Demonstration Program Scrap Listing

```
// ****************************************************************************************
// Scrap.c                                                                  CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program utilises Carbon Scrap Manager functions to allow the user to:
//
// •  Cut, copy, clear, and paste text and pictures from and to two document windows opened by
//    the program.
//
// •  Paste text and pictures cut or copied from another application to the two document
//    windows.
//
// •  Open and close a Clipboard window, in which the current contents of the scrap are
//    displayed.
//
// The program's preferred scrap flavour type is 'TEXT'.  Thus, if the scrap contains data in
// both the 'TEXT' and 'PICT' flavour types, only the 'TEXT' flavour will be used for pastes
// to the document windows and and for display in the Clipboard window.
//
// In order to keep that part of the source code that is not related to the Carbon Scrap
// Manager to a minimum, the windows do not display insertion points, nor can the pictures be
// dragged within the windows.  The text and pictures are not inserted into a document as
// such.  When the Paste item in the Edit menu is chosen:
//
// •  The text or picture on the Clipboard is simply drawn in the centre of the active window.
//
// •  A handle to the text or picture is assigned to fields in a document structure associated
//    with the window.  (The demonstration program at MonoTextEdit shows how to cut, copy, and
//    paste text from and to a TextEdit structure using the scrap.)
//
// For the same reason, highlighting the selected text or picture in a window is achieved by
// simply inverting the image.
//
// The program utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus (preload,
//    non-purgeable).
//
// •  Three 'WIND' resources (purgeable) (initially visible), two for the program's main
//    windows and one for the Clipboard window.
//
// •  A 'TEXT' resource (non-purgeable) containing text displayed in the left window at
//    program  start.
//
// •  A 'PICT' resource (non-purgeable) containing a picture displayed in the right window at
//    program start.
//
// •  A 'STR#' resource (purgeable) containing strings to be displayed in the error Alert.
//
// •  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  iClose           4
#define  iQuit            12
#define mEdit             130
#define  iCut             3
#define  iCopy            4
#define  iPaste           5
#define  iClear           6
#define  iClipboard       8
#define rWindow           128
#define rClipboardWindow  130
#define rText             128
#define rPicture          128
#define rErrorStrings     128
#define  eFailMenu        1
#define  eFailWindow      2
#define  eFailDocStruc    3
#define  eFailMemory      4
#define  eClearScrap      5
#define  ePutScrapFlavor  6
#define  eGetScrapSize    7
#define  eGetScrapData    8
#define kDocumentType     1
#define kClipboardType    2
#define MAX_UINT32        0xFFFFFFFF

// ..... typedefs

typedef struct
{
  PicHandle pictureHdl;
  Handle    textHdl;
  Boolean   selectFlag;
  SInt16    windowType;
} docStructure, **docStructureHandle;

// ..... global variables

Boolean   gRunningOnX         = false;
Boolean   gDone;
WindowRef gWindowRefs[2];
WindowRef gClipboardWindowRef = NULL;
Boolean   gClipboardShowing   = false;
SInt16    gPixelDepth;
Boolean   gIsColourDevice     = false;

// ..... function prototypes

void  main                  (void);
void  doPreliminaries       (void);
OSErr quitAppEventHandler   (AppleEvent *,AppleEvent *,SInt32);
void  doEvents              (EventRecord *);
void  doUpdate              (EventRecord *);
void  doOSEvent             (EventRecord *);
void  doAdjustMenus         (void);
void  doMenuChoice          (SInt32);
void  doErrorAlert          (SInt16);
void  doOpenWindows         (void);
void  doCloseWindow         (void);
void  doInContent           (Point);
void  doCutCopyCommand      (Boolean);
void  doPasteCommand        (void);
void  doClearCommand        (void);
void  doClipboardCommand    (void);
void  doDrawClipboardWindow (void);
void  doDrawDocumentWindow  (WindowRef);
Rect  doSetDestRect         (Rect *,WindowRef);
void  doGetDepthAndDevice   (void);

// ********************************************************************************* main
```

```
void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  Boolean       gotEvent;
  EventRecord   eventStructure;

  // ..... do preliminaries

  doPreliminaries();

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

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

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;
  }

