Demonstration Program Drag Listing

```
// ****************************************************************************************
// Drag.h                                                                  CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program demonstrates drag and drop utilising the Drag Manager.  Support for Undo and
// Redo of drag and drop within the source window is included.
//
// The bulk of the code in the source code file Drag.c is identical to the code in the file
// Text1.c in the Chapter 20 TextEdit demonstration program MonoTextEdit.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus.
//
// •  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 '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 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  eTextEdit        4
#define  eExceedChara     5
#define  eNoSpaceCut      6
#define  eNoSpacePaste    7
#define  eDragHandler     8
#define  eDrag            9
#define  eDragUndo        10
#define kMaxTELength      32767
#define kTab              0x09
#define kDel              0x7F
```

```
#define kReturn           0x0D
#define topLeft(r)        (((Point *) &(r))[0])
#define botRight(r)       (((Point *) &(r))[1])

// ..... typedefs

typedef struct
{
  TEHandle   textEditStrucHdl;
  ControlRef vScrollbarRef;
  WindowRef  windowRef;
  Boolean    windowTouched;
  Handle     preDragText;
  SInt16     preDragSelStart;
  SInt16     preDragSelEnd;
  SInt16     postDropSelStart;
  SInt16     postDropSelEnd;
} docStructure, *docStructurePointer;

// ..... 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      doErrorAlert           (SInt16);

OSErr     doStartDrag            (EventRecord *,RgnHandle,TEHandle);
OSErr     dragTrackingHandler    (DragTrackingMessage,WindowRef,void *,DragRef);
SInt16    doGetOffset            (Point,TEHandle);
SInt16    doIsOffsetAtLineStart  (SInt16,TEHandle);
void      doDrawCaret            (SInt16,TEHandle);
SInt16    doGetLine              (SInt16,TEHandle);

OSErr     dragReceiveHandler     (WindowRef,void *,DragRef);
Boolean   doIsWhiteSpaceAtOffset (SInt16,TEHandle);
Boolean   doIsWhiteSpace         (char);
char      doGetCharAtOffset      (SInt16,TEHandle);
SInt16    doInsertTextAtOffset   (SInt16,Ptr,SInt32,TEHandle);
Boolean   doSavePreInsertionText (docStructurePointer);
void      doUndoRedoDrag         (WindowRef);

// ********************************************************************************
// Drag.c
// ********************************************************************************
```

```
// ..... includes

#include "Drag.h"

// ..... global variables

ControlActionUPP       gScrollActionFunctionUPP;
TEClickLoopUPP         gCustomClickLoopUPP;
DragTrackingHandlerUPP gDragTrackingHandlerUPP;
DragReceiveHandlerUPP  gDragReceiveHandlerUPP;
Boolean                gRunningOnX = false;
Boolean                gDone;
RgnHandle              gCursorRegion;
SInt16                 gNumberOfWindows = 0;
SInt16                 gOldControlValue;
Boolean                gEnableDragUndoRedoItem = false;
Boolean                gUndoFlag;

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

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

  // ..... do preliminaries

  doPreliminaries();

  // ..... create universal procedure pointers

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

  gDragTrackingHandlerUPP  = NewDragTrackingHandlerUPP((DragTrackingHandlerProcPtr)
                                                        dragTrackingHandler);
  gDragReceiveHandlerUPP   = NewDragReceiveHandlerUPP((DragReceiveHandlerProcPtr)
                                                      dragReceiveHandler);

  // ..... 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);
    }

    gRunningOnX = true;
  }

  // ..... 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)
{
  docStructurePointer docStrucPtr;
  WindowRef           windowRef;

  windowRef = FrontWindow();

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  if(docStrucPtr != NULL)
    TEIdle(docStrucPtr->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;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              selectionLength;

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  gEnableDragUndoRedoItem = false;

  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);
  }
}

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

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

  windowRef = GetControlOwner(controlRef);
```

```
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;
  textEditStrucHdl = docStrucPtr->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);
  }
}

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

void  doInContent(EventRecord *eventStrucPtr)
{
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  Point               mouseXY;
  ControlRef          controlRef;
  SInt16              partCode;
  RgnHandle           hiliteRgn;
  OSErr               osError;
  Boolean             shiftKeyPosition = false;

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) GetWRefCon(windowRef);
  textEditStrucHdl = docStrucPtr->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))
  {
    hiliteRgn = NewRgn();

    TEGetHiliteRgn(hiliteRgn,textEditStrucHdl);

    if(!EmptyRgn(hiliteRgn) && PtInRgn(mouseXY,hiliteRgn))
    {
      if(WaitMouseMoved(eventStrucPtr->where))
      {
        osError = doStartDrag(eventStrucPtr,hiliteRgn,textEditStrucHdl);
        if(osError != noErr)
          doErrorAlert(eDrag);
      }
    }
    else
    {
      if((eventStrucPtr->modifiers & shiftKey) != 0)
        shiftKeyPosition = true;
      TEClick(mouseXY,shiftKeyPosition,textEditStrucHdl);

      gEnableDragUndoRedoItem = false;

      doAdjustCursor(windowRef);
    }

    DisposeRgn(hiliteRgn);
  }
}

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

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

  windowRef = (WindowRef) eventStrucPtr->message;
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  BeginUpdate((WindowRef) eventStrucPtr->message);

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

  UpdateControls(windowRef,visibleRegionHdl);

  GetWindowPortBounds(windowRef,&portRect);
  TEUpdate(&(*textEditStrucHdl)->viewRect,textEditStrucHdl);

  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)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->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(docStrucPtr->vScrollbarRef);
    doAdjustScrollbar(windowRef);
    doAdjustCursor(windowRef);
  }
  else
  {
    TEDeactivate(textEditStrucHdl);
    DeactivateControl(docStrucPtr->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;
```

```
  docStructurePointer docStrucPtr;
  Rect                portRect, destAndViewRect;
  OSErr               osError;

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

  SetPortWindowPort(windowRef);
  TextSize(10);

  if(!(docStrucPtr = (docStructurePointer) NewPtr(sizeof(docStructure))))
  {
    doErrorAlert(eDocStructure);
    return NULL;
  }

  SetWRefCon(windowRef,(SInt32) docStrucPtr);
  SetWindowProxyCreatorAndType(windowRef,0,'TEXT',kUserDomain);

  gNumberOfWindows ++;

  docStrucPtr->windowRef     = windowRef;
  docStrucPtr->windowTouched = false;
  docStrucPtr->preDragText   = NULL;
  docStrucPtr->vScrollbarRef = GetNewControl(rVScrollbar,windowRef);

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

  if(!(docStrucPtr->textEditStrucHdl = TENew(&destAndViewRect,&destAndViewRect)))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposePtr((Ptr) docStrucPtr);
    doErrorAlert(eTextEdit);
    return NULL;
  }

  TESetClickLoop(gCustomClickLoopUPP,docStrucPtr->textEditStrucHdl);
  TEAutoView(true,docStrucPtr->textEditStrucHdl);
