## Demonstration Program MonoTextEdit Listing

```
// *******************************************************************************************
// MonoTextEdit.c                                                          CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program demonstrates:
//
// •  A "bare-bones" monostyled text editor.
//
// •  A Help dialog which features the integrated scrolling of multistyled text and pictures.
//
// In the monostyled text editor demonstration, a panel is displayed at the bottom of all
// opened windows.  This panel displays the edit record length, number of lines, line height,
// destination rectangle (top), scroll bar value, and scroll bar maximum value.
//
// The bulk of the source code for the Help dialog is contained in the file HelpDialog.c.
// The dialog itself displays information intended to assist the user in adapting the Help
// dialog source code and resources to the requirements of his/her own application.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Help dialog pop-up
//    menus (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially visible).
//
// •  'CNTL' resources (purgeable) for the vertical scroll bars in the text editor window and
//    Help dialog, and for the pop-up menu in the Help Dialog.
//
// •  A 'DLOG' resource (purgeable, initially invisible) and associated 'dctb' resource
//    (purgeable) for the Help dialog.
//
// •  'TEXT' and associated 'styl' resources (all purgeable) for the Help dialog.
//
// •  'PICT' resources (purgeable) for the Help dialog.
//
// •  A 'STR#' resource  (purgeable) containing error text strings.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************

// .......................................................................................... includes

#include <Carbon.h>

// ........................................................................................... defines

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define  iHelp            2
#define mFile             129
#define  iNew             1
#define  iOpen            2
#define  iClose           4
#define  iSaveAs          6
#define  iQuit            12
#define mEdit             130
#define  iUndo            1
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
#define  iClear           6
```

```
#define  iSelectAll       7
#define rWindow           128
#define rVScrollbar       128
#define rErrorStrings     128
#define  eMenuBar         1
#define  eWindow          2
#define  eDocStructure    3
#define  eEditRecord      4
#define  eExceedChara     5
#define  eNoSpaceCut      6
#define  eNoSpacePaste    7
#define kMaxTELength      32767
#define kTab              0x09
#define kDel              0x7F
#define kReturn           0x0D
#define kFileCreator      'kjbB'
#define topLeft(r)        (((Point *) &(r))[0])
#define botRight(r)       (((Point *) &(r))[1])

// ........................................................................................ typedefs

typedef struct
{
  TEHandle   textEditStrucHdl;
  ControlRef vScrollbarRef;
} docStructure, **docStructureHandle;

// ................................................................................ global variables

Boolean          gRunningOnX = false;
MenuID           gHelpMenu;
ControlActionUPP gScrollActionFunctionUPP;
TEClickLoopUPP   gCustomClickLoopUPP;
Boolean          gDone;
RgnHandle        gCursorRegion;
SInt16           gNumberOfWindows = 0;
SInt16           gOldControlValue;

// ............................................................................. function prototypes

void      main                  (void);
void      doPreliminaries       (void);
OSErr     quitAppEventHandler   (AppleEvent *,AppleEvent *,SInt32);
void      eventLoop             (void);
void      doIdle                (void);
void      doEvents              (EventRecord *);
void      doKeyEvent            (SInt8);
void      scrollActionFunction  (ControlRef,SInt16);
void      doInContent           (EventRecord *);
void      doUpdate              (EventRecord *);
void      doActivate            (EventRecord *);
void      doActivateDocWindow   (WindowRef,Boolean);
void      doOSEvent             (EventRecord *);
WindowRef doNewDocWindow        (void);
Boolean   customClickLoop       (void);
void      doSetScrollBarValue   (ControlRef,SInt16 *);
void      doAdjustMenus         (void);
void      doMenuChoice          (SInt32);
void      doFileMenu            (MenuItemIndex);
void      doEditMenu            (MenuItemIndex);
SInt16    doGetSelectLength     (TEHandle);
void      doAdjustScrollbar     (WindowRef);
void      doAdjustCursor        (WindowRef);
void      doCloseWindow         (WindowRef);
void      doSaveAsFile          (TEHandle);
void      doOpenCommand         (void);
void      navEventFunction      (NavEventCallbackMessage,NavCBRecPtr,NavCallBackUserData);
void      doOpenFile            (FSSpec);
void      doDrawDataPanel       (WindowRef);
void      doErrorAlert         (SInt16);

extern void doHelp             (void);

// ************************************************************************************ main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;

  // ........................................................................ do preliminaries

  doPreliminaries();

  // ................................................................ set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(eMenuBar);
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
    }

    menuRef = GetMenuRef(mAppleApplication);
    DeleteMenuItem(menuRef,iHelp);

    HMGetHelpMenu(&menuRef,NULL);
    InsertMenuItem(menuRef,"\pMonoTextEdit Help",0);
    gHelpMenu = GetMenuID(menuRef);

    gRunningOnX = true;
  }

  // ...................................................... create universal procedure pointers

  gScrollActionFunctionUPP = NewControlActionUPP((ControlActionProcPtr) scrollActionFunction);
  gCustomClickLoopUPP       = NewTEClickLoopUPP((TEClickLoopProcPtr) customClickLoop);

  // .................................................................. open an untitled window

  doNewDocWindow();

  // ......................................................................... enter eventLoop

  eventLoop();
}

// *************************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(192);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                             NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                             0L,false);
  if(osError != noErr)
    ExitToShell();
}

// *************************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                               &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ******************************************************************************** eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;
  Boolean     gotEvent;
  SInt32      sleepTime;

  gDone = false;
  gCursorRegion = NewRgn();
  doAdjustCursor(FrontWindow());
  sleepTime = GetCaretTime();

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,sleepTime,gCursorRegion);

    if(gotEvent)
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
        if(gNumberOfWindows > 0)
          doIdle();
    }
  }
}

// *********************************************************************************** doIdle

void  doIdle(void)
{
  docStructureHandle docStrucHdl;
  WindowRef          windowRef;

  windowRef = FrontWindow();

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  if(docStrucHdl != NULL)
    TEIdle((*docStrucHdl)->textEditStrucHdl);
}
```

```
// ******************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  SInt8          charCode;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          doAdjustMenus();
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          else
            doInContent(eventStrucPtr);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          doAdjustCursor(windowRef);
          break;

        case inGoAway:
          if(TrackGoAway(windowRef,eventStrucPtr->where))
            doCloseWindow(FrontWindow());
          break;
      }
      break;

    case keyDown:
      charCode = eventStrucPtr->message & charCodeMask;
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      else
        doKeyEvent(charCode);
      break;

    case autoKey:
      charCode = eventStrucPtr->message & charCodeMask;
      if((eventStrucPtr->modifiers & cmdKey) == 0)
        doKeyEvent(charCode);
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// *************************************************************************** doKeyEvent

void  doKeyEvent(SInt8 charCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             selectionLength;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(charCode == kTab)
  {
    // Do tab key handling here if required.
  }
  else if(charCode == kDel)
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(selectionLength == 0)
      (*textEditStrucHdl)->selEnd += 1;
    TEDelete(textEditStrucHdl);
    doAdjustScrollbar(windowRef);
  }
  else
  {
    selectionLength = doGetSelectLength(textEditStrucHdl);
    if(((*textEditStrucHdl)->teLength - selectionLength + 1) < kMaxTELength)
    {
      TEKey(charCode,textEditStrucHdl);
      doAdjustScrollbar(windowRef);
    }
    else
      doErrorAlert(eExceedChara);
  }

  doDrawDataPanel(windowRef);
}

// ******************************************************************* scrollActionFunction

void  scrollActionFunction(ControlRef controlRef,SInt16 partCode)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             linesToScroll;
  SInt16             controlValue, controlMax;

  windowRef = GetControlOwner(controlRef);
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  if(partCode)
  {
    if(partCode != kControlIndicatorPart)
    {
      switch(partCode)
      {
        case kControlUpButtonPart:
```

```
        case kControlDownButtonPart:
          linesToScroll = 1;
          break;

        case kControlPageUpPart:
        case kControlPageDownPart:
          linesToScroll = (((*textEditStrucHdl)->viewRect.bottom -
                           (*textEditStrucHdl)->viewRect.top) /
                           (*textEditStrucHdl)->lineHeight) - 1;
          break;
      }

      if((partCode == kControlDownButtonPart) || (partCode == kControlPageDownPart))
        linesToScroll = -linesToScroll;

      linesToScroll = controlValue - linesToScroll;
      if(linesToScroll < 0)
        linesToScroll = 0;
      else if(linesToScroll > controlMax)
        linesToScroll = controlMax;

      SetControlValue(controlRef,linesToScroll);

      linesToScroll = controlValue - linesToScroll;
    }
    else
    {
      linesToScroll = gOldControlValue - controlValue;
      gOldControlValue = controlValue;
    }

    if(linesToScroll != 0)
    {
      TEScroll(0,linesToScroll * (*textEditStrucHdl)->lineHeight,textEditStrucHdl);
      doDrawDataPanel(windowRef);
    }
  }
}

// ******************************************************************************** doInContent

void  doInContent(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Point              mouseXY;
  ControlRef         controlRef;
  SInt16             partCode;
  Boolean            shiftKeyPosition = false;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  mouseXY = eventStrucPtr->where;
  SetPortWindowPort(windowRef);
  GlobalToLocal(&mouseXY);

  if((partCode = FindControl(mouseXY,windowRef,&controlRef)) != 0)
  {
    gOldControlValue = GetControlValue(controlRef);
    TrackControl(controlRef,mouseXY,gScrollActionFunctionUPP);
  }
  else if(PtInRect(mouseXY,&(*textEditStrucHdl)->viewRect))
  {
    if((eventStrucPtr->modifiers & shiftKey) != 0)
      shiftKeyPosition = true;
    TEClick(mouseXY,shiftKeyPosition,textEditStrucHdl);
  }
}

// ********************************************************************************* doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  GrafPtr            oldPort;
  RgnHandle          visibleRegionHdl = NewRgn();
  Rect               portRect;

  windowRef = (WindowRef) eventStrucPtr->message;
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  BeginUpdate((WindowRef) eventStrucPtr->message);

  GetPortVisibleRegion(GetWindowPort(windowRef),visibleRegionHdl);
  EraseRgn(visibleRegionHdl);

  UpdateControls(windowRef,visibleRegionHdl);

  GetWindowPortBounds(windowRef,&portRect);
  TEUpdate(&portRect,textEditStrucHdl);

  doDrawDataPanel(windowRef);

  EndUpdate((WindowRef) eventStrucPtr->message);

  DisposeRgn(visibleRegionHdl);
  SetPort(oldPort);
}

// ******************************************************************************* doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
  doActivateDocWindow(windowRef,becomingActive);
}

// ********************************************************************** doActivateDocWindow