  // ..... open windows

  doOpenWindows();

  // ..... get pixel depth and whether colour device for function SetThemeTextColor

  doGetDepthAndDevice();

  // ..... enter eventLoop

  gDone = false;

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL);

    if(gotEvent)
      doEvents(&eventStructure);
  }
}

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

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(96);
  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)
  {
    CallInScrapPromises();
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ************************************************************************************** doEvents

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

  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->where);
          break;

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

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

    case keyDown:
```

```
      if((eventStrucPtr->modifiers & cmdKey) != 0)
      {
        doAdjustMenus();
        doMenuChoice(MenuEvent(eventStrucPtr));
      }
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      windowRef = (WindowRef) eventStrucPtr->message;
      if(windowRef == gClipboardWindowRef)
        doDrawClipboardWindow();
      break;

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

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

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  SInt32             windowType;

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

  BeginUpdate(windowRef);

  if(windowType == kDocumentType)
  {
    if((*docStrucHdl)->pictureHdl != NULL || (*docStrucHdl)->textHdl != NULL)
      doDrawDocumentWindow(windowRef);
  }
  else if(windowType == kClipboardType)
    doDrawClipboardWindow();

  EndUpdate(windowRef);
}

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

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
      {
        SetThemeCursor(kThemeArrowCursor);
        if(gClipboardWindowRef && gClipboardShowing)
          ShowWindow(gClipboardWindowRef);
      }
      else
      {
        if(gClipboardWindowRef && gClipboardShowing)
        ShowHide(gClipboardWindowRef,false);
      }
      break;
  }
}
```

```
// ********************************************************************************** doAdjustMenus

void  doAdjustMenus(void)
{
  MenuRef            fileMenuRef, editMenuRef;
  docStructureHandle docStrucHdl;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   scrapFlavorFlags;
  Boolean            scrapHasText = false, scrapHasPicture = false;

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

  docStrucHdl = (docStructureHandle) GetWRefCon(FrontWindow());

  if((*docStrucHdl)->windowType == kClipboardType)
    EnableMenuItem(fileMenuRef,iClose);
  else
    DisableMenuItem(fileMenuRef,iClose);

  if(((*docStrucHdl)->pictureHdl || (*docStrucHdl)->textHdl) && ((*docStrucHdl)->selectFlag))
  {
    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)
    scrapHasText = true;

  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypePicture,&scrapFlavorFlags);
  if(osError == noErr)
    scrapHasPicture = true;

  if((scrapHasText || scrapHasPicture) && ((*docStrucHdl)->windowType != kClipboardType))
    EnableMenuItem(editMenuRef,iPaste);
  else
    DisableMenuItem(editMenuRef,iPaste);

  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:
      if(menuItem == iClose)
        doCloseWindow();
      else if(menuItem == iQuit)
      {
        CallInScrapPromises();
        gDone = true;
      }
      break;

    case mEdit:
      switch(menuItem)
      {
        case iCut:
          doCutCopyCommand(true);
          break;

        case iCopy:
          doCutCopyCommand(false);
          break;

        case iPaste:
          doPasteCommand();
          break;

        case iClear:
          doClearCommand();
          break;

        case iClipboard:
          doClipboardCommand();
          break;
      }
      break;
  }

  HiliteMenu(0);
}

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

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

  GetIndString(errorString,rErrorStrings,errorCode);
  StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
  ExitToShell();
}

// ********************************************************************************* doOpenWindows

void  doOpenWindows(void)
{
  SInt16             a;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  Rect               theRect;

  for(a=0;a<2;a++)
  {
    if(!(windowRef = GetNewCWindow(rWindow + a,NULL,(WindowRef) -1)))
      doErrorAlert(eFailWindow);
    gWindowRefs[a] = windowRef;
```

```
    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
      doErrorAlert(eFailDocStruc);