  TEFeatureFlag(teFOutlineHilite,teBitSet,docStrucPtr->textEditStrucHdl);

  if(osError = InstallTrackingHandler(gDragTrackingHandlerUPP,windowRef,docStrucPtr))
  {
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposePtr((Ptr) docStrucPtr);
    doErrorAlert(eDragHandler);
    return NULL;
  }

  if(osError = InstallReceiveHandler(gDragReceiveHandlerUPP,windowRef,docStrucPtr))
  {
    RemoveTrackingHandler(gDragTrackingHandlerUPP,windowRef);
    DisposeWindow(windowRef);
    gNumberOfWindows --;
    DisposePtr((Ptr) docStrucPtr);
    doErrorAlert(eDragHandler);
    return NULL;
  }

  return windowRef;
}
```

```
// ************************************************************************** customClickLoop

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

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->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(docStrucPtr->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }
  else if(mouseXY.v > portRect.bottom)
  {
    linesToScroll = -1;
    doSetScrollBarValue(docStrucPtr->vScrollbarRef,&linesToScroll);
    if(linesToScroll != 0)
      TEScroll(0,linesToScroll * ((*textEditStrucHdl)->lineHeight),textEditStrucHdl);
  }

  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             fileMenuRef, editMenuRef;
  WindowRef           windowRef;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  ScrapRef            scrapRef;
  OSStatus            osError;
  ScrapFlavorFlags    scrapFlavorFlags;

  fileMenuRef = GetMenuRef(mFile);
  editMenuRef = GetMenuRef(mEdit);

  if(gNumberOfWindows > 0)
  {
    windowRef = FrontWindow();
    docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;
    textEditStrucHdl = docStrucPtr->textEditStrucHdl;

    EnableMenuItem(fileMenuRef,iClose);

    if(gEnableDragUndoRedoItem)
    {
      EnableMenuItem(editMenuRef,iUndo);
      if(gUndoFlag)
        SetMenuItemText(editMenuRef,iUndo,"\pUndo Drag & Drop");
      else
        SetMenuItemText(editMenuRef,iUndo,"\pRedo Drag & Drop");
    }
    else
    {
      DisableMenuItem(editMenuRef,iUndo);
      SetMenuItemText(editMenuRef,iUndo,"\pRedo Drag & Drop");
    }

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

    GetCurrentScrap(&scrapRef);

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

    if((*textEditStrucHdl)->teLength > 0)
    {
      EnableMenuItem(fileMenuRef,iSaveAs);
      EnableMenuItem(editMenuRef,iSelectAll);
    }
    else
    {
      DisableMenuItem(fileMenuRef,iSaveAs);
      DisableMenuItem(editMenuRef,iSelectAll);
    }
  }
  else
```

```
  {
    DisableMenuItem(fileMenuRef,iClose);
    DisableMenuItem(fileMenuRef,iSaveAs);
    DisableMenuItem(editMenuRef,iClear);
    DisableMenuItem(editMenuRef,iSelectAll);
  }

  DrawMenuBar();
}

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

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

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

  if(menuID == 0)
    return;

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

    case mFile:
      doFileMenu(menuItem);
      break;

    case mEdit:
      doEditMenu(menuItem);
      break;
  }

  HiliteMenu(0);
}

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

void  doFileMenu(MenuItemIndex menuItem)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;

  switch(menuItem)
  {
    case iNew:
      doNewDocWindow();
      break;

    case iOpen:
      doOpenCommand();
      break;

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

    case iSaveAs:
      docStrucPtr = (docStructurePointer) (GetWRefCon(FrontWindow()));
      textEditStrucHdl = docStrucPtr->textEditStrucHdl;
      doSaveAsFile(textEditStrucHdl);
      break;
```

```
    case iQuit:
      gDone = true;
      break;
  }
}

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

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

  windowRef = FrontWindow();
  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->textEditStrucHdl;

  switch(menuItem)
  {
    case iUndo:
      doUndoRedoDrag(windowRef);
      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;
  }
}

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

SInt16  doGetSelectLength(TEHandle textEditStrucHdl)
{
  SInt16 selectionLength;

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

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

void  doAdjustScrollbar(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              numberOfLines, controlMax, controlValue;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;
  textEditStrucHdl = docStrucPtr->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(docStrucPtr->vScrollbarRef,controlMax);

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

  SetControlValue(docStrucPtr->vScrollbarRef,controlValue);

  SetControlViewSize(docStrucPtr->vScrollbarRef,(*textEditStrucHdl)->viewRect.bottom -
                     (*textEditStrucHdl)->viewRect.top);

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

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

void  doAdjustCursor(WindowRef windowRef)
{
  GrafPtr             oldPort;
  RgnHandle           arrowRegion, iBeamRegion, hiliteRgn;
  Rect                portRect, cursorRect;
  docStructurePointer docStrucPtr;
```

```
  Point               offset, mouseXY;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

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

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

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

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  TEGetHiliteRgn(hiliteRgn,docStrucPtr->textEditStrucHdl);
  LocalToGlobal(&topLeft(portRect));
  offset = topLeft(portRect);
  OffsetRgn(hiliteRgn,offset.h,offset.v);
  DiffRgn(iBeamRegion,hiliteRgn,iBeamRegion);

  GetGlobalMouse(&mouseXY);

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

  DisposeRgn(arrowRegion);
  DisposeRgn(iBeamRegion);
  DisposeRgn(hiliteRgn);

  SetPort(oldPort);
}

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

void  doCloseWindow(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));;

  DisposeControl(docStrucPtr->vScrollbarRef);
  TEDispose(docStrucPtr->textEditStrucHdl);
  DisposePtr((Ptr) docStrucPtr);

  if(docStrucPtr->preDragText == NULL)
    DisposeHandle(docStrucPtr->preDragText);

  RemoveTrackingHandler(gDragTrackingHandlerUPP,windowRef);
  RemoveReceiveHandler(gDragReceiveHandlerUPP,windowRef);
```

```
  DisposeWindow(windowRef);

  gNumberOfWindows --;
}

// ********************************************************************************** doSaveAsFile

void  doSaveAsFile(TEHandle textEditStrucHdl)
{
  OSErr            osError = noErr;
  NavDialogOptions dialogOptions;
  WindowRef        windowRef;
  NavEventUPP      navEventFunctionUPP;
  OSType           fileType;
  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,
                         'kkkB',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,'kkkB',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)
    {
      if(osError == noErr)
      {
        osError = AECountItems(&(navReplyStruc.selection),&count);

        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;
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt16              fileRefNum;
  SInt32              textLength;
  Handle              textBuffer;

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

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));
  textEditStrucHdl = docStrucPtr->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;
}

// ************************************************************************** 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);
  }
}

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