void  doActivateDocWindow(WindowRef windowRef,Boolean becomingActive)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(becomingActive)
  {
    SetPortWindowPort(windowRef);
    (*textEditStrucHdl)->viewRect.bottom = ((((*textEditStrucHdl)->viewRect.bottom -
                                             (*textEditStrucHdl)->viewRect.top) /
                                             (*textEditStrucHdl)->lineHeight) *
                                             (*textEditStrucHdl)->lineHeight) +
                                             (*textEditStrucHdl)->viewRect.top;
    (*textEditStrucHdl)->destRect.bottom = (*textEditStrucHdl)->viewRect.bottom;

    TEActivate(textEditStrucHdl);
    ActivateControl((*docStrucHdl)->vScrollbarRef);
    doAdjustScrollbar(windowRef);
    doAdjustCursor(windowRef);
  }
  else
  {
    TEDeactivate(textEditStrucHdl);
    DeactivateControl((*docStrucHdl)->vScrollbarRef);
  }
}

// ******************************************************************************** doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
        SetThemeCursor(kThemeArrowCursor);
      break;

    case mouseMovedMessage:
      doAdjustCursor(FrontWindow());
      break;
  }
}

// **************************************************************************** doNewDocWindow

WindowRef  doNewDocWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  Rect               portRect, destAndViewRect;

  if(!(windowRef = GetNewCWindow(rWindow,NULL,(WindowRef) -1)))
  {
    doErrorAlert(eWindow);
    return NULL;
  }

  SetPortWindowPort(windowRef);
  TextSize(10);

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    doErrorAlert(eDocStructure);
    return NULL;
  }
  SetWRefCon(windowRef,(SInt32) docStrucHdl);

  gNumberOfWindows ++;

  (*docStrucHdl)->vScrollbarRef = GetNewControl(rVScrollbar,windowRef);

  GetWindowPortBounds(windowRef,&portRect);
  destAndViewRect = portRect;
  destAndViewRect.right -= 15;
  destAndViewRect.bottom -= 15;
  InsetRect(&destAndViewRect,2,2);

  MoveHHi((Handle) docStrucHdl);
  HLock((Handle) docStrucHdl);

  if(!((*docStrucHdl)->textEditStrucHdl = TENew(&destAndViewRect,&destAndViewRect)))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposeHandle((Handle) docStrucHdl);
    doErrorAlert(eEditRecord);
    return NULL;
  }

  HUnlock((Handle) docStrucHdl);

  TESetClickLoop(gCustomClickLoopUPP,(*docStrucHdl)->textEditStrucHdl);
  TEAutoView(true,(*docStrucHdl)->textEditStrucHdl);
  TEFeatureFlag(teFOutlineHilite,1,(*docStrucHdl)->textEditStrucHdl);

  return windowRef;
}

// ************************************************************************ customClickLoop

Boolean  customClickLoop(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  GrafPtr            oldPort;
  RgnHandle          oldClip;
  Rect               tempRect, portRect;
  Point              mouseXY;
  SInt16             linesToScroll = 0;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);
  oldClip = NewRgn();
  GetClip(oldClip);
  SetRect(&tempRect,-32767,-32767,32767,32767);
  ClipRect(&tempRect);

  GetMouse(&mouseXY);
  GetWindowPortBounds(windowRef,&portRect);

  if(mouseXY.v < portRect.top)
  {
    linesToScroll = 1;
    doSetScrollBarValue((*docStrucHdl)->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }
  else if(mouseXY.v > portRect.bottom)
  {
    linesToScroll = -1;
    doSetScrollBarValue((*docStrucHdl)->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }

  if(linesToScroll != 0)
    doDrawDataPanel(windowRef);

  SetClip(oldClip);
  DisposeRgn(oldClip);
  SetPort(oldPort);

  return true;
}
```

```
// ********************************************************************** doSetScrollBarValue

void  doSetScrollBarValue(ControlRef controlRef,SInt16 *linesToScroll)
{
  SInt16 controlValue, controlMax;

  controlValue = GetControlValue(controlRef);
  controlMax = GetControlMaximum(controlRef);

  *linesToScroll = controlValue - *linesToScroll;
  if(*linesToScroll < 0)
    *linesToScroll = 0;
  else if(*linesToScroll > controlMax)
    *linesToScroll = controlMax;

  SetControlValue(controlRef,*linesToScroll);
  *linesToScroll = controlValue - *linesToScroll;
}

// **************************************************************************** doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef            fileMenuHdl, editMenuHdl;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   scrapFlavorFlags;

  fileMenuHdl = GetMenuRef(mFile);
  editMenuHdl = GetMenuRef(mEdit);

  if(gNumberOfWindows > 0)
  {
    windowRef = FrontWindow();
    docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
    textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

    EnableMenuItem(fileMenuHdl,iClose);

    if((*textEditStrucHdl)->selStart < (*textEditStrucHdl)->selEnd)
    {
      EnableMenuItem(editMenuHdl,iCut);
      EnableMenuItem(editMenuHdl,iCopy);
      EnableMenuItem(editMenuHdl,iClear);
    }
    else
    {
      DisableMenuItem(editMenuHdl,iCut);
      DisableMenuItem(editMenuHdl,iCopy);
      DisableMenuItem(editMenuHdl,iClear);
    }

    GetCurrentScrap(&scrapRef);

    osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&scrapFlavorFlags);
    if(osError == noErr)
      EnableMenuItem(editMenuHdl,iPaste);
    else
      DisableMenuItem(editMenuHdl,iPaste);

    if((*textEditStrucHdl)->teLength > 0)
    {
      EnableMenuItem(fileMenuHdl,iSaveAs);
      EnableMenuItem(editMenuHdl,iSelectAll);
    }
    else
    {
      DisableMenuItem(fileMenuHdl,iSaveAs);
      DisableMenuItem(editMenuHdl,iSelectAll);
    }
  }
  else
  {
    DisableMenuItem(fileMenuHdl,iClose);
    DisableMenuItem(fileMenuHdl,iSaveAs);
    DisableMenuItem(editMenuHdl,iClear);
    DisableMenuItem(editMenuHdl,iSelectAll);
  }

  DrawMenuBar();
}

// ******************************************************************************* doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID   = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  if(gRunningOnX)
    if(menuID == gHelpMenu)
      if(menuItem == 1)
        doHelp();

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        SysBeep(10);
      else if(menuItem == iHelp)
        doHelp();
      break;

    case mFile:
      doFileMenu(menuItem);
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;
  }

  HiliteMenu(0);
}

// ********************************************************************************* doFileMenu

void  doFileMenu(MenuItemIndex menuItem)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;

  switch(menuItem)
  {
    case iNew:
      if(windowRef = doNewDocWindow())
        ShowWindow(windowRef);
      break;
    case iOpen:
      doOpenCommand();
      break;

    case iClose:
      doCloseWindow(FrontWindow());
      break;

    case iSaveAs:
      docStrucHdl = (docStructureHandle) (GetWRefCon(FrontWindow()));
      textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
      doSaveAsFile(textEditStrucHdl);
      break;

    case iQuit:
      gDone = true;
      break;
  }
}

// ******************************************************************************* doEditMenu

void  doEditMenu(MenuItemIndex menuItem)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt32             totalSize, contigSize, newSize;
  SInt16             selectionLength;
  ScrapRef           scrapRef;
  Size               sizeOfTextData;

  windowRef = FrontWindow();
  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  switch(menuItem)
  {
    case iUndo:
      break;

    case iCut:
      if(ClearCurrentScrap() == noErr)
      {
        PurgeSpace(&totalSize,&contigSize);
        selectionLength = doGetSelectLength(textEditStrucHdl);
        if(selectionLength > contigSize)
          doErrorAlert(eNoSpaceCut);
        else
        {
          TECut(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
          if(TEToScrap() != noErr)
            ClearCurrentScrap();
        }
      }
      break;

    case iCopy:
      if(ClearCurrentScrap() == noErr)
        TECopy(textEditStrucHdl);
      if(TEToScrap() != noErr)
        ClearCurrentScrap();
      break;

    case iPaste:
      GetCurrentScrap(&scrapRef);;
      GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
      newSize = (*textEditStrucHdl)->teLength + sizeOfTextData;
      if(newSize > kMaxTELength)
        doErrorAlert(eNoSpacePaste);
      else
      {
        if(TEFromScrap() == noErr)
        {
          TEPaste(textEditStrucHdl);
          doAdjustScrollbar(windowRef);
        }
      }
      break;

    case iClear:
      TEDelete(textEditStrucHdl);
      doAdjustScrollbar(windowRef);
      break;

    case iSelectAll:
      TESetSelect(0,(*textEditStrucHdl)->teLength,textEditStrucHdl);
      break;
  }

  doDrawDataPanel(windowRef);
}

// ********************************************************************** doGetSelectLength

SInt16  doGetSelectLength(TEHandle textEditStrucHdl)
{
  SInt16 selectionLength;

  selectionLength = (*textEditStrucHdl)->selEnd - (*textEditStrucHdl)->selStart;
  return selectionLength;
}

// ********************************************************************** doAdjustScrollbar

void  doAdjustScrollbar(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             numberOfLines, controlMax, controlValue;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  numberOfLines = (*textEditStrucHdl)->nLines;
  if(*(*(*textEditStrucHdl)->hText + (*textEditStrucHdl)->teLength - 1) == kReturn)
    numberOfLines += 1;

  controlMax = numberOfLines - (((*textEditStrucHdl)->viewRect.bottom -
             (*textEditStrucHdl)->viewRect.top) /
             (*textEditStrucHdl)->lineHeight);
  if(controlMax < 0)
    controlMax = 0;
  SetControlMaximum((*docStrucHdl)->vScrollbarRef,controlMax);

  controlValue = ((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) /
                (*textEditStrucHdl)->lineHeight;
  if(controlValue < 0)
    controlValue = 0;
  else if(controlValue > controlMax)
    controlValue = controlMax;

  SetControlValue((*docStrucHdl)->vScrollbarRef,controlValue);

  SetControlViewSize((*docStrucHdl)->vScrollbarRef,(*textEditStrucHdl)->viewRect.bottom -
                    (*textEditStrucHdl)->viewRect.top);
  TEScroll(0,((*textEditStrucHdl)->viewRect.top - (*textEditStrucHdl)->destRect.top) -
               (GetControlValue((*docStrucHdl)->vScrollbarRef) *
              (*textEditStrucHdl)->lineHeight),textEditStrucHdl);
}

// *************************************************************************** doAdjustCursor

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr   oldPort;
  RgnHandle arrowRegion, iBeamRegion;
  Rect      portRect, cursorRect;
  Point     mouseXY;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  arrowRegion = NewRgn();
  iBeamRegion = NewRgn();
  SetRectRgn(arrowRegion,-32768,-32768,32766,32766);

  GetWindowPortBounds(windowRef,&portRect);
  cursorRect = portRect;
  cursorRect.bottom -= 15;
  cursorRect.right  -= 15;
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  RectRgn(iBeamRegion,&cursorRect);
  DiffRgn(arrowRegion,iBeamRegion,arrowRegion);

  GetGlobalMouse(&mouseXY);

  if(PtInRgn(mouseXY,iBeamRegion))
  {
    SetThemeCursor(kThemeIBeamCursor);
    CopyRgn(iBeamRegion,gCursorRegion);
  }
  else
  {
    SetThemeCursor(kThemeArrowCursor);
    CopyRgn(arrowRegion,gCursorRegion);
  }

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);

  SetPort(oldPort);
}

// **************************************************************************** doCloseWindow

void  doCloseWindow(WindowRef windowRef)
{
  docStructureHandle  docStrucHdl;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;

  DisposeControl((*docStrucHdl)->vScrollbarRef);
  TEDispose((*docStrucHdl)->textEditStrucHdl);
  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);

  gNumberOfWindows --;
}

// ***************************************************************************** doSaveAsFile
```

```
void  doSaveAsFile(TEHandle textEditStrucHdl)
{
  OSErr            osError = noErr;
  NavDialogOptions dialogOptions;
  WindowRef        windowRef;
  OSType           fileType;
  NavEventUPP      navEventFunctionUPP;
  NavReplyRecord   navReplyStruc;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  SInt16           fileRefNum;
  Size             actualSize;
  SInt32           dataLength;
  Handle           editTextHdl;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    windowRef = FrontWindow();

    fileType = 'TEXT';

    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavPutFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,fileType,
                         kFileCreator,NULL);
    DisposeNavEventUPP(navEventFunctionUPP);

    if(navReplyStruc.validRecord && osError == noErr)
    {
      if((osError = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize)) == noErr)

      {
        if(!navReplyStruc.replacing)
        {
          osError = FSpCreate(&fileSpec,kFileCreator,fileType,navReplyStruc.keyScript);
          if(osError != noErr)
          {
