Appendix B

MiniEdit Code Listing

Here, for comparison, is the complete code of my example Macintosh application program, MiniEdit, from which the Windows program WiniEdit was derived. The code was translated from the original Pascal version of the program as published in *Macintosh Revealed,* Volume Two. Again, the code uses C++-style comments and a few other syntactic conveniences, but is essentially written in straight C.

Here is **`MiniEdit.make`**, the MPW (Macintosh Programmer's Workshop) make file for the C version of MiniEdit:

```
#   File:       "MiniEdit.make"
#   Target:     "MiniEdit"
#   Sources:    "MiniEdit.cp" "MiniEdit.r"
#   Created:    Wednesday, October 26, 1994 3:58:40 PM


OBJECTS = "MiniEdit.cp.o"


"MiniEdit" ff "MiniEdit.make" "MiniEdit.r" "MiniEdit.rsrc"
   Rez "MiniEdit.r" -append -o "MiniEdit"

"MiniEdit" ff "MiniEdit.make" {OBJECTS}
   Link -t APPL -c MEDT ∂
      {OBJECTS} ∂
      #"{CLibraries}"CSANELib.o ∂
      #"{CLibraries}"Math.o ∂
      #"{CLibraries}"CplusLib.o ∂
      #"{CLibraries}"Complex.o ∂
      #"{CLibraries}"StdCLib.o ∂
      "{Libraries}"Runtime.o ∂
      "{Libraries}"Interface.o ∂
      -o "MiniEdit"
   SetFile "MiniEdit" -a B
"MiniEdit.cp.o" ƒ "MiniEdit.make" "MiniEdit.cp"
    CPlus   "MiniEdit.cp"
```

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

```
//
//                                  MiniEdit
//                        Example Macintosh application program
//                          S. Chernicoff    24 October 1994
//

//     Multiple-window text editor to illustrate event-driven program structure


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


// Global Constants


   const short ChangeFlag = 0x0002;                  // Mask for extracting "change bit" from event modifiers

   const short MenuBarHeight = 20;                   // Height of menu bar in pixels
   const short TitleBarSize  = 18;                   // Size of window title bar in pixels
   const short ScreenMargin  =  4;                   // Width of "safety margin" around edge of screen

   const short MinWidth   = 80;                      // Minimum width of window in pixels
   const short MinHeight  = 80;                      // Minimum height of window in pixels
   const short SBarWidth  = 16;                      // Width of scroll bars in pixels
   const short TextMargin = 4;                    // Inset from window to text rectangle

   const short DlgTop  = 100;                        // Top edge of dialog box for Get and Put dialogs
   const short DlgLeft =  85;                        // Left edge of dialog box for Get and Put dialogs

   const short AppleID = 1;                       // Menu ID for Apple menu
      const short AboutItem  = 1;                    // Item number for About... command

   const short FileID  = 2;                       // Menu ID for File menu
      const short NewItem    =  1;                      // Item number for New command
      const short OpenItem   =  2;                      // Item number for Open... command
      const short CloseItem  =  3;                      // Item number for Close command
      const short SaveItem   =  5;                      // Item number for Save command
      const short SaveAsItem =  6;                      // Item number for Save As... command
      const short RevertItem =  7;                      // Item number for Revert to Saved command
      const short SetupItem  =  9;                      // Item number for Page Setup... command
      const short PrintItem  = 10;                      // Item number for Print... command
      const short QuitItem   = 12;                      // Item number for Quit command

   const short EditID  = 3;                       // Menu ID for Edit menu
      const short UndoItem   = 1;                    // Item number for Undo command
      const short CutItem    = 3;                    // Item number for Cut command
```

```
      const short CopyItem   = 4;                  // Item number for Copy command
      const short PasteItem  = 5;                  // Item number for Paste command
      const short ClearItem  = 7;                  // Item number for Clear command

   const short WindowID  = 1000;                   // Resource ID for window template
   const short ScrollID  = 1000;                   // Resource ID for scroll bar template
   const short NoTitleID = 1000;                   // Resource ID of title string for empty window

   const short AboutID     = 1000;                 // Resource ID for About alert
   const short SaveID      = 1001;                 // Resource ID for Save alert
   const short RevertID    = 1002;                 // Resource ID for Revert alert
   const short CantPrintID = 1003;                 // Resource ID for Can't Print alert
   const short WrongTypeID = 1004;                 // Resource ID for Wrong Type alert
   const short TooLongID   = 1005;                 // Resource ID for File Too Long alert
   const short OpWrID      = 1006;                 // Resource ID for Already Open alert
   const short IOErrID     = 1007;                 // Resource ID for I/O Error alert

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


// Global Types


   struct WindowData
      {
         TEHandle       editRec;                  // Handle to edit record
         ControlHandle  scrollBar;                // Handle to scroll bar
         Boolean        dirty;                  // Document changed since last saved?
         Byte           padding;                  // Extra byte for padding
         short          volNumber;                // Volume reference number
         short          fileNumber;               // File reference number

      }; /* end WindowData */


   typedef WindowData   *WDPtr,
                        **WDHandle;

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


// Global Variables


   EventRecord  TheEvent;                        // Current event

   WindowPtr      TheWindow    = nil;               // Pointer to currently active window
   ControlHandle  TheScrollBar = nil;               // Handle to active window's scroll bar
   TEHandle       TheText      = nil;               // Handle to active window's edit record

   Rect   ScreenBounds;                             // Boundary rectangle of screen
```

```
short  ScreenWidth;                    // Width of screen in pixels
short  ScreenHeight;                      // Height of screen in pixels
```

```
   short  OldMask;                              // Original value of system event mask

   MenuHandle  AppleMenu;                      // Handle to Apple menu
   MenuHandle  FileMenu;                       // Handle to File menu
   MenuHandle  EditMenu;                       // Handle to Edit menu

   CursHandle  IBeam;                          // Handle to I-beam cursor
   CursHandle  Watch;                          // Handle to wristwatch cursor

   short  OpenCount  = 0;                      // Number of windows opened
   short  CloseCount = 0;                      // Number of windows closed

   short    ScrapCompare;                      // Previous scrap count for comparison
   Boolean  ScrapDirty = false;                   // Has scrap been changed?

   Boolean  Quitting  = false;                 // Closing up shop?
   Boolean  Finished  = false;                 // All closed?
   Boolean  ErrorFlag = false;                 // I/O error flag


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

// Function Prototypes

void Initialize (void);
      // Do one-time-only initialization.
   void SetUpMenus (void);
         // Set up menus.
   void SetUpCursors (void);
         // Set up cursors.
   void DoStartup (void);
         // Process Finder startup information.
void MainLoop (void);
      // Execute one pass of main program loop.
   void FixCursor (void);
         // Adjust cursor for region of screen.
   void DoEvent (void);
         // Get and process one event.
      void DoMouseDown (void);
            // Handle mouse-down event.
         void DoMenuClick (void);
               // Handle mouse-down event in menu bar.
            void DoMenuChoice (long menuChoice);
                  // Handle user's menu choice.
               void DoAppleChoice (short theItem);
                     // Handle choice from Apple menu.
                  void DoAbout (void);
                        // Handle About MiniEdit... command.
               void DoFileChoice (short theItem);
                     // Handle choice from File menu.
                  void DoNew (void);
                        // Handle New command.
```

```
void OffsetWindow (WindowPtr whichWindow);
      // Offset location of new window.
```

```
        void DoOpen (void);
              // Handle Open... command.
           void OpenFile (Str255 fileName, short vNum);
                 // Open document file.
        void DoClose (void);
              // Handle Close command.
           void CloseAppWindow (void);
                 // Close application window.
           void CloseSysWindow (void);
                 // Close system window.
        void DoSave (void);
              // Handle Save command.
        void DoSaveAs (void);
              // Handle Save As... command.
           void WriteFile (short theFile, short volNum);
                 // 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 DoQuit (void);
              // Handle Quit command.
     void DoEditChoice (short theItem);
           // Handle choice from Edit menu.
        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 DoClear (void);
              // Handle Clear command.
void DoSysClick (WindowPtr whichWindow);
     // Handle mouse-down event in system window.
void DoContent (WindowPtr whichWindow);
     // Handle mouse-down event in content region of active window.
   void DoScroll (short thePart, Point thePoint);
        // Handle mouse-down event in scroll bar.
     pascal void ScrollText (ControlHandle theControl, short thePart);
           // Scroll text within window.
     void AdjustText (void);
           // Adjust text within window to match scroll bar setting.
     pascal Boolean AutoScroll (void);
           // Handle automatic scrolling during text selection.
   void DoSelect (Point thePoint);
        // Handle mouse-down event in text rectangle.
     void FixEditMenu (void);
           // Enable/disable editing commands.
void DoDrag (WindowPtr whichWindow);
```

```
  // Handle mouse-down event in drag region.
```

```
        void DoGrow (WindowPtr whichWindow);
              // Handle mouse-down event in size region.
           void FixScrollBar (void);
                // Resize window's scroll bar.
           void FixText (void);
                // Resize window's text rectangle.
        void DoGoAway (WindowPtr whichWindow);
              // Handle mouse-down event in close region.
        void DoZoom (WindowPtr whichWindow, short inOrOut);
              // Handle mouse-down event in zoom region.
     void DoKeystroke (void);
           // Handle keystroke.
     void DoTyping (unsigned char ch);
           // Handle character typed from keyboard.
     void DoUpdate (void);
           // Handle update event.
     void DoActivate (void);
           // Handle activate (or deactivate) event.
        void ActWindow (WindowPtr whichWindow);
              // Activate window.
        void DeactWindow (WindowPtr whichWindow);
              // Deactivate window.
void Finalize (void);
     // Do one-time-only finalization.
   void WindowDirty (Boolean isDirty);
        // Mark window dirty or clean.
   void AdjustScrollBar (void);
        // Adjust scroll bar to length of document.
   void ScrollToSelection (void);
        // Scroll current selection into view.
   void ScrollCharacter (short theCharacter, Boolean toBottom);
        // Scroll character into view.
   void ReadDeskScrap (void);
        // Copy desk scrap to Toolbox scrap.
   void WriteDeskScrap (void);
        // Copy Toolbox scrap to desk scrap.
   void IOCheck (OSErr resultCode);
        // Check for I/O error.
```