```
// StartAndTrackDrag.c
// ********************************************************************************************

// ..... includes

#include "Drag.h"

// ..... global variables

Boolean gCursorInContent, gCanAcceptItems, gCaretShowFlag;
SInt16  gInsertPosition, gLastOffset, gCaretOffset;
UInt32  gSystemCaretTime, gCaretStartTime;

extern Boolean gRunningOnX;

// ************************************************************************** doStartDrag

OSErr  doStartDrag(EventRecord *eventStrucPtr,RgnHandle hiliteRgnHdl,TEHandle textEditStrucHdl)
{
  OSErr         osError;
  DragReference dragRef;
  Rect          originalHiliteRect, zeroedHiliteRect;
  RgnHandle     maskRgnHdl;
  Point         offsetPoint;
  QDErr         qdError;
  CGrafPtr      savedPortPtr;
  GDHandle      saveDeviceHdl;
  GWorldPtr     dragGWorldPtr = NULL;
  PixMapHandle  dragPixMapHdl, windPixMapHdl;
  RgnHandle     dragRgnHdl, tempRgnHdl;

  // ..... create new drag

  if(osError = NewDrag(&dragRef))
    return osError;

  // ..... add 'TEXT' flavour

  osError = AddDragItemFlavor(dragRef,1,'TEXT',
                              (*(*textEditStrucHdl)->hText) + (*textEditStrucHdl)->selStart,
                              (*textEditStrucHdl)->selEnd - (*textEditStrucHdl)->selStart,0);


  // ..... get and set drag image for translucent drag and drop

  if(!gRunningOnX)
  {
    maskRgnHdl =  dragRgnHdl = tempRgnHdl = NULL;

    GetRegionBounds(hiliteRgnHdl,&originalHiliteRect);
    zeroedHiliteRect = originalHiliteRect;
    OffsetRect(&zeroedHiliteRect,-originalHiliteRect.left,-originalHiliteRect.top);

    GetGWorld(&savedPortPtr,&saveDeviceHdl);

    qdError = NewGWorld(&dragGWorldPtr,8,&zeroedHiliteRect,NULL,NULL,0);
    if(dragGWorldPtr != NULL && qdError == noErr)
    {
      SetGWorld(dragGWorldPtr,NULL);
      EraseRect(&zeroedHiliteRect);

      dragPixMapHdl = GetGWorldPixMap(dragGWorldPtr);
      LockPixels(dragPixMapHdl);
      windPixMapHdl = GetGWorldPixMap(savedPortPtr);

      CopyBits((BitMap *) *windPixMapHdl,(BitMap *) *dragPixMapHdl,
               &originalHiliteRect,&zeroedHiliteRect,srcCopy,NULL);

      UnlockPixels(dragPixMapHdl);
```

```
      SetGWorld(savedPortPtr,saveDeviceHdl);

      maskRgnHdl = NewRgn();
      if(maskRgnHdl != NULL)
      {
        CopyRgn(hiliteRgnHdl,maskRgnHdl);
        OffsetRgn(maskRgnHdl,-originalHiliteRect.left,-originalHiliteRect.top);

        SetPt(&offsetPoint,originalHiliteRect.left,originalHiliteRect.top);
        LocalToGlobal(&offsetPoint);

        SetDragImage(dragRef,dragPixMapHdl,maskRgnHdl,offsetPoint,kDragStandardTranslucency);
      }
    }
  }

  // ..... get drag region

  dragRgnHdl = NewRgn();
  if(dragRgnHdl == NULL)
    return MemError();

  CopyRgn(hiliteRgnHdl,dragRgnHdl);
  SetPt(&offsetPoint,0,0);
  LocalToGlobal(&offsetPoint);
  OffsetRgn(dragRgnHdl,offsetPoint.h,offsetPoint.v);

  tempRgnHdl = NewRgn();
  if(tempRgnHdl == NULL)
    return MemError();

  CopyRgn(dragRgnHdl,tempRgnHdl);
  InsetRgn(tempRgnHdl,1,1);
  DiffRgn(dragRgnHdl,tempRgnHdl,dragRgnHdl);
  DisposeRgn(tempRgnHdl);

  // ..... perform the drag

  osError = TrackDrag(dragRef,eventStrucPtr,dragRgnHdl);
  if(osError != noErr && osError != userCanceledErr)
    return osError;

  if(dragRef)       DisposeDrag(dragRef);
  if(maskRgnHdl)    DisposeRgn(maskRgnHdl);
  if(dragGWorldPtr) DisposeGWorld(dragGWorldPtr);
  if(dragRgnHdl)    DisposeRgn(dragRgnHdl);
  if(tempRgnHdl)    DisposeRgn(tempRgnHdl);

  return noErr;
}

// ********************************************************************** dragTrackingHandler

OSErr dragTrackingHandler(DragTrackingMessage trackingMessage,WindowRef windowRef,
                          void *handlerRefCon,DragRef dragRef)
{
  docStructurePointer docStrucPtr;
  DragAttributes      dragAttributes;
  UInt32              theTime;
  UInt16              numberOfDragItems, index;
  ItemReference       itemRef;
  OSErr               result;
  FlavorFlags         flavorFlags;
  Point               mousePt, localMousePt;
  RgnHandle           windowHiliteRgn;
  Rect                correctedViewRect;
  SInt16              theOffset;

  if((trackingMessage != kDragTrackingEnterHandler) && !gCanAcceptItems)
```

```
  return noErr;

docStrucPtr = (docStructurePointer) handlerRefCon;

GetDragAttributes(dragRef,&dragAttributes);
gSystemCaretTime = GetCaretTime();
theTime = TickCount();

switch(trackingMessage)
{
  // ..... enter handler

  case kDragTrackingEnterHandler:
    gCanAcceptItems = true;

    CountDragItems(dragRef,&numberOfDragItems);

    for(index=1;index <= numberOfDragItems;index++)
    {
      GetDragItemReferenceNumber(dragRef,index,&itemRef);
      result = GetFlavorFlags(dragRef,itemRef,'TEXT',&flavorFlags);
      if(result != noErr)
      {
        gCanAcceptItems = false;
        break;
      }
    }
    break;

    // ..... enter window

  case kDragTrackingEnterWindow:
    gCaretStartTime  = theTime;
    gCaretOffset     = gLastOffset = -1;
    gCaretShowFlag   = true;
    gCursorInContent = false;
    break;

  // ..... in window

  case kDragTrackingInWindow:

    GetDragMouse(dragRef,&mousePt,NULL);
    localMousePt = mousePt;
    GlobalToLocal(&localMousePt);

    if(dragAttributes & kDragHasLeftSenderWindow)
    {
      if(PtInRect(localMousePt,&(**(docStrucPtr->textEditStrucHdl)).viewRect))
      {
        if(!gCursorInContent)
        {
          windowHiliteRgn = NewRgn();
          correctedViewRect = (**(docStrucPtr->textEditStrucHdl)).viewRect;
          InsetRect(&correctedViewRect,-2,-2);
          RectRgn(windowHiliteRgn,&correctedViewRect);
          ShowDragHilite(dragRef,windowHiliteRgn,true);
          DisposeRgn(windowHiliteRgn);
        }
        gCursorInContent = true;
      }
      else
      {
        if(gCursorInContent)
          HideDragHilite(dragRef);
        gCursorInContent = false;
      }
    }
```

```
  // … … … … … … … … … … … … start caret drawing stuff, first get the offset into the text

  theOffset = doGetOffset(mousePt,docStrucPtr->textEditStrucHdl);

  // … … … … … … … … … … … … … … … …  if in sender window, defeat caret drawing in selection

  if(dragAttributes & kDragInsideSenderWindow)
  {
    if((theOffset >= (*(docStrucPtr->textEditStrucHdl))->selStart) &&
       (theOffset <= (*(docStrucPtr->textEditStrucHdl))->selEnd))
    {
      theOffset = -1;
    }
  }

  // … … … … … … … … … … … … … … … save the offset to a global for use by dragReceiveHandler

  gInsertPosition = theOffset;

  // … … … … … … … … … … … … … … … … … if offset has changed, reset the caret flashing timer

  if(theOffset != gLastOffset)
  {
    gCaretStartTime = theTime;
    gCaretShowFlag = true;
  }

  gLastOffset = theOffset;

