Demonstration Program Files Listing

```
// *******************************************************************************************
// Files.h                                                              CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program demonstrates:
//
// •  File operations associated with:
//
//    •  The user invoking the File menu Open…, Close, Save, Save As…, Revert, and Quit
//      commands of a typical application.
//
//    •  Handling of the required Apple events Open Application, Re-open Application, Open
//      Documents, Print Documents, and Quit Application.
//
// •  File synchronisation.
//
// •  The creation, display, and handling of Open, Save Location, Choose a Folder, Save
//    Changes and Discard Changes dialogs and alerts using the new model introduced with
//    Navigation Services 3.0.
//
// To keep the code not specifically related to files and file-handling to a minimum, an item
// is included in the Demonstration menu which allows the user to simulate "touching" a window
// (that is, modifying the contents of the associated document).  Choosing the first menu item
// in this menu sets the window-touched flag in the window's document structure to true and
// draws the text "WINDOW TOUCHED" in the window in a large font size, this latter so that the
// user can keep track of which windows have been "touched".
//
// This program is also, in part, an extension of the demonstration program Windows2 in that
// it also demonstrates certain file-related Window Manager features introduced with the Mac
// OS 8.5 Window Manager.  These features are:
//
// •  Window proxy icons.
//
// •  Window path pop-up menus.
//
// Those sections of the source code relating to these features are identified with ///// at
// the right of each line.
//
// The program utilises the following resources:
//
// •  A 'plst' resource containing an information property list which provides information
//    to the Mac OS X Finder.
//
// •  An 'MBAR' resource, and 'MENU' and 'xmnu' resources for Apple, File, Edit and
//    Demonstration menus (preload, non-purgeable).
//
// •  A 'WIND' resource (purgeable) (initially not visible).
//
// •  A 'STR ' resource containing the "missing application name" string, which is copied to
//    all document files created by the program.
//
// •  'STR#' resources (purgeable) containing error strings, the application's name (for
//    certain Navigation Services functions), and a message  string for the Choose a Folder
//    dialog.
//
// •  An 'open' resource (purgeable) containing the file type list for the Open dialog.
//
// •  A 'kind' resource (purgeable) describing file types, which is used by Navigation
//    Services to  build the native file types section of the Show pop-up menu in the Open
//    dialog.
//
// •  Two 'pnot' resources (purgeable) which, together with an associated 'PICT' resource
//    (purgeable) and a 'TEXT' resource created by the program, provide the previews for
//    the PICT and, on Mac OS 8/9, TEXT files.
//
// •  The 'BNDL' resource (non-purgeable), 'FREF' resources (non-purgeable), signature
//    resource (non-purgeable), and icon family resources (purgeable), required to support the
//    built application on Mac OS 8/9.
//
// •  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  File_New                 'new '
#define  File_Open                'open'
#define  File_Close               'clos'
#define  File_Save                'save'
#define  File_SaveAs              'sava'
#define  File_Revert              'reve'
#define  File_Quit                'quit'
#define  iQuit                    12
#define mDemonstration            131
#define  Demo_TouchWindow         'touc'
#define  Demo_ChooseAFolderDialog 'choo'
#define rNewWindow                128
#define rErrorStrings              128
#define  eInstallHandler          1000
#define  eMaxWindows              1001
#define  eCantFindFinderProcess   1002                                              /////
#define rMiscStrings              129
#define  sApplicationName         1
#define  sChooseAFolder           2
#define rOpenResource             128
#define kFileCreator              'kJbb'
#define kFileTypeTEXT             'TEXT'
#define kFileTypePICT             'PICT'
#define kMaxWindows               10
#define kOpen                     0
#define kPrint                    1
#define MIN(a,b)                  ((a) < (b) ? (a) : (b))

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

typedef struct
{
  TEHandle     editStrucHdl;
  PicHandle    pictureHdl;
  SInt16       fileRefNum;
  FSSpec       fileFSSpec;
  AliasHandle  aliasHdl;
  Boolean      windowTouched;
  NavDialogRef modalToWindowNavDialogRef;
  NavEventUPP  askSaveDiscardEventFunctionUPP;
  Boolean      isAskSaveChangesDialog;
}  docStructure, *docStructurePointer, **docStructureHandle;

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

void   main                        (void);
void   eventLoop                   (void);
void   doPreliminaries             (void);
void   doInstallAEHandlers         (void);
void   doEvents                    (EventRecord *);
void   doMouseDown                 (EventRecord *);
void   doBringFinderToFront         (void);                                         /////
OSErr  doFindProcess                (OSType,OSType,ProcessSerialNumber *);          /////
void   doUpdate                     (EventRecord *);
void   doMenuChoice                 (SInt32);
void   doCommand                    (MenuCommand);
void   doAdjustMenus                (void);
void   doErrorAlert                 (SInt16);
void   doCopyPString                (Str255,Str255);
void   doConcatPStrings             (Str255,Str255);
void   doTouchWindow                (void);
OSErr  openAppEventHandler          (AppleEvent *,AppleEvent *,SInt32);
OSErr  reopenAppEventHandler        (AppleEvent *,AppleEvent *,SInt32);
OSErr  openAndPrintDocsEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSErr  quitAppEventHandler          (AppleEvent *,AppleEvent *,SInt32);
OSErr  doHasGotRequiredParams       (AppleEvent *);
SInt16 doReviewChangesAlert         (void);

OSErr  doNewCommand                 (void);
OSErr  doOpenCommand                (void);
OSErr  doCloseCommand               (NavAskSaveChangesAction);
OSErr  doSaveCommand                (void);
OSErr  doSaveAsCommand              (void);
OSErr  doRevertCommand              (void);

OSErr  doNewDocWindow               (Boolean,OSType,WindowRef *);
OSErr  doCloseDocWindow             (WindowRef);
OSErr  doOpenFile                   (FSSpec,OSType);
OSErr  doReadTextFile               (WindowRef);
OSErr  doReadPictFile               (WindowRef);
OSErr  doCreateAskSaveChangesDialog (WindowRef,docStructureHandle,NavAskSaveChangesAction);
OSErr  doSaveUsingFSSpec            (WindowRef,NavReplyRecord *);
OSErr  doSaveUsingFSRef             (WindowRef,NavReplyRecord *);
OSErr  doWriteFile                  (WindowRef);
OSErr  doWriteTextData              (WindowRef,SInt16);
OSErr  doWritePictData              (WindowRef,SInt16);

void   getFilePutFileEventFunction  (NavEventCallbackMessage,NavCBRecPtr,NavCallBackUserData);
void   askSaveDiscardEventFunction  (NavEventCallbackMessage,NavCBRecPtr,NavCallBackUserData);

OSErr  doCopyResources              (WindowRef);
OSErr  doCopyAResource              (ResType,SInt16,SInt16,SInt16);

void   doSynchroniseFiles           (void);
OSErr  doChooseAFolderDialog        (void);

// *******************************************************************************************
// Files.c