            NavDisposeReply(&navReplyStruc);
          }
        }

        if(osError == noErr)
          osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum);

        if(osError == noErr)
        {
          SetWTitle(windowRef,fileSpec.name);
          dataLength = (*textEditStrucHdl)->teLength;
          editTextHdl = (*textEditStrucHdl)->hText;
          FSWrite(fileRefNum,&dataLength,*editTextHdl);
        }

        NavCompleteSave(&navReplyStruc,kNavTranslateInPlace);
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// ****************************************************************************** doOpenCommand

void  doOpenCommand(void)
{
  OSErr            osError  = noErr;
  NavDialogOptions dialogOptions;
  NavEventUPP      navEventFunctionUPP;
  NavReplyRecord   navReplyStruc;
  SInt32           index, count;
  AEKeyword        theKeyword;
  DescType         actualType;
  FSSpec           fileSpec;
  Size             actualSize;
  FInfo            fileInfo;

  osError = NavGetDefaultDialogOptions(&dialogOptions);

  if(osError == noErr)
  {
    navEventFunctionUPP = NewNavEventUPP((NavEventProcPtr) navEventFunction);
    osError = NavGetFile(NULL,&navReplyStruc,&dialogOptions,navEventFunctionUPP,NULL,NULL,
                         NULL,0);
    DisposeNavEventUPP(navEventFunctionUPP);

    if(osError == noErr && navReplyStruc.validRecord)
    {
      osError = AECountItems(&(navReplyStruc.selection),&count);
      if(osError == noErr)
      {
        for(index=1;index<=count;index++)
        {
          osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,&theKeyword,
                                &actualType,&fileSpec,sizeof(fileSpec),&actualSize);
          {
            if((osError = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
              doOpenFile(fileSpec);
          }
        }
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

// *************************************************************************** navEventFunction

void  navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                       NavCallBackUserData callBackUD)
{
  WindowRef windowRef;

  if(callBackParms != NULL)
  {
    switch(callBackSelector)
    {
      case kNavCBEvent:
        switch(callBackParms->eventData.eventDataParms.event->what)
        {
          case updateEvt:
            windowRef = (WindowRef) callBackParms->eventData.eventDataParms.event->message;
            if(GetWindowKind(windowRef) != kDialogWindowKind)
              doUpdate((EventRecord*) callBackParms->eventData.eventDataParms.event);
            break;
        }
        break;
    }
  }
}

// ********************************************************************************* doOpenFile

void  doOpenFile(FSSpec fileSpec)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             fileRefNum;
  SInt32             textLength;
  Handle             textBuffer;

  if((windowRef = doNewDocWindow()) == NULL)
    return;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  SetWTitle(windowRef,fileSpec.name);

  FSpOpenDF(&fileSpec,fsCurPerm,&fileRefNum);

  SetFPos(fileRefNum,fsFromStart,0);
  GetEOF(fileRefNum,&textLength);

  if(textLength > 32767)
    textLength = 32767;

  textBuffer = NewHandle((Size) textLength);

  FSRead(fileRefNum,&textLength,*textBuffer);

  MoveHHi(textBuffer);
  HLock(textBuffer);

  TESetText(*textBuffer,textLength,textEditStrucHdl);

  HUnlock(textBuffer);
  DisposeHandle(textBuffer);

  FSClose(fileRefNum);

  (*textEditStrucHdl)->selStart = 0;
  (*textEditStrucHdl)->selEnd = 0;
}

// *************************************************************************** doDrawDataPanel

void  doDrawDataPanel(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  RGBColor           whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  RGBColor           blackColour = { 0x0000, 0x0000, 0x0000 };
  RGBColor           blueColour = { 0x1818, 0x4B4B, 0x8181 };
  ControlRef         controlRef;
  Rect               panelRect;
  Str255             textString;

  docStrucHdl = (docStructureHandle) (GetWRefCon(windowRef));;
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  controlRef = (*docStrucHdl)->vScrollbarRef;

  MoveTo(0,282);
  LineTo(495,282);

  RGBForeColor(&whiteColour);
  RGBBackColor(&blueColour);
  SetRect(&panelRect,0,283,495,300);
  EraseRect(&panelRect);

  MoveTo(3,295);
  DrawString("\pteLength               nLines          lineHeight");

  MoveTo(225,295);
  DrawString("\pdestRect.top              controlValue           contrlMax");

  SetRect(&panelRect,47,284,88,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,124,284,149,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,204,284,222,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,286,284,323,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,389,284,416,299);
  EraseRect(&panelRect);
  SetRect(&panelRect,472,284,495,299);
  EraseRect(&panelRect);

  NumToString((SInt32) (*textEditStrucHdl)->teLength,textString);
  MoveTo(47,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->nLines,textString);
  MoveTo(124,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->lineHeight,textString);
  MoveTo(204,295);
  DrawString(textString);

  NumToString((SInt32) (*textEditStrucHdl)->destRect.top,textString);
  MoveTo(286,295);
  DrawString(textString);

  NumToString((SInt32) GetControlValue(controlRef),textString);
  MoveTo(389,295);
  DrawString(textString);

  NumToString((SInt32) GetControlMaximum(controlRef),textString);
  MoveTo(472,295);
  DrawString(textString);

  RGBForeColor(&blackColour);
  RGBBackColor(&whiteColour);
}

// ******************************************************************************** doErrorAlert

void  doErrorAlert(SInt16 errorCode)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorCode);

  if(errorCode < eWindow)
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
  else
  {
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  }
}

// *********************************************************************************************
// HelpDialog.c
// *********************************************************************************************

// ............................................................................................. includes
```

```
#include <Carbon.h>

// ............................................................................ defines

#define rHelpModal            128
#define  iUserPane            2
#define  iScrollBar           3
#define  iPopupMenu           4
#define  eHelpDialog          9
#define  eHelpDocRecord       10
#define  eHelpText            11
#define  eHelpPicture         12
#define rTextIntroduction     128
#define rTextCreatingText     129
#define rTextModifyHelp       130
#define rPictIntroductionBase 128
#define rPictCreatingTextBase 129
#define kTextInset            4

// ............................................................................ typedefs

typedef struct
{
  Rect      bounds;
  PicHandle pictureHdl;
} pictInfoStructure;

typedef struct
{
  TEHandle          textEditStrucHdl;
  ControlRef        scrollbarHdl;
  SInt16            pictCount;
  pictInfoStructure *pictInfoStructurePtr;
}  docStructure, ** docStructureHandle;

typedef struct
{
  RGBColor     backColour;
  PixPatHandle backPixelPattern;
  Pattern      backBitPattern;
} backColourPattern;

// ..................................................................... global variables

ModalFilterUPP         eventFilterUPP;
ControlUserPaneDrawUPP userPaneDrawFunctionUPP;
ControlActionUPP       actionFunctionUPP;
backColourPattern      gBackColourPattern;
SInt16                 gTextResourceID;
SInt16                 gPictResourceBaseID;
RgnHandle              gSavedClipRgn  = NULL;

// ................................................................. function prototypes

void           doHelp               (void);
void           doCloseHelp          (DialogPtr,GrafPtr);
void           doDisposeDescriptors (void);
pascal void    userPaneDrawFunction (ControlRef,SInt16);
Boolean        doGetText            (DialogPtr,SInt16,Rect);
Boolean        doGetPictureInfo     (DialogPtr,SInt16);
pascal void    actionFunction       (ControlRef,SInt16);
void           doScrollTextAndPicts (DialogPtr);
void           doDrawPictures       (DialogPtr,Rect *);
pascal Boolean eventFilter          (DialogPtr,EventRecord *,SInt16 *);
void           doSetBackgroundWhite (void);

extern void    doUpdate             (EventRecord *);
extern void    doErrorAlert         (SInt16);
```

```
// ******************************************************************************** doHelp

void  doHelp(void)
{
  DialogPtr          dialogPtr;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  ControlRef         controlRef;
  SInt16             itemHit, menuItem;
  Rect               userPaneRect, destRect, viewRect;

  // ............................................................ create universal procedure pointers

  eventFilterUPP          = NewModalFilterUPP(eventFilter);
  userPaneDrawFunctionUPP = NewControlUserPaneDrawUPP(userPaneDrawFunction);
  actionFunctionUPP       = NewControlActionUPP(actionFunction);

  // ........................................................ create dialog and attach document structure

  if(!(dialogPtr = GetNewDialog(rHelpModal,NULL,(WindowRef) -1)))
  {
    doErrorAlert(eHelpDialog);
    doDisposeDescriptors();
    return;
  }

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    doErrorAlert(eHelpDocRecord);
    DisposeDialog(dialogPtr);
    doDisposeDescriptors();
    return;
  }

  windowRef = GetDialogWindow(dialogPtr);
  SetWRefCon(windowRef,(SInt32) docStrucHdl);

  // ............................................................ set graphics port and default button

  GetPort(&oldPort);
  SetPortDialogPort(dialogPtr);
  SetDialogDefaultItem(dialogPtr,kStdOkItemIndex);

  // ................................................................ set user pane drawing function

  GetDialogItemAsControl(dialogPtr,iUserPane,&controlRef);
  SetControlData(controlRef,kControlEntireControl,kControlUserPaneDrawProcTag,
                 sizeof(userPaneDrawFunctionUPP),(Ptr) &userPaneDrawFunctionUPP);

  // ........................................ set destination and view rectangles, create edit structure

  GetControlBounds(controlRef,&userPaneRect);
  InsetRect(&userPaneRect,kTextInset,kTextInset / 2);
  destRect = viewRect = userPaneRect;
  (*docStrucHdl)->textEditStrucHdl = TEStyleNew(&destRect,&viewRect);

  // .............................. assign handle to scroll bar to relevant document structure field

  GetDialogItemAsControl(dialogPtr,iScrollBar,&controlRef);
  (*docStrucHdl)->scrollbarHdl = controlRef;

  // .......................... initialise picture information structure field of document structure

  (*docStrucHdl)->pictInfoStructurePtr = NULL;

  // ...................................... assign resource IDs of first topic's 'TEXT'/'styl' resources

  gTextResourceID      = rTextIntroduction;
```

```
gPictResourceBaseID  = rPictIntroductionBase;

// ....................................................... load text resources and insert into edit structure

if(!(doGetText(dialogPtr,gTextResourceID,viewRect)))
{
  doCloseHelp(dialogPtr,oldPort);
  doDisposeDescriptors();
  return;
}

// ............ search for option-space charas in text and load same number of 'PICT' resources

if(!(doGetPictureInfo(dialogPtr,gPictResourceBaseID)))
{
  doCloseHelp(dialogPtr,oldPort);
  doDisposeDescriptors();
  return;
}

// ............................................. create an empty region for saving the old clipping region

gSavedClipRgn = NewRgn();

// ....................................................... show window and save background colour and pattern

ShowWindow(GetDialogWindow(dialogPtr));

// ........................................................................................... enter ModalDialog loop

do
{
  ModalDialog(eventFilterUPP,&itemHit);

  if(itemHit == iPopupMenu)
  {
    SetControlValue((*docStrucHdl)->scrollbarHdl,0);

    GetDialogItemAsControl(dialogPtr,iPopupMenu,&controlRef);
    menuItem = GetControlValue(controlRef);

    switch(menuItem)
    {
      case 1:
        gTextResourceID     = rTextIntroduction;
        gPictResourceBaseID = rPictIntroductionBase;
        break;

      case 2:
        gTextResourceID     = rTextCreatingText;
        gPictResourceBaseID = rPictCreatingTextBase;
        break;

      case 3:
        gTextResourceID     = rTextModifyHelp;
        break;
    }

    if(!(doGetText(dialogPtr,gTextResourceID,viewRect)))
    {
      doCloseHelp(dialogPtr,oldPort);
      doDisposeDescriptors();
      return;
    }
    if(!(doGetPictureInfo(dialogPtr,gPictResourceBaseID)))
    {
      doCloseHelp(dialogPtr,oldPort);
      doDisposeDescriptors();
      return;
      }

      doDrawPictures(dialogPtr,&viewRect);
    }