    SetWRefCon(windowRef,(SInt32) docStrucHdl);

    (*docStrucHdl)->pictureHdl = NULL;
    (*docStrucHdl)->textHdl    = NULL;
    (*docStrucHdl)->windowType = kDocumentType;
    (*docStrucHdl)->selectFlag = false;

    SetPortWindowPort(windowRef);
    TextSize(10);

    if(gRunningOnX)
    {
      GetWindowPortBounds(windowRef,&theRect);
      InsetRect(&theRect,40,40);
      ClipRect(&theRect);
    }

    if(a == 0)
      (*docStrucHdl)->textHdl = (Handle) GetResource('TEXT',rText);
    else
      (*docStrucHdl)->pictureHdl = GetPicture(rPicture);
  }
}

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

void  doCloseWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  MenuRef            editMenuRef;

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

  if((*docStrucHdl)->windowType == kClipboardType)
  {
    DisposeWindow(windowRef);
    gClipboardWindowRef = NULL;
    gClipboardShowing = false;
    editMenuRef = GetMenuRef(mEdit);
    SetMenuItemText(editMenuRef,iClipboard,"\pShow Clipboard");
  }
}

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

void  doInContent(Point mouseXY)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  Rect               theRect;

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

  if((*docStrucHdl)->windowType == kClipboardType)
    return;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if((*docStrucHdl)->textHdl != NULL || (*docStrucHdl)->pictureHdl != NULL)
  {
    if((*docStrucHdl)->textHdl != NULL)
```

```
    {
      GetWindowPortBounds(windowRef,&theRect);
      InsetRect(&theRect,40,40);
    }
    else if((*docStrucHdl)->pictureHdl != NULL)
    {
      theRect = doSetDestRect(&(*(*docStrucHdl)->pictureHdl)->picFrame,windowRef);
    }

    GlobalToLocal(&mouseXY);

    if(PtInRect(mouseXY,&theRect) && (*docStrucHdl)->selectFlag == false)
    {
      (*docStrucHdl)->selectFlag = true;
      InvertRect(&theRect);
    }
    else if(!PtInRect(mouseXY,&theRect) && (*docStrucHdl)->selectFlag == true)
    {
      (*docStrucHdl)->selectFlag = false;
      InvertRect(&theRect);
    }
  }

  SetPort(oldPort);
}

// ********************************************************************************* doCutCopyCommand

void  doCutCopyCommand(Boolean cutFlag)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  OSStatus           osError;
  ScrapRef           scrapRef;
  Size               dataSize;
  GrafPtr            oldPort;
  Rect               portRect;

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

  if((*docStrucHdl)->selectFlag == false)
    return;

  osError = ClearCurrentScrap();
  if(osError == noErr)
  {
    GetCurrentScrap(&scrapRef);

    if((*docStrucHdl)->textHdl != NULL)  // ..........'TEXT'
    {
      dataSize = GetHandleSize((Handle) (*docStrucHdl)->textHdl);
      HLock((*docStrucHdl)->textHdl);

      osError = PutScrapFlavor(scrapRef,kScrapFlavorTypeText,kScrapFlavorMaskNone,
                               dataSize,*((*docStrucHdl)->textHdl));
      if(osError != noErr)
        doErrorAlert(ePutScrapFlavor);
    }
    else if((*docStrucHdl)->pictureHdl != NULL)  // .......... 'PICT'
    {
      dataSize = GetHandleSize((Handle) (*docStrucHdl)->pictureHdl);
      HLock((Handle) (*docStrucHdl)->pictureHdl);

      osError = PutScrapFlavor(scrapRef,kScrapFlavorTypePicture,kScrapFlavorMaskNone,
                               dataSize,*((Handle) (*docStrucHdl)->pictureHdl));
      if(osError != noErr)
        doErrorAlert(ePutScrapFlavor);
    }
```

```
    if((*docStrucHdl)->textHdl != NULL)
      HUnlock((*docStrucHdl)->textHdl);
    if((*docStrucHdl)->pictureHdl != NULL)
      HUnlock((Handle) (*docStrucHdl)->pictureHdl);
  }
  else
    doErrorAlert(eClearScrap);

  if(cutFlag)
  {
    GetPort(&oldPort);
    SetPortWindowPort(windowRef);

    if((*docStrucHdl)->pictureHdl != NULL)
    {
      DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
      (*docStrucHdl)->pictureHdl = NULL;
      (*docStrucHdl)->selectFlag = false;
    }
    if((*docStrucHdl)->textHdl != NULL)
    {
      DisposeHandle((*docStrucHdl)->textHdl);
      (*docStrucHdl)->textHdl = NULL;
      (*docStrucHdl)->selectFlag = false;
    }

    GetWindowPortBounds(windowRef,&portRect);
    EraseRect(&portRect);

    SetPort(oldPort);
  }

  if(gClipboardWindowRef != NULL)
    doDrawClipboardWindow();
}

// ********************************************************************************* doPasteCommand

void  doPasteCommand(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  ScrapRef           scrapRef;
  OSStatus           osError;
  ScrapFlavorFlags   flavorFlags;
  Size               sizeOfPictData = 0, sizeOfTextData = 0;
  Handle             newTextHdl, newPictHdl;
  CFStringRef        stringRef;