  // … … … … if caret-flashing interval has elapsed, toggle caret "show" flag, reset timer

  if(theTime - gCaretStartTime > gSystemCaretTime)
  {
    gCaretShowFlag = !gCaretShowFlag;
    gCaretStartTime = theTime;
  }

  // … … … … … … … … … … if caret "show" flag is off, set variable to defeat caret drawing

  if(!gCaretShowFlag)
    theOffset = -1;

  // … … … … if offset has changed, erase previous caret, draw new caret at current offset

  if(theOffset != gCaretOffset)
  {
    // … … … … … … … … if first pass this window, don't erase, otherwise erase at old offset

    if(gCaretOffset != -1)
      doDrawCaret(gCaretOffset,docStrucPtr->textEditStrucHdl);

    // … … … … … … … …  … … …  … … …  if "show" flag says show, draw caret at current offset

    if(theOffset != -1)
      doDrawCaret(theOffset,docStrucPtr->textEditStrucHdl);
  }

  gCaretOffset = theOffset;

  break;

// …… leave window

case kDragTrackingLeaveWindow:

  if(gCaretOffset != -1)
  {
    doDrawCaret(gCaretOffset,docStrucPtr->textEditStrucHdl);
    gCaretOffset = -1;
```

```
      }

      if(gCursorInContent && dragAttributes & kDragHasLeftSenderWindow)
        HideDragHilite(dragRef);

      break;

    // ..... leave handler

    case kDragTrackingLeaveHandler:
      break;
  }

  return noErr;
}

// ********************************************************************************* doGetOffset

SInt16  doGetOffset(Point mousePt,TEHandle textEditStrucHdl)
{
  WindowRef windowRef;
  SInt16    theOffset;
  Point     thePoint;

  theOffset = -1;

  if(FindWindow(mousePt,&windowRef) == inContent)
  {
    SetPortWindowPort(windowRef);
    GlobalToLocal(&mousePt);

    if(PtInRect(mousePt,&((*textEditStrucHdl)->viewRect)))
    {
      theOffset = TEGetOffset(mousePt,textEditStrucHdl);
      thePoint  = TEGetPoint(theOffset - 1,textEditStrucHdl);

      if((theOffset) &&
         (doIsOffsetAtLineStart(theOffset,textEditStrucHdl)) &&
         ((*(*textEditStrucHdl)->hText)[theOffset - 1] != 0x0D) &&
         (thePoint.h < mousePt.h))
      {
        theOffset--;
      }
    }
  }

  return theOffset;
}

// ********************************************************************** doIsOffsetAtLineStart

SInt16  doIsOffsetAtLineStart(SInt16 offset,TEHandle textEditStrucHdl)
{
  SInt16 line = 0;

  if((*textEditStrucHdl)->teLength == 0)
    return(true);

  if(offset >= (*textEditStrucHdl)->teLength)
    return((*((*textEditStrucHdl)->hText))[(*textEditStrucHdl)->teLength - 1] == 0x0D);

  while((*textEditStrucHdl)->lineStarts[line] < offset)
    line++;

  return ((*textEditStrucHdl)->lineStarts[line] == offset);
}

// ********************************************************************************* doDrawCaret
```

```
void  doDrawCaret(SInt16 theOffset,TEHandle textEditStrucHdl)
{
  Point  thePoint;
  SInt16 theLine, lineHeight;

  thePoint = TEGetPoint(theOffset,textEditStrucHdl);
  theLine  = doGetLine(theOffset,textEditStrucHdl);

  if((theOffset == (*textEditStrucHdl)->teLength) &&
      (*((*textEditStrucHdl)->hText))[(*textEditStrucHdl)->teLength - 1] == 0x0D)
  {
    thePoint.v += TEGetHeight(theLine,theLine,textEditStrucHdl);
  }

  PenMode(patXor);
  lineHeight = TEGetHeight(theLine,theLine,textEditStrucHdl);
  MoveTo(thePoint.h - 1,thePoint.v - 1);
  Line(0,1 - lineHeight);

  PenNormal();
}

// ********************************************************************************** doGetLine

SInt16  doGetLine(SInt16 theOffset,TEHandle textEditStrucHdl)
{
  SInt16 theLine = 0;

  if(theOffset > (*textEditStrucHdl)->teLength)
    return ((*textEditStrucHdl)->nLines);

  while((*textEditStrucHdl)->lineStarts[theLine] < theOffset)
    theLine++;

  return theLine;
}

// ********************************************************************************************
// ReceiveAndUndoDrag.c
// ********************************************************************************************

// ..... includes

#include "Drag.h"

// ..... global variables

extern Boolean gEnableDragUndoRedoItem;
extern Boolean gUndoFlag;
extern Boolean gCanAcceptItems;
extern SInt16  gInsertPosition, gCaretOffset;

// ************************************************************************ dragReceiveHandler

OSErr  dragReceiveHandler(WindowRef windowRef,void *handlerRefCon,DragRef dragRef)
{
  docStructurePointer docStrucPtr;
  TEHandle            textEditStrucHdl;
  SInt32              totalTextStart;
  Size                totalTextSize;
  Boolean             wasActive, moveText, gotUndoMemory = false;
  DragAttributes      dragAttributes;
  SInt16              mouseDownModifiers, mouseUpModifiers, selStart, selEnd;
  UInt16              numberOfDragItems, index;
  ItemReference       itemReference;
  OSErr               osError;
  Size                textSize;
  Ptr                 textDataPtr;
  SInt32              additionalChars;
```

```
if((!gCanAcceptItems) || (gInsertPosition == -1))
  return dragNotAcceptedErr;

docStrucPtr = (docStructurePointer) handlerRefCon;
textEditStrucHdl = docStrucPtr->textEditStrucHdl;

// … set graphics port to this window's port and, if necessary, activate text edit structure

SetPortWindowPort(windowRef);

wasActive = (*textEditStrucHdl)->active != 0;
if(!wasActive)
  TEActivate(textEditStrucHdl);

// …… get drag attributes and keyboard modifiers

GetDragAttributes(dragRef,&dragAttributes);
GetDragModifiers(dragRef,0L,&mouseDownModifiers,&mouseUpModifiers);

// … … …  in case their are multiple items, save first insertion point for later TESetSelect

totalTextStart = gInsertPosition;
totalTextSize = 0;

// … … for all items in drag, get 'TEXT' data, insert into this window's text edit structure

CountDragItems(dragRef,&numberOfDragItems);

for(index=1;index <= numberOfDragItems;index++)
{
  GetDragItemReferenceNumber(dragRef,index,&itemReference);

  osError = GetFlavorDataSize(dragRef,itemReference,'TEXT',&textSize);
  if(osError == noErr)
  {
    // …  if addition of drag to the text edit structure would exceed TextEdit limit, return

    if(((*textEditStrucHdl)->teLength + textSize) > kMaxTELength)
      return dragNotAcceptedErr;

    // … … … … … … … … … … … … … … create nonrelocatable block and get the 'TEXT' data into it

    textDataPtr = NewPtr(textSize);
    if(textDataPtr == NULL)
      return dragNotAcceptedErr;

    GetFlavorData(dragRef,itemReference,'TEXT',textDataPtr,&textSize,0);