  } while(itemHit != kStdOkItemIndex);

  // ......................................................................................... clean up

  doCloseHelp(dialogPtr,oldPort);
  doDisposeDescriptors();
}

// ***************************************************************************** doCloseHelp

void  doCloseHelp(DialogPtr dialogPtr,GrafPtr oldPort)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             a;

  docStrucHdl = (docStructureHandle) GetWRefCon(GetDialogWindow(dialogPtr));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  if(gSavedClipRgn)
    DisposeRgn(gSavedClipRgn);

  if((*docStrucHdl)->textEditStrucHdl)
    TEDispose((*docStrucHdl)->textEditStrucHdl);

  if((*docStrucHdl)->pictInfoStructurePtr)
  {
    for(a=0;a<(*docStrucHdl)->pictCount;a++)
      ReleaseResource((Handle) (*docStrucHdl)->pictInfoStructurePtr[a].pictureHdl);
    DisposePtr((Ptr) (*docStrucHdl)->pictInfoStructurePtr);
  }

  DisposeHandle((Handle) docStrucHdl);
  DisposeDialog(dialogPtr);

  SetPort(oldPort);
}

// ******************************************************************** doDisposeDescriptors

void  doDisposeDescriptors(void)
{
  DisposeModalFilterUPP(eventFilterUPP);
  DisposeControlUserPaneDrawUPP(userPaneDrawFunctionUPP);
  DisposeControlActionUPP(actionFunctionUPP);
}

// ******************************************************************** userPaneDrawFunction

pascal void  userPaneDrawFunction(ControlRef controlRef,SInt16 thePart)
{
  Rect               itemRect, viewRect;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  DialogPtr          dialogPtr;
  Boolean            inState;

  windowRef = GetControlOwner(controlRef);
  dialogPtr = GetDialogFromWindow(windowRef);

  GetControlBounds(controlRef,&itemRect);
  InsetRect(&itemRect,1,1);
  itemRect.right += 15;
  if(IsWindowVisible(windowRef))
    inState = IsWindowHilited(windowRef);
  DrawThemeListBoxFrame(&itemRect,inState);

  doSetBackgroundWhite();
  EraseRect(&itemRect);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  viewRect = (*textEditStrucHdl)->viewRect;

  TEUpdate(&viewRect,textEditStrucHdl);
  doDrawPictures(dialogPtr,&viewRect);
}

// ********************************************************************************** doGetText

Boolean  doGetText(DialogPtr dialogPtr,SInt16 textResourceID,Rect viewRect)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Handle             helpTextHdl;
  StScrpHandle       stylScrpStrucHdl;
  SInt16             numberOfLines, heightOfText, heightToScroll;

  doSetBackgroundWhite();

  windowRef = GetDialogWindow(dialogPtr);
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  TESetSelect(0,32767,textEditStrucHdl);
  TEDelete(textEditStrucHdl);

  (*textEditStrucHdl)->destRect = (*textEditStrucHdl)->viewRect;
  SetControlValue((*docStrucHdl)->scrollbarHdl,0);

  helpTextHdl = GetResource('TEXT',textResourceID);
  if(helpTextHdl == NULL)
  {
    doErrorAlert(eHelpText);
    return false;
  }

  stylScrpStrucHdl = (StScrpHandle) GetResource('styl',textResourceID);
  if(stylScrpStrucHdl == NULL)
  {
    doErrorAlert(eHelpText);
    return false;
  }

  TEStyleInsert(*helpTextHdl,GetHandleSize(helpTextHdl),stylScrpStrucHdl,textEditStrucHdl);

  ReleaseResource(helpTextHdl);
  ReleaseResource((Handle) stylScrpStrucHdl);

  numberOfLines = (*textEditStrucHdl)->nLines;
  heightOfText = TEGetHeight((SInt32) numberOfLines,1,textEditStrucHdl);

  if(heightOfText > (viewRect.bottom - viewRect.top))
  {
    heightToScroll = TEGetHeight((SInt32) numberOfLines,1,textEditStrucHdl) -
                           (viewRect.bottom - viewRect.top);
    SetControlMaximum((*docStrucHdl)->scrollbarHdl,heightToScroll);
    ActivateControl((*docStrucHdl)->scrollbarHdl);
    SetControlViewSize((*docStrucHdl)->scrollbarHdl,(*textEditStrucHdl)->viewRect.bottom -
                       (*textEditStrucHdl)->viewRect.top);
  }
  else
  {
    DeactivateControl((*docStrucHdl)->scrollbarHdl);
  }

  return true;
}

// *********************************************************************** doGetPictureInfo

Boolean  doGetPictureInfo(DialogPtr dialogPtr,SInt16 firstPictID)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  Handle             textHdl;
  SInt32             offset, textSize;
  SInt16             numberOfPicts, a, lineHeight, fontAscent;
  SInt8              optionSpace[1] = "\xCA";
  pictInfoStructure  *pictInfoPtr;
  Point              picturePoint;
  TextStyle          whatStyle;

  windowRef = GetDialogWindow(dialogPtr);
  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->pictInfoStructurePtr != NULL)
  {
    for(a=0;a<(*docStrucHdl)->pictCount;a++)
      ReleaseResource((Handle) (*docStrucHdl)->pictInfoStructurePtr[a].pictureHdl);

    DisposePtr((Ptr) (*docStrucHdl)->pictInfoStructurePtr);
    (*docStrucHdl)->pictInfoStructurePtr = NULL;
  }

  (*docStrucHdl)->pictCount = 0;

  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
  textHdl = (*textEditStrucHdl)->hText;

  textSize = GetHandleSize(textHdl);
  offset = 0;
  numberOfPicts = 0;

  HLock(textHdl);

  offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
  while((offset >= 0) && (offset <= textSize))
  {
    numberOfPicts++;
    offset++;
    offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
  }

  if(numberOfPicts == 0)
  {
    HUnlock(textHdl);
    return true;
  }

  pictInfoPtr = (pictInfoStructure *) NewPtr(sizeof(pictInfoStructure) * numberOfPicts);
  (*docStrucHdl)->pictInfoStructurePtr = pictInfoPtr;

  offset = 0L;

  for(a=0;a<numberOfPicts;a++)
  {
    pictInfoPtr[a].pictureHdl = GetPicture(firstPictID + a);
    if(pictInfoPtr[a].pictureHdl == NULL)
    {
      doErrorAlert(eHelpPicture);
      return false;
    }

    offset = Munger(textHdl,offset,optionSpace,1,NULL,0);
    picturePoint = TEGetPoint((SInt16)offset,textEditStrucHdl);

    TEGetStyle(offset,&whatStyle,&lineHeight,&fontAscent,textEditStrucHdl);
    picturePoint.v -= lineHeight;
    offset++;
    pictInfoPtr[a].bounds = (**pictInfoPtr[a].pictureHdl).picFrame;

    OffsetRect(&pictInfoPtr[a].bounds,
               (((*textEditStrucHdl)->destRect.right + (*textEditStrucHdl)->destRect.left) -
               (pictInfoPtr[a].bounds.right + pictInfoPtr[a].bounds.left) ) / 2,
               - pictInfoPtr[a].bounds.top + picturePoint.v);
  }

  (*docStrucHdl)->pictCount = a;

  HUnlock(textHdl);

  return true;
}

// *************************************************************************** actionFunction

pascal void  actionFunction(ControlRef scrollbarHdl,SInt16 partCode)
{
  WindowRef          windowRef;
  DialogPtr          dialogPtr;
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             delta, oldValue, offset, lineHeight, fontAscent;
  Point              thePoint;
  Rect               viewRect, portRect;
  TextStyle          style;

  if(partCode)
  {
    windowRef = GetControlOwner(scrollbarHdl);
    dialogPtr = GetDialogFromWindow(windowRef);
    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
    textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;
    viewRect = (*textEditStrucHdl)->viewRect;
    thePoint.h = viewRect.left + kTextInset;

    if(partCode != kControlIndicatorPart)
    {
      switch(partCode)
      {
        case kControlUpButtonPart:
          thePoint.v = viewRect.top - 4;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          delta = thePoint.v - lineHeight - viewRect.top;
          break;

        case kControlDownButtonPart:
          thePoint.v = viewRect.bottom + 2;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          delta = thePoint.v - viewRect.bottom;
          break;

        case kControlPageUpPart:
          thePoint.v = viewRect.top + 2;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          thePoint.v += lineHeight - fontAscent;
          thePoint.v -= viewRect.bottom - viewRect.top;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          delta = thePoint.v - viewRect.top;
          if(offset == 0)
            delta -= lineHeight;
          break;

        case kControlPageDownPart:
          thePoint.v = viewRect.bottom - 2;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          thePoint.v -= fontAscent;
          thePoint.v += viewRect.bottom - viewRect.top;
          offset = TEGetOffset(thePoint,textEditStrucHdl);
          thePoint = TEGetPoint(offset,textEditStrucHdl);
          TEGetStyle(offset,&style,&lineHeight,&fontAscent,textEditStrucHdl);
          delta =  thePoint.v - lineHeight - viewRect.bottom;
          if(offset == (**textEditStrucHdl).teLength)
            delta += lineHeight;
          break;
      }

      oldValue = GetControlValue(scrollbarHdl);

      if(((delta < 0) && (oldValue > 0)) || ((delta > 0) &&
         (oldValue < GetControlMaximum(scrollbarHdl))))
      {
        GetClip(gSavedClipRgn);
        GetWindowPortBounds(windowRef,&portRect);
        ClipRect(&portRect);

        SetControlValue(scrollbarHdl,oldValue + delta);
        SetClip(gSavedClipRgn);
      }
    }

    doScrollTextAndPicts(dialogPtr);
  }
}

// ********************************************************************** doScrollTextAndPicts

void  doScrollTextAndPicts(DialogPtr dialogPtr)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             scrollDistance, oldScroll;
  Rect               updateRect;

  doSetBackgroundWhite();

  docStrucHdl = (docStructureHandle) GetWRefCon(GetDialogWindow(dialogPtr));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  oldScroll = (*textEditStrucHdl)->viewRect.top -(*textEditStrucHdl)->destRect.top;
  scrollDistance = oldScroll - GetControlValue((*docStrucHdl)->scrollbarHdl);
  if(scrollDistance == 0)
    return;

  TEScroll(0,scrollDistance,textEditStrucHdl);
  if((*docStrucHdl)->pictCount == 0)
    return;

  updateRect = (*textEditStrucHdl)->viewRect;

  if(scrollDistance > 0)
  {
    if(scrollDistance < (updateRect.bottom - updateRect.top))
      updateRect.bottom = updateRect.top + scrollDistance;
  }
  else
  {
    if( - scrollDistance < (updateRect.bottom - updateRect.top))
      updateRect.top = updateRect.bottom + scrollDistance;
  }

  doDrawPictures(dialogPtr,&updateRect);
}

// ************************************************************************** doDrawPictures

void  doDrawPictures(DialogPtr dialogPtr,Rect *updateRect)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditStrucHdl;
  SInt16             pictCount, pictIndex, vOffset;
  PicHandle          thePictHdl;
  Rect               pictLocRect, dummyRect;

  docStrucHdl = (docStructureHandle) GetWRefCon(GetDialogWindow(dialogPtr));
  textEditStrucHdl = (*docStrucHdl)->textEditStrucHdl;

  vOffset = (*textEditStrucHdl)->destRect.top - (*textEditStrucHdl)->viewRect.top -
            kTextInset;
  pictCount = (*docStrucHdl)->pictCount;

  for(pictIndex = 0;pictIndex < pictCount;pictIndex++)
  {
    pictLocRect = (*docStrucHdl)->pictInfoStructurePtr[pictIndex].bounds;
    OffsetRect(&pictLocRect,0,vOffset);

    if(!SectRect(&pictLocRect,updateRect,&dummyRect))
      continue;

    thePictHdl = (*docStrucHdl)->pictInfoStructurePtr[pictIndex].pictureHdl;

    LoadResource((Handle) thePictHdl);
    HLock((Handle) thePictHdl);

    GetClip(gSavedClipRgn);
    ClipRect(updateRect);
    DrawPicture(thePictHdl,&pictLocRect);

    SetClip(gSavedClipRgn);
    HUnlock((Handle) thePictHdl);
  }
}

// ***************************************************************************** eventFilter

pascal Boolean  eventFilter(DialogPtr dialogPtr,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean            handledEvent;
  GrafPtr            oldPort;
  Point              mouseXY;
  ControlRef         controlRef;
  docStructureHandle docStrucHdl;

  handledEvent = false;
  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogPtr)))
  {
    doUpdate(eventStrucPtr);
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogPtr);

    if(eventStrucPtr->what == mouseDown)
    {
      mouseXY = eventStrucPtr->where;
      GlobalToLocal(&mouseXY);
      if(FindControl(mouseXY,GetDialogWindow(dialogPtr),&controlRef))
      {
        docStrucHdl = (docStructureHandle) GetWRefCon(GetDialogWindow(dialogPtr));
        if(controlRef == (*docStrucHdl)->scrollbarHdl)
        {
          TrackControl((*docStrucHdl)->scrollbarHdl,mouseXY,actionFunctionUPP);
          *itemHit = iScrollBar;
          handledEvent = true;
        }
      }
    }
    else
    {
      handledEvent = StdFilterProc(dialogPtr,eventStrucPtr,itemHit);
    }

    SetPort(oldPort);
  }

  return handledEvent;
}

// ********************************************************************** doSetBackgroundWhite

void  doSetBackgroundWhite(void)
{
  RGBColor whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  Pattern  whitePattern;

  RGBBackColor(&whiteColour);
  BackPat(GetQDGlobalsWhite(&whitePattern));
}

// *******************************************************************************************
```