// *******************************************************************************************

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

#include "Files.h"

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

Boolean     gRunningOnX = false;
Boolean     gDone;
SInt16      gAppResFileRefNum;
NavEventUPP gGetFilePutFileEventFunctionUPP ;
Boolean     gQuittingApplication = false;

extern SInt16 gCurrentNumberOfWindows;
extern Rect   gDestRect,gViewRect;

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

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

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

  doPreliminaries();

  // ................................................ save application's resource file file reference number

  gAppResFileRefNum = CurResFile();

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

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

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

    gRunningOnX = true;
  }

  // ......................................................................... install required Apple event handlers

  doInstallAEHandlers();

  // .......... get universal procedure pointer to main Navigation Services services event function

  gGetFilePutFileEventFunctionUPP  =
                              NewNavEventUPP((NavEventProcPtr) getFilePutFileEventFunction);

  // ........................................................................ enter event loop

  eventLoop();
}

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

void  eventLoop(void)
{
  EventRecord eventStructure;

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
    else
      doSynchroniseFiles();

    if(gRunningOnX && gQuittingApplication)
      doCloseCommand(kNavSaveChangesQuittingApplication);
  }
}

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

void  doPreliminaries(void)
```

```
{
  MoreMasterPointers(448);
  InitCursor();
  FlushEvents(everyEvent,0);
}

// ******************************************************************* doInstallAEHandlers

void  doInstallAEHandlers(void)
{
  OSErr osError;

  osError = AEInstallEventHandler(kCoreEventClass,kAEOpenApplication,
                NewAEEventHandlerUPP((AEEventHandlerProcPtr) openAppEventHandler),
                0L,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

  osError = AEInstallEventHandler(kCoreEventClass,kAEReopenApplication,
                NewAEEventHandlerUPP((AEEventHandlerProcPtr) reopenAppEventHandler),
                0L,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

  osError = AEInstallEventHandler(kCoreEventClass,kAEOpenDocuments,
                NewAEEventHandlerUPP((AEEventHandlerProcPtr) openAndPrintDocsEventHandler),
                kOpen,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

  osError = AEInstallEventHandler(kCoreEventClass,kAEPrintDocuments,
                NewAEEventHandlerUPP((AEEventHandlerProcPtr) openAndPrintDocsEventHandler),
                kPrint,false);
  if(osError != noErr)  doErrorAlert(eInstallHandler);

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

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

void  doEvents(EventRecord *eventStrucPtr)
{
  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      doMouseDown(eventStrucPtr);
      break;

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

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if((eventStrucPtr->message & resumeFlag) == 1)
```

```
            SetThemeCursor(kThemeArrowCursor);
          break;
      }
      break;
  }
}

// ****************************************************************************** doMouseDown

void  doMouseDown(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;
  OSStatus       osStatus;                                                        /////
  Boolean        handled = false;                                                 /////
  SInt32         itemSelected;                                                    /////

  partCode = FindWindow(eventStrucPtr->where,&windowRef);

  switch(partCode)
  {
    case inMenuBar:
      doAdjustMenus();
      doMenuChoice(MenuSelect(eventStrucPtr->where));
      break;

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

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

    case inProxyIcon:                                                             /////
      osStatus = TrackWindowProxyDrag(windowRef,eventStrucPtr->where);            /////
      if(osStatus == errUserWantsToDragWindow)                                    /////
        handled = false;                                                          /////
      else if(osStatus == noErr)                                                  /////
        handled = true;                                                           /////

    case inDrag:                                                                  /////
      if(!handled)                                                                /////
      {                                                                           /////
        if(IsWindowPathSelectClick(windowRef,eventStrucPtr))                      /////
        {                                                                         /////
          if(WindowPathSelect(windowRef,NULL,&itemSelected) == noErr)             /////
          {                                                                       /////
            if(LoWord(itemSelected) > 1)                                          /////
              doBringFinderToFront();                                             /////
          }                                                                       /////
                                                                                  /////
          handled = true;                                                         /////
        }                                                                         /////
      }                                                                           /////
      if(!handled)                                                                /////
        DragWindow(windowRef,eventStrucPtr->where,NULL);                          /////
      break;
  }
}

// ********************************************************************* doBringFinderToFront

void  doBringFinderToFront(void)                                                  /////
{                                                                                 /////
  ProcessSerialNumber finderProcess;                                              /////
                                                                                  /////
  if(doFindProcess('MACS','FNDR',&finderProcess) == noErr)                     /////
    SetFrontProcess(&finderProcess);                                           /////
  else                                                                         /////
    doErrorAlert(eCantFindFinderProcess);                                      /////
}                                                                              /////

// ********************************************************************** doFindProcess

OSErr  doFindProcess(OSType creator,OSType type,ProcessSerialNumber *outProcSerNo)
{                                                                              /////
  ProcessSerialNumber procSerialNo;                                            /////
  ProcessInfoRec      procInfoStruc;                                           /////
  OSErr               osError = 0;                                             /////