  Rect               destRect, portRect;

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

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  GetCurrentScrap(&scrapRef);

  // ..................................................................................................... 'TEXT'

  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&flavorFlags);
  if(osError == noErr)
  {
    osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
    if(osError == noErr && sizeOfTextData > 0)
    {
      newTextHdl = NewHandle(sizeOfTextData);
```

```
    osError = MemError();
    if(osError == memFullErr)
      doErrorAlert(eFailMemory);

    HLock(newTextHdl);

    osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypeText,&sizeOfTextData,*newTextHdl);
    if(osError != noErr)
      doErrorAlert(eGetScrapData);

    // ..... draw text in window

    GetWindowPortBounds(windowRef,&portRect);
    EraseRect(&portRect);
    InsetRect(&portRect,40,40);

    if(!gRunningOnX)
    {
      TETextBox(*newTextHdl,sizeOfTextData,&portRect,teFlushLeft);
    }
    else
    {
      stringRef =  CFStringCreateWithBytes(NULL,(UInt8 *) *newTextHdl,sizeOfTextData,
                                           smSystemScript,false);
      DrawThemeTextBox(stringRef,kThemeSmallSystemFont,kThemeStateActive,true,&portRect,
                       teFlushLeft,NULL);
      if(stringRef != NULL)
        CFRelease(stringRef);
    }

    HUnlock(newTextHdl);

    (*docStrucHdl)->selectFlag = false;

    // ..... assign handle to new text to window's document structure

    if((*docStrucHdl)->textHdl != NULL)
      DisposeHandle((*docStrucHdl)->textHdl);
    (*docStrucHdl)->textHdl = newTextHdl;

    if((*docStrucHdl)->pictureHdl != NULL)
      DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
    (*docStrucHdl)->pictureHdl = NULL;
  }
  else
    doErrorAlert(eGetScrapSize);
}

// ..... ' PICT'

else
{
  (osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypePicture,&flavorFlags));
  if(osError == noErr)
  {
    osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData);
    if(osError == noErr && sizeOfPictData > 0)
    {
      newPictHdl = NewHandle(sizeOfPictData);
      osError = MemError();
      if(osError == memFullErr)
        doErrorAlert(eFailMemory);

      HLock(newPictHdl);

      osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData,
                                   *newPictHdl);
      if(osError != noErr)
        doErrorAlert(eGetScrapData);
```

```
        // ..... draw picture in window

        GetWindowPortBounds(windowRef,&portRect);
        EraseRect(&portRect);
        (*docStrucHdl)->selectFlag = false;
        destRect = doSetDestRect(&(*(PicHandle) newPictHdl)->picFrame,windowRef);
        DrawPicture((PicHandle) newPictHdl,&destRect);

        HUnlock(newPictHdl);

        (*docStrucHdl)->selectFlag = false;

        // ..... assign handle to new picture to window's document structure

        if((*docStrucHdl)->pictureHdl != NULL)
          DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
        (*docStrucHdl)->pictureHdl = (PicHandle) newPictHdl;

        if((*docStrucHdl)->textHdl != NULL)
          DisposeHandle((Handle) (*docStrucHdl)->textHdl);
        (*docStrucHdl)->textHdl = NULL;
      }
      else
        doErrorAlert(eGetScrapSize);
    }
  }

  SetPort(oldPort);
}

// ************************************************************************** doClearCommand

void  doClearCommand(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  Rect               portRect;

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

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  if((*docStrucHdl)->textHdl != NULL)
  {
    DisposeHandle((*docStrucHdl)->textHdl);
    (*docStrucHdl)->textHdl = NULL;
  }

  if((*docStrucHdl)->pictureHdl != NULL)
  {
    DisposeHandle((Handle) (*docStrucHdl)->pictureHdl);
    (*docStrucHdl)->pictureHdl = NULL;
  }

  (*docStrucHdl)->selectFlag = false;

  GetWindowPortBounds(windowRef,&portRect);
  EraseRect(&portRect);

  SetPort(oldPort);
}

// ********************************************************************** doClipboardCommand

void  doClipboardCommand(void)
```

```
{
  MenuRef            editMenuRef;
  docStructureHandle docStrucHdl;

  editMenuRef = GetMenuRef(mEdit);

  if(gClipboardWindowRef == NULL)
  {
    if(!(gClipboardWindowRef = GetNewCWindow(rClipboardWindow,NULL,(WindowRef)-1)))
      doErrorAlert(eFailWindow);
    if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
      doErrorAlert(eFailDocStruc);

    SetWRefCon(gClipboardWindowRef,(SInt32) docStrucHdl);
    (*docStrucHdl)->windowType = kClipboardType;

    SetMenuItemText(editMenuRef,iClipboard,"\pHide Clipboard");

    gClipboardShowing = true;