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

```
//
//                                  MiniEdit
//                      Example Macintosh application program
//                        S. Chernicoff    24 October 1994
//

//     Multiple-window text editor to illustrate event-driven program structure
```

```
   #include <SysEqu.h>                          // System globals
   #include <Types.h>                           // Elementary data types
   #include <QuickDraw.h>                       // QuickDraw graphics routines
   #include <Events.h>                          // Event-handling routines
   #include <Windows.h>                            // Window-handling routines
   #include <Controls.h>                        // Control-handling routines
   #include <Menus.h>                           // Menu-handling routines
   #include <Dialogs.h>                            // Dialog-handling routines
   #include <TextEdit.h>                        // Text-editing routines
   #include <Fonts.h>                           // Font-handling routines
   #include <Desk.h>                               // Desk accessory support routines
   #include <Scrap.h>                           // Clipboard-management routines
   #include <Files.h>                           // File-handling routines
   #include <StandardFile.h>                       // Standard File Package
   #include <Memory.h>                          // Memory-management routines
   #include <SegLoad.h>                            // Segment Loader routines
   #include <ToolUtils.h>                       // Toolbox utility routines
   #include <OSUtils.h>                            // Operating System utility routines
   #include <Errors.h>                          // Error codes

   #include "MiniEdit.h"


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

main ()

   // Main program

   {
      Initialize ();                               // Do one-time-only initialization

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

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

      return noErr;                             // Signal successful completion

   }  /* end main */

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

void Initialize (void)

   // Do one-time-only initialization.

   {
```

```
short        masterBlocks = 4;              // Number of master pointer blocks to preallocate
short       *sysMaskPtr;                    // Pointer for finding old event mask
short        newMask;                       // New value for system event mask
```

```
    PScrapStuff  scrapInfo;                         // Pointer to scrap information record


    InitGraf ( Ptr(&qd.thePort) );              // Initialize QuickDraw
    InitFonts ();                               // Initialize fonts
    InitWindows ();                             // Initialize windows
    InitMenus ();                               // Initialize menus
    TEInit ();                                  // Initialize text editing
    InitDialogs (nil);                          // Initialize dialogs

    MaxApplZone ();                             // Expand heap to maximum size
    while ( (masterBlocks--) > 0 )              // Preallocate master pointers to
       MoreMasters ();                          //    minimize heap fragmentation

    ScreenBounds = qd.screenBits.bounds;                // Get boundary rectangle of screen
    ScreenWidth  = ScreenBounds.right  - ScreenBounds.left; // Set screen dimensions
    ScreenHeight = ScreenBounds.bottom - ScreenBounds.top;

    sysMaskPtr = (short*)(SysEvtMask);              // Point to system event mask
    OldMask    = *sysMaskPtr;                       // Save old mask value

    newMask = everyEvent - keyUpMask - mUpMask;     // Disable key-up and mouse-up events
    SetEventMask (newMask);                         // Set the mask
    FlushEvents  (everyEvent, 0);                   // Clear out event queue

    SetUpMenus ();                                  // Create program's menus
    SetUpCursors ();                            // Get standard cursors

    scrapInfo    = InfoScrap();                 // Get scrap info
    ScrapCompare = scrapInfo->scrapCount + 1;       // Force scrap transfer
    ReadDeskScrap ();                               // Read desk scrap into Toolbox scrap

    DoStartup ();                               // Process Finder startup information

  }  /* end Initialize */

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

void SetUpMenus (void)

  // Set up menus.

  {
    AppleMenu = GetMenu(AppleID);                   // Get Apple menu from resource file
    AddResMenu (AppleMenu, 'DRVR');                 // Add names of available desk accessories
    InsertMenu (AppleMenu, 0);                      // Install at end of menu bar

    FileMenu = GetMenu(FileID);                 // Get File menu from resource file
    InsertMenu (FileMenu, 0);                       // Install at end of menu bar

    EditMenu = GetMenu(EditID);                 // Get Edit menu from resource file
    InsertMenu (EditMenu, 0);                       // Install at end of menu bar
```

```
DrawMenuBar ();                         // Show new menu bar on screen
```

```
   }  /* end SetUpMenus */

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

void SetUpCursors (void)

   // Set up cursors.

   {
      IBeam = GetCursor(iBeamCursor);                 // Get cursors from system resource file
      Watch = GetCursor(watchCursor);

      InitCursor ();                                  // Set standard arrow cursor

   }  /* end SetUpCursors */

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

void DoStartup (void)

   // Process Finder startup information.

   {
      short      theMessage;                       // Open or print?
      short      nDocs;                               // Number of documents selected in Finder
      short      thisDoc;                          // Index number of document
      AppFile    docInfo;                          // Startup information about one document
      StringPtr  fileName;                            // Name of document to be opened
      short      volNum;                           // Reference number of document's volume
      OSType     fileType;                         // Document's file type


      CountAppFiles (&theMessage, &nDocs);         // Get number of documents and startup message

      if (theMessage == appPrint)                  // Did user choose Print in Finder?
         {
            StopAlert (CantPrintID, nil);             // Post alert
            ExitToShell ();                        // Return to Finder

         }  /* end if (theMessage == appPrint) */

      if ( nDocs == 0 )                               // If no documents selected,
         DoNew ();                                 //    just open an empty window

      else
         for ( (thisDoc = 1); (thisDoc <= nDocs); (thisDoc++) )  // Otherwise loop through documents
            {
               GetAppFiles (thisDoc, &docInfo);       // Get startup information

               fileName = docInfo.fName;              // Get file name
```

```
  volNum   = docInfo.vRefNum;         // Get volume reference number
  fileType = docInfo.fType;             // Get file type
```

```
            if ( fileType == 'TEXT' )              // Is it a text file?
               {
                  ErrorFlag = false;             // Clear I/O error flag

                  OpenFile (fileName, volNum);      // Read file into a window
                  ClrAppFiles (thisDoc);         // Tell Finder it's been processed

               }  /* end if ( fileType == 'TEXT' ) */

            else
               {
                  ParamText (fileName, "\p", "\p", "\p");   // Merge in file name
                  InitCursor ();                    // Set arrow cursor
                  StopAlert (WrongTypeID, nil);     // Post alert

               }  /* end else */

         }  /* end for (thisDoc) */

   }  /* end DoStartup */

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

void MainLoop (void)

   // Execute one pass of main program loop.

   {
      if ( FrontWindow() == nil )                 // Is the desktop empty?
         {
            DisableItem (EditMenu, UndoItem);      // Disable inapplicable menu commands
            DisableItem (EditMenu, CutItem);
            DisableItem (EditMenu, CopyItem);
            DisableItem (EditMenu, PasteItem);
            DisableItem (EditMenu, ClearItem);

            DisableItem (FileMenu, CloseItem);
            DisableItem (FileMenu, SaveItem);
            DisableItem (FileMenu, SaveAsItem);
            DisableItem (FileMenu, RevertItem);
            DisableItem (FileMenu, SetupItem);
            DisableItem (FileMenu, PrintItem);

         }  /* end if ( FrontWindow() == nil ) */

      FixCursor ();                               // Adjust cursor for region of screen
      SystemTask ();                                 // Do system idle processing

      if ( TheText != nil )
         TEIdle (TheText);                           // Blink cursor

      DoEvent ();                                 // Get and process one event
```

```
}  /* end MainLoop */
```

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

void FixCursor (void)

   // Adjust cursor for region of screen.

   {
      Point  mousePoint;                        // Current mouse position in window coordinates
      Rect   textRect;                          // Active window's text rectangle


      if ( Quitting )                           // Skip cursor adjustment during quit sequence
         return;

      if ( FrontWindow() == nil )               // Screen empty?
         InitCursor ();                            // Set arrow cursor

      else if ( FrontWindow() == TheWindow )       // Is one of our windows active?
         {
            GetMouse (&mousePoint);                // Get mouse position
            textRect = (**TheText).viewRect;       // Get window's text rectangle

            if ( PtInRect(mousePoint, &textRect) )    // Is mouse in text rectangle?
               SetCursor ( &(**IBeam) );              // Set I-beam cursor
            else
               InitCursor ();                         // Set arrow cursor

         }  /* end if ( FrontWindow() == TheWindow ) */

      else                                      // A system window is active:
         /* Do nothing */;                         //    let desk accessory set cursor

   }  /* end FixCursor */

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

void DoEvent (void)

   // Get and process one event.

   {
      ErrorFlag = false;                        // Clear I/O error flag

      if ( GetNextEvent(everyEvent, &TheEvent) )     // Get next event

         switch ( TheEvent.what )
            {
               case mouseDown:
                  if ( !Quitting )
                     DoMouseDown();                  // Handle mouse-down event
```

```
    break;

  case keyDown:
```

```
            case autoKey:
               if ( !Quitting )
                  DoKeystroke ();               // Handle keystroke
               break;

            case updateEvt:
               DoUpdate ();                     // Handle update event
               break;

            case activateEvt:
               DoActivate ();                      // Handle activate/deactivate event
               break;

            default:
               break;

         }  /* end switch ( TheEvent.what ) */

      else if ( Quitting && (TheEvent.what == nullEvent) ) // Closing up shop after a Quit command?
         {
            if ( FrontWindow() != nil )            // Any windows on the screen?
               DoClose ();                         // Close the frontmost
            else
               Finished = true;                    // Signal end of program

         }  /* end if ( Quitting && (TheEvent.what == nullEvent) ) */

   }  /* end DoEvent */

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

void DoMouseDown (void)

   // Handle mouse-down event.

   {
      WindowPtr  whichWindow;                      // Window where mouse was pressed
      short      thePart;                          // Part of screen where mouse was pressed


      thePart = FindWindow(TheEvent.where, &whichWindow);  // Where on the screen was mouse pressed?

      switch ( thePart )
         {
            case inDesk:
               break;                              // Nothing to do for click on desktop

            case inMenuBar:
               DoMenuClick ();                     // Handle click in menu bar
               break;

            case inSysWindow:
```

```
  DoSysClick (whichWindow);             // Handle click in system window
  break;
```

```
            case inContent:
               DoContent (whichWindow);            // Handle click in content region
               break;

            case inDrag:
               DoDrag (whichWindow);               // Handle click in drag region
               break;

            case inGrow:
               DoGrow (whichWindow);               // Handle click in size region
               break;

            case inGoAway:
               DoGoAway (whichWindow);                // Handle click in close region
               break;

            case inZoomIn:
               DoZoom (whichWindow, inZoomIn);        // Handle click in zoom region
               break;

            case inZoomOut:
               DoZoom (whichWindow, inZoomOut);       // Handle click in zoom region
               break;

         }  /* end switch ( thePart ) */

   }  /* end DoMouseDown */

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

void DoMenuClick (void)

   // Handle mouse-down event in menu bar.

   {
      long  menuChoice;                            // Menu ID and item number


      menuChoice = MenuSelect(TheEvent.where);       // Track mouse
      DoMenuChoice (menuChoice);                     // Handle user's menu choice


   }  /* end DoMenuClick */

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

void DoMenuChoice (long menuChoice)

   // Handle user's menu choice.
```

```
{
   const short  noMenu = 0;                    // No menu selected
```

```
    short  theMenu = HiWord(menuChoice);        // Menu ID of selected menu
    short  theItem = LoWord(menuChoice);        // Item number of selected item


    switch ( theMenu )
       {
          case noMenu:
             break;                             // No menu selected, nothing to do

          case AppleID:
             DoAppleChoice (theItem);           // Handle choice from Apple menu
             break;

          case FileID:
             DoFileChoice (theItem);               // Handle choice from File menu
             break;

          case EditID:
             DoEditChoice (theItem);               // Handle choice from Edit menu
             break;

       }  /* end switch ( theMenu ) */

    HiliteMenu (0);                             // Unhighlight menu title

  }  /* end DoMenuChoice */

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

void DoAppleChoice (short theItem)

  // Handle choice from Apple menu.