                                                                               /////
  procSerialNo.highLongOfPSN = 0;                                              /////
  procSerialNo.lowLongOfPSN  = kNoProcess;                                     /////
                                                                               /////
  procInfoStruc.processInfoLength = sizeof(ProcessInfoRec);                    /////
  procInfoStruc.processName       = NULL;                                      /////
  procInfoStruc.processAppSpec    = NULL;                                      /////
  procInfoStruc.processLocation   = NULL;                                      /////
                                                                               /////
  while(true)                                                                  /////
  {                                                                            /////
    osError = GetNextProcess(&procSerialNo);                                   /////
    if(osError != noErr)                                                       /////
      break;                                                                   /////
                                                                               /////
    osError = GetProcessInformation(&procSerialNo,&procInfoStruc);             /////
    if(osError != noErr)                                                       /////
      break;                                                                   /////
    if((procInfoStruc.processSignature == creator) && (procInfoStruc.processType == type)) ///
      break;                                                                   /////
  }                                                                            /////
                                                                               /////
  *outProcSerNo = procSerialNo;                                                /////
                                                                               /////
  return osError;                                                              /////
}                                                                              /////

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

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  GrafPtr            oldPort;
  Rect               destRect;

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

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  BeginUpdate(windowRef);

  if((*docStrucHdl)->pictureHdl)
  {
    destRect = (*(*docStrucHdl)->pictureHdl)->picFrame;
    OffsetRect(&destRect,170,54);
    HLock((Handle) (*docStrucHdl)->pictureHdl);
    DrawPicture((*docStrucHdl)->pictureHdl,&destRect);
    HUnlock((Handle) (*docStrucHdl)->pictureHdl);
  }
  else if((*docStrucHdl)->editStrucHdl)
  {
    HLock((Handle) (*docStrucHdl)->editStrucHdl);
    TEUpdate(&gDestRect,(*docStrucHdl)->editStrucHdl);
    HUnlock((Handle) (*docStrucHdl)->editStrucHdl);
  }

  if((*docStrucHdl)->windowTouched)
  {
    TextSize(48);
    MoveTo(30,170);
    DrawString("\pWINDOW TOUCHED");
    TextSize(12);
  }

  EndUpdate((WindowRef) eventStrucPtr->message);

  SetPort(oldPort);
}

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

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSErr         osError;
  MenuCommand   commandID;

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

  if(menuID == 0)
    return;

  if(menuID == mAppleApplication)
  {
    if(menuItem == iAbout)
      SysBeep(10);
  }
  else
  {
    osError = GetMenuItemCommandID(GetMenuRef(menuID),menuItem,&commandID);
    if(osError == noErr && commandID != 0)
      doCommand(commandID);
  }

  HiliteMenu(0);
}

// ********************************************************************************** doCommand

void  doCommand(MenuCommand commandID)
{
  OSErr osError;

  switch(commandID)
  {
    // .................................................................................. File menu

    case File_New:
      if(osError = doNewCommand())
        doErrorAlert(osError);
      break;

    case File_Open:
      if(osError = doOpenCommand())
        doErrorAlert(osError);
      break;

    case File_Close:
      if(osError = doCloseCommand(kNavSaveChangesClosingDocument))
        doErrorAlert(osError);
      break;

    case File_Save:
      if(osError = doSaveCommand())
        doErrorAlert(osError);
      break;

    case File_SaveAs:
      if(osError = doSaveAsCommand())
        doErrorAlert(osError);
      break;

    case File_Revert:
      if(osError = doRevertCommand())
        doErrorAlert(osError);
      break;

    case File_Quit:
      gQuittingApplication = true;
      doCloseCommand(kNavSaveChangesQuittingApplication);
      break;

    // .................................................................................. Demonstration menu

    case Demo_TouchWindow:
      doTouchWindow();
      break;

    case Demo_ChooseAFolderDialog:
      if(osError = doChooseAFolderDialog())
        doErrorAlert(osError);
      break;
  }
}

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

void  doAdjustMenus(void)
{
  OSErr              osError;
  MenuRef            menuRef;
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;

  if(gCurrentNumberOfWindows > 0)
  {
    if(gRunningOnX)
    {
      if((osError = GetSheetWindowParent(FrontWindow(),&windowRef)) == noErr)
      {
        menuRef = GetMenuRef(mFile);
        DisableMenuCommand(menuRef,File_Close);
        DisableMenuCommand(menuRef,File_Save);
        DisableMenuCommand(menuRef,File_SaveAs);
        DisableMenuCommand(menuRef,File_Revert);
        menuRef = GetMenuRef(mDemonstration);
        DisableMenuCommand(menuRef,Demo_TouchWindow);
        return;
      }
      else
        windowRef = FrontWindow();
    }
    else
      windowRef = FrontWindow();

    if(GetWindowKind(windowRef) == kApplicationWindowKind)
    {
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
      menuRef = GetMenuRef(mFile);
      EnableMenuCommand(menuRef,File_Close);
      if((*docStrucHdl)->windowTouched)
      {
        EnableMenuCommand(menuRef,File_Save);
        EnableMenuCommand(menuRef,File_Revert);
      }
      else
      {
        DisableMenuCommand(menuRef,File_Save);
        DisableMenuCommand(menuRef,File_Revert);
      }

      if(((*docStrucHdl)->pictureHdl != NULL) ||
         ((*(*docStrucHdl)->editStrucHdl)->teLength > 0))
        EnableMenuCommand(menuRef,File_SaveAs);
      else
        DisableMenuCommand(menuRef,File_SaveAs);

      menuRef = GetMenuRef(mDemonstration);

      if(((*docStrucHdl)->pictureHdl != NULL) ||
         ((*(*docStrucHdl)->editStrucHdl)->teLength > 0))
      {
        if((*docStrucHdl)->windowTouched == false)
          EnableMenuCommand(menuRef,Demo_TouchWindow);
        else
          DisableMenuCommand(menuRef,Demo_TouchWindow);
      }
      else
        DisableMenuCommand(menuRef,Demo_TouchWindow);
    }
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuCommand(menuRef,File_Close);
    DisableMenuCommand(menuRef,File_Save);
    DisableMenuCommand(menuRef,File_SaveAs);
    DisableMenuCommand(menuRef,File_Revert);
    menuRef = GetMenuRef(mDemonstration);
    DisableMenuCommand(menuRef,Demo_TouchWindow);
  }

  DrawMenuBar();
}

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

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

  if(errorCode == eInstallHandler)
    GetIndString(errorString,rErrorStrings,1);
  else if(errorCode == eMaxWindows)
    GetIndString(errorString,rErrorStrings,2);
  else if(errorCode == eCantFindFinderProcess)
    GetIndString(errorString,rErrorStrings,3);