  }
  else
  {
    if(gClipboardShowing)
    {
      HideWindow(gClipboardWindowRef);
      gClipboardShowing = false;
      SetMenuItemText(editMenuRef,iClipboard,"\pShow Clipboard");
    }
    else
    {
      ShowWindow(gClipboardWindowRef);
      gClipboardShowing = true;
      SetMenuItemText(editMenuRef,iClipboard,"\pHide Clipboard");
    }
  }
}

// ********************************************************************** doDrawClipboardWindow

void  doDrawClipboardWindow(void)
{
  GrafPtr          oldPort;
  Rect             theRect, textBoxRect;
  ScrapRef         scrapRef;
  OSStatus         osError;
  ScrapFlavorFlags flavorFlags;
  CFStringRef      stringRef;
  Handle           tempHdl;
  Size             sizeOfPictData = 0, sizeOfTextData = 0;
  RGBColor         blackColour = { 0x0000, 0x0000, 0x0000 };

  GetPort(&oldPort);
  SetPortWindowPort(gClipboardWindowRef);

  GetWindowPortBounds(gClipboardWindowRef,&theRect);
  EraseRect(&theRect);

  SetRect(&theRect,-1,-1,597,24);
  DrawThemeWindowHeader(&theRect,gClipboardWindowRef == FrontWindow());

  if(gClipboardWindowRef == FrontWindow())
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  SetRect(&textBoxRect,10,5,120,20);
  DrawThemeTextBox(CFSTR("Clipboard Contents:"),kThemeSmallSystemFont,0,true,&textBoxRect,
                   teJustLeft,NULL);
```

```
GetCurrentScrap(&scrapRef);

// ..... 'TEXT'

osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypeText,&flavorFlags);
if(osError == noErr)
{
  osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypeText,&sizeOfTextData);
  if(osError == noErr && sizeOfTextData > 0)
  {
    SetRect(&textBoxRect,120,5,597,20);
    DrawThemeTextBox(CFSTR("Text"),kThemeSmallSystemFont,0,true,&textBoxRect,teJustLeft,
                     NULL);

    tempHdl = NewHandle(sizeOfTextData);
    osError = MemError();
    if(osError == memFullErr)
      doErrorAlert(eFailMemory);

    HLock(tempHdl);

    osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypeText,&sizeOfTextData,*tempHdl);
    if(osError != noErr)
      doErrorAlert(eGetScrapData);

    // ..... draw text in clipboard window

    GetWindowPortBounds(gClipboardWindowRef,&theRect);
    theRect.top += 24;
    InsetRect(&theRect,2,2);

    if(sizeOfTextData >= 965)
      sizeOfTextData = 965;
    stringRef =  CFStringCreateWithBytes(NULL,(UInt8 *) *tempHdl,sizeOfTextData,
                                         CFStringGetSystemEncoding(),true);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,true,&theRect,teFlushLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    HUnlock(tempHdl);
    DisposeHandle(tempHdl);
  }
  else
    doErrorAlert(eGetScrapSize);
}

// ..... 'PICT'

else
{
  osError = GetScrapFlavorFlags(scrapRef,kScrapFlavorTypePicture,&flavorFlags);
  if(osError == noErr)
  {
    osError = GetScrapFlavorSize(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData);
    if(osError == noErr && sizeOfPictData > 0)
    {
      SetRect(&textBoxRect,120,5,597,20);
      DrawThemeTextBox(CFSTR("Picture"),kThemeSmallSystemFont,0,true,&textBoxRect,
                       teJustLeft,NULL);

      tempHdl = NewHandle(sizeOfPictData);
      osError = MemError();
      if(osError == memFullErr)
        doErrorAlert(eFailMemory);

      HLock(tempHdl);

      osError = GetScrapFlavorData(scrapRef,kScrapFlavorTypePicture,&sizeOfPictData,
                                   *tempHdl);
```

```
        if(osError != noErr)
          doErrorAlert(eGetScrapData);

        // ..... draw picture in clipboard window

        theRect = (*(PicHandle) tempHdl)->picFrame;
        OffsetRect(&theRect,-((*(PicHandle) tempHdl)->picFrame.left - 2),
                            -((*(PicHandle) tempHdl)->picFrame.top - 26));
        DrawPicture((PicHandle) tempHdl,&theRect);

        HUnlock(tempHdl);
        DisposeHandle(tempHdl);
      }
      else
        doErrorAlert(eGetScrapSize);
    }
  }

  TextMode(srcOr);
  SetPort(oldPort);
}

// ********************************************************************** doDrawDocumentWindow

void  doDrawDocumentWindow(WindowRef windowRef)
{
  GrafPtr            oldPort;
  docStructureHandle docStrucHdl;
  Rect               destRect;
  CFStringRef        stringRef;

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->textHdl != NULL)
  {
    GetWindowPortBounds(windowRef,&destRect);
    InsetRect(&destRect,40,40);

    stringRef =  CFStringCreateWithBytes(NULL,(UInt8 *) *(*docStrucHdl)->textHdl,
                                         GetHandleSize((*docStrucHdl)->textHdl),
                                         smSystemScript,false);
    DrawThemeTextBox(stringRef,kThemeSmallSystemFont,0,true,&destRect,teFlushLeft,NULL);
    if(stringRef != NULL)
      CFRelease(stringRef);