    // … … … … … … … … … … … … … … … … … …  if caret or highlighting is on screen, remove it

    if(gCaretOffset != -1)
    {
      doDrawCaret(gCaretOffset,textEditStrucHdl);
      gCaretOffset = -1;
    }

    if(dragAttributes & kDragHasLeftSenderWindow)
      HideDragHilite(dragRef);

    // save current text and selection start/end for Undo, and set Redo/Undo menu item flags

    if(dragAttributes & kDragInsideSenderWindow)
    {
      gotUndoMemory = doSavePreInsertionText(docStrucPtr);
      if(gotUndoMemory)
      {
        gEnableDragUndoRedoItem = true;
        gUndoFlag = true;
```

```
      }
    }
    else
      gEnableDragUndoRedoItem = false;

    // … … … … … if in sender window, ensure selected text is deleted if option key not down

    moveText = (dragAttributes & kDragInsideSenderWindow) &&
               (!((mouseDownModifiers & optionKey) | (mouseUpModifiers & optionKey)));

    if(moveText)
    {
      selStart = (*textEditStrucHdl)->selStart;
      selEnd   = (*textEditStrucHdl)->selEnd;

      // … … … … …  extend selection by one chara if space charas just before and just after

      if(doIsWhiteSpaceAtOffset(selStart - 1,textEditStrucHdl) &&
        !doIsWhiteSpaceAtOffset(selStart,textEditStrucHdl) &&
        !doIsWhiteSpaceAtOffset(selEnd - 1,textEditStrucHdl) &&
         doIsWhiteSpaceAtOffset(selEnd,textEditStrucHdl))
      {
        if(doGetCharAtOffset(selEnd,textEditStrucHdl) == ' ')
          (*textEditStrucHdl)->selEnd++;
      }

      // if insertion is after selected text, move insertion point back by size of selection

      if(gInsertPosition > selStart)
      {
        selEnd = (*textEditStrucHdl)->selEnd;
        gInsertPosition -= (selEnd - selStart);
        totalTextStart -= (selEnd - selStart);
      }

      // … … … …  … … … … … … … … … … … … … … … … … … … … … … … … … … delete the selection

      TEDelete(textEditStrucHdl);
    }

    // … … … … … … … … … … … … … … … … … … … … … … insert the 'TEXT' data at the insertion point

    additionalChars = doInsertTextAtOffset(gInsertPosition,textDataPtr,textSize,
                                           textEditStrucHdl);

    // … … … … … if inserting multiple blocks of text, update insertion point for next block

    gInsertPosition += textSize + additionalChars;
    totalTextSize += textSize + additionalChars;

    // … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … … dispose of nonrelocatable block

    DisposePtr(textDataPtr);
  }
}

// ………………………………………………………………………………… select total inserted text and adjust scrollbar

TESetSelect(totalTextStart,totalTextStart + totalTextSize,textEditStrucHdl);
doAdjustScrollbar(windowRef);

// ……………… set window's "touched" flag, and save post-insert selection start and end for Redo

docStrucPtr->windowTouched = true;

if(dragAttributes & kDragInsideSenderWindow)
{
  docStrucPtr->postDropSelStart = totalTextStart;
  docStrucPtr->postDropSelEnd = totalTextStart + totalTextSize;
```

```
  }

  // ……..…… if text edit structure had to be activated earlier, deactivate it

  if(!wasActive)
    TEDeactivate(textEditStrucHdl);

  return noErr;
}

// ********************************************************************* doIsWhiteSpaceAtOffset

Boolean  doIsWhiteSpaceAtOffset(SInt16 offset,TEHandle textEditStrucHdl)
{
  char theChar;

  if((offset < 0) || (offset > (*textEditStrucHdl)->teLength - 1))
    return true;

  theChar = ((char *) *((*textEditStrucHdl)->hText))[offset];

  return (doIsWhiteSpace(theChar));
}

// ************************************************************************** doIsWhiteSpace

Boolean  doIsWhiteSpace(char theChar)
{
  return ((theChar == ' ') || (theChar == 0x0D));
}

// ************************************************************************ doGetCharAtOffset

char  doGetCharAtOffset(SInt16 offset,TEHandle textEditStrucHdl)
{
  if(offset < 0)
    return 0x0D;

  return (((char *) *((*textEditStrucHdl)->hText))[offset]);
}

// ********************************************************************** doInsertTextAtOffset

SInt16  doInsertTextAtOffset(SInt16 textOffset,Ptr textDataPtr,SInt32 textSize,
                             TEHandle textEditStrucHdl)
{
  SInt16 charactersAdded = 0;

  if(textSize == 0)
    return charactersAdded;

  // …… if inserting at end of word, and selection does not begin with a space, insert a space

  if(!doIsWhiteSpaceAtOffset(textOffset - 1,textEditStrucHdl) &&
       doIsWhiteSpaceAtOffset(textOffset,textEditStrucHdl) &&
     !doIsWhiteSpace(textDataPtr[0]))
  {
    TESetSelect(textOffset,textOffset,textEditStrucHdl);
    TEKey(' ',textEditStrucHdl);
    ++textOffset;
    ++charactersAdded;
  }

  // … if inserting at beginning of word and selection does not end with a space, insert space

  if(doIsWhiteSpaceAtOffset(textOffset - 1,textEditStrucHdl) &&
    !doIsWhiteSpaceAtOffset(textOffset,textEditStrucHdl) &&
    !doIsWhiteSpace(textDataPtr[textSize - 1]))
  {
```

```
    TESetSelect(textOffset,textOffset,textEditStrucHdl);
    TEKey(' ',textEditStrucHdl);
    ++charactersAdded;
  }

  // ..... before inserting, set selection range to a zero

  TESetSelect(textOffset,textOffset,textEditStrucHdl);
  TEInsert(textDataPtr,textSize,textEditStrucHdl);

  return charactersAdded;
}

// ********************************************************************* doSavePreInsertionText

Boolean  doSavePreInsertionText(docStructurePointer  docStrucPtr)
{
  OSErr  osError;
  Size   tempSize;
  Handle tempTextHdl;

  if(docStrucPtr->preDragText == NULL)
    docStrucPtr->preDragText = NewHandle(0);

  tempTextHdl = (*(docStrucPtr->textEditStrucHdl))->hText;
  tempSize = GetHandleSize(tempTextHdl);
  SetHandleSize(docStrucPtr->preDragText,tempSize);
  osError = MemError();
  if(osError != noErr)
  {
    doErrorAlert(eDragUndo);
    return false;
  }

  BlockMove(*tempTextHdl,*(docStrucPtr->preDragText),tempSize);

  docStrucPtr->preDragSelStart = (*((docStrucPtr)->textEditStrucHdl))->selStart;
  docStrucPtr->preDragSelEnd   = (*((docStrucPtr)->textEditStrucHdl))->selEnd;

  return true;
}

// ************************************************************************** doUndoRedoDrag

void  doUndoRedoDrag(WindowRef windowRef)
{
  docStructurePointer docStrucPtr;
  Handle              tempTextHdl;
  Rect                portRect;

  docStrucPtr = (docStructurePointer) (GetWRefCon(windowRef));

  tempTextHdl = (*(docStrucPtr->textEditStrucHdl))->hText;
  (*(docStrucPtr->textEditStrucHdl))->hText = docStrucPtr->preDragText;
  docStrucPtr->preDragText = tempTextHdl;

  if(gUndoFlag)
  {
    (*((docStrucPtr)->textEditStrucHdl))->selStart = docStrucPtr->preDragSelStart;
    (*((docStrucPtr)->textEditStrucHdl))->selEnd = docStrucPtr->preDragSelEnd;
  }
  else
  {
    (*((docStrucPtr)->textEditStrucHdl))->selStart = docStrucPtr->postDropSelStart;
    (*((docStrucPtr)->textEditStrucHdl))->selEnd = docStrucPtr->postDropSelEnd;
  }

  gUndoFlag = !gUndoFlag;
```

```
  TECalText(docStrucPtr->textEditStrucHdl);
  GetWindowPortBounds(windowRef,&portRect);
  InvalWindowRect(windowRef,&portRect);
}