  else if(errorCode == opWrErr)
    GetIndString(errorString,rErrorStrings,4);
  else
  {
    GetIndString(errorString,rErrorStrings,5);
    NumToString((SInt32) errorCode,theString);
    doConcatPStrings(errorString,theString);
  }
  if(errorCode != memFullErr)
  {
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  }
  else
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}

// ***************************************************************************** doCopyPString

void  doCopyPString(Str255 sourceString,Str255 destinationString)
{
  SInt16 stringLength;

  stringLength = sourceString[0];
  BlockMove(sourceString + 1,destinationString + 1,stringLength);
  destinationString[0] = stringLength;
}

// ************************************************************************** doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16  appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ***************************************************************************** doTouchWindow

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

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

  SetPortWindowPort(windowRef);

  TextSize(48);
  MoveTo(30,170);
  DrawString("\pWINDOW TOUCHED");
  TextSize(12);

  (*docStrucHdl)->windowTouched = true;

  SetWindowModified(windowRef,true);                                                      /////
}

// ************************************************************************ openAppEventHandler

OSErr  openAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSErr  osError;

  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
    osError = doNewCommand();
  return osError;
}

// ******************************************************************** reopenAppEventHandler

OSErr  reopenAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                              SInt32 handlerRefCon)
{
  OSErr osError;

  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
    if(!FrontWindow())
      osError = doNewCommand();

  return osError;
}

// ************************************************************* openAndPrintDocsEventHandler

OSErr  openAndPrintDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                   SInt32 handlerRefcon)
{
  FSSpec     fileSpec;
  AEDescList docList;
  OSErr      osError, ignoreErr;
  SInt32     index, numberOfItems;
  Size       actualSize;
  AEKeyword  keyWord;
  DescType   returnedType;
  FInfo      fileInfo;

  osError = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osError == noErr)
  {
    osError = doHasGotRequiredParams(appEvent);
    if(osError == noErr)
    {
      osError = AECountItems(&docList,&numberOfItems);
      if(osError == noErr)
      {
        for(index=1;index<=numberOfItems;index++)
        {
          osError = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,
                                &fileSpec,sizeof(fileSpec),&actualSize);
          if(osError == noErr)
          {
            osError = FSpGetFInfo(&fileSpec,&fileInfo);
            if(osError == noErr)
            {
              if(osError = doOpenFile(fileSpec,fileInfo.fdType))
                doErrorAlert(osError);

              if(osError == noErr && handlerRefcon == kPrint)
              {
                // Call printing function here
              }
            }
          }
          else
            doErrorAlert(osError);
        }
      }
    }
    else
      doErrorAlert(osError);

    ignoreErr = AEDisposeDesc(&docList);
  }
  else
    doErrorAlert(osError);

  return osError;
}

// ******************************************************************** quitAppEventHandler

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr              osError;
WindowRef          windowRef, previousWindowRef;
  docStructureHandle docStrucHdl;
  SInt16             touchedWindowsCount = 0;
  SInt16             itemHit;

  osError = doHasGotRequiredParams(appEvent);
  if(osError == noErr)
  {
    if(!gRunningOnX)
    {
      gQuittingApplication = true;
      doCloseCommand(kNavSaveChangesQuittingApplication);
    }
    else
    {
      // …………… if any window has a sheet, bring to front, play system alert sound, and return

      windowRef = GetFrontWindowOfClass(kSheetWindowClass,true);
      if(windowRef)
      {
        SelectWindow(windowRef);
        SysBeep(10);
        return noErr;
      }

      // ………………………………………………………………………………………………………………………………………… count touched windows

      windowRef = FrontWindow();
      do
      {
        docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
        if((*docStrucHdl)->windowTouched == true)
          touchedWindowsCount++;
        previousWindowRef = windowRef;
      } while(windowRef = GetNextWindowOfClass(previousWindowRef,kDocumentWindowClass,true));

      // ……………………………………………………………………………………………………… if no touched windows, simply close down

      if(touchedWindowsCount == 0)
        gDone = true;

      // ……… if one touched window, cause Save Changes alert on that window, close all others

      if(touchedWindowsCount == 1)
        gQuittingApplication = true;

      // …… if more than one touched window, create Review Changes alert, handle button clicks

      else if(touchedWindowsCount > 1)
      {
        itemHit = doReviewChangesAlert(touchedWindowsCount);

        if(itemHit == kAlertStdAlertOKButton)
          gQuittingApplication = true;
        else if(itemHit == kAlertStdAlertCancelButton)
          gQuittingApplication = false;
        else if(itemHit == kAlertStdAlertOtherButton)
```

```
          gDone = true;
      }
    }
  }

  return noErr;
}

// ******************************************************************* doHasGotRequiredParams

OSErr  doHasGotRequiredParams(AppleEvent *appEvent)
{
  DescType returnedType;
  Size     actualSize;
  OSErr    osError;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,
                              NULL,0,&actualSize);
  if(osError == errAEDescNotFound)
    osError = noErr;
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************* doReviewChangesAlert

SInt16  doReviewChangesAlert(SInt16 touchedWindowsCount)
{
  AlertStdCFStringAlertParamRec  paramRec;
  Str255      messageText1 = "\pYou have ";
  Str255      messageText2 = "\p Files documents with unsaved changes. ";
  Str255      messageText3 = "\pDo you want to review these changes before quitting?";
  Str255      countString;
  CFStringRef messageText;
  CFStringRef informativeText =
              CFSTR("If you don't review your documents, all your changes will be lost.");
  DialogRef        dialogRef;
  DialogItemIndex  itemHit;

  NumToString(touchedWindowsCount,countString);
  doConcatPStrings(messageText1,countString);
  doConcatPStrings(messageText1,messageText2);
  doConcatPStrings(messageText1,messageText3);
  messageText = CFStringCreateWithPascalString(NULL,messageText1,CFStringGetSystemEncoding());

  GetStandardAlertDefaultParams(&paramRec,kStdCFStringAlertVersionOne);
  paramRec.movable     = true;
  paramRec.defaultText = CFSTR("Review Changes…");
  paramRec.cancelText  = CFSTR("Cancel");
  paramRec.otherText   = CFSTR("Discard Changes");

  CreateStandardAlert(kAlertStopAlert,messageText,informativeText,&paramRec,&dialogRef);
  RunStandardAlert(dialogRef,NULL,&itemHit);

  if(messageText != NULL)
    CFRelease(messageText);

  return itemHit;
}

// ******************************************************************************************
// NewOpenCloseSave.c
// ******************************************************************************************

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

#include "Files.h"
```