    if((*docStrucHdl)->selectFlag)
      InvertRect(&destRect);
  }
  else if((*docStrucHdl)->pictureHdl != NULL)
  {
    destRect = doSetDestRect(&(*(*docStrucHdl)->pictureHdl)->picFrame,windowRef);
    DrawPicture((*docStrucHdl)->pictureHdl,&destRect);
    if((*docStrucHdl)->selectFlag)
      InvertRect(&destRect);
  }

  SetPort(oldPort);
}

// ************************************************************************** doSetDestRect

Rect  doSetDestRect(Rect *picFrame,WindowRef windowRef)
{
  Rect   destRect, portRect;
  SInt16 diffX, diffY;
```

```
  destRect = *picFrame;
  GetWindowPortBounds(windowRef,&portRect);

  OffsetRect(&destRect,-(*picFrame).left,-(*picFrame).top);

  diffX = (portRect.right - portRect.left) - ((*picFrame).right - (*picFrame).left);
  diffY = (portRect.bottom - portRect.top) - ((*picFrame).bottom - (*picFrame).top);

  OffsetRect(&destRect,diffX / 2,diffY / 2);

  return destRect;
}

// ********************************************************************** doGetDepthAndDevice

void doGetDepthAndDevice(void)
{
  GDHandle deviceHdl;

  deviceHdl = GetMainDevice();
  gPixelDepth = (*(*deviceHdl)->gdPMap)->pixelSize;
  if(((1 << gdDevType) & (*deviceHdl)->gdFlags) != 0)
    gIsColourDevice = true;
}

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

Demonstration Program Scrap Comments

When this program is run, the user should choose the Edit menu's Show Clipboard item to open the Clipboard window. The user should then cut, copy, clear and paste the supplied text or picture from/to the two document windows opened by the program, noting the effect on the scrap as displayed in the Clipboard window. (To indicate selection, the text or picture inverts when the user clicks on it with the mouse.)

The user should also copy text and pictures from another application's window, observing the changes to the contents of the Clipboard window when the demonstration application is brought to the front, and paste that text and those pictures to the document windows. (A simple way to get a picture into the scrap on Mac OS 8/9 is to use Command-Shift-Control-4 to copy an area of the screen to the scrap.)

The program's preferred scrap flavour type is 'TEXT'. Thus, if the scrap contains data in both the 'TEXT' and 'PICT' flavour types, only the 'TEXT' flavour will be used for pastes to the document windows and for display in the Clipboard window. The user can prove this behaviour by copying a picture object containing text in an application such as Adobe Illustrator, bringing the demonstration program to the front, noting the contents of the Clipboard window, pasting to one of the document windows, and noting what is pasted.

The user should note that, when the Clipboard window is open and showing, it will be hidden when the program is sent to the background and shown again when the program is brought to the foreground.

defines

kDocumentType and kClipboardType will enable the program to distinguish between the two "document" windows opened by the program and the Clipboard window.

typedefs

Document structures will be attached to the two document windows and the Clipboard window. docStructure is the associated data type. The windowType field will be used to differentiate between the document windows and the Clipboard window.

Global Variables

The WindowRefs for the two document windows will be copied into the fields of gWindowRefs. gClipBoardWindowRef will be assigned the WindowRef for the Clipboard window when it is opened by the user. gClipBoardShowing will keep track of whether the Clipboard window is currently hidden or showing.

quitAppEventHandler

Note that, before gDone is set to true, CallInScrapPromises is called to force any outstanding promises to be kept.

doEvents

Note that, in the case of an activate event for the Clipboard window, the function doDrawClipBoardWindow is called.

doUpdate

If the window is of the document type (as opposed to the Clipboard type), and if the window's document structure currently contains a picture or text, doDrawDocumentWindow is called to draw that picture or text. If the window is the Clipboard window, doDrawClipboardWindow is called to draw the Clipboard window.

doOSEvent

In the case of a resume event, if the Clipboard window has been opened and was showing when the program was sent to the background, ShowWindow is called to show the Clipboard window. In the case of a suspend event, if the Clipboard window has been opened and is currently showing, ShowHideWindow is called to hide the Clipboard window. ShowHide, rather than HideWindow is used in this instance to prevent activation of the first document window in the list when the Clipboard window is in front and the application is switched out.