## Demonstration Program MonoTextEdit Comments

When this program is run, the user should explore both the text editor and the Help dialog.

### Text Editor

In the text editor, the user should perform all the actions usually associated with a simple text editor, that is:

- Open a new document window, open an existing 'TEXT' file for display in a new document window, and save a document to a 'TEXT' file. (A 'TEXT' file titled "MonoTextEdit Document" is included.)

- Enter new text and use the Edit menu Cut, Copy, Paste, and Clear commands to edit the text. (Pasting between documents and other applications is supported.)

- Select text by clicking and dragging, double-clicking a word, shift-clicking, and choosing the Select All command from the Edit menu. Also select large amounts of text by clicking in the text and dragging the cursor above or below the window so as to invoke auto-scrolling.

- Scroll a large document by dragging the scroll box/scroller (live scrolling is used), clicking once in a scroll arrow or gray area/track, and holding the mouse down in a scroll arrow or gray area/track.

Whenever any action is taken, the user should observe the changes to the values displayed in the data panel at the bottom of each window. In particular, the relationship between the destination rectangle and scroll bar control value should be noted.

The user should also note that outline highlighting is activated for all windows and that the forward-delete key is supported by the application. (The forward-delete key is not supported by TextEdit.)

### Help Dialog

The user should choose MonoTextEdit Help from the Mac OS 8/9 Apple menu or Mac OS X Help menu to open the Help dialog and then scroll through the three help topics, which may be chosen in the pop-up menu at the bottom of the dialog. The help topics contain documentation on the Help dialog which supplements the source code comments below.