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

NavDialogRef gModalToApplicationNavDialogRef;
SInt16       gCurrentNumberOfWindows = 0;
Rect         gDestRect, gViewRect;
Boolean      gCloseDocWindow = false;

extern NavEventUPP gGetFilePutFileEventFunctionUPP;
extern SInt16      gAppResFileRefNum;
extern Boolean     gDone;
extern Boolean     gQuittingApplication;
extern Boolean     gRunningOnX;

// *************************************************************************** doNewCommand

OSErr  doNewCommand(void)
{
  OSErr     osError = noErr;
  WindowRef windowRef;
  OSType    documentType = kFileTypeTEXT;

  osError = doNewDocWindow(true,documentType,&windowRef);

  if(osError == noErr)
    SetWindowProxyCreatorAndType(windowRef,kFileCreator,kFileTypeTEXT,kUserDomain);      /////

  return osError;
}

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

OSErr  doOpenCommand(void)
{
  OSErr                    osError = noErr;
  NavDialogCreationOptions dialogOptions;
  Str255                   applicationName;
  NavTypeListHandle        fileTypeListHdl = NULL;

  // ........................................................... create application-modal Open dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    GetIndString(applicationName,rMiscStrings,sApplicationName);
    dialogOptions.clientName = CFStringCreateWithPascalString(NULL,applicationName,
                                                       CFStringGetSystemEncoding());
    dialogOptions.modality = kWindowModalityAppModal;
    fileTypeListHdl = (NavTypeListHandle) GetResource('open',rOpenResource);

    osError = NavCreateGetFileDialog(&dialogOptions,fileTypeListHdl,
                                     gGetFilePutFileEventFunctionUPP ,NULL,NULL,NULL,
                                     &gModalToApplicationNavDialogRef);
    if(osError == noErr && gModalToApplicationNavDialogRef != NULL)
    {
      osError = NavDialogRun(gModalToApplicationNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose(gModalToApplicationNavDialogRef);
        gModalToApplicationNavDialogRef = NULL;
      }
    }

    if(dialogOptions.clientName != NULL)
      CFRelease(dialogOptions.clientName);

    if(fileTypeListHdl != NULL)
      ReleaseResource((Handle) fileTypeListHdl);
  }
  return osError;
}

// *************************************************************************** doCloseCommand

OSErr  doCloseCommand(NavAskSaveChangesAction action)
{
  WindowRef          windowRef;
  SInt16             windowKind;
  docStructureHandle docStrucHdl;
  OSErr              osError = noErr;

  windowRef = FrontWindow();
  windowKind = GetWindowKind(windowRef);

  switch(windowKind)
  {
    case kApplicationWindowKind:
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

      // ................................ if window has unsaved changes, create Save Changes alert and return

      if((*docStrucHdl)->windowTouched == true)
      {
        if(IsWindowCollapsed(windowRef))
          CollapseWindow(windowRef,false);

        osError = doCreateAskSaveChangesDialog(windowRef,docStrucHdl,action);
      }

      // ............................................................................................. otherwise close file and clean up

      else
        osError = doCloseDocWindow(windowRef);
      break;

    case kDialogWindowKind:
      // Hide or close modeless dialog, as required.
      break;
  }

  return osError;
}

// **************************************************************************** doSaveCommand

OSErr  doSaveCommand(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  OSErr              osError = noErr;
  Rect               portRect;

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

  // ......... if the document has a file ref number, write the file, otherwise call doSaveAsCommand

  if((*docStrucHdl)->fileRefNum)
  {
    osError = doWriteFile(windowRef);

    SetPortWindowPort(windowRef);
    GetWindowPortBounds(windowRef,&portRect);
    EraseRect(&portRect);
    InvalWindowRect(windowRef,&portRect);
  }
  else
```

```
    osError = doSaveAsCommand();

  if(osError == noErr)                                                        /////
    SetWindowModified(windowRef,false);                                        /////

  return osError;
}

// ************************************************************************** doSaveAsCommand

OSErr  doSaveAsCommand(void)
{
  OSErr                     osError = noErr;
  NavDialogCreationOptions dialogOptions;
  WindowRef                 windowRef;
  Str255                    windowTitle, applicationName;
  docStructureHandle        docStrucHdl;
  OSType                    fileType;

  // .................................................. create window-modal Save Location dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    dialogOptions.optionFlags ^= kNavAllowStationery;

    windowRef = FrontWindow();

