

   }  /* end DoQuery */

//-----------------------------------------------------------------------------------------------------------
--

VOID DoDestroy (HWND thisWindow, UINT wParam, LONG lParam)

   // Handle WM_DESTROY message.

   {
      DeleteObject (BackgroundBrush);               // Destroy background brush

      PostQuitMessage (NO_ERROR);                // Signal program completion

   }  /* end DoDestroy */
```

```
//-----------------------------------------------------------------------------------------------------------
--

VOID Finalize (VOID)
```

```
   // Do one-time-only finalization.

   {
      /* Nothing to do */

   }  /* end Finalize */

//-----------------------------------------------------------------------------------------------------------
--

INT ShowUserMessage (UINT msgID, UINT iconStyle, UINT buttonStyle, LPSTR msgTitle, LPSTR argString)

   // Display user message on screen.

   {
      CHAR     msgFormat[MsgMax];                 // Format for confirmation message
      CHAR     msgText  [MsgMax];                 // Text of confirmation message
      UINT     msgStyle;                          // Style options for message box
      INT      msgResult;                         // Result value returned by message box


      MessageBeep (MB_OK);                           // Beep for attention


      LoadString (ThisInstance, msgID,               // Get message text from resource
                  msgFormat, MsgMax );
      wsprintf   (msgText, msgFormat, argString);    // Merge argument string into message

      msgStyle = MB_APPLMODAL |                      // Application-modal message box
                 iconStyle |                        // Merge in icon style
                buttonStyle |                       // Merge in button style
                MB_SETFOREGROUND;                   // Bring to foreground

      msgResult = MessageBox (TheWindow, msgText,    // Display message box
                              msgTitle, msgStyle);


      return msgResult;                              // Pass result back to caller


   }  /* end ShowUserMessage */

//-----------------------------------------------------------------------------------------------------------
--

VOID IOCheck (VOID)

   // Check for I/O error.

   {
      LONG    errorCode;                         // Code number identifying error
      CHAR    errorString[10];                   // Error code in string form
```

```
errorCode = GetLastError();                // Get error code
```

```
   if ( errorCode == NO_ERROR )                  // Was there an error?
      ErrorFlag = FALSE;                        // If not, just continue normal processing

   else
      {
         ultoa (errorCode, errorString, 10);     // Convert to string
         ShowUserMessage (IOError_Msg,           // Display generic I/O error message
                          MB_ICONEXCLAMATION,   // Exclamation-point icon
                          MB_OK,                 // OK button
                         NULL,                  // Use default title for message box
                          errorString);         // Merge error code into message

         ErrorFlag = TRUE;                       // Force exit to main message loop

      }  /* end else */

}  /* end IOCheck */
```

The resource header file, **WiniEdit Resources.h**, defines the symbolic identification codes for the program's resources. This file was generated automatically by the Visual C++ onscreen resource editors, using the symbolic names and ID numbers I specified when building the program's resources with the onscreen resource editors. For better readability, I have done some editing on the raw output generated by the development software, such as grouping together resources of the same type and indenting definitions to reflect their hierarchical relationships.

```
//
//                                  WiniEdit Resources.h
//                  Resource header for example Windows application program
//                           S. Chernicoff         15 January 1995
//

//            Global resource header file for WiniEdit example application program


//-------------------------------------------------------------------------------


// Icon

#define ProgIcon_ID                     1000


//-------------------------------------------------------------------------------


// Menus
```

```
#define Main_Menu                       1000

#define File_Menu                    0
#define    New_Item                     1001
#define    Open_Item                    1002
#define    Close_Item                   1003
#define    Save_Item                    1004
#define    SaveAs_Item                  1005
#define    Revert_Item                  1006
#define    Setup_Item                   1007
#define    Print_Item                   1008
#define    Exit_Item                    1009

#define Edit_Menu                    1
#define    Undo_Item                    1101
#define    Cut_Item                     1102
#define    Copy_Item                    1103
#define    Paste_Item                   1104
#define    Delete_Item                  1105
#define    SelectAll_Item               1106