  {
     Str255  accName;                           // Name of desk accessory


     switch ( theItem )
        {
           case AboutItem:
              DoAbout ();                       // Handle About MiniEdit... command
              break;

           default:
              {
                 if ( FrontWindow() == nil )    // Is the desktop empty?
                    {
                       EnableItem (FileMenu, CloseItem);// Enable Close command

                       EnableItem (EditMenu, UndoItem);    // Enable standard
                       EnableItem (EditMenu, CutItem);     //    editing commands
                       EnableItem (EditMenu, CopyItem);    //    for desk accessory
```

```
EnableItem (EditMenu, PasteItem);
EnableItem (EditMenu, ClearItem);
```

```
            }  /* end if ( FrontWindow() == nil ) */

            GetItem (AppleMenu, theItem, accName);    // Get accessory name
            OpenDeskAcc (accName);                 // Open desk accessory

            break;

         }  /* end default */

      }  /* end switch ( theItem ) */

   }  /* end DoAppleChoice */

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

void DoAbout (void)

   // Handle About MiniEdit... command.

   {
      InitCursor ();                              // Set arrow cursor
      Alert (AboutID, nil);                     // Post alert

   }  /* end DoAbout */

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

void DoFileChoice (short theItem)

   // Handle choice from File menu.

   {
      switch ( theItem )
         {
            case NewItem:
               DoNew ();                          // Handle New command
               break;

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

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

            case SaveItem:
               DoSave ();                         // Handle Save command
               break;
```

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

```
            case RevertItem:
               DoRevert ();                          // Handle Revert to Saved command
               break;

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

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

            case QuitItem:
               DoQuit ();                            // Handle Quit command
               break;

         }  /* end switch ( theItem ) */

   }  /* end DoFileChoice */

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

void DoNew (void)

   // Handle New command.

   {
      WDHandle  theData;                             // Handle to window's data record
      Handle    dataHandle;                          // Untyped handle for creating data record
      Rect      destRect;                            // Wrapping rectangle for window's text
      Rect      viewRect;                            // Clipping rectangle for window's text


      TheWindow = GetNewWindow(WindowID, nil, WindowPtr(-1)); // Make new window from template

      OffsetWindow (TheWindow);                        // Offset from location of previous window
      ShowWindow   (TheWindow);                        // Make window visible

      SetPort  (TheWindow);                          // Get into the window's port
      TextFont (geneva);                             // Set text font
      TextSize (9);                                  // Set text size

      SetRect (&viewRect, 0,                         // Set up clipping rectangle
                          0,
                          (*TheWindow).portRect.right  - (SBarWidth - 1),
                          (*TheWindow).portRect.bottom - (SBarWidth - 1));
      destRect = viewRect;                             // Set up wrapping rectangle
      InsetRect (&destRect, TextMargin, 0);            // Inset by text margin

      dataHandle = NewHandle( sizeof(WindowData) );  // Allocate window data record
      theData    = WDHandle(dataHandle);               // Convert to typed handle
```

```
SetWRefCon (TheWindow, long(theData));          // Store as reference constant
```

```
    TheText = TENew (&destRect, &viewRect);        // Make edit record
    (**theData).editRec = TheText;              // Store handle in data record
    SetClikLoop (AutoScroll, TheText);             // Install auto-scroll routine

    TheScrollBar = GetNewControl(ScrollID, TheWindow);   // Make scroll bar
    (**theData).scrollBar = TheScrollBar;          // Store handle in data record

    (**theData).dirty      = false;                // Document is initially clean
    (**theData).fileNumber = 0;                 // Window has no associated file
    (**theData).volNumber  = 0;                 //    or volume

    EnableItem (FileMenu, CloseItem);           // Enable Close command on menu

  }  /* end DoNew */

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

void OffsetWindow (WindowPtr whichWindow)

  // Offset location of new window.

  {
    const short  hOffset = 20;                     // Horizontal offset from previous window, in pixels
    const short  vOffset = 20;                     // Vertical offset from previous window, in pixels

    short  windowWidth;                         // Width of window in pixels
    short  windowHeight;                           // Height of window in pixels
    short  hExtra;                                 // Excess screen width in pixels
    short  vExtra;                                 // Excess screen height in pixels
    short  hMax;                                // Maximum number of windows horizontally
    short  vMax;                                // Maximum number of windows vertically
    short  windowLeft;                          // Left edge of window in global coordinates
    short  windowTop;                              // Top edge of window in global coordinates


    windowWidth   = whichWindow->portRect.right  - whichWindow->portRect.left; // Get window dimensions
    windowHeight  = whichWindow->portRect.bottom - whichWindow->portRect.top;  //    from port rectangle
    windowHeight += TitleBarSize;                                              // Adjust for title bar

    hExtra =  ScreenWidth  - windowWidth;                  // Find excess screen width
    vExtra = (ScreenHeight - MenuBarHeight) - windowHeight; // Find excess screen height

    hMax = (hExtra / hOffset) + 1;                      // Find maximum number of windows horizontally
    vMax = (vExtra / vOffset) + 1;                      // Find maximum number of windows vertically

    OpenCount++;                                        // Increment open window count
    windowLeft = (OpenCount % hMax) * hOffset;             // Calculate horizontal
    windowTop  = (OpenCount % vMax) * vOffset;             //    and vertical offset

    windowLeft += TitleBarSize;                         // Adjust for title bar
    windowTop  += MenuBarHeight;                           //    and menu bar
```

```
MoveWindow (whichWindow, windowLeft, windowTop, false); // Move window to new location
```

```
   }  /* end OffsetWindow */

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

void DoOpen (void)

   // Handle Open... command.

   {
      Point       dlgOrigin;                    // Top-left corner of dialog box
      SFTypeList  theTypeList;                  // List of file types to display
      SFReply     theReply;                     // Data returned by Get dialog


      SetPt (&dlgOrigin, DlgLeft, DlgTop);      // Set up dialog origin
      theTypeList[0] = 'TEXT';                  // Display text files only

      SFGetFile (dlgOrigin, "\p", nil, 1, theTypeList, nil, &theReply);  // Get file name from user

      if ( theReply.good )                        // Did user confirm file selection?
         OpenFile (theReply.fName, theReply.vRefNum); // Open file and read into window

   }  /* end DoOpen */

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

void OpenFile (Str255 fileName, short vNum)

   // Open document file.

   {
      long      theRefCon;                        // Window's reference constant
      WDHandle  theData;                        // Handle to window data record
      short     theFile;                        // Reference number of file
      OSErr     resultCode;                     // I/O error code


      resultCode = FSOpen (fileName, vNum, &theFile); // Open the file
      IOCheck (resultCode);                     // Check for error
      if (ErrorFlag) return;                    // On error, exit to main event loop

      DoNew ();                                 // Open a new window

      theRefCon = GetWRefCon(TheWindow);           // Get reference constant
      theData   = WDHandle(theRefCon);             // Convert to data handle

      (**theData).volNumber  = vNum;            // Save volume and file number
      (**theData).fileNumber = theFile;         //    in window data record
      SetWTitle (TheWindow, fileName);             // File name becomes window title

      DoRevert ();                              // Read file into window
```

```
if ( ErrorFlag )                        // Error reading file?
   CloseAppWindow ();                   // Close and dispose of the window
```

```
   }  /* end OpenFile */

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

void DoClose (void)

   // Handle Close command.

   {
      if ( FrontWindow() == TheWindow )           // Is the active window one of ours?
         CloseAppWindow ();                       // Close application window
      else
         CloseSysWindow ();                       // Close system window

   }  /* end DoClose */

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

void CloseAppWindow (void)

   // Close application window.

   {
      const short saveItem    = 1;                  // Item number for Save button
      const short discardItem = 2;                  // Item number for Discard button
      const short cancelItem  = 3;                  // Item number for Cancel button

      WindowPtr  thisWindow;                     // Pointer to window being closed
      long       theRefCon;                      // Window's reference constant
      WDHandle   theData;                        // Handle to window data record
      Handle     dataHandle;                     // Untyped handle for destroying data record
      Str255     theTitle;                          // Title of window
      short      theFile;                        // Reference number of window's file
      TEHandle   theEditRec;                     // Handle to window's edit record
      short      theItem;                        // Item number for Save alert
      OSErr      resultCode;                     // I/O error code


      theRefCon = GetWRefCon(TheWindow);             // Get reference constant
      theData   = WDHandle(theRefCon);               // Convert to data handle

      if ( (**theData).dirty )                   // Have window contents been changed?
         {
            GetWTitle (TheWindow, theTitle);         // Get window title
            ParamText (theTitle, "\p", "\p", "\p");  // Substitute into alert text

            InitCursor ();                           // Set arrow cursor
            theItem = CautionAlert(SaveID, nil);  // Post alert
            switch ( theItem )
               {
```

```
case saveItem:
   DoSave ();                    // Save window contents to disk
   if ( ErrorFlag )              // Check for I/O error
```

```
                  return;                    // Exit to main event loop
               break;

            case discardItem:
               break;                        // Nothing to do

            case cancelItem:
               Quitting = false;                // Cancel Quit command, if any
               return;                       // Exit to main event loop

         }  /* end switch ( theItem ) */

      }  /* end if ( (**theData).dirty ) */

   theFile = (**theData).fileNumber;           // Get file reference number
   if ( theFile != 0 )                         // Is window associated with a file?
      {
         resultCode = FSClose(theFile);        // Close file
         IOCheck (resultCode);                 // Post error alert, if any

      }  /* end if ( theFile != 0 ) */

   thisWindow = TheWindow;                        // Save window pointer (DoActivate will change TheWindow
   HideWindow (TheWindow);                        // Force deactivate event

   if ( GetNextEvent(activMask, &TheEvent) )      // Get deactivate event
      DoActivate ();                              //    and handle it
   if ( GetNextEvent(activMask, &TheEvent) )      // Get activate event
      DoActivate ();                              //    and handle it

   CloseCount++;                               // Increment closed window count
   if ( CloseCount == OpenCount )              // Closing last application window on screen?
      {
         OpenCount  = 0;                       // Reset window offset to zero
         CloseCount = 0;

      }  /* end if ( CloseCount == OpenCount ) */

   theEditRec = (**theData).editRec;           // Get handle to edit record
   TEDispose (theEditRec);                        // Dispose of edit record

   dataHandle = Handle(theData);                  // Convert to raw handle
   DisposHandle (dataHandle);                     // Dispose of window data record

   DisposeWindow (thisWindow);                 // Dispose of window and scroll bar

}  /* end CloseAppWindow */

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

void CloseSysWindow (void)
```

```
// Close sys window.
```

```
   {
      WindowPeek  whichWindow;                    // Pointer for access to window's fields
      short       accNumber;                      // Reference number of desk accessory


      whichWindow = WindowPeek( FrontWindow() );     // Convert to a WindowPeek

      accNumber = whichWindow->windowKind;        // Get reference number of desk accessory
      CloseDeskAcc (accNumber);                      // Close desk accessory

   }  /* end CloseSysWindow */

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

void DoSave (void)

   // Handle Save command.

   {
      long      theRefCon;                           // Window's reference constant
      WDHandle  theData;                          // Handle to window data record
      short     theVolume;                           // Volume reference number
      short     theFile;                          // File reference number


      theRefCon = GetWRefCon(TheWindow);              // Get reference constant
      theData   = WDHandle(theRefCon);                // Convert to data handle

      theVolume = (**theData).volNumber;              // Get volume reference number
      theFile   = (**theData).fileNumber;             // Get file reference number

      if ( theFile == 0 )                         // Is window associated with a file?
         DoSaveAs ();                             // Get file name from user
      else
         WriteFile (theFile, theVolume);              // Write to window's file

   }  /* end DoSave */

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

void DoSaveAs (void)

   // Handle Save As... command.