    GetWTitle(windowRef,windowTitle);
    dialogOptions.saveFileName = CFStringCreateWithPascalString(NULL,windowTitle,
                                                               CFStringGetSystemEncoding());
    GetIndString(applicationName,rMiscStrings,sApplicationName);
    dialogOptions.clientName = CFStringCreateWithPascalString(NULL,applicationName,
                                                             CFStringGetSystemEncoding());
    dialogOptions.parentWindow = windowRef;
    dialogOptions.modality = kWindowModalityWindowModal;

    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
    if((*docStrucHdl)->editStrucHdl != NULL)
      fileType = kFileTypeTEXT;
    else if((*docStrucHdl)->pictureHdl != NULL)
      fileType = kFileTypePICT;

    osError = NavCreatePutFileDialog(&dialogOptions,fileType,kFileCreator,
                                     gGetFilePutFileEventFunctionUPP ,
                                     windowRef,&(*docStrucHdl)->modalToWindowNavDialogRef);

    if(osError == noErr && (*docStrucHdl)->modalToWindowNavDialogRef != NULL)
    {
      osError = NavDialogRun((*docStrucHdl)->modalToWindowNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
      }
    }

    if(dialogOptions.saveFileName != NULL)
      CFRelease(dialogOptions.saveFileName);
    if(dialogOptions.clientName != NULL)
      CFRelease(dialogOptions.clientName);
  }

  return osError;
}

// ************************************************************************** doRevertCommand

OSErr  doRevertCommand(void)
```

```
{
  OSErr                     osError = noErr;
  NavDialogCreationOptions dialogOptions;
  WindowRef                 windowRef;
  Str255                    windowTitle;
  docStructureHandle        docStrucHdl;

  // ................................................ create window-modal Discard Changes dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    windowRef = FrontWindow();

    GetWTitle(windowRef,windowTitle);
    dialogOptions.saveFileName = CFStringCreateWithPascalString(NULL,windowTitle,
                                                              CFStringGetSystemEncoding());
    dialogOptions.parentWindow = windowRef;
    dialogOptions.modality = kWindowModalityWindowModal;

    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
    if((*docStrucHdl)->askSaveDiscardEventFunctionUPP != NULL)
    {
      DisposeNavEventUPP((*docStrucHdl)->askSaveDiscardEventFunctionUPP);
      (*docStrucHdl)->askSaveDiscardEventFunctionUPP = NULL;
    }
    (*docStrucHdl)->askSaveDiscardEventFunctionUPP =
                            NewNavEventUPP((NavEventProcPtr) askSaveDiscardEventFunction);

    HLock((Handle) docStrucHdl);

    osError = NavCreateAskDiscardChangesDialog(&dialogOptions,
                                              (*docStrucHdl)->askSaveDiscardEventFunctionUPP,
                                              windowRef,
                                              &(*docStrucHdl)->modalToWindowNavDialogRef);
    HUnlock((Handle) docStrucHdl);

    if(osError == noErr && (*docStrucHdl)->modalToWindowNavDialogRef != NULL)
    {
      osError = NavDialogRun((*docStrucHdl)->modalToWindowNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
      }
    }

    if(dialogOptions.saveFileName != NULL)
      CFRelease(dialogOptions.saveFileName);
  }

  return osError;
}

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

OSErr  doNewDocWindow(Boolean showWindow,OSType documentType,WindowRef * windowRef)
{
  docStructureHandle docStrucHdl;
  Rect               portRect;

  if(gCurrentNumberOfWindows == kMaxWindows)
    return eMaxWindows;

  // .......................................... create window and associated document structure

  if(!(*windowRef = GetNewCWindow(rNewWindow,NULL,(WindowRef)-1)))
    return MemError();
  SetPortWindowPort(*windowRef);

  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
  {
    DisposeWindow(*windowRef);
    return MemError();
  }

  SetWRefCon(*windowRef,(SInt32) docStrucHdl);

  // ................................................ initialise fields of document structure

  (*docStrucHdl)->editStrucHdl                   = NULL;
  (*docStrucHdl)->pictureHdl                     = NULL;
  (*docStrucHdl)->fileRefNum                     = 0;
  (*docStrucHdl)->aliasHdl                       = NULL;                                   /////
  (*docStrucHdl)->windowTouched                  = false;
  (*docStrucHdl)->modalToWindowNavDialogRef      = NULL;
  (*docStrucHdl)->askSaveDiscardEventFunctionUPP = NULL;
  (*docStrucHdl)->isAskSaveChangesDialog         = false;

  // ...... if document's file type is to be 'TEXT', associate TextEdit structure with window

  if(documentType == kFileTypeTEXT)
  {
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);

    GetWindowPortBounds(*windowRef,&portRect);
    gDestRect = portRect;
    InsetRect(&gDestRect,6,6);
    gViewRect = gDestRect;

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

    if(!((*docStrucHdl)->editStrucHdl = TENew(&gDestRect,&gViewRect)))
    {
      DisposeWindow(*windowRef);
      DisposeHandle((Handle) docStrucHdl);
      return MemError();
    }

    HUnlock((Handle) docStrucHdl);
  }

  // ................................................ show window and increment window count

  if(showWindow)
    ShowWindow(*windowRef);

  gCurrentNumberOfWindows ++;

  return noErr;
}

// ************************************************************************** doCloseDocWindow

OSErr  doCloseDocWindow(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  // .................. close file, flush volume, dispose of window and associated memory

  if((*docStrucHdl)->fileRefNum != 0)
  {
    if(!(osError = FSClose((*docStrucHdl)->fileRefNum)))
    {
      osError = FlushVol(NULL,(*docStrucHdl)->fileFSSpec.vRefNum);
      (*docStrucHdl)->fileRefNum = 0;
    }
  }

  if((*docStrucHdl)->editStrucHdl != NULL)
    TEDispose((*docStrucHdl)->editStrucHdl);
  if((*docStrucHdl)->pictureHdl != NULL)
    KillPicture((*docStrucHdl)->pictureHdl);