#define Format_Menu                  2
#define    Format_Item                  1201
#define    Default_Item                 1202
#define    Background_Item              1203

#define Help_Menu                    3
#define    Help_Item                    1301
#define    About_Item                   1302


//----------------------------------------------------------------------------


// Accelerators

#define Accel_ID                        1000


//----------------------------------------------------------------------------


// Control ID

#define Edit_Control                    1000


//----------------------------------------------------------------------------


// Dialog

#define About_Dialog                    1000
```

```
//------------------------------------------------------------------------------


// Strings

#define ProgName_Str                    1001
#define NoTitle_Str                     1002
#define FileFilter_Str                  1003
#define FileExt_Str                     1004

#define Save_Msg                        2001
#define Revert_Msg                      2002
#define DefaultFormat_Msg               2003
#define WrongType_Msg                   2004
#define TooLong_Msg                     2005
#define OutOfMem_Msg                    2006
#define IOError_Msg                     2007


//------------------------------------------------------------------------------


// Next default values for new objects

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40004
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           102
#endif
#endif
```

The resource description file, **WiniEdit.rc**, was generated automatically by the Visual C++ development software to define the resources I built onscreen with the interactive resource editors.

```
//Microsoft Visual C++ generated resource script.
//
#include "WiniEdit Resources.h"

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS


#ifdef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "WiniEdit Resources.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

////////////////////////////////////////////////////////////////////////////
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

About_Dialog DIALOG DISCARDABLE  32, 32, 163, 96
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE
FONT 8, "MS Sans Serif"
BEGIN
    ICON            ProgIcon_ID,About_Dialog,8,8,18,20
    CTEXT           "WiniEdit 1.0",IDC_STATIC,61,12,41,8
```

```
    CTEXT           "Example Windows application",IDC_STATIC,32,32,98,8
    LTEXT           "S. Chernicoff",IDC_STATIC,8,52,44,8
    RTEXT           "15 January 1995",IDC_STATIC,100,52,55,8
    DEFPUSHBUTTON   "Continue",IDOK,60,68,43,20
END


//////////////////////////////////////////////////////////////////////////
//
// Menu
//
```

```
Main_Menu MENU PRELOAD DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New\tCtrl+N",                New_Item
        MENUITEM "&Open...\tCtrl+O",            Open_Item
        MENUITEM "&Close\tCtrl+W",              Close_Item
        MENUITEM SEPARATOR
        MENUITEM "&Save\tCtrl+S",               Save_Item
        MENUITEM "Save &As...\tCtrl+Alt+S",     SaveAs_Item
        MENUITEM "&Revert to Saved...\tCtrl+R", Revert_Item
        MENUITEM SEPARATOR
        MENUITEM "Page Set&up...\tCtrl+Alt+P",  Setup_Item
        MENUITEM "&Print...\tCtrl+P",           Print_Item
        MENUITEM SEPARATOR
        MENUITEM "E&xit\tCtrl+Q",               Exit_Item
    END
    POPUP "&Edit"
    BEGIN
        MENUITEM "&Undo\tCtrl+Z",               Undo_Item
        MENUITEM SEPARATOR
        MENUITEM "Cu&t\tCtrl+X",                Cut_Item
        MENUITEM "&Copy\tCtrl+C",               Copy_Item
        MENUITEM "&Paste\tCtrl+V",              Paste_Item
        MENUITEM "&Delete\tDelete",             Delete_Item
        MENUITEM SEPARATOR
        MENUITEM "Select &All\tCtrl+A",         SelectAll_Item
    END
    POPUP "&Format"
    BEGIN
        MENUITEM "Text &Format...\tCtrl+F",     Format_Item
        MENUITEM "&Default Format\tCtrl+D",     Default_Item
        MENUITEM SEPARATOR
        MENUITEM "&Background Color...\tCtrl+B", Background_Item
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&Help\tCtrl+?",               Help_Item
        MENUITEM SEPARATOR
        MENUITEM "&About WiniEdit...",          About_Item
    END
END