doAdjustMenus

If the front window is the Clipboard window, the Close item is enabled, otherwise it is disabled. If the document contains a picture and that picture is currently selected, the Cut, Copy, and Clear items are enabled, otherwise they are disabled.

If the scrap contains data of flavour type 'PICT' or flavour type 'TEXT', and the front window is not the Clipboard window, the Paste item is enabled, otherwise it is disabled. In this section, GetCurrentScrap is called to obtain a reference to the current scrap. This reference is then passed in a call to two calls to GetScrapFlavorFlags, which determine whether the scrap contains data of the flavour type 'PICT'

and/or flavour type 'TEXT'. If it does, and if the front window is not the Clipboard window, the Paste item is enabled.

doMenuChoice

Note that, at the mFile case, before gDone is set to true, CallInScrapPromises is called to force any outstanding promises to be kept.

doOpenWindows

doOpenWindows opens the two document windows, creates document structures for each window, attaches the document structures to the windows and initialises the fields of the document structures.

The textHdl field of the first window's document structure is assigned a handle to a 'TEXT' resource and the textHdl field of the second window's document structure is assigned a handle to a 'PICT' resource.

doCloseWindow

doCloseWindow closes the Clipboard window (the only window that can be closed from within the program).

If the window is the Clipboard window, The window is disposed of, the global variable which contains its reference is set to NULL, the global variable which keeps track of whether the window is showing or hidden is set to false, and the text of the Show/Hide Clipboard menu item is set to Show Clipboard.

doInContent

doInContent handles mouse-down events in the content region of a document window. If the window contains text or a picture, and if the mouse-down was inside the text or picture, the text or picture is selected. If the window contains a text or picture, and if the mouse-down was outside the text or picture, the text or picture is deselected.

The first two lines get a reference to the front window and a handle to its document structure. If the front window is the Clipboard window, the function returns immediately. GetPort and SetPortWindowPort save the current graphics port and make the graphics port associated with the front window the current graphics port.

If the front window contains text or a picture the following occurs. If the window contains text, a rectangle equal to the port rectangle minus 40 pixels all round is established. If the window contains a picture, the function doSetDestRect is called to return a rectangle of the same dimensions as that contained in the picture structure's picFrame field, but centred laterally and vertically in the window. GlobalToLocal converts the mouse-down coordinates to local coordinates preparatory to the call to PtInRect. If the mouse-down occurred within the rectangle and the text or picture has not yet been selected, the document structure's selectFlag field is set to true and the text or picture is inverted. If the mouse-down occurred outside the rectangle and the text or picture is currently selected, the document structure's selectFlag field is set to false, and the text or picture is re-inverted.

doCutCopyCommand

doCutCopyCommand handles the user's choice of the Cut and Copy items in the Edit menu.

The first two lines get a reference to the front window and a handle to that window's document structure.

If the selectFlag field of the document structure contains false (meaning that the text or picture has not been selected), the function returns immediately. (Note that no check is made as to whether the front window is the Clipboard window because the menu adjustment function disables the Cut and Copy items when the Clipboard window is the front window, meaning that this function can never be called when the Clipboard window is in front.)

ClearCurrentScrap attempts to clear the current scrap. (This call should always be made immediately the user chooses Cut or Copy.) If the call is successful, GetCurrentScrap then gets a reference to the scrap.

If the selected item is text, GetHandleSize gets the size of the text from the window's document structure. (In a real application, code which gets the size of the selection would appear here.) PutScrapFlavor copies the "selected" text to the scrap. If the call to PutScrapFlavor is not successful, an alert is displayed to advise the user of the error.

If the selected item is a picture, GetHandleSize gets the size of the picture from the window's document structure. PutScrapFlavor copies the selected picture to the scrap. If the call to PutScrapFlavor is not successful, an alert is displayed to advise the user of the error.

If the menu choice was the Cut item, additional action is taken. Preparatory to a call to EraseRect, the current graphics port is saved and the front window's port is made the current port. DisposeHandle is called to dispose of the text or picture and the document structure's textHdl or pictureHdl field, and

selectFlag field, are set to NULL and false respectively. EraseRect then erases the picture. (In a real application, the action taken in this block would be to remove the selected text or picture from the document.)

Finally, and importantly, if the Clipboard window has previously been opened by the user, doDrawClipboardWindow is called to draw the current contents of the scrap in the Clipboard window.

doPasteCommand

doPasteCommand handles the user's choice of the Paste item from the Edit menu. Note that no check is made as to whether the front window is the Clipboard window because the menu adjustment function disables the Paste item when the Clipboard window is the front window, meaning that this function can never be called when the Clipboard window is in front.