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

Demonstration Program Drag Comments

When this program is run, the user should open the document "Drag Document" and drag selections to other locations within the document, to other demonstration program windows, to the windows of other applications that accept 'TEXT' format data, and to the desktop and Finder windows (to create text clippings). The user should also drag text from the windows of other applications to the demonstration program's windows.

The user should note the following:

- The highlighting of the demonstration program's windows (and the window's proxy icon) when items containing data of the 'TEXT' flavour are dragged over them.
- The movement of the insertion point caret with the cursor when items are dragged within the demonstration program's windows, and the "hiding" of the caret when the drag originates in a demonstration program window and the cursor is moved over the selection.
- When dragging and dropping within the demonstration program's windows:
  - The program's implementation of "smart drag and drop" (For example, if there is a space character immediately to the left and right of the selection, the deletion is extended to include the second space character, thus leaving a single space character between the two words which previously bracketed the selection.)
  - The availability and effect of the Undo/Redo item in the Edit menu.

The non-drag and drop aspects of this program are based on the demonstration program MonoTextEdit (Chapter 21), and the contents of Drag.h and Drag.c are very similar to the contents of MonoTextEdit.c. Accordingly, comments on the content of Drag.h and Drag.c are restricted to those areas where modifications have been made to the code contained in MonoTextEdit.c.

Drag.h

defines

Three additional constants are established for drag and drop errors.

typedefs

The docStructure data type has been extended to include fields to store the owning window's window reference, a Boolean which is set to true when the contents of the window have been modified, and five fields to support drag and drop undo/redo.

Drag.c

Global Variables

dragTrackingHandlerUPP and dragTrackingReceiverUPP will be assigned universal procedure pointers to the tracking and receive handlers. gEnableDragUndoRedoItem and gUndoFlag will be used to control enabling/disabling of the Drag and Drop Undo/Redo item in the Edit menu.

main

dragTrackingHandlerUPP and dragTrackingReceiverUPP are assigned universal procedure pointers to the tracking and receive handlers.

doKeyEvent

The global variable gEnableDragUndoRedoItem is set to false. This causes the Drag and Drop Undo/Redo item in the Edit menu to be disabled.

doInContent

If the mouse-down was within the TextEdit view rectangle, TEGetHiliteRgn is called to attempt to get the highlight region. If there is a highlight region (that is, a selection), and if the mouse-down was within that region, WaitMouseMoved is called. WaitMouseMoved waits for either the mouse to move from the given initial mouse location or for the mouse button to be released. If the mouse moves away from the initial mouse location before the mouse button is released, WaitMouseMoved returns true, in which case doStartDrag is called.

doNewDocWindow

A nonrelocatable block is created for the window's document structure.

SetWindowProxyCreatorAndType is called with 0 passed in the fileCreator parameter and 'TEXT' passed in the fileType parameter to cause the system's default icon for a document file to be displayed as the proxy icon. In this program, the proxy icon is used solely for the purpose of demonstrating proxy icon highlighting when ShowDragHilite tis called to indicate that the window is a valid drag-and-drop target.

After gNumberOfWindows is incremented, four of the fields of the document structure are initialised.

Following the call to TEFeatureFlag, the drag tracking and receive handlers are installed on the window. (Note that the pointer to the window's document structure is passed in the handlerRefCon parameter of the installer functions.) If either installation is unsuccessful, the window and document structure are disposed of, the tracking handler also being removed in the case of a failure to install the receive handler.

doAdjustMenus

When the global variable gEnableDragUndoRedoItem is set to true, the Drag and Drop Undo/Redo item is enabled, otherwise it is disabled. If the item is enabled, the global variable gUndoFlag controls the item text, setting it to either Undo or Redo.

doEditMenu

If the Drag and Drop Undo/Redo item is chosen from the Edit menu, doUndoRedoDrag is called.

doAdjustCursor

After the first call to DiffRgn (which establishes the arrow and IBeam regions), a pointer to the window's document structure is retrieved. This allows the handle to the window's TextEdit structure to be passed in a call to TEGetHiliteRgn. The region returned by TEGetHiliteRgn is in local coordinates, so the next three lines change it to global coordinates preparatory to a call to DiffRgn. The DiffRgn call, in effect, cuts the equivalent of the highlight region out of the IBeam region.

If the location of the mouse (returned by the call to GetGlobalMouse) is within the highlight region, the cursor is set to the arrow shape.

doCloseWindow

If the preDragText field of the window's document structure does not contain NULL, DisposeHandle is called to release memory assigned in support of drag and drop redo/undo.

The calls to RemoveTrackingHandler and RemoveReceiveHandler remove the tracking and receive handlers before the window is disposed of.

StartAndTrackDrag.c

doStartDrag

The call to NewDrag allocates a new drag object.

The call to AddDragItemFlavor creates a drag item and adds a data flavour (specifically 'TEXT') to that item. Note that the item reference number passed is 1. If additional flavours were to be added to the item, this same item reference number would be passed in the additional calls to AddDragItemFlavor.

The next block gets and sets the drag image for translucent dragging, but executes only if the program is running on Mac OS 8/9. GetRegionBounds gets the bounding rectangle of the highlight region and OffsetRect adjusts the coordinates so that top left is 0,0. The call to NewGWorld creates an 8-bit deep offscreen graphics world the same size as this rectangle. CopyBits is then called to copy the highlight region area to the offscreen graphics world. The calls to CopyRgn and OffsetRgn create a region the same shape as the highlight region and adjusted so that the top left of the bounding rectangle is 0,0. The next two lines establish the offset point required by the following call to SetDragImage. This offset is required to move the pixel map in the offscreen graphics world to the global coordinates where the drag image is to initially appear. Finally, the handles to the offscreen graphics world and mask, the offset, and a constant specifying the required level of translucency are passed in the call to SetDragImage, which associates the image with the drag reference.

Any errors which might occur in setting up the translucent drag are not critical on Mac OS 8/9 because the next block creates the drag region for the alternative visual representation (a grey outline). On Mac OS X, this is the block that creates the grey outline. The received highlight region (which is in local coordinates) is copied to dragRgnHdl, which is then converted to global coordinates. This region, in turn, is copied to tempRgnHdl, which is then inset by one pixel. The call to DiffRgn subtracts the inset

region from dragRgnHdl, leaving the latter with the same outline as the highlight region but only one pixel thick. The newly defined drag region is passed in the theRegion parameter of the call to TrackDrag.