   {
      Point         dlgOrigin;                    // Top-left corner of dialog box
      SFReply       theReply;                        // Data returned by Put dialog

      short         oldFile;                      // Reference number of window's previous file
      short         newFile;                      // Reference number of file being saved
      short         newVol;                       // Volume reference number for file being saved
```

```
StringPtr     newName;                  // Name of file being saved
FInfo         theInfo;                  // File's Finder information
```

```
long          theRefCon;                    // Window's reference constant
WDHandle      theData;                      // Handle to window data record
Str255        theTitle;                        // Title of window

StringHandle  untitled;                        // Handle to title string for empty window
OSErr         resultCode;                      // I/O error code


SetPt     (&dlgOrigin, DlgLeft, DlgTop);       // Set up dialog origin
GetWTitle (TheWindow, theTitle);               // Get current file name from window title
SFPutFile (dlgOrigin, "\pSave under what name?", theTitle, nil, &theReply);
                                               // Get new file name from user

if ( !(theReply.good) )                        // Did user confirm file selection?
   {
      Quitting  = false;                       // Cancel Quit command, if any
      ErrorFlag = true;                           // Force exit to main event loop
      return;                                  // Skip rest of operation
   }  /* end if ( !(theReply.good) ) */

newName = theReply.fName;                         // Get file name
newVol  = theReply.vRefNum;                    // Get volume reference number

resultCode = GetFInfo (newName, newVol, &theInfo);   // Get Finder info
switch ( resultCode )
   {
      case noErr:                              // File already exists
         if ( theInfo.fdType != 'TEXT' )           // Not a text file?
            {
               ParamText (newName, "\p", "\p", "\p"); // Substitute file name into text of alert
               StopAlert (WrongTypeID, nil);     // Post alert

               ErrorFlag = true;                   // Force exit to main event loop
               return;                          // Skip rest of operation

            }  /* end if ( theInfo.fdType != 'TEXT' ) */
         break;

      case fnfErr:                             // File not found
         resultCode = Create (newName, newVol, 'MEDT', 'TEXT');  // Create the file
         IOCheck (resultCode);                 // Check for error
         if (ErrorFlag) return;                // On error, exit to main event loop
         break;

      default:                                 // Unanticipated error
         IOCheck (resultCode);                 // Post error alert
         return;                               // Exit to main event loop

   }  /* end switch ( resultCode ) */

theRefCon = GetWRefCon(TheWindow);                // Get reference constant
theData   = WDHandle(theRefCon);                  // Convert to data handle
```

```
SetCursor ( &(**Watch) );                    // Indicate delay
```

```
    oldFile = (**theData).fileNumber;           // Get file reference number from data record
    if ( oldFile != 0 )                         // Does window already have a file?
       {
          resultCode = FSClose (oldFile);          // Close old file
          IOCheck (resultCode);                 // Check for error
          if (ErrorFlag) return;                // On error, exit to main event loop

       }  /* end if ( oldFile != 0 ) */

    resultCode = FSOpen (newName, newVol, &newFile);  // Open new file
    IOCheck (resultCode);                       // Check for error
    if (ErrorFlag)                                 // Error detected during open?
       {
          (**theData).volNumber  = 0;           // Window is left with no file: clear volume
          (**theData).fileNumber = 0;           //    and file numbers in window data

          untitled = GetString (NoTitleID);     // Get string from resource file
          newName  = &(**untitled);                // Convert from handle

       }  /* end if (ErrorFlag) */

    else
       {
          (**theData).volNumber  = newVol;          // Save new volume and file
          (**theData).fileNumber = newFile;     //    numbers in window data

          WriteFile (newFile, newVol);              // Write window's contents to file

       }  /* end else */

    SetWTitle (TheWindow, newName);                // Set new window title

  }  /* end DoSaveAs */

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

void WriteFile (short theFile, short volNum)

  // Write window contents to a file.

  {
     Handle  textHandle;                        // Handle to text of file
     long    textLength;                        // Length of text in bytes
     OSErr   resultCode;                        // I/O error code


     SetCursor ( &(**Watch) );                     // Indicate delay

     textHandle = (**TheText).hText;               // Get text handle and current length
     textLength = (**TheText).teLength;            //    from edit record
```

```
resultCode = SetFPos (theFile, fsFromStart, 0);// Reset mark to beginning of file
IOCheck (resultCode);                          // Check for error
```

```
    if (ErrorFlag) return;                      // On error, exit to main event loop

    MoveHHi (textHandle);                       // Move text to top of heap
    HLock   (textHandle);                       // Lock text
       resultCode = FSWrite ( theFile, &textLength, &(**textHandle) ); // Write text to file
    HUnlock (textHandle);                       // Unlock text
    IOCheck (resultCode);                       // Check for error
    if (ErrorFlag) return;                      // On error, exit to main event loop

    resultCode = SetEOF (theFile, textLength);     // Set length of file
    IOCheck (resultCode);                       // Check for error
    if (ErrorFlag) return;                      // On error, exit to main event loop

    resultCode = FlushVol (nil, volNum);        // Flush volume buffer
    IOCheck (resultCode);                       // Check for error
    if (ErrorFlag) return;                      // On error, exit to main event loop

    WindowDirty (false);                           // Mark window as clean

  }  /* end WriteFile */

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

void DoRevert (void)

  // Handle Revert to Saved command.

  {
     const short  maxLength = 32767;               // Maximum document length in bytes

     long      theRefCon;                          // Window's reference constant
     WDHandle  theData;                         // Handle to window data record
     Str255    fileName;                        // Title of window
     short     theFile;                         // Reference number of file
     Handle    textHandle;                      // Text handle from window's edit record
     long      textLength;                      // Length of file in bytes
     short     theItem;                         // Item number returned by alert
     OSErr     resultCode;                      // I/O error code


     theRefCon = GetWRefCon(TheWindow);            // Get reference constant
     theData   = WDHandle(theRefCon);              // Convert to data handle

     if ( (**theData).dirty )                   // Have window contents been changed?
        {
           GetWTitle (TheWindow, fileName);         // Get file name from window title
           ParamText (fileName, "\p", "\p", "\p");  // Substitute into text of alert

           InitCursor ();                           // Set arrow cursor
           theItem = CautionAlert (RevertID, nil);  // Post alert

           if ( theItem == cancel )              // Did user cancel?
```

```
  {
     ErrorFlag = true;                  // Force exit to main event loop
```

```
            return;                          // Skip rest of operation

          }  /* end if ( theItem == cancel ) */

      }  /* end if ( (**theData).dirty ) */

   SetCursor ( &(**Watch) );                    // Indicate delay

   theFile = (**theData).fileNumber;           // Get file reference number from data record
   resultCode = GetEOF ( theFile, &textLength );  // Get length of file
   if ( textLength > maxLength )                // File too long?
      {
         GetWTitle (TheWindow, fileName);       // Get file name from window title
         ParamText (fileName, "\p", "\p", "\p");   // Substitute into text of alert

         InitCursor ();                         // Set arrow cursor
         theItem = StopAlert (TooLongID, nil);     // Post alert

         ErrorFlag = true;                      // Force exit

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

   resultCode = SetFPos (theFile, fsFromStart, 0);// Set mark at beginning of file
   IOCheck (resultCode);                        // Check for error
   if (ErrorFlag) return;                       // On error, exit to main event loop

   textHandle = (**TheText).hText;                 // Get text handle from edit record
   SetHandleSize (textHandle, textLength);         // Adjust text to length of file
   (**TheText).teLength = textLength;              // Set text length in edit record

   MoveHHi (textHandle);                        // Move block to top of heap
   HLock   (textHandle);                        // Lock text handle
      resultCode = FSRead ( theFile, &textLength, &(**textHandle) );  // Read text of file into block
   HUnlock (textHandle);                        // Unlock text handle
   IOCheck (resultCode);                        // Check for error
   if (ErrorFlag) return;                       // On error, exit to main event loop

   TECalText (TheText);                            // Wrap text to window
   AdjustScrollBar ();                          // Adjust scroll bar to length of text
   TESetSelect (0, 0, TheText);                    // Set insertion point at beginning

   InvalRect ( &(TheWindow->portRect) );           // Force update to redraw text
   WindowDirty (false);                            // Mark window as clean

}  /* end DoRevert */

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

void DoSetup (void)
```

```
// Handle Page Setup... command.
```

```
   {
      SysBeep (1);                              // Page Setup... command not implemented

   }  /* end DoSetup */

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

void DoPrint (void)

   // Handle Print... command.

   {
      SysBeep (1);                              // Print... command not implemented

   }  /* end DoPrint */

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

void DoQuit (void)

   // Handle Quit command.

   {
      Quitting = true;                          // Start closing down windows

   }  /* end DoQuit */

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

void DoEditChoice (short theItem)

   // Handle choice from Edit menu.

   {
      const short  undoCmd  = 0;                   // Code representing Undo command
      const short  cutCmd   = 2;                   // Code representing Cut command
      const short  copyCmd  = 3;                   // Code representing Copy command
      const short  pasteCmd = 4;                   // Code representing Paste command
      const short  clearCmd = 5;                   // Code representing Clear command


      switch ( theItem )
         {
            case UndoItem:
               if ( !SystemEdit(undoCmd) )         // Intercepted by a desk accessory?
                  DoUndo ();                       // Handle Undo command
               break;

            case CutItem:
```

```
  if ( !SystemEdit(cutCmd) )              // Intercepted by a desk accessory?
     DoCut ();                           // Handle Cut command
  break;
```

```
            case CopyItem:
               if ( !SystemEdit(copyCmd) )         // Intercepted by a desk accessory?
                  DoCopy ();                       // Handle Copy command
               break;

            case PasteItem:
               if ( !SystemEdit(pasteCmd) )           // Intercepted by a desk accessory?
                  DoPaste ();                      // Handle Paste command
               break;

            case ClearItem:
               if ( !SystemEdit(clearCmd) )           // Intercepted by a desk accessory?
                  DoClear ();                      // Handle Clear command
               break;

         }  /* end switch ( theItem ) */

   }  /* end DoEditChoice */

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

void DoUndo (void)

   // Handle Undo command.

   {
      SysBeep (1);                                 // Undo command not implemented

   }  /* end DoUndo */

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

void DoCut (void)

   // Handle Cut command.

   {
      ScrollToSelection ();                        // Make sure selection is visible

      TECut (TheText);                             // Cut the selection

      AdjustScrollBar ();                          // Adjust scroll bar to length of text
      AdjustText ();                                  // Adjust text to match scroll bar
      ScrollToSelection ();                        // Keep insertion point visible

      DisableItem (EditMenu, CutItem);                // Disable menu items that operate
      DisableItem (EditMenu, CopyItem);            //    on a nonempty selection
      DisableItem (EditMenu, ClearItem);

      EnableItem  (EditMenu, PasteItem);              // Enable Paste command
```

```
ScrapDirty = true;                    // Mark scrap as dirty
WindowDirty (true);                   // Mark window as dirty
```

```
   }  /* end DoCut */

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

void DoCopy (void)

   // Handle Copy command.