## MonoTextEdit.c

### defines

kMaxTELength represents the maximum allowable number of bytes in a TextEdit structure. kTab, kDel, and kReturn representing the character codes generated by the tab, delete and return keys.

### typedefs

The docStructure data type will be used for a small document structure comprising a handle to a TextEdit structure and a handle to a (vertical) scroll bar.

### Global Variables

gScrollActionFunctionUPP will be assigned a universal procedure pointer to an action function for the scroll bar. gCustomClickLoopUPP will be assigned a universal procedure pointer to a custom click loop function. gCursorRegion is related to the WaitNextEvent's mouseRgn parameter. gNumberOfWindows will keep track of the number of windows open at any one time. gOldControlValue will be assigned the scroll bar's control value.

### main

The main function creates universal procedure pointers for the application-defined scroll action and custom click loop functions and opens a new document window.

### eventLoop

Note that WaitNextEvent's sleep parameter is assigned the value returned by the call to GetCaretTime.

If WaitNextEvent returns a null event, and provided at least one window is open, the function doIdle is called.

### doIdle

doIdle is called whenever a NULL event is received.

The first line gets a reference to the front window, allowing the next line to attempt to retrieve a handle to that window's document structure.  If the attempt is successful, TEIdle is called to blink the insertion point caret.

### doEvents

In the case of a mouse-down event in the content region, the function doInContent is called.

in the case of a keyDown event, if the Command key is down, the menus are adjusted and doMenuChoice is called.  If the Command key is not down, the function doKeyEvent is called.  In the case of an autoKey event, doKeyEvent is called provided the Command key is not down.

### doKeyEvent

doKeyEvent handles all key-down events that are not Command key equivalents.

The first three lines get a handle to the TextEdit structure whose handle is stored in the front window's document structure.

The next line filters out the tab key character code.  (TextEdit does not support the tab key and some applications may need to provide a tab key handler.)

The next character code to be filtered out is the forward-delete key character code.  TextEdit does not recognise this key, so this else if block provides forward-delete key support for the program.  The first line in this block gets the current selection length from the TextEdit structure.  If this is zero (that is, there is no selection range and an insertion point is being displayed), the selEnd field is increased by one.  This, in effect, creates a selection range comprising the character following the insertion point.  TEDelete deletes the current selection range from the TextEdit structure.  Such deletions could change the number of text lines in the TextEdit structure, requiring the vertical scroll bar to be adjusted; hence the call to the function doAdjustScrollbar.

Processing of those character codes which have not been filtered out is performed in the else block.  A new character must not be allowed to be inserted if the TextEdit limit of 32,767 characters will be exceeded.  Accordingly, and given that TEKey replaces the selection range with the character passed to it, the first step is to get the current selection length.  If the current number of characters minus the selection length plus 1 is less than 32,767, the character code is passed to TEKey for insertion into the TextEdit structure.  In addition, and since all this could change the number of lines in the TextEdit structure, the scroll bar adjustment function is called.

If the TextEdit limit will be exceeded by accepting the character, an alert is invoked advising the user of the situation.

The last line calls a function which prints data extracted from the edit text and control structures at the bottom of the window.

### scrollActionFunction

scrollActionFunction is associated with the vertical scroll bar.  It is the callback function which will be repeatedly called by TrackControl while the mouse button remains down in the scroll box/scroller, scroll arrows or gray areas/track of the vertical scroll bar.

The first line gets a reference to the window object for the window which "owns" the control.  The next two lines get a handle to the TextEdit structure associated with the window.

Within the outer if block, the first if block executes if the control part is not the scroll box/scroller (that is, the indicator).  The purpose of the switch is to get a value into the variable linesToScroll.  If the mouse-down was in a scroll arrow, that value will be 1.  If the mouse-down was in a gray area/track, that value will be equivalent to one less than the number of text lines that will fit in the view rectangle.  (Subtracting 1 from the total number of lines that will fit in the view rectangle ensures that the line of text at the bottom/top of the view rectangle prior to a gray area/track scroll will be visible at the top/bottom of the window after the scroll.)

Immediately after the switch, the value in linesToScroll is changed to a negative value if the mouse-down occurred in either the down scroll arrow or down gray area/track.

The next block ensures that no scrolling action will occur if the document is currently scrolled fully up (control value equals control maximum) or fully down (control value equals 0).  In either case, linesToScroll will be set to 0, meaning that the call to TEScroll near the end of the function will not occur.

SetControlValue sets the control value to the value just previously calculated, that is, to the current control value minus the value in linesToScroll.

The next line sets the value in linesToScroll back to what it was before the line linesToScroll = controlvalue - linesToScroll executed. This value, multiplied by the value in the lineHeight field of the TextEdit structure, is later passed to TEScroll as the parameter which specifies the number of pixels to scroll.

If the control part is the scroll box/scroller (that is, the indicator), the variable linesToScroll is assigned a value equal to the control's value as it was last time this function was called minus the control's current value. The global variable which holds the control's "old" value is then assigned the control's current value preparatory to the next call to this function.

With the number of lines to scroll determined, TEScroll is called to scroll the text within the view rectangle by the number of pixels specified in the second parameter. (Positive values scroll the text towards the bottom of the screen. Negative values scroll the text towards the top.)

The last line is for demonstration purposes only. It calls the function which prints data extracted from the edit and control structures at the bottom of the window.

### doInContent

doInContent continues mouse-down processing.

The first three lines retrieve a handle to the TextEdit structure associated with the front window.

The next three lines convert the mouse-down coordinates from global to local coordinates. (Local coordinates will be required by upcoming calls to FindControl, TrackControl, and PtInRect.)

If the mouse-down was in a control (that is, the scroll bar), the global variable gOldControlValue is assigned the value of the control to cater for the case of the mouse-down being within the scroll box/scroller (indicator), and TrackControl is called with the universal procedure pointer to the application-defined scroll action function passed in the third parameter. TrackControl retains control until the mouse button is released, during which time the previously described action function scrollActionFunction is repeatedly called.

If the mouse-down was not in a control, PtInRect checks whether it occurred within the view rectangle. (Note that the view rectangle is in local coordinates, so the mouse-down coordinates passed as the first parameter to the PtInRect call must also be in local coordinates.) If the mouse-down was in the view rectangle, a check is made of the shift key position at the time of the mouse-down. The result is passed as the second parameter in the call to TEClick. (TEClick's behaviour depends on the position of the shift key.)

### doUpdate

doUpdate handles update events. The first three lines get the handle to the TextEdit structure associated with the window.

Between the usual BeginUpdate and EndUpdate calls, TEUpdate is called to draw the text in the TextEdit structure, UpdateControls is called to draw the scroll bar, and the data panel is redrawn.

### doActivateDocWindow

doActivateDocWindow handles window activation/deactivation.

The first two lines retrieve a handle to the TextEdit structure for the window.

If the window is becoming active, its graphics port is set as the current graphics port. The bottom of the view rectangle is then adjusted so that the height of the view rectangle is an exact multiple of the value in the lineHeight field of the TextEdit structure. (This avoids the possibility of only part of the full height of a line of text appearing at the bottom of the view rectangle.) TEActivate activates the TextEdit structure associated with the window, ActivateControl activates the scroll bar, doAdjustScrollbar adjusts the scroll bar, and doAdjustCursor adjusts the cursor shape.

If the window is becoming inactive, TEDeactivate deactivates the TextEdit structure associated with the window and DeactivateControl deactivates the scroll bar.

### doNewDocWindow

doNewDocWindow is called at program launch and when the user chooses New or Open from the File menu. It opens a new window, associates a document structure with that window, creates a vertical scroll bar, creates a monostyled TextEdit structure, installs the custom click loop function, enables automatic scrolling, and enables outline highlighting.

GetNewCWindow opens a new window and sets its graphics port as the current graphics port. (Since the TextEdit structure assumes the drawing environment specified in the graphics port structure, setting the graphics port must be done before the call to TENew to create the TextEdit structure.)

The next line sets the text size. (These will be copied from the graphics port to the TextEdit structure when TENew is called.)

The next block creates a document structure and stores the handle to that structure in the window's window object. The following line increments the global variable which keeps track of the number of open windows. GetNewControl creates a vertical scroll bar and assigns a handle to it to the appropriate field of the document structure. The next block establishes the view and destination rectangles two pixels inside the window's port rectangle less the scroll bar.

MoveHHi and HLock move the document structure high and lock it. A monostyled TextEdit structure is then created by TENew and its handle is assigned to the appropriate field of the document structure. (If this call is not successful, the window and scroll bar are disposed of, an error alert is displayed, and the function returns.) The handle to the document structure is then unlocked.

TESetClickLoop installs the universal procedure pointer to the custom click loop function customClickLoop in the clickLoop field of the TextEdit structure. TEAutoView enables automatic scrolling for the TextEdit structure. TEFeatureFlag enables outline highlighting for the TextEdit structure.

The last line returns a reference to the newly opened window's window object.

### *customClickLoop*

customClickLoop replaces the default click loop function so as to provide for scroll bar adjustment in concert with automatic scrolling. Following a mouse-down within the view rectangle, customClickLoop is called repeatedly by TEClick as long as the mouse button remains down.

The first three lines retrieve a handle to the TextEdit structure associated with the window. The next two lines save the current graphics port and set the window's graphics port as the current port.

The window's current clip region will have been set by TextEdit to be equivalent to the view rectangle. Since the scroll bar has to be redrawn, the clipping region must be temporarily reset to include the scroll bar. Accordingly, GetClip saves the current clipping region and the following two lines set the clipping region to the bounds of the coordinate plane.

GetMouse gets the current position of the cursor. If the cursor is above the top of the port rectangle, the text must be scrolled downwards. Accordingly, the variable linesToScroll is set to 1. The subsidiary function doSetScrollBarValue (see below) is then called to, amongst other things, reset the scroll bar's/scroller's value. Note that the value in linesToScroll may be modified by doSetScrollBarValue. If linesToScroll is not set to 0 by doSetScrollBarValue, TEScroll is called to scroll the text by a number of pixels equivalent to the value in the lineHeight field of the TextEdit structure, and in a downwards direction.