TrackDrag performs the drag. During the drag, the Drag Manager follows the cursor on the screen with the translucent image or dithered 50% grey pattern drag feedback (Mac OS 8/9) or grey outline (Mac OS X) and sends tracking messages to applications that have registered drag tracking handlers. When the user releases the mouse button, the Drag Manager calls any receive drop handlers that have been registered on the destination window.

The TrackDrag function returns noErr in situations where the user selected a destination for the drag and the destination received data from the Drag Manager. If the user drops over a non-aware application or the receiver does not accept any data from the Drag Manager, the Drag Manager automatically provides a "zoom back" animation and returns userCanceledErr. Thus the first return will execute only if an error other than userCanceledErr was returned.

dragTrackingHandler

dragTrackingHandler is the drag tracking handler.

Firstly, if the message received is not the enter handler message, and if it was determined at the time of receipt of the enter handler message that the drop cannot be accepted, the function returns immediately.

The pointer received in the handlerRefCon formal parameter is cast to a pointer to the window's document structure so that certain fields in this structure can later be accessed.

GetDragAttributes gets the current set of drag attribute flags. GetCaretTime gets the insertion point caret blink interval. TickCount gets the current system tick count. These latter two values will be used by the caret drawing code.

enter handler

Within the switch, the "enter handler" message is processed at the first case.

CountDragItems returns the number of items in the drag. Then, for each of the items in the drag, GetDragItemReferenceNumber is called to retrieve the item reference number, allowing the call to GetFlavorFlags to determine whether the item contains data of the 'TEXT' flavour. (GetFlavorFlags will return an error if the flavour does not exist.) gCanAcceptItems is assigned false if any item does not contain 'TEXT' data, meaning that the drag will only be accepted if all items contain data of the 'TEXT' flavour.

enter window

If the message is the "enter window" message, several global variables are initialised. These globals will be used when the "in window" message is received to control insertion point caret drawing and window highlighting.

in window

Each time the "in window" message is received, GetDragMouse is called to get the current mouse location in global coordinates. The location is copied to a local Point variable, which is then converted from global to local coordinates.

The first if block executes if the drag has left the sender window. If the mouse cursor is within the window's TextEdit view rectangle, and if gCursorInContent has previously been set to false (recall that it is set to false at the "enter window" message), ShowDragHilite is called to draw the standard drag and drop highlight around the view rectangle. gCursorInContent is then set to true to defeat further ShowDragHilite calls while the drag remains in this window.

If the mouse cursor is not within the window's TextEdit view rectangle, and if gCursorContent has previously been set to true (causing a highlight draw), HideDragHilite is called to remove the highlighting, and gCursorInContent is set to false again so that ShowDragHilite is called if the drag moves back inside a view rectangle again.

With window highlighting attended to, the next task is to perform insertion point caret drawing.

The function doGetOffset takes the mouse location in global coordinates and the window's TextEdit structure handle and returns the offset into the text corresponding to the mouse location. (Note that doGetOffset will return -1 if the cursor is not within the content region and the view rectangle of the window under the cursor. Note also that, if the cursor is within the content region of the window under the cursor, doGetOffset sets that window's graphics port as the current port.)

if the drag is currently inside the sender window, and the offset returned by doGetOffset indicates that the cursor is within the TextEdit selection, theOffset is set to -1. As will be seen, this defeats the drawing of the caret when the cursor is within the selection.

The offset returned by doGetOffset is then saved to a global variable. As will be seen, the value assigned to this variable the last time the "in window" case executes (when the user releases the mouse button) will be used by the receive handler dragReceiveHandler.

If the offset has changed since the last time the "in window" case executed, the caret flashing timer is reset to the time the handler was entered this time around and the caret show/hide flag is set to "show". The current offset is then assigned to the global gLastOffset preparatory to the execution of the "in window" case.

If the caret flashing interval has elapsed, the show/hide flag is toggled and the caret flashing timer is reset.

If the caret show/hide flag indicates that the caret should be "hidden", theOffset will be set to -1 to defeat caret drawing.

If the offset has changed since the last execution of the "in window" case, the function for drawing/erasing the caret is called in certain circumstances. Firstly, if this is not the first execution of the "in window" case since entering the window, doDrawCaret is called to erase the caret previously drawn at the old offset. Secondly, if the show/hide flag indicates that the caret should be drawn, doDrawCaret is called to draw the caret at the current offset.

The global which stores the old offset is assigned the current offset before the case exits.

leave window

When the "leave window" message is received, if the caret is on the screen, it is erased. If the window highlighting has previously been drawn, HideDragHilite is called to remove the highlighting.

doGetOffset

doGetOffset is called by dragTrackingHandler to returns the offset into the text corresponding to the specified mouse location. doGetOffset also sets the graphics port to the port associated with the window under the cursor.

If the part code returned by FindWindow indicates that the cursor is within the content region of a window, that window's graphics port is set as the current port and the cursor location is converted to local coordinates preparatory to the call to PtInRect.

If the cursor is within the view rectangle of the TextEdit structure associated with the window, TEGetOffset is called to get the offset into the text corresponding to the cursor location and TEGetPoint is called to get the local coordinates of the character immediately before the offset.

If the offset is at a TextEdit line start, and the character immediately before the offset is not a carriage return, and the horizontal coordinate of character immediately before the offset is less than the horizontal coordinate of the cursor, theOffset is decremented by one. In the situation where the cursor is dragged to right of the rightmost character of a line, this will cause the caret to continue to be drawn immediately to the right of that character instead of "jumping" to the beginning of the next line.

isOffsetAtLineStart

isOffsetAtLineStart is called by doGetOffset. It returns true if the specified offset is at a TextEdit line start.

doDrawCaret

doDrawCaret is called by dragTrackingHandler to draw and erase the caret. The transfer mode is et to patXor so that two successive calls will, in effect, draw and erase the image.

The call to TEGetPoint gets the coordinates of the bottom left of the character at the specified offset. The call to doGetLine returns the TextEdit line number that contains the specified offset.

The next block accommodates a quirk of TextEdit. For some reason, TextEdit does not return the proper coordinates of the last offset in the field if the last character in the record is a carriage return. TEGetPoint returns a point that is one line higher than expected. This block fixes the problem.

Following the call to PenMode, TEGetHeight is called to get the height of a single line of text. A line is then drawn from a point one pixel to the left and above the bottom left of the character at the offset

to a point vertically above the starting point. The length of the line is one pixel less that the line height.

doGetLine

doGetLine is called by doDrawCaret. It returns the TextEdit line number that contains the specified offset.

ReceiveAndUndoDrag.c

dragReceiveHandler

dragReceiveHandler is the drag receive handler.

Recall that dragTrackingHandler sets gCanAcceptItems to true if all items in the drag contained data of the 'TEXT' flavour. Recall also that dragTrackingHandler sets gInsertPosition to -1 if the cursor is over the selection. Thus, if at least one item does not contain data of the 'TEXT' flavour, or if the cursor is over the selection at the time the mouse button is released, the function exits and dragNotAcceptedErr is returned to the Drag Manager. dragNotAcceptedErr causes the Drag Manager to execute a "zoomback" animation of the drag region to the source location.

The pointer received in the handlerRefCon formal parameter is cast to a pointer to the window's document structure. The handle to the TextEdit structure associated with the window is then retrieved from the document structure.