GetCurrentScrap gets a reference to the scrap.

The first call to GetScrapFlavorFlags determines whether the scrap contains data of flavour type 'TEXT'. If so, GetScrapFlavorSize is called to get the size of the 'TEXT' data. NewHandle creates a relocatable block of a size equivalent to the 'TEXT' data. GetScrapFlavorData is called to copy the 'TEXT' data in the scrap to this block.

TETextBox or DrawThemeTextBox is called to draw the text in a rectangle equal to the port rectangle minus 40 pixels all round.

If the textHdl field of the window's document structure does not contain NULL, the associated block is disposed of, following which the handle to the block containing the new 'TEXT' data is then assigned to the textHdl field. (In a real application, this block would copy the text into the document at the insertion point.) (The last three lines in this section simply ensure that, if the window's "document" contains text, it cannot also contain a picture. This is to prevent a picture overdrawing the text when the window contents are updated.)

If the scrap does not contain data of flavour type 'TEXT', GetScrapFlavorFlags is called again to determine whether the scrap contains data of flavour type 'PICT'. If it does, much the same procedure is followed, except that rectangle in which the picture is drawn is extracted from the 'PICT' data itself and adjusted to the middle of the window via a call to the function doSetDestRec.

It is emphasized that the scrap is only checked for flavour type 'PICT' if the scrap does not contain flavour type 'TEXT'. Thus, if both flavours exist in the scrap, only the 'TEXT' flavour will be used to draw the Clipboard.

doClearCommand

doClearCommand handles the user's choice of the Clear item in the Edit menu.

Note that, as was the case in the doCutCopyCommand function, no check is made as to whether the front window is the Clipboard window because the menu adjustment function disables the Clear item when the Clipboard window is the front window.

If the front window's document structure indicates that the window contains text or a picture, the block containing the TextEdit structure or Picture structure is disposed of and the relevant field of the document structure is set to NULL. In addition, the selectFlag field of the document structure to false and the window's port rectangle is erased.

doClipboardCommand

doClipboardCommand handles the user's choice of the Show/Hide Clipboard command in the Edit menu.

The first line gets the reference to the Edit menu. This will be required in order to toggle the Show/Hide Clipboard item's text between Show Clipboard and Hide Clipboard.

The if statement checks whether the Clipboard window has been created. If not, the Clipboard window is created by the call to GetNewCWindow, a document structure is created and attached to the window, the windowType field of the document structure is set to indicate that the window is of the Clipboard type, a global variable which keeps track of whether the Clipboard window is currently showing or hidden is set to true, and the text of the menu item is set to Hide Clipboard.

If the Clipboard window has previously been created, and if the window is currently showing, the window is hidden, the Clipboard-showing flag is set to false, and the item's text is set to Show Clipboard. If the window is not currently showing, the window is made visible, the Clipboard-showing flag is set to true, and the item's text is set to Hide Clipboard.

doDrawClipboardWindow

doDrawClipboardWindow draws the contents of the scrap in the Clipboard window. It supports the drawing of both 'PICT' and 'TEXT' flavour type data.

The first four lines save the current graphics port, make the Clipboard window's graphics port the current graphics port and erase the window's content region.

DrawThemeWindowHeader draws a window header in the top of the window. Text describing the type of data in the scrap will be drawn in this header.

The next four lines set the text-drawing source mode according to whether or not the Clipboard window is the front window. The next block draws the words "Clipboard Contents:" in the header.

The code for getting a reference to the current scrap, checking for the 'TEXT' and 'PICT' flavour types, getting the flavour size, getting the flavour data, and drawing the text and picture in the window is much the same as in the function doPasteCommand. The differences are: the rectangle in which the text is drawn is the port rectangle minus a few pixels all round and with the top further adjusted downwards by the height of the window header: the left/top of the rectangle in which the picture is drawn is two pixels inside the left side of the content region and two pixels below the window header respectively; The words "Text" or "Picture" are drawn in the window header as appropriate.

Note that, as was the case in the function doPasteCommand, the scrap is only checked for flavour type 'PICT' if the scrap does not contain flavour type 'TEXT'. Thus, if both flavours exist in the scrap, only the 'TEXT' flavour will be used to draw the Clipboard.

doDrawDocumentWindow

doDrawDocumentWindow draws the text or picture belonging to that window in the window. It is called when an update event is received for the window.

doSetDestRect

doSetDestRect takes the rectangle contained in the picFrame field of a picture structure and returns a rectangle of the same dimensions but centred in the window's port rectangle.