   {
      ScrollToSelection ();                     // Make sure selection is visible

      TECopy (TheText);                            // Copy the selection

      EnableItem (EditMenu, PasteItem);         // Enable Paste command

      ScrapDirty = true;                        // Mark scrap as dirty

   }  /* end DoCopy */

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

void DoPaste (void)

   // Handle Paste command.

   {
      ScrollToSelection ();                     // Make sure selection is visible

      TEPaste (TheText);                        // Paste the scrap

      AdjustScrollBar ();                       // Adjust scroll bar to length of text
      AdjustText ();                               // Adjust text to match scroll bar
      ScrollToSelection ();                     // Keep selection visible

      DisableItem (EditMenu, CutItem);             // Disable menu items that operate
      DisableItem (EditMenu, CopyItem);         //    on a nonempty selection
      DisableItem (EditMenu, ClearItem);

      WindowDirty (true);                       // Mark window as dirty

   }  /* end DoPaste */

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

void DoClear (void)

   // Handle Clear command.

   {
      ScrollToSelection ();                     // Make sure selection is visible
```

```
TEDelete (TheText);                    // Delete the selection

AdjustScrollBar ();                    // Adjust scroll bar to length of text
```

```
      AdjustText ();                                  // Adjust text to match scroll bar
      ScrollToSelection ();                          // Keep insertion point visible

      DisableItem (EditMenu, CutItem);                // Disable menu items that operate
      DisableItem (EditMenu, CopyItem);              //    on a nonempty selection
      DisableItem (EditMenu, ClearItem);

      WindowDirty (true);                            // Mark window as dirty

   }  /* end DoClear */

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

void DoSysClick (WindowPtr whichWindow)

   // Handle mouse-down event in system window.

   {
      SystemClick (&TheEvent, whichWindow);          // Pass event to Toolbox for handling}

   }  /* end DoSysClick */

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

void DoContent (WindowPtr whichWindow)

   // Handle mouse-down event in content region of active window.

   {
      Point          thePoint;                      // Location of mouse click in window coordinates
      ControlHandle  theControl;                       // Handle to control
      short          thePart;                          // Part of control where mouse was pressed
      Rect           textRect;                      // Window's text rectangle


      if ( whichWindow != FrontWindow() )                  // Is it an inactive window?
         SelectWindow (whichWindow);                      // If so, just activate it

      else
         {
            thePoint = TheEvent.where;                     // Get point in screen coordinates
            GlobalToLocal (&thePoint);                     // Convert to window coordinates

            thePart = FindControl (thePoint, whichWindow, &theControl);  // Was mouse pressed in a control?

            if ( theControl == TheScrollBar )             // Was it in the scroll bar?
               DoScroll (thePart, thePoint);                  // Go scroll the window

            else
               {
                  textRect = (**TheText).viewRect;            // Get text rectangle
```

```
    if ( (theControl == nil)                  // Not in a control?
         && ( PtInRect(thePoint, &textRect) ) ) // In text rectangle?
       DoSelect (thePoint);                     // Go handle text selection
```

```
            }  /* end else */

         }  /* end else */

   }  /* end DoContent */

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

void DoScroll (short thePart, Point thePoint)

   // Handle mouse-down event in scroll bar.

   {
      if ( thePart == inThumb )                          // Dragging the indicator?

         {
            TrackControl (TheScrollBar, thePoint, nil);   // Track mouse with no action procedure
            AdjustText ();                                // Adjust text to new setting

         }  /* end if ( thePart == inThumb ) */

      else

         TrackControl (TheScrollBar, thePoint, ProcPtr(ScrollText));  // Track mouse with continuous scroll

   }  /* end DoScroll */

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

pascal void ScrollText (ControlHandle theControl, short thePart)

   // Scroll text within window.

   {
      short  windowHeight;                          // Height of text rectangle in pixels
      short  windowLines;                          // Height of text rectangle in lines
      short  delta;                                // Amount to scroll by, in lines
      short  oldValue;                             // Previous setting of scroll bar


      windowHeight = (**TheText).viewRect.bottom - (**TheText).viewRect.top;  // Find height of text rectangl
      windowLines  = windowHeight / (**TheText).lineHeight;                   // Convert to lines

      switch ( thePart )
         {
            case inUpButton:
               delta = -1;                          // Scroll up one line at a time
               break;

            case inDownButton:
```

```
  delta = +1;                    // Scroll down one line at a time
  break;
```

```
            case inPageUp:
               delta = 1 - windowLines;            // Scroll up by height of text rectangle
               break;

            case inPageDown:
               delta = windowLines - 1;            // Scroll down by height of text rectangle
               break;

            default:
               break;

         }  /* end switch ( thePart ) */

      if ( thePart != 0 )                          // Is mouse still in the original part?
         {
            oldValue = GetCtlValue (theControl);   // Get old setting
            SetCtlValue (theControl, oldValue + delta);  // Adjust by scroll amount

            AdjustText ();                            // Scroll text to match new setting

         }  /* end if (thePart != 0) */

   }  /* end ScrollText */

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

void AdjustText (void)

   // Adjust text within window to match scroll bar setting.

   {
      short  viewTop;                              // Top of view (clipping) rectangle
      short  destTop;                              // Top of destination (wrapping) rectangle
      short  topLine;                              // First line currently visible in window
      short  heightPerLine;                        // Height of each text line in pixels
      short  oldScroll;                               // Old text offset in pixels
      short  newScroll;                               // New text offset in pixels


      viewTop       = (**TheText).viewRect.top;      // Get top of clip
      destTop       = (**TheText).destRect.top;      // Get top of wrap
      topLine       = GetCtlValue(TheScrollBar);     // Get current top line
      heightPerLine = (**TheText).lineHeight;        // Get height per line

      oldScroll = viewTop - destTop;               // Find current offset
      newScroll = topLine * heightPerLine;         // Scroll bar gives new offset

      if ( oldScroll != newScroll )                   // Any difference?
         TEScroll (0, (oldScroll - newScroll), TheText);// Scroll by difference

   }  /* end AdjustText */
```

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

```
pascal Boolean AutoScroll (void)

   // Handle automatic scrolling during text selection.

   {
      Point      mousePoint;                     // Mouse location in local (window) coordinates
      Rect       textRect;                          // Active window's text rectangle
      RgnHandle  saveClip;                          // Original clipping region on entry


      saveClip = NewRgn ();                        // Create temporary region
      GetClip  (saveClip);                          // Set it to existing clipping region
      ClipRect ( &(TheWindow->portRect) );         // Clip to entire port rectangle

         GetMouse (&mousePoint);                    // Find mouse location
         textRect = (**TheText).viewRect;           // Get text rectangle

         if ( mousePoint.v < textRect.top )         // Above top of rectangle?
            ScrollText (TheScrollBar, inUpButton);  // Scroll up one line

         else if ( mousePoint.v > textRect.bottom ) // Below bottom of rectangle?
            ScrollText (TheScrollBar, inDownButton); // Scroll down one line

         /* else do nothing */

      SetClip    (saveClip);                       // Restore original clipping region
      DisposeRgn (saveClip);                       // Dispose of temporary region

      return true;                                 // Continue tracking mouse

   }  /* end AutoScroll */

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

void DoSelect (Point thePoint)

   // Handle mouse-down event in text rectangle.

   {
      Boolean  extend;                              // Extend existing selection (Shift-click)?


      extend = ( (TheEvent.modifiers & shiftKey) != 0); // Shift key down?

      TEClick (thePoint, extend, TheText);          // Do text selection

      FixEditMenu ();                               // Enable/disable menu items

   }  /* end DoSelect */

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

```
void FixEditMenu (void)
```

```
   // Enable/disable editing commands.

   {
      short  selFirst;                              // Character position at start of selection
      short  selLast;                               // Character position at end of selection


      DisableItem (EditMenu, UndoItem);             // Disable Undo command


      selFirst = (**TheText).selStart;                 // Get start of selection
      selLast  = (**TheText).selEnd;                // Get end of selection

      if ( selFirst == selLast )                       // Is selection empty?
         {
            DisableItem (EditMenu, CutItem);           // Disable menu items that operate
            DisableItem (EditMenu, CopyItem);       //    on a nonempty selection
            DisableItem (EditMenu, ClearItem);

         }  /* end if ( selFirst == selLast ) */

      else
         {
            EnableItem (EditMenu, CutItem);            // Enable menu items that operate
            EnableItem (EditMenu, CopyItem);           //    on a nonempty selection
            EnableItem (EditMenu, ClearItem);

         }  /* end else */


      if ( TEGetScrapLen() == 0 )                   // Is scrap empty?
         DisableItem (EditMenu, PasteItem);            // Disable Paste command
      else
         EnableItem  (EditMenu, PasteItem);            // Enable Paste command

   }  /* end FixEditMenu */

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

void DoDrag (WindowPtr whichWindow)

   // Handle mouse-down event in drag region.

   {
      Rect  limitRect;                           // Limit rectangle for dragging


      SetRect    (&limitRect, 0, MenuBarHeight, ScreenWidth, ScreenHeight); // Set limit rectangle
      InsetRect  (&limitRect, ScreenMargin, ScreenMargin);                  // Inset by screen margin

      DragWindow (whichWindow, TheEvent.where, &limitRect);                    // Let user drag the window
```

```
}  /* end DoDrag */
```

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

void DoGrow (WindowPtr whichWindow)

   // Handle mouse-down event in size region.

   {
      Rect   sizeRect;                              // Minimum and maximum dimensions of window
      long   newSize;                               // Coded representation of new dimensions
      short  newWidth;                              // New width of window
      short  newHeight;                                // New height of window


      SetRect (&sizeRect,                           // Set size rectangle
                 MinWidth,
                 MinHeight,
                 ScreenWidth,                       // Maximum width is full screen
                 (ScreenHeight - MenuBarHeight) );// Maximum height is full screen minus menu bar

      newSize = GrowWindow (whichWindow, TheEvent.where, &sizeRect);  // Let user drag size region

      if ( newSize != 0 )                           // Was size changed?
         {
            EraseRect ( &(whichWindow->portRect) );   // Clear window to white

            newWidth  = LoWord(newSize);               // Extract width from low word
            newHeight = HiWord(newSize);               // Extract height from high word
            SizeWindow (whichWindow, newWidth, newHeight, true); // Adjust size of window

            InvalRect ( &(whichWindow->portRect) );   // Force update of window's contents

            FixScrollBar ();                        // Resize scroll bar
            FixText ();                             // Resize text rectangle

         }  /* end if ( newSize != 0 ) */

   }  /* end DoGrow */

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

void FixScrollBar (void)

   // Resize window's scroll bar.