SetPortWindowPort sets the window's graphics port as the current port. The next line saves whether the TextEdit structure is currently active or inactive so that that state can later be restored. If the TextEdit structure is currently not active, it is made active.

GetDragAttributes and GetDragModifiers are called to get the drag's attributes and the modifier keys that were pressed at mouse-down and mouse-up time.

If there are multiple items in the drag, the initial insertion point, which is set in the drag tracking handler, is saved to a local variable for later use by TESetSelect. (If there are multiple items to insert, the offset in gInsertPosition will be updated each time main if block executes.)

CountDragItems returns the number of items in the drag and the main for loop executes for each item. Within the loop, the first call (to GetDragItemReferenceNumber) retrieves the item's item reference number, which is passed in the call to GetFlavorDataSize to get the size of the 'TEXT' data. If GetFlavorDataSize does not return an error, the following occurs:

- The data size returned by GetFlavorDataSize is added to the current size of the text in the TextEdit structure. If adding the 'TEXT' data to the current text would exceed the TextEdit limit, the handler exits, returning dragNotAcceptedErr to the Drag Manager.

- A nonrelocatable block the size of the 'TEXT' data is created and GetFlavorData is called to get the 'TEXT' data into that block.

- If the insertion point caret is on the screen, it is removed. (Recall that the global variables used here are set in the drag tracking handler.) If the window is currently highlighted, the highlighting is removed.

- If the drop is within the sender window, doSavePreInsertionText is called to save the current TextEdit text and the current selection start and end. This is to support drag undo/redo. If doSavePreInsertionText returns true, flags are set to cause the Undo/Redo item in the Edit menu to be enabled and to cause the initial item text to be set to "Undo".

- If there are multiple items in the drag, the initial insertion point is saved for later use by TESetSelect. (If there are multiple items to insert, the offset in gInsertPosition will be updated each time around

- The variable moveText is assigned true if the drop is inside the sender window and the option key was not down when the mouse button went down or was released. If moveText is true, the current selection must be deleted, so the if block executes.

  - Firstly, the current selection start and end are saved to two local variables.

  - The next block implements "smart drag and drop" If the character just before selection start is a space or CR character, and if the first character in the selection is not such a character, and if the last character in the selection is not such a character, and if the character just after the

selection is such a character, then, if the character just after the selection is a space character, the current end of the selection is extended to include that space character. This means that if, for example, just the characters of a word are currently selected, the space character immediately after the selection is added to the selection so that only one space character will remain between the words which bracket the selection when the selection is deleted by TEDelete.

- If the current drop insertion offset is after the selection start offset, the local variable holding the selection end offset is updated (it may have been increased by the "smart drag and drop" code), and gInsertPosition and totalTextStart offsets are moved back by the length of the selection.

- TEDelete then deletes the selected text (perhaps extended by one by the "smart drag and drop" code) from the TextEdit structure and redraws the text.

- The doInsertTextAtOffset is called to insert the 'TEXT' data at the current insertion point. This function implements another aspect of "smart drag and drop" which can possibly add additional characters to the TextEdit structure. The number of additional characters added (if any) is returned by the function.

- If the main if block executes again (meaning that there are multiple items in the drag), gInsertPosition must be updated to the offset for the next insertion. This is achieved by adding the size of the last insertion, plus the number of any additional characters added by doInsertTextAtOffset to the current value in gInsertPosition. The value in totalTextSize is increased by the same amount.

When all items have been inserted the main if block exits. TESetSelect is then called to set the selection range to the total inserted text. This call unhighlights the previous selection and highlights the new selection range. doAdjustScrollbar is called to adjust the scroll bars.

The windowTouched field in the document structure is set to true to record the fact that the contents of the window have been modified. The post-insertion selection start and selection end offsets are saved to the relevant fields of the document structure for possible use in the Undo/Redo function.

Finally, if the TextEdit structure was not active when dragReceiveHandler was entered, it is restored to that state.

isWhiteSpaceAtOffset

isWhiteSpaceAtOffset is called by dragReceiveHandler and doInsertTextAtOffset. Given an offset into a TextEdit structure, it determines if the character at that offset is a "white space: character, that is, a space character or a carriage return character.

isWhiteSpace

isWhiteSpace is called by isWhiteSpaceAtOffset and doInsertTextAtOffset. Given a character, it returns true if that character is either a space character or a carriage return character.

doGetCharAtOffset

doGetCharAtOffset is called by dragReceiveHandler. Given an offset and a handle to a TextEdit structure, it returns the character at that offset.

doInsertTextAtOffset

doInsertTextAtOffset is called by dragReceiveHandler to insert the drag text at the specified offset. In addition to inserting the text, dragReceiveHandler implements additional aspects of "smart drag and drop", returning the number of space characters, if any, added to the TextEdit structure text by this process.

If there is no text in the buffer, the function simply returns.

If the character to the left of the insertion offset is not a space, and if the character at the offset is a space, the insertion is at the end of a word. If, in addition, the first character in the drag is not a space, TESetSelect is called to set collapse the selection range to an insertion point at the received offset, TEKey is called to insert a space character in the TextEdit structure at that offset, the offset is incremented by one to accommodate that added character, and the variable which keeps track of the number of added characters is incremented.

The next block is similar, except that it inserts a space character if the insertion is at the beginning of a word and the text to be inserted does not end with a space character. Also, in this block, the offset is not incremented.

With the "smart drag and drop" segment completed, TESetSelect is called to ensure that the selection range is collapsed to an insertion point at the (possibly incremented) offset, and TEInsert is called to insert the text into the TextEdit structure at that insertion point.

doSavePreInsertionText

doSavePreInsertionText is called by dragReceiveHandler to save the document's pre-drop text and selection in support of drag and drop undo/redo.

If the preDragText field of the window's document structure cuurently contains NULL, a new empty relocatable block is created and its handle assigned to the preDragText field.

The next block copies the handle to the TextEdit structure's text to a local variable, gets the size TextEdit structure's text, and expands the newly-created block to that size.

BlockMove is then called to copy the TextEdit structure's text to the newly-created block. In addition, the current (pre-drop) selection start and end are saved to the window's document structure.

doUndoRedoDrag

doUndoRedoDrag is called by doEditMenu in the event of the user choosing the Drag and Drop Undo/Redo item in the Edit menu. That item will only be enabled when the user has released the mouse button and the drop is within the sender window.

The first block means that, each time this function is called, the text block whose handle is assigned to the TextEdit structure will toggle between the pre-drop text and the post-drop text. The first time this function is called, the item text in the Edit Menu will be "Drag and Drop Undo" and the TextEdit structure will be assigned the handle to pre-drop text saved in the
document structure's preDragText field. The next time the function is called, the item text in the Edit menu will be "Drag and Drop Redo" and the TextEdit structure will be assigned the handle to post-drop text, and so on.

The global variable gUndoFlag is toggled by this function. At the next block, and depending on the value in gUndoFlag, either the pre-drop or post-drop selection start and end offsets are assigned to the TextEdit structure, as appropriate. (gUndoFlag also controls the text in the Drag and Drop Undo/Redo item in the Edit menu. See doAdjustMenus.)

With the appropriate text and selection assigned to the window's TextEdit structure, TECalText is called to wrap the text to the width of the view rectangle and re-calculate the line-starts. Finally, InvalWindowRect is called to force a call to TEUpdate (in the doUpdate function) to redraw the text.