If the cursor is below the bottom of the port rectangle, the same process occurs except that the variable linesToScroll is set to -1, thus causing an upwards scroll of the text (assuming that the value in linesToScroll is not changed to 0 by doSetScrollBarValue).

If scrolling has occurred, doDrawDataPanel redraws the data panel. SetClip restores the clipping region to that established by the view rectangle and SetPort restores the saved graphics port. Finally, the last line returns true. (A return of false would cause TextEdit to stop calling customClickLoop, as if the user had released the mouse button.)

### *doSetScrollBarValue*

doSetScrollBarValue is called from customClickLoop. Apart from setting the scroll bar's/scroller's value so as to cause the scroll box/scroller to follow up automatic scrolling, the function checks whether the limits of scrolling have been reached.

The first two lines get the current control value and the current control maximum value. At the next block, the value in the variable linesToScroll will be set to either 0 (if the current control value is 0) or equivalent to the control maximum value (if the current control value is equivalent to the control maximum value. If these modifications do not occur, the value in linesToScroll will remain as established at the first line in this block, that is, the current control value minus the value in linesToScroll as passed to the function.

SetControlValue sets the control's value to the value in linesToScroll. The last line sets the value in linesToScroll to 0 if the limits of scrolling have already been reached, or to the value as it was when the doSetScrollBarValue function was entered.

### *doAdjustMenus*

doAdjustMenus adjusts the menus. Much depends on whether any windows are currently open.

If at least one window is open, the first three lines in the if block get a handle to the TextEdit structure associated with the front window and the first call to EnableMenuItem enables the Close item in the File menu. If there is a current selection range, the Cut, Copy, and Clear items are enabled, otherwise they are disabled. If there is data of flavour type 'TEXT' in the scrap (the call to GetScrapFlavourFlags), the Paste item is enabled, otherwise it is disabled. If there is any text in the TextEdit structure, the SaveAs and Select All items are enabled, otherwise they are disabled.

If no windows are open, the Close, SaveAs, Clear, and Select All items are disabled.

### *doFileMenu*

doFileMenu handles File menu choices, calling the appropriate functions according to the menu item chosen. In the SaveAs case, a handle to the TextEdit structure associated with the front window is retrieved and passed as a parameter to the appropriate function.

Note that, because TextEdit, rather than file operations, is the real focus of this program, the file-related code has been kept to a minimum, even to the extent of having no Save-related, as opposed to SaveAs-related, code.

### *doEditMenu*

doEditMenu handles choices from the Edit menu. Recall that, in the case of monostyled TextEdit structures, TECut, TECopy, and TEPaste do not copy/paste text to/from the scrap. This program, however, supports copying/pasting to/from the scrap.

Before the usual switch is entered, a handle to the TextEdit structure associated with the front window is retrieved.

The iCut case handles the Cut command. Firstly, the call to ClearCurrentScrap attempts to clear the scrap. If the call succeeds, PurgeSpace establishes the size of the largest block in the heap that would be available if a general purge were to occur. The next line gets the current selection length. If the selection length is greater than the available memory, the user is advised via an error message. Otherwise, TECut is called to remove the selected text from the TextEdit structure and copy it to the TextEdit private scrap. The scroll bar is adjusted, and TEToScrap is called to copy the private scrap to the scrap. If the TEToScrap call is not successful, ClearCurrentScrap cleans up as best it can by emptying the scrap.

The iCopy case handles the Copy command. If the call to ClearCurrentScrap to empty the scrap is successful, TECopy is called to copy the selected text from the TextEdit structure to the TextEdit private scrap. TEToScrap then copies the private scrap to the scrap. If the TEToScrap call is not successful, ClearCurrentScrap cleans up as best it can by emptying the scrap.

The iPaste case handles the Paste command, which must not proceed if the paste would cause the TextEdit limit of 32,767 bytes to be exceeded. The third line establishes a value equal to the number of bytes in the TextEdit structure plus the number of bytes of the 'TEXT' flavour type in the scrap. If this value exceeds the TextEdit limit, the user is advised via an error message. Otherwise, TEFromScrap copies the scrap to TextEdit's private scrap, TEPaste inserts the private scrap into the TextEdit structure, and the following line adjusts the scroll bar.

The iClear case handles the Clear command. TEDelete deletes the current selection range from the TextEdit structure and the following line adjusts the scroll bar.

The iSelectAll case handle the Select All command. TESetSelect sets the selection range according to the first two parameters (selStart and selEnd).

### *doGetSelectLength*

doGetSelectLength returns a value equal to the length of the current selection.

### *doAdjustScrollbar*

doAdjustScrollbar adjusts the vertical scroll bar.

The first two lines retrieve handles to the document structure and TextEdit structure associated with the window in question.

At the next block, the value in the nLines field of the TextEdit structure is assigned to the numberOfLines variable. The next action is somewhat of a refinement and is therefore not essential. If the last character in the TextEdit structure is the return character, numberOfLines is incremented by one. This will ensure that, when the document is scrolled to its end, a single blank line will appear below the last line of text.

At the next block, the variable controlMax is assigned a value equal to the number of lines in the TextEdit structure less the number of lines which will fit in the view rectangle. If this value is less than 0 (indicating that the number of lines in the TextEdit structure is less than the number of lines that will fit in the view rectangle), controlMax is set to 0. SetControlMaximum then sets the control maximum value. If controlMax is 0, the scroll bar is automatically unhighlighted by the SetControlMaximum call.

The first line of the next block assigns to the variable controlValue a value equal to the number of text lines that the top of the destination rectangle is currently "above" the top of the view rectangle. If the calculation returns a value less than 0 (that is, the document has been scrolled fully down), controlValue is set to 0. If the calculation returns a value greater than the current control maximum value (that is, the document has been scrolled fully up), controlValue is set to equal that value. SetControlValue sets the control value to the value in controlValue. For example, if the top of the view rectangle is 2, the top of the destination rectangle is -34 and the lineHeight field of the TextEdit structure contains the value 13, the control value will be set to 3.

SetControlViewSize is called to advise the Control Manager of the height of the view rectangle. This will cause the scroll box/scroller to be a proportional scroll box/scroller. (On Mac OS 8/9, this assumes that the user has selected Smart Scrolling on in the Appearance control panel.)

With the control maximum value and the control value set, TEScroll is called to make sure the text is scrolled to the position indicated by the scroll box/scroller. Extending the example in the previous paragraph, the second parameter in the TEScroll call is 2 - (34 - (3 * 13)), that is, 0. In that case, no corrective scrolling actually occurs.

### doAdjustCursor

doAdjustCursor adjusts the cursor to the I-Beam shape when the cursor is over the content region less the scroll bar area, and to the arrow shape when the cursor is outside that region. It is similar to the cursor adjustment function in the demonstration program GworldPicCursIcn (Chapter 13).

### doCloseWindow

doCloseWindow disposes of the specified window. The associated scroll bar, the associated TextEdit structure and the associated document structure are disposed of before the call to DisposeWindow.

### doSaveAsFile, doOpenCommand, doOpenFile

The functions doSaveAsFile, doOpenCommand, and doOpenFile are document saving and opening functions, enabling the user to open and save 'TEXT' documents. Since the real focus of this program is TextEdit, not file operations, the code is "bare bones" and as brief as possible.

> For a complete example of opening and saving monostyled 'TEXT' documents, see the demonstration program Files (Chapter 18).

### doDrawDataPanel

doDrawDataPanel draws the data panel at the bottom of each window. Displayed in this panel are the values in the teLength, nLines, lineHeight and destRect.top fields of the TextEdit structure and the contrlValue and contrlMax fields of the scroll bar's control structure.

## HelpDialog.c

### defines

Constants are established for the 'DLOG' resource ID and items in the dialog, the index of error strings within a 'STR#' resource, and 'TEXT', 'styl', and 'PICT' resource IDs. kTextInset which will be used to inset the view and destination rectangles a few pixels inside the user pane's rectangle.

### typedefs

The first two data types are for a picture information structure and a document structure. (Note that one field in the document structure is a pointer to a picture information structure.) The third data type will be used for saving and restoring the background colour and pattern.

### doHelp

doHelp is called when the user chooses the MonoTextEdit Help item in the Help menu.

The dialog will utilise an event filter function, a user pane drawing function, and an action function. The first three lines create the associated routine descriptors.

GetNewDialog creates the dialog.  NewHandle creates a block for a document structure and the handle is stored in the dialog's window object.  SetDialogDefaultItem sets the dialog's push button as the default item and ensures that the default ring will be drawn around that item.

At the next block, SetControlData sets the user pane drawing function.

At the next block, the destination and view rectangles are both made equal to the user pane's rectangle, but inset four pixels from the left and right and two pixels from the top and bottom.  The call to TEStyleNew creates a multistyled TextEdit structure based on those two rectangles.

The next block assigns the reference to the scrollbar to the appropriate field of the dialog's document structure.

A pointer to a picture information structure will eventually be assigned to a field in the document structure.  For the moment, that field is set to NULL.

At the next block, two global variables are assigned the resource IDs relating to the first Help topic's 'TEXT'/'styl' resource and associated 'PICT' resources.

The next block calls the function doGetText which, amongst other things, loads the specified 'TEXT'/'styl' resources and inserts the text and style information into the TextEdit structure.

The next block calls the function doGetPictureInfo which, amongst other things, searches for option-space characters in the 'TEXT' resource and, if option-space characters are found, loads a like number of 'PICT' resources beginning with the specified ID.

NewRgn creates an empty region, which will be used to save the dialog's graphic's port's clipping region.

To complete the initial setting up, ShowWindow is called to makes the dialog visible.

The do-while ModalDialog loop continues until the user clicks the "Done" (OK) button or hits the Return key.

The dialog uses an application-defined filter function which, as will be seen, handles mouse-down events within the scrollbar.  The only other event of interest is a hit on the pop-up menu button.  Accordingly, if ModalDialog returns a hit on that control, SetControlValue is called to set the scroll bar's value to 0, the menu item chosen is determined, and the switch assigns the appropriate 'TEXT'/'styl' and 'PICT' resource IDs to the global variables which keep track of which of those resources are to be loaded and displayed.