   {
      HideControl (TheScrollBar);                   // Hide scroll bar

      MoveControl (TheScrollBar,                       // Move top-left corner
                   TheWindow->portRect.right
                        - (SBarWidth - 1),          // Allow for 1-pixel overlap at right
                   -1);                             // Overlap window top by 1 pixel
```

```
SizeControl (TheScrollBar,                    // Adjust bottom-right corner
             SBarWidth,
```

```
                (TheWindow->portRect.bottom + 1)
                    - (TheWindow->portRect.top - 1)
                    - (SBarWidth - 1) );          // Allow room for size box

      ShowControl (TheScrollBar);                 // Redisplay scroll bar

      ValidRect ( &( (**TheScrollBar).contrlRect ) );// Avoid updating again

   }  /* end FixScrollBar */

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

void FixText (void)

   // Resize window's text rectangle.

   {
      short  topLine;                             // First line visible in window
      short  firstChar;                              // Character position of first character in window
      short  heightPerLine;                       // Height of each line in pixels
      Rect  *textRect;                            // Pointer to text rectangle


      SetCursor ( &(**Watch) );                      // Indicate delay

      topLine       = GetCtlValue (TheScrollBar);    // Get previous first line
      firstChar     = (**TheText).lineStarts[topLine];  // Find first character previously visible
      heightPerLine = (**TheText).lineHeight;        // Get line height

      textRect  = &( (**TheText).destRect );         // Point to wrapping rectangle
      *textRect = TheWindow->portRect;               // Wrap text to window's port rectangle
      (*textRect).right  -= (SBarWidth - 1);         // Exclude scroll bar, allowing 1-pixel overlap
      (*textRect).bottom -= (SBarWidth - 1);         // Leave space for scroll bar at bottom
      (*textRect).bottom /= heightPerLine;        // Truncate to a whole
      (*textRect).bottom *= heightPerLine;        //    number of lines

      (**TheText).viewRect = *textRect;           // Clip to same rectangle
      InsetRect (textRect, TextMargin, TextMargin);  // Inset wrap by text margin

      TECalText (TheText);                           // Recalibrate line starts
      AdjustScrollBar ();                         // Adjust scroll bar to new length

      ScrollCharacter (firstChar, false);            // Scroll same character to top of window

   }  /* end FixText */

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

void DoGoAway (WindowPtr whichWindow)

   // Handle mouse-down event in close region.
```

```
{
   if ( TrackGoAway(whichWindow, TheEvent.where) )    // Track mouse in close region and
```

```
      DoClose ();                                 //    close window if necessary

   }  /* end DoGoAway */

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

void DoZoom (WindowPtr whichWindow, short inOrOut)

   // Handle mouse-down event in zoom region.

   {
      if ( TrackBox(whichWindow, TheEvent.where, inOrOut) )      // Track mouse in zoom region
         {
            EraseRect  ( &(whichWindow->portRect) );          // Clear window to white

            ZoomWindow (whichWindow, inOrOut, false);         // Zoom the window

            InvalRect  ( &(whichWindow->portRect) );          // Force update of window's contents

            FixScrollBar ();                                 // Resize scroll bar
            FixText ();                                      // Resize text rectangle

         } /* end if ( TrackBox(whichWindow, TheEvent.where, inOrOut) ) */

   }  /* end DoZoom */

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

void DoKeystroke (void)

   // Handle keystroke.

   {
      unsigned char  ch;                            // Character that was typed
      long           menuChoice;                    // Menu ID and item number for Command combination


      ch = TheEvent.message & charCodeMask;          // Extract character code

      if ( (TheEvent.modifiers & cmdKey) != 0 )      // Command key down?
         {
            if ( TheEvent.what != autoKey )          // Ignore repeats
               {
                  menuChoice = MenuKey(ch);          // Get menu equivalent
                  DoMenuChoice (menuChoice);         // Handle as menu choice

               }  /* end if ( TheEvent.what != autoKey ) */

         }  /* end if ( (TheEvent.modifiers & cmdKey) != 0 ) */

      else
```

```
      DoTyping (ch);                          // Handle as normal character

}  /* end DoKeystroke */
```

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

void DoTyping (unsigned char ch)

   // Handle character typed from keyboard.

   {
      if ( TheText == nil )                        // Is screen empty?
         {
            SysBeep(1);                              // Just beep
            return;                                  //    and exit
         } /* end if ( TheText == nil ) */

      ScrollToSelection ();                        // Make sure insertion point is visible

      TEKey (ch, TheText);                            // Process character

      AdjustScrollBar ();                             // Adjust scroll bar to length of text
      AdjustText ();                                  // Adjust text to match scroll bar
      ScrollToSelection ();                        // Keep insertion point visible

      DisableItem (EditMenu, CutItem);                // Disable menu items that operate
      DisableItem (EditMenu, CopyItem);               //    on a nonempty selection
      DisableItem (EditMenu, ClearItem);

      WindowDirty (true);                             // Mark window as dirty

   }  /* end DoTyping */

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

void DoUpdate (void)

   // Handle update event.

   {
      GrafPtr    savePort;                         // Pointer to previous current port
      WindowPtr  whichWindow;                      // Pointer to window to be updated
      long       theRefCon;                     // Window's reference constant
      WDHandle   theData;                       // Handle to window's data record
      TEHandle   theEditRec;                    // Handle to window's edit record
      Rect       theViewRect;                      // Edit record's view rectangle


      GetPort (&savePort);                          // Save previous port

         whichWindow = WindowPtr(TheEvent.message);   // Convert long integer to pointer
         SetPort (whichWindow);                    // Make window the current port

         BeginUpdate (whichWindow);                   // Restrict visible region to update region
```

```
EraseRect ( &(whichWindow->portRect) );   // Clear update region
```

```
            DrawGrowIcon (whichWindow);            // Redraw size box
            DrawControls (whichWindow);            // Redraw scroll bar

            theRefCon   = GetWRefCon(whichWindow);    // Get reference constant
            theData     = WDHandle(theRefCon);        // Convert to data handle
            theEditRec  = (**theData).editRec;        // Get edit record
            theViewRect = (**theEditRec).viewRect;    // Get view rectangle
            TEUpdate (&theViewRect, theEditRec);   // Redraw the text

         EndUpdate (whichWindow);                  // Restore original visible region

      SetPort (savePort);                          // Restore original port

   }  /* end DoUpdate */

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

void DoActivate (void)

   // Handle activate (or deactivate) event.

   {
      WindowPtr  whichWindow;                        // Pointer to the window


      whichWindow = WindowPtr( TheEvent.message );   // Convert long integer to pointer

      if ( (TheEvent.modifiers & activeFlag) != 0 )  // Test activate/deactivate bit
         ActWindow   (whichWindow);                  // Activate window
      else
         DeactWindow (whichWindow);                  // Deactivate window

   }  /* end DoActivate */

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

void ActWindow (WindowPtr whichWindow)

   // Activate window.

   {
      const short  active = 0;                    // Highlighting code for active scroll bar

      long      theRefCon;                           // Window's reference constant
      WDHandle  theData;                          // Handle to window data record
      short     theFile;                          // Reference number of window's file


      theRefCon = GetWRefCon(whichWindow);        // Get reference constant
      theData   = WDHandle(theRefCon);               // Convert to data handle
```

```
TheWindow    = whichWindow;                // Set global pointers/handles
TheScrollBar = (**theData).scrollBar;
TheText      = (**theData).editRec;
```

```
   SetPort (TheWindow);                          // Make window the current port

   DrawGrowIcon  (TheWindow);                    // Highlight size box
   HiliteControl (TheScrollBar, active);         // Activate scroll bar
   TEActivate    (TheText);                    // Highlight selection

   if ( (TheEvent.modifiers & ChangeFlag) != 0 )  // Coming from a system window?
      ReadDeskScrap ();                          // Copy desk scrap to Toolbox scrap

   FixEditMenu ();                             // Enable/disable editing commands

   EnableItem (FileMenu, SaveAsItem);            // Enable Save As... command
   if ( (**theData).dirty )                    // Is document dirty?
      {
         EnableItem (FileMenu, SaveItem);        // Enable Save command

         theFile = (**theData).fileNumber;     // Get file reference number
         if ( theFile != 0 )                   // Is there a file to revert to?
            EnableItem (FileMenu, RevertItem);    // Enable Revert command

      }  /* end if ( (**theData).dirty ) */

   }  /* end ActWindow */

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

void DeactWindow (WindowPtr whichWindow)

   // Deactivate window.

   {
      const short  inactive = 255;               // Highlighting code for inactive scroll bar

      long      theRefCon;                       // Window's reference constant
      WDHandle  theData;                         // Handle to window data record


      theRefCon = GetWRefCon(whichWindow);       // Get reference constant
      theData   = WDHandle(theRefCon);              // Convert to data handle

      TheWindow    = nil;                        // Clear global pointers/handles
      TheScrollBar = nil;
      TheText      = nil;

      SetPort (whichWindow);                     // Make window the current port

      DrawGrowIcon  (whichWindow);                  // Unhighlight size box
      HiliteControl ((**theData).scrollBar, inactive);  // Deactivate scroll bar
      TEDeactivate  ((**theData).editRec);       // Unhighlight selection

      if ( (TheEvent.modifiers & ChangeFlag) != 0 )  // Exiting to a system window?
```

```
{
   WriteDeskScrap ();                    // Copy Toolbox scrap to desk scrap
```

```
            EnableItem (EditMenu, UndoItem);          // Enable standard editing commands
            EnableItem (EditMenu, CutItem);           //    for desk accessory
            EnableItem (EditMenu, CopyItem);
            EnableItem (EditMenu, PasteItem);
            EnableItem (EditMenu, ClearItem);

         }  /* end if ( (TheEvent.modifiers & ChangeFlag) != 0 ) */

      DisableItem (FileMenu, SaveItem);           // Disable filing commands for desk
      DisableItem (FileMenu, SaveAsItem);            //    accessory or empty desk
      DisableItem (FileMenu, RevertItem);

   }  /* end DeactWindow */

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

void Finalize (void)

   // Do one-time-only finalization.

   {
      WriteDeskScrap ();                          // Copy Toolbox scrap to desk scrap

      SetEventMask (OldMask);                        // Restore original value of system event mask

   }  /* end Finalize */

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

void WindowDirty (Boolean isDirty)

   // Mark window dirty or clean.

   {
      long      theRefCon;                          // Window's reference constant
      WDHandle  theData;                         // Handle to window data record
      short     theFile;                         // Reference number of window's file


      theRefCon = GetWRefCon(TheWindow);             // Get reference constant
      theData   = WDHandle(theRefCon);               // Convert to data handle

      (**theData).dirty = isDirty;                   // Set flag in data record

      if ( isDirty )                                 // Is window becoming dirty or clean?
         {
            EnableItem (FileMenu, SaveItem);         // Enable Save command

            theFile = (**theData).fileNumber;      // Get file reference number
            if ( theFile != 0 )                    // Is window associated with a file?
```

```
    EnableItem (FileMenu, RevertItem);     // Enable Revert command

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

```
      else
         {
            DisableItem (FileMenu, SaveItem);      // Disable menu items
            DisableItem (FileMenu, RevertItem);

         }  /* end else */

   }  /* end WindowDirty */

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

void AdjustScrollBar (void)

   // Adjust scroll bar to length of document.