  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);

  gCurrentNumberOfWindows --;

  // .................................................................. if quitting application

  if(gQuittingApplication)
  {
    if(FrontWindow() == NULL)
      gDone = true;
    else
      doCloseCommand(kNavSaveChangesQuittingApplication);
  }

  return osError;
}

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

OSErr  doOpenFile(FSSpec fileSpec,OSType documentType)
{
  WindowRef          windowRef;
  OSErr              osError;
  SInt16             fileRefNum;
  docStructureHandle docStrucHdl;

  // ........................................................................ create new window

  if(osError = doNewDocWindow(false,documentType,&windowRef))
    return osError;

  SetWTitle(windowRef,fileSpec.name);

  // .................................................................... open file's data fork

  if(osError = FSpOpenDF(&fileSpec,fsRdWrPerm,&fileRefNum))
  {
    DisposeWindow(windowRef);
    gCurrentNumberOfWindows --;
    return osError;
  }

  // ........... store file reference number and FSSpec in window's document structure

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  (*docStrucHdl)->fileRefNum = fileRefNum;
  (*docStrucHdl)->fileFSSpec = fileSpec;

  // ......................................................................... read in the file

  if(documentType == kFileTypeTEXT)
  {
    if(osError = doReadTextFile(windowRef))
      return osError;
  }
  else if(documentType == kFileTypePICT)
  {
    if(osError = doReadPictFile(windowRef))
      return osError;
  }

  // .......................................................... set up window's proxy icon, and show window

  SetWindowProxyFSSpec(windowRef,&fileSpec);                                              /////
  GetWindowProxyAlias(windowRef,&((*docStrucHdl)->aliasHdl));                             /////
  SetWindowModified(windowRef,false);                                                     /////

  ShowWindow(windowRef);

  return noErr;
}

// ************************************************************ doCreateAskSaveChangesDialog

OSErr  doCreateAskSaveChangesDialog(WindowRef windowRef,docStructureHandle docStrucHdl,
                                    NavAskSaveChangesAction action)
{
  OSErr                    osError;
  NavDialogCreationOptions dialogOptions;
  Str255                   windowTitle, applicationName;

  // ...................................................... create window-modal Save Changes Changes dialog

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    GetWTitle(windowRef,windowTitle);
    dialogOptions.saveFileName = CFStringCreateWithPascalString(NULL,windowTitle,
                                                               CFStringGetSystemEncoding());
    GetIndString(applicationName,rMiscStrings,sApplicationName);
    dialogOptions.clientName = CFStringCreateWithPascalString(NULL,applicationName,
                                                             CFStringGetSystemEncoding());
    dialogOptions.parentWindow = windowRef;
    dialogOptions.modality = kWindowModalityWindowModal;

    if((*docStrucHdl)->askSaveDiscardEventFunctionUPP != NULL)
    {
      DisposeNavEventUPP((*docStrucHdl)->askSaveDiscardEventFunctionUPP);
      (*docStrucHdl)->askSaveDiscardEventFunctionUPP = NULL;
    }
    (*docStrucHdl)->askSaveDiscardEventFunctionUPP =
                             NewNavEventUPP((NavEventProcPtr) askSaveDiscardEventFunction);

    HLock((Handle) docStrucHdl);

    osError = NavCreateAskSaveChangesDialog(&dialogOptions,action,
                                            (*docStrucHdl)->askSaveDiscardEventFunctionUPP,
                                            windowRef,
                                            &(*docStrucHdl)->modalToWindowNavDialogRef);

    HUnlock((Handle) docStrucHdl);

    if(osError == noErr && (*docStrucHdl)->modalToWindowNavDialogRef != NULL)
    {
      (*docStrucHdl)->isAskSaveChangesDialog = true;

      osError = NavDialogRun((*docStrucHdl)->modalToWindowNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
        (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
        (*docStrucHdl)->isAskSaveChangesDialog = false;
      }

      if(!gRunningOnX)
      {
        if(gCloseDocWindow)
        {
          osError = doCloseDocWindow(windowRef);
          if(osError != noErr)
            doErrorAlert(osError);
          gCloseDocWindow = false;
        }
      }
    }

    if(dialogOptions.saveFileName != NULL)
      CFRelease(dialogOptions.saveFileName);
    if(dialogOptions.clientName != NULL)
      CFRelease(dialogOptions.clientName);
  }

  return osError;
}

// ************************************************************************ doSaveUsingFSSpec

OSErr  doSaveUsingFSSpec(WindowRef windowRef,NavReplyRecord *navReplyStruc)
{
  OSErr              osError = noErr;
  AEKeyword          theKeyword;
  DescType           actualType;
  FSSpec             fileSpec;
  Size               actualSize;
  docStructureHandle docStrucHdl;
  OSType             fileType;
  CFStringRef        fileName;
  SInt16             fileRefNum;
  Rect               portRect;

  if((*navReplyStruc).validRecord)
  {
    // .......................................................................... get FSSpec

    if((osError = AEGetNthPtr(&(*navReplyStruc).selection,1,typeFSS,&theKeyword,
                              &actualType,&fileSpec,sizeof(fileSpec),&actualSize)) == noErr)
    {
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

      // ............ get file name, convert to Pascal string, assign to name field of FSSpec

      fileName = NavDialogGetSaveFileName((*docStrucHdl)->modalToWindowNavDialogRef);
      if(fileName != NULL)
        osError = CFStringGetPascalString(fileName,&fileSpec.name[0],sizeof(FSSpec),
                                          CFStringGetSystemEncoding());

      // ..................................................... if not replacing, first create a new file

      if(!((*navReplyStruc).replacing))
      {
        if((*docStrucHdl)->editStrucHdl != NULL)
          fileType = kFileTypeTEXT;
        else if((*docStrucHdl)->pictureHdl != NULL)
          fileType = kFileTypePICT;