The next block then performs the same "get text" and "get picture information" actions as were preformed at start-up, but this time with the 'TEXT'/'styl' and 'PICT' resources as determined within the preceding switch.

The call to doDrawPictures draws any pictures that might initially be located in the view rectangle.

When the user clicks the "Done" (OK) button or hits the Return key, functions are called to close down the Help dialog and dispose of the routine descriptors.

### doCloseHelp

doCloseHelp closes down the Help dialog.

The first two lines retrieve a handle to the dialog's document structure.  The next two lines dispose of the region used to save the clipping region.  TEDispose disposes of the TextEdit structure.  The next block disposes of any 'PICT' resources currently in memory, together with the picture information structure.  Finally, the dialog's document structure is disposed of, the dialog itself is disposed of, and the graphics port saved in doHelp is restored.

### userPaneDrawFunction

userPaneDrawFunction is the user pane drawing function set within doHelp.  It will be called automatically whenever the dialog receives an update event.

The first block gets a pointer to the dialog to which the user pane control belongs, gets the user pane's rectangle, insets that rectangle by one pixel all round, and then further expands it to the right to the right edge of the scroll bar.  At the next block, a list box frame is drawn in the appropriate state, depending on whether the window is the movable modal dialog is currently the active window.

The next block erases the previously defined rectangle with the white colour using the white pattern.

The next three lines retrieve the view rectangle from the TextEdit structure.  The call to TEUpdate draws the text in the TextEdit structure in the view rectangle.  The call to drawPictures draws any pictures which might currently be located in the view rectangle.

### *doGetText*

doGetText is called when the dialog is first opened and when the user chooses a new item from the pop-up menu.  Amongst other things, it loads the 'TEXT'/'styl' resources associated with the current menu item and inserts the text and style information into the TextEdit structure.

The first two lines get a handle to the TextEdit structure.  The next two lines set the selection range to the maximum value and then delete that selection.  The destination rectangle is then made equal to the view rectangle and the scroll bar's value is set to 0.

GetResource is called twice to load the specified 'TEXT'/'styl' resources, following which TEStyleInsert is called to insert the text and style information into the TextEdit structure.  Two calls to ReleaseResource then release the 'TEXT'/'styl' resources.

The next block gets the total height of the text in pixels.

At the next block, if the height of the text is greater than the height of the view rectangle, the local variable heightToScroll is made equal to total height of the text minus the height of the view rectangle.  This value is then used to set the scroll bar's maximum value.  The scroll bar is then made active.

SetControlViewSize is called to advise the Control Manager of the height of the view rectangle.  If, on Mac OS 8/9, Smart Scrolling is selected on in the Options tab of the Appearance control panel, this will cause the scroll box/scroller to be a proportional scroll box/scroller.

If the height of the text is less than the height of the view rectangle, the scroll bar is made inactive.

true is returned if the GetResource calls did not return with false.

### *doGetPictureInfo*

doGetPictureInfo is called after getText when the dialog is opened and when the user chooses a new item from the pop-up menu.  Amongst other things, it searches for option-space characters in the 'TEXT' resource and, if option-space characters are found, loads a like number of 'PICT' resources beginning with the specified ID.

The first line gets a handle to the dialog's document structure.

If the picInfoRecPtr field of the document structure does not contain NULL, the currently loaded 'PICT' resources are released, the picture information structures are disposed of, and the picInfoRecPtr field of the document structure is set to NULL.

The next line sets to 0 the field of the document structure which keeps track of the number of pictures associated with the current 'TEXT' resource.

The next two lines get a handle to the TextEdit structure, then a handle to the block containing the actual text.  This latter is then used to assign the size of that block to a local variable.  After two local variables are initialised, the block containing the text is locked.

The next block counts the number of option-space characters in the text block.  At the following block, if there are no option-space characters in the block, the block is unlocked and the function returns.

A call to NewPtr then allocates a nonrelocatable block large enough to accommodate a number of picture information structures equal to the number of option-space characters found.  The pointer to the block is then assigned to the appropriate field of the dialog's document structure.

The next line resets the offset value to 0.

The for loop repeats for each of the option-space characters found.  GetPicture loads the specified 'PICT' resource (the resource ID being incremented from the base ID at each pass through the loop) and assigns the handle to appropriate field of the relevant picture information structure.  Munger finds the offset to the next option-space character and TEGetPoint gets the point, based on the destination rectangle, of the bottom left of the character at that offset. TEGetStyle is called to obtain the line height of the character at the offset and this value is subtracted from the value in the point's v field. The offset is incremented and the rectangle in the picture structure's picFrame field is assigned to the bounds field of the picture information structure. The next block then offsets this rectangle so that it is centred laterally in the destination rectangle with its top offset from the top of the destination rectangle by the amount established at the line picturePoint.v -= lineHeight;.

The third last line assigns the number of pictures loaded to the appropriate field of the dialog's document structure. The block containing the text is then unlocked. The function returns true if false has not previously been returned within the for loop.

### actionFunction

actionFunction is the action function called from within the event filter function eventFilter. It is repeatedly called by TrackControl while the mouse button remains down within the scroll bar. Its ultimate purpose is to determine the new scrollbar value when the mouse-down is within the scroll arrows or gray areas/track of the scroll bar, and then call a separate function to effect the actual scrolling of the text and pictures based on the new scrollbar value. (The scroll bar is the live scrolling variant, so the CEDF automatically updates the control's value while the mouse remains down in the scroll box/scroller.)

Firstly, if the cursor is still not within the control, execution falls through to the bottom of the function and the action function exits.

The first block gets a pointer to the owner of the scrollbar, retrieves a handle to the dialog's document structure, gets a handle to the TextEdit structure, gets the view rectangle, and assigns a value to the h field of a point variable equal to the left of the view rectangle plus 4 pixels.

The switch executes only if the mouse-down is not in the scroll box/scroller.

In the case of the Up scroll arrow, the variable delta is assigned a value which will ensure that, after the scroll, the top of the incoming line of text will be positioned cleanly at top of the view rectangle.

In the case of the Down scroll arrow, the variable delta is assigned a value which will ensure that, after the scroll, the bottom of the incoming line of text will be positioned cleanly at bottom of the view rectangle.

In the case of the Up gray area/track, the variable delta is assigned a value which will ensure that, after the scroll, the top of the top line of text will be positioned cleanly at the top of the view rectangle and the line of text which was previously at the top will still be visible at the bottom of the view rectangle.

In the case of the Down gray area/track, the variable delta is assigned a value which will ensure that, after the scroll, the bottom of the bottom line of text will be positioned cleanly at the bottom of the view rectangle and the line of text which was previously at the bottom will still be visible at the top of the view rectangle.

The first line after the switch gets the pre-scroll scroll bar value. If the text is not fully scrolled up and a scroll up is called for, or if the text is not fully scrolled down and a scroll down is called for, the current clipping region is saved, the clipping region is set to the dialog's port rectangle, the scroll bar value is set to the required new value, and the saved clipping region is restored. (TextEdit may have set the clipping region to the view rectangle, so it must be changed to include the scroll bar area, otherwise the scroll bar will not be drawn.)

With the scroll bar's new value set and the scroll box/scroller redrawn in its new position, the function for scrolling the text and pictures is called. Note that this last line will also be called if the mouse-down was within the scroll box/scroller.

### doScrollTextAndPicts

doScrollTextAndPicts is called from actionFunction. It scrolls the text within the view rectangle and calls another function to draw any picture whose rectangle intersects the "vacated" area of the view rectangle.

The first line sets the background colour to white and the background pattern to white.

The next two lines get a handle to the TextEdit structure. The next line determines the difference between the top of the destination rectangle and the top of the view rectangle and the next subtracts from this value the scroll bar's new value. If the result is zero, the text must be fully scrolled in one direction or the other, so the function simply returns.

If the text is not already fully scrolled one way or the other, TEScroll scrolls the text in the view rectangle by the number of pixels determined at the fifth line.

If there are no pictures associated with the 'TEXT' resource in use, the function returns immediately after the text is scrolled.

The next consideration is the pictures and whether any of their rectangles, as stored in the picture information structure, intersect the area of the view rectangle "vacated" by the scroll. At the if/else block, a rectangle is made equal to the "vacated" area of the view rectangle, the if block catering for the scrolling up case and the else block catering for the scrolling down case. (Of course, if the scroll box/scroller has been dragged by the user over a large distance, the "vacated" area will equate to the whole view rectangle.) This rectangle is passed as a parameter in the call to drawPictures.

### *doDrawPictures*

doDrawPictures determines whether any pictures intersect the rectangle passed to it as a formal parameter and draws any pictures that do.

The first two lines get handles to the dialog's document structure and the TextEdit structure.

The next line determines the difference between the top of the destination rectangle and the top of the view rectangle. This will be used later to offset the picture's rectangle from destination rectangle coordinates to view rectangle coordinates. The next line determines the number of pictures associated with the current 'TEXT' resource, a value which will be used to control the number of passes through the following for loop.

Within the loop, the picture's rectangle is retrieved from the picture information structure and offset to the coordinates of the view rectangle. SectRect determines whether this rectangle intersects the rectangle passed to drawPictures from scrollTextAndPicts. If it does not, the loop returns for the next iteration. If it does, the picture's handle is retrieved, LoadResource checks whether the 'PICT' resource is in memory and, if necessary, loads it, HLock locks the handle, DrawPicture draws the picture, and HUnlock unlocks the handle. Before DrawPicture is called, the clipping region is temporarily adjusted to equate to the rectangle passed to drawPictures from scrollTextAndPicts so as to limit drawing to that rectangle.

### *eventFilter*

eventFilter is the application-defined filter function for the dialog. It is essentially the same as the event filter introduced in the demonstration program DialogsAndAlerts (Chapter 8) except that it also intercepts mouse-down events.

In the case of a mouse-down event, the location of the event is determined in local coordinates. If FindControl and the following two lines determine that the mouse-down was within the scrollbar, TrackControl is called to handle user interaction while the mouse button remains down. While the mouse button remains down, TrackControl continually calls actionFunction.

### *doSetBackgroundWhite*

doSetBackgroundWhite sets the background colour to white and the background pattern to the pattern white.