   {
      const short  active   =   0;                  // Highlighting code for active scroll bar
      const short  inactive = 255;                  // Highlighting code for inactive scroll bar

      short  windowTop;                             // Top of window's text rectangle
      short  windowBottom;                          // Bottom of window's text rectangle
      short  heightPerLine;                        // Height of each text line in pixels
      short  windowHeight;                          // Height of text rectangle in pixels
      short  windowLines;                          // Height of text rectangle in lines
      short  textLines;                             // Total lines of text
      short  maxTop;                                // Maximum value for top line in window


      windowTop     = (**TheText).viewRect.top;     // Get top of rectangle
      windowBottom  = (**TheText).viewRect.bottom;  // Get bottom of rectangle
      textLines     = (**TheText).nLines;           // Get total lines
      heightPerLine = (**TheText).lineHeight;       // Get height per line

      windowHeight = windowBottom - windowTop;      // Find height of text rectangle
      windowLines  = windowHeight / heightPerLine;  // Convert to lines
      maxTop       = textLines - windowLines;       // Avoid white space at bottom

      if ( maxTop <= 0 )                           // Is text smaller than window?
         {
            maxTop = 0;                            // Show all of text
            HiliteControl (TheScrollBar, inactive);  // Disable scroll bar

         }  /* end if ( maxTop <= 0 ) */
      else
         HiliteControl (TheScrollBar, active);      // Enable scroll bar

      SetCtlMax (TheScrollBar, maxTop);            // Adjust range of scroll bar

   }  /* end AdjustScrollBar */

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

```
void ScrollToSelection (void)
```

```
   // Scroll current selection into view.

   {
      short  windowTop;                              // Top of window's text rectangle
      short  windowBottom;                           // Bottom of window's text rectangle
      short  heightPerLine;                        // Height of each text line in pixels
      short  windowHeight;                           // Height of text rectangle in pixels
      short  windowLines;                          // Height of text rectangle in lines
      short  topLine;                              // First line visible in window
      short  bottomLine;                           // First line beyond bottom of window
      short  topChar;                              // First character position visible in window
      short  bottomChar;                           // First character position beyond bottom of window
      short *lineBreaks;                             // Pointer to array of line starts
      short  selFirst;                             // Character position at start of selection
      short  selLast;                              // Character position at end of selection


      windowTop     = (**TheText).viewRect.top;      // Get top of rectangle
      windowBottom  = (**TheText).viewRect.bottom;   // Get bottom of rectangle
      heightPerLine = (**TheText).lineHeight;        // Get height per line

      windowHeight = windowBottom - windowTop;       // Find height of text rectangle in pixels
      windowLines  = windowHeight / heightPerLine;   // Convert to lines

      topLine    = GetCtlValue(TheScrollBar);        // Get current top line
      bottomLine = topLine + windowLines;            // Find line beyond bottom

      lineBreaks = (**TheText).lineStarts;         // Point to start of line array
      topChar    = lineBreaks[topLine];            // Find first visible character
      bottomChar = lineBreaks[bottomLine];         // Find first character beyond bottom

      selFirst = (**TheText).selStart;               // Get start of selection
      selLast  = (**TheText).selEnd;               // Get end of selection

      if ( GetCtlMax(TheScrollBar) == 0 )            // Not enough text to fill the window?
         AdjustText ();                              // Start of text to top of window
      else if ( selLast < topChar )                  // Whole selection above window top?
         ScrollCharacter (selFirst, false);          // Start of selection to top of window
      else if ( selFirst >= bottomChar )             // Whole selection below window bottom?
         ScrollCharacter (selLast, true);            // End of selection to bottom of window

   }  /* end ScrollToSelection */

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

void ScrollCharacter (short theCharacter, Boolean toBottom)

   // Scroll character into view.

   {
      short  windowTop;                                // Top of window's text rectangle
      short  windowBottom;                             // Bottom of window's text rectangle
```

```
short  heightPerLine;                    // Height of each text line in pixels
short  windowHeight;                        // Height of text rectangle in pixels
```

```
      short  windowLines;                          // Height of text rectangle in lines
      short *lineBreaks;                          // Pointer to array of line starts
      short  thisLine;                            // Number of line containing character


      lineBreaks = (**TheText).lineStarts;        // Point to start of line array
      for ( (thisLine = 0); (lineBreaks[thisLine + 1] <= theCharacter); (thisLine++) )
         {  /* Search for line containing character */ }

      if ( toBottom )                             // Scrolling to bottom of window?
         {
            windowTop     = (**TheText).viewRect.top;    // Get top of rectangle
            windowBottom  = (**TheText).viewRect.bottom; // Get bottom of rectangle
            heightPerLine = (**TheText).lineHeight;      // Get height per line

            windowHeight = windowBottom - windowTop;     // Find height of text rectangle in pixels
            windowLines  = windowHeight / heightPerLine; // Convert to lines
            thisLine -= (windowLines - 1);               // Offset for window height

         }  /* end if ( toBottom ) */

      SetCtlValue (TheScrollBar, thisLine);             // Adjust setting of scroll bar
      AdjustText ();                                    // Scroll text to match new setting

   }  /* end ScrollCharacter */

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

void ReadDeskScrap (void)

   // Read desk scrap into Toolbox scrap.

   {
      PScrapStuff  scrapInfo;                       // Pointer to scrap information record
      long         scrapLength;                     // Length of desk text scrap in bytes
      long         ignore;                          // Dummy variable for scrap offset
      OSErr        result;                          // Result code from scrap transfer


      scrapInfo = InfoScrap();                    // Get scrap info

      if ( ScrapCompare != scrapInfo->scrapCount )   // Has scrap count changed?
         {
            scrapLength = GetScrap(nil, 'TEXT', &ignore);  // Check desk scrap for a text item

            if ( scrapLength >= 0 )                 // Is there a text item?
               {
                  result = TEFromScrap();           // Transfer desk scrap to Toolbox scrap
                  if ( result != noErr )          // Was there an error?
                     scrapLength = result;        // Make sure scrap length is negative

            }  /* end if ( scrapLength >= 0 ) */
```

```
if ( scrapLength > 0 )                  // Was scrap nonempty?
```

```
            EnableItem (EditMenu, PasteItem);   // Enable Paste command
         else
            {
               TESetScrapLen (0);               // Mark Toolbox scrap as empty
               DisableItem (EditMenu, PasteItem);  // Disable Paste command

            }  /* end else */

         scrapInfo    = InfoScrap();            // Get scrap info
         ScrapCompare = scrapInfo->scrapCount;     // Save scrap count for later comparison

      }  /* end if ( ScrapCompare() <> scrapInfo->scrapCount ) */

   }  /* end ReadDeskScrap */

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

void WriteDeskScrap (void)

   // Write Toolbox scrap to desk scrap.

   {
      PScrapStuff  scrapInfo;                     // Pointer to scrap information record
      long         scrapResult;                   // Result code from ZeroScrap
      OSErr        teResult;                   // Result code from scrap transfer


      if ( ScrapDirty )                           // Has scrap changed since last read?
         {
            scrapResult = ZeroScrap();            // Change scrap count
            if ( scrapResult == noErr )         // Was there an error?
               {
                  scrapInfo    = InfoScrap();      // Get scrap info
                  ScrapCompare = scrapInfo->scrapCount;  // Save new scrap count for comparison

               }  /* end if ( scrapResult == noErr ) */

            teResult = TEToScrap ();            // Transfer Toolbox scrap to desk scrap

            ScrapDirty = false;                 // Toolbox and desk scraps now agree

         }  /* end if ( ScrapDirty ) */

   }  /* end WriteDeskScrap */

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

void IOCheck (OSErr resultCode)

   // Check for I/O error.
```

```
{
   short   alertID;                        // Resource ID of alert
   Str255  errorString;                       // Error code in string form
```

```
   if ( resultCode == noErr )                    // Just return if no error
      return;

   switch ( resultCode )
      {
         case opWrErr:                          // File already open?
            alertID = OpWrID;                   // Use Already Open alert
            break;

         /* Insert code here to handle any other specific errors */

         default:
            alertID = IOErrID;                      // Use general I/O Error alert
            NumToString (resultCode, errorString);      // Convert error code to a string
            ParamText (errorString, "\p", "\p", "\p");  // Substitute into text of alert
            break;

      }  /* end switch ( resultCode ) */

   InitCursor ();                                   // Restore normal cursor
   StopAlert (alertID, nil);                        // Post alert

   Quitting  = false;                           // Cancel Quit command, if any
   ErrorFlag = true;                                // Force exit to main event loop

}  /* end IOCheck */
```