////////////////////////////////////////////////////////////////////////////
//
// Icon
//

ProgIcon_ID             ICON    DISCARDABLE     "WiniEdit.ico"

////////////////////////////////////////////////////////////////////////////
//
```

```
// Accelerator
```

```
//

Accel_ID ACCELERATORS PRELOAD DISCARDABLE
BEGIN
    "A",            SelectAll_Item,         VIRTKEY, CONTROL, NOINVERT
    "B",            Background_Item,        VIRTKEY, CONTROL, NOINVERT
    "C",            Copy_Item,              VIRTKEY, CONTROL, NOINVERT
    "D",            Default_Item,           VIRTKEY, CONTROL, NOINVERT
    "F",            Format_Item,            VIRTKEY, CONTROL, NOINVERT
    "N",            New_Item,               VIRTKEY, CONTROL, NOINVERT
    "O",            Open_Item,              VIRTKEY, CONTROL, NOINVERT
    "P",            Print_Item,             VIRTKEY, CONTROL, NOINVERT
    "P",            Setup_Item,             VIRTKEY, CONTROL, ALT, NOINVERT
    "Q",            Exit_Item,              VIRTKEY, CONTROL, NOINVERT
    "R",            Revert_Item,            VIRTKEY, CONTROL, NOINVERT
    "S",            Save_Item,              VIRTKEY, CONTROL, NOINVERT
    "S",            SaveAs_Item,            VIRTKEY, CONTROL, ALT, NOINVERT
    "V",            Paste_Item,             VIRTKEY, CONTROL, NOINVERT
    VK_BACK,        Undo_Item,              VIRTKEY, ALT, NOINVERT
    VK_DELETE,      Cut_Item,               VIRTKEY, SHIFT, NOINVERT
    VK_F1,          Help_Item,              VIRTKEY, NOINVERT
    VK_F2,          Cut_Item,               VIRTKEY, NOINVERT
    VK_F3,          Copy_Item,              VIRTKEY, NOINVERT
    VK_F4,          Paste_Item,             VIRTKEY, NOINVERT
    VK_HELP,        Help_Item,              VIRTKEY, CONTROL, NOINVERT
    VK_HELP,        Help_Item,              VIRTKEY, SHIFT, CONTROL, NOINVERT
    VK_INSERT,      Copy_Item,              VIRTKEY, CONTROL, NOINVERT
    VK_INSERT,      Paste_Item,             VIRTKEY, SHIFT, NOINVERT
    "W",            Close_Item,             VIRTKEY, CONTROL, NOINVERT
    "X",            Cut_Item,               VIRTKEY, CONTROL, NOINVERT
    "Z",            Undo_Item,              VIRTKEY, CONTROL, NOINVERT
END


////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    ProgName_Str            "WiniEdit"
    NoTitle_Str             "Untitled"
    FileFilter_Str          "WiniEdit files (*.wed)|*.wed|Plain text files (*.txt)|
*.txt|ASCII files (*.asc)|*.asc*|All text files (*.wed, *.txt, *.asc)|
*.wed;*.txt;*.asc|All files (*.*)|*.*||"
    FileExt_Str             "wed"
END

STRINGTABLE DISCARDABLE
BEGIN
    Save_Msg                "Save document ""%s"" before closing?"
    Revert_Msg              "Revert to most recently saved version of document
```

```
""%s""?"
```

```
   DefaultFormat_Msg       "Revert document ""%s"" to standard text format?"
    WrongType_Msg           "Sorry, WiniEdit works with text documents only. Can't
read or write document ""%s""."
    TooLong_Msg             "Sorry, document ""%s"" is too long for WiniEdit to read."
    OutOfMem_Msg            "Out of memory!"
    IOError_Msg             "Unanticipated input/output error #%s."
END


#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```