The resource description file, **MiniEdit.r**, was generated by the DeRez resource decompiler from the resources I built onscreen with the interactive resource editor ResEdit.

```
//
//                                  MiniEdit
//                        Example Macintosh application program
//                           S. Chernicoff    24 October 1994
//
//
//    Multiple-window text editor to illustrate event-driven program structure
//
//
//    MiniEdit.r - Rez source file


#include "systypes.r"
#include "types.r"

data 'MENU' (1, "Apple") {
   $"0001 0000 0000 0000 0000 FFFF FFFB 0114"            /* ..........ˇˇˇ°.. */
   $"0F41 626F 7574 204D 696E 6945 6469 74C9"            /* .About MiniEdit… */
   $"0000 0000 012D 0000 0000 00"                        /* .....-..... */
};

data 'MENU' (2, "File") {
   $"0002 0000 0000 0000 0000 FFFF F007 0446"            /* ..........ˇˇ..F */
   $"696C 6503 4E65 7700 4E00 0005 4F70 656E"            /* ile.New.N...Open */
   $"C900 4F00 0005 436C 6F73 6500 5700 0001"            /* ….O...Close.W... */
   $"2D00 0000 0004 5361 7665 0053 0000 0853"            /* -.....Save.S...S */
   $"6176 6520 4173 C900 4100 000F 5265 7665"            /* ave As….A...Reve */
   $"7274 2074 6F20 5361 7665 6400 5200 0001"            /* rt to Saved.R... */
   $"2D00 0000 000B 5061 6765 2053 6574 7570"            /* -.....Page Setup */
   $"C900 5500 0006 5072 696E 74C9 0050 0000"            /* ….U...Print….P.. */
   $"012D 0000 0000 0451 7569 7400 5100 0000"            /* .-.....Quit.Q... */
};

data 'MENU' (3, "Edit") {
   $"0003 0000 0000 0000 0000 FFFF FF01 0445"            /* ..........ˇˇˇ..E */
   $"6469 7404 556E 646F 005A 0000 012D 0000"            /* dit.Undo.Z...-.. */
   $"0000 0343 7574 0058 0000 0443 6F70 7900"            /* ...Cut.X...Copy. */
   $"4300 0005 5061 7374 6500 5600 0001 2D00"            /* C...Paste.V...-. */
   $"0000 0005 436C 6561 7200 4200 0000"                 /* ....Clear.B... */
};

data 'ALRT' (1000, "About MiniEdit") {
   $"003C 005A 00DC 01A4 03E8 4444"                      /* .<.Z.‹.§.ËDD */
};

data 'ALRT' (1001, "Save") {
   $"003C 0078 00B8 0166 03E9 5555"                      /* .<.x.∏.f.ÈUU */
};

data 'ALRT' (1002, "Revert") {
   $"003C 0057 009A 01A9 03EA 5555"                      /* .<.W.ö.©.ÍUU */
};
```

```
data 'ALRT' (1003, "Can’t Print") {
   $"003C 0064 00A4 019A 03EB 4444"                    /* .<.d.§.ö.ÎDD */
};

data 'ALRT' (1004, "Wrong Type") {
   $"003C 0064 00AC 019C 03EC 5555"                    /* .<.d.¨.ú.ÏUU */
};

data 'ALRT' (1005, "File Too Long") {
   $"003C 0070 00AA 0190 03ED 5555"                    /* .<.p.™.ê.ÌUU */
};

data 'ALRT' (1006, "Already Open") {
   $"003C 005C 008C 01A4 03EE 5555"                    /* .<.\.å.§.ÓUU */
};

data 'ALRT' (1007, "I/O Error") {
```

```
   $"003C 00A5 008C 015B 03EF 5555"                    /* .<.•.å.[.ÔUU */
};

data 'DITL' (1000, "About MiniEdit") {
   $"0001 0000 0000 0082 0082 0096 00C8 0408"          /* .......Ç.Ç.ñ.».. */
   $"436F 6E74 696E 7565 0000 0000 000A 0014"          /* Continue........ */
   $"007E 0136 88AC 2020 2020 2020 2020 2020"          /* .~.6à¨          */
   $"2020 2020 2020 2020 2020 2020 2020 2020"          /*                 */
   $"4D69 6E69 4564 6974 2043 2B2B 0D0D D24D"          /* MiniEdit C++¬¬“M */
   $"6163 696E 746F 7368 2052 6576 6561 6C65"          /* acintosh Reveale */
   $"64D3 2065 7861 6D70 6C65 2061 7070 6C69"          /* d” example appli */
   $"6361 7469 6F6E 0D0D 2020 2020 2020 2020"          /* cation¬¬         */
   $"2020 2020 2020 2020 2020 2020 2020 2020"          /*                 */
   $"2020 432B 2B20 7665 7273 696F 6E0D 0D20"          /*   C++ version¬¬  */
   $"532E 2043 6865 726E 6963 6F66 6620 2020"          /* S. Chernicoff    */
   $"2020 2020 2020 2020 2020 2020 2020 2020"          /*                 */
   $"2020 2032 3420 4F63 746F 6265 7220 3139"          /*    24 October 19 */
   $"3934"                                             /* 94 */
};

data 'DITL' (1001, "Save") {
   $"0003 0000 0000 0040 0014 0054 005A 0404"          /* .......@...T.Z.. */
   $"5361 7665 0000 0000 005E 0014 0072 005A"          /* Save.....^...r.Z */
   $"0407 4469 7363 6172 6400 0000 0000 005E"          /* ..Discard......^ */
   $"009B 0072 00E1 0406 4361 6E63 656C 0000"          /* .õ.r.·..Cancel.. */
   $"0000 000A 003E 003C 00E1 881E 5361 7665"          /* .....>.<.·à.Save */
   $"2066 696C 6520 D25E 30D3 0D62 6566 6F72"          /*  file “^0”¬befor */
   $"6520 636C 6F73 696E 673F"                         /* e closing? */
};

data 'DITL' (1002, "Revert") {
   $"0002 0000 0000 0040 0014 0054 005A 0406"          /* .......@...T.Z.. */
   $"5265 7665 7274 0000 0000 0040 00F8 0054"          /* Revert.....@.¯.T */
   $"013E 0406 4361 6E63 656C 0000 0000 000A"          /* .>..Cancel...... */
   $"003C 003C 013E 8833 5265 7665 7274 2074"          /* .<.<.>à3Revert t */
   $"6F20 6D6F 7374 2072 6563 656E 746C 7920"          /* o most recently  */
   $"7361 7665 6420 7665 7273 696F 6E20 6F66"          /* saved version of */
   $"2066 696C 6520 D25E 30D3 3F00"                    /*  file “^0”?. */
};

data 'DITL' (1003, "Can’t Print") {
   $"0001 0000 0000 0046 00D4 0058 011A 0404"          /* .......F.‘.X.... */
   $"5175 6974 0000 0000 000A 003C 002A 0121"          /* Quit.......<.*.! */
   $"8834 536F 7272 792C 2074 6869 7320 7665"          /* à4Sorry, this ve */
   $"7273 696F 6E20 6F66 204D 696E 6945 6469"          /* rsion of MiniEdi */
   $"7420 0D63 616E D574 2070 7269 6E74 2061"          /* t ¬can’t print a */
   $"2066 696C 652E"                                   /*  file. */
};

data 'DITL' (1004, "Wrong Type") {
   $"0001 0000 0000 0052 00DE 0066 0124 0402"          /* .......R.fi.f.$.. */
   $"4F4B 0000 0000 000A 003E 004E 0124 884B"          /* OK.......>.N.$àK */
   $"536F 7272 792C 204D 696E 6945 6469 7420"          /* Sorry, MiniEdit  */
```

```
   $"776F 726B 7320 7769 7468 2074 6578 7420"           /* works with text  */
```

```
   $"6669 6C65 7320 6F6E 6C79 2E20 2043 616E"          /* files only.  Can */
   $"2774 2072 6561 6420 6F72 2077 7269 7465"          /* 't read or write */
   $"2066 696C 6520 D25E 30D3 2E00"                    /*  file “^0”.. */
};

data 'DITL' (1005, "File Too Long") {
   $"0001 0000 0000 0050 00CE 0064 0114 0406"          /* .......P.Œ.d.... */
   $"4361 6E63 656C 0000 0000 000A 003E 004C"          /* Cancel.......>.L */
   $"010C 8831 536F 7272 792C 2066 696C 6520"          /* ..à1Sorry, file  */
   $"D25E 30D3 0D69 7320 746F 6F20 6C6F 6E67"          /* “^0”¬is too long */
   $"2066 6F72 204D 696E 6945 6469 7420 746F"          /*  for MiniEdit to */
   $"2072 6561 642E"                                   /*  read. */
};

data 'DITL' (1006, "Already Open") {
   $"0001 0000 0000 0032 00EE 0046 0134 0406"          /* .......2.Ó.F.4.. */
   $"4361 6E63 656C 0000 0000 000A 003E 001C"          /* Cancel.......>.. */
   $"0134 8825 536F 7272 792C 2063 616E 2774"          /* .4à%Sorry, can't */
   $"206F 7065 6E20 7468 6520 7361 6D65 2066"          /*  open the same f */
   $"696C 6520 7477 6963 652E"                         /* ile twice. */
};

data 'DITL' (1007, "I/O Error") {
   $"0001 0000 0000 0032 005C 0046 00A2 0406"          /* .......2.\.F.¢.. */
   $"4361 6E63 656C 0000 0000 000A 003E 001C"          /* Cancel.......>.. */
   $"00A2 880C 492F 4F20 4572 726F 7220 5E30"          /* .¢à.I/O Error ^0 */
};

data 'WIND' (1000, "MiniEdit window template") {
   $"0046 0032 0116 015E 0008 0000 0100 0000"          /* .F.2...^........ */
   $"0000 0875 6E74 6974 6C65 64"                      /* ...untitled */
};

data 'STR ' (1000, "Default window title") {
   $"0875 6E74 6974 6C65 64"                           /* .untitled */
};

data 'CNTL' (1000, "Scroll bar template") {
   $"FFFF 011D 00C2 012D 0000 0100 0000 0000"          /* ˘˘...¬.-........ */
   $"0010 0000 0000 012A"                              /* .......* */
};

data 'BNDL' (128, "MiniEdit bundle") {
   $"4D45 4454 0000 0001 4652 4546 0001 0000"          /* MEDT....FREF.... */
   $"0080 0001 0081 4943 4E23 0001 0000 0080"          /* .Ä...ÅICN#.....Ä */
   $"0001 0081"                                        /* ...Å */
};

data 'FREF' (128, "APPL") {
   $"4150 504C 0000 00"                                /* APPL... */
};

data 'FREF' (129, "TEXT") {
```

```
  $"5445 5854 0001 00"                          /* TEXT... */
```

```
};

data 'ICN#' (128, "MiniEdit APPL file icon") {
   $"FFFF FFFF 8000 0001 8000 0001 FFFF FFFF"            /* ˇˇˇˇÄ...Ä...ˇˇˇˇ */
   $"D555 5555 AAAA AAAB FFFD 5555 A006 AAAB"            /* ’UUU™™™´ˇ˝UU†.™´ */
   $"FFFD 7FFD A006 C00B E005 7FFD A006 C00B"            /* ˇ˝.˝†.¿.‡..˝†.¿. */
   $"E7FF C00D A400 400B E7FF C00D A400 400B"            /* Áˇ¿¬§.@.Áˇ¿¬§.@. */
   $"E400 400D A400 400B FC00 400D AC00 7FFB"            /* ‰.@¬§.@.¸.@¬¨..˚ */
   $"D400 00D5 AC00 00AB D400 00D5 AC00 00AB"            /* ‘..’¨..´‘..’¨..´ */
   $"D400 00D5 AC00 00AB D400 00D5 AC00 00AB"            /* ‘..’¨..´‘..’¨..´ */
   $"D400 00D5 AFFF FFAB D555 5555 FFFF FFFF"            /* ‘..’Øˇˇ´’UUUˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
   $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"            /* ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ */
};

data 'ICN#' (129, "MiniEdit TEXT file icon", purgeable, preload) {
   $"0FFF FC00 0800 0600 0800 0500 0800 0480"            /* .ˇ¸............Ä */
   $"08F8 0440 0800 0420 081F E7F0 0800 0010"            /* .¯.@... ..Á.... */
   $"081F E010 0800 0010 0803 FC10 0800 0010"            /* ..‡.......¸..... */
   $"081F 0010 0800 0010 081F FE10 0800 0010"            /* ..........˛..... */
   $"08FF E010 0800 0010 083C 0010 0800 0010"            /* .ˇ‡......<...... */
   $"080F C010 0800 0010 080F F810 0800 0010"            /* ..¿.......¯..... */
   $"08FC 0010 0800 0010 083F E010 0800 0010"            /* .¸.......?‡..... */
   $"0807 FE10 0800 0010 0800 0010 0FFF FFF0"            /* ..˛...........ˇˇ */
   $"0FFF FC00 0FFF FE00 0FFF FF00 0FFF FF80"            /* .ˇ¸...ˇ˛...ˇˇ...ˇˇÄ */
   $"0FFF FFC0 0FFF FFE0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ¿.ˇˇ‡.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0 0FFF FFF0 0FFF FFF0"            /* .ˇˇ.ˇˇ.ˇˇ.ˇˇ */
   $"0FFF FFF0 0FFF FFF0"                                /* .ˇˇ.ˇˇ */
};

data 'MEDT' (0, "MiniEdit autograph") {
   $"514D 696E 6945 6469 740D 4578 616D 706C"            /* QMiniEdit¬Exampl */
   $"6520 4D61 6369 6E74 6F73 6820 6170 706C"            /* e Macintosh appl */
   $"6963 6174 696F 6E20 7072 6F67 7261 6D0D"            /* ication program¬ */
   $"532E 2043 6865 726E 6963 6F66 662C 2032"            /* S. Chernicoff, 2 */
   $"3420 4F63 746F 6265 7220 3139 3934"                 /* 4 October 1994 */
};
```