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

      // ................... assign FSSpec to fileFSSpec field of window's document structure
      (*docStrucHdl)->fileFSSpec = fileSpec;

      // ........................................................... if file currently exists for document, close it

      if((*docStrucHdl)->fileRefNum != 0)
      {
        osError = FSClose((*docStrucHdl)->fileRefNum);
        (*docStrucHdl)->fileRefNum = 0;
      }

      // ............................................................................ open file's data fork and write file

      if(osError == noErr)
        osError = FSpOpenDF(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm,&fileRefNum);

      if(osError == noErr)
      {
        (*docStrucHdl)->fileRefNum = fileRefNum;
        SetWTitle(windowRef,fileSpec.name);

        // … … … … … … … … … … … … … … … … … … … … … … proxy icon and file synchronisation stuff

        SetPortWindowPort(windowRef);                                         /////
        SetWindowProxyFSSpec(windowRef,&fileSpec);                            /////
        GetWindowProxyAlias(windowRef,&((*docStrucHdl)->aliasHdl));           /////
        SetWindowModified(windowRef,false);                                   /////

        // … … … … … … … … … … … … … … … … … … … … … … … … … … … …  write file using safe save

        osError = doWriteFile(windowRef);

        NavCompleteSave(&(*navReplyStruc),kNavTranslateInPlace);
      }
    }
  }

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

  return osError;
}

// ************************************************************************ doSaveUsingFSRef

OSErr  doSaveUsingFSRef(WindowRef windowRef,NavReplyRecord *navReplyStruc)
{
  OSErr              osError = noErr;
  AEDesc             aeDesc;
  Size               dataSize;
  FSRef              fsRefParent, fsRefDelete;
  UniCharCount       nameLength;
  UniChar            *nameBuffer;
  FSSpec             fileSpec;
  docStructureHandle docStrucHdl;
  FInfo              fileInfo;
  SInt16             fileRefNum;
  Rect               portRect;

  osError = AECoerceDesc(&(*navReplyStruc).selection,typeFSRef,&aeDesc);
  if(osError == noErr)
  {
    // ..................................................................................................... get FSRef

    dataSize = AEGetDescDataSize(&aeDesc);
    if(dataSize > 0)
      osError = AEGetDescData(&aeDesc,&fsRefParent,sizeof(FSRef));
    if(osError == noErr)
```

```
{
  // ............................................................ get file name from saveFileName field of NavReplyRecord

  nameLength = (UniCharCount) CFStringGetLength((*navReplyStruc).saveFileName);
  nameBuffer = (UniChar *) NewPtr(nameLength);
  CFStringGetCharacters((*navReplyStruc).saveFileName,CFRangeMake(0,nameLength),
                        &nameBuffer[0]);
  if(nameBuffer != NULL)
  {
    // ............................................................ if replacing, delete the file being replaced

    if((*navReplyStruc).replacing)
    {
      osError = FSMakeFSRefUnicode(&fsRefParent,nameLength,nameBuffer,
                                   kTextEncodingUnicodeDefault,&fsRefDelete);
      {
        if(osError == noErr)
          osError = FSDeleteObject(&fsRefDelete);
        if(osError == fBsyErr)
        {
          DisposePtr((Ptr) nameBuffer);
          return osError;
        }
      }
    }

    // ...................... create file with Unicode name (but it can be written with an FSSpec)

    if(osError == noErr)
    {
      osError = FSCreateFileUnicode(&fsRefParent,nameLength,nameBuffer,kFSCatInfoNone,
                                    NULL,NULL,&fileSpec);
      if(osError == noErr)
      {
        docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

        osError = FSpGetFInfo(&fileSpec,&fileInfo);

        if((*docStrucHdl)->editStrucHdl != NULL)
          fileInfo.fdType = kFileTypeTEXT;
        else if((*docStrucHdl)->pictureHdl != NULL)
          fileInfo.fdType = kFileTypePICT;
        fileInfo.fdCreator = kFileCreator;

        if(osError == noErr)
          osError = FSpSetFInfo(&fileSpec,&fileInfo);

        (*docStrucHdl)->fileFSSpec = fileSpec;

        // ............................................................ open file's data fork and write file

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

        if(osError == noErr)
        {
          (*docStrucHdl)->fileRefNum = fileRefNum;
          SetWTitle(windowRef,fileSpec.name);

          // … … … … … … … … … … … … … … … … … … proxy icon and file synchronisation stuff

          SetPortWindowPort(windowRef);                                        /////
          SetWindowProxyFSSpec(windowRef,&fileSpec);                           /////
          GetWindowProxyAlias(windowRef,&((*docStrucHdl)->aliasHdl));          /////
          SetWindowModified(windowRef,false);                                  /////

          // … … … … … … … … … … … … … … … … … … … … … … … … …  write file using safe save

          osError = doWriteFile(windowRef);
```

```
              NavCompleteSave(&(*navReplyStruc),kNavTranslateInPlace);
            }
          }
        }
      }

      DisposePtr((Ptr) nameBuffer);
    }

    AEDisposeDesc(&aeDesc);
  }

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

  return osError;
}

// ******************************************************************************** doWriteFile

OSErr  doWriteFile(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  FSSpec             fileSpecActual, fileSpecTemp;
  UInt32             currentTime;
  Str255             tempFileName;
  SInt16             tempFileVolNum, tempFileRefNum;
  SInt32             tempFileDirID;
  OSErr              osError = noErr;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileSpecActual = (*docStrucHdl)->fileFSSpec;

  GetDateTime(&currentTime);
  NumToString((SInt32) currentTime,tempFileName);

  osError = FindFolder(fileSpecActual.vRefNum,kTemporaryFolderType,kCreateFolder,
                       &tempFileVolNum,&tempFileDirID);
  if(osError == noErr)
    osError = FSMakeFSSpec(tempFileVolNum,tempFileDirID,tempFileName,&fileSpecTemp);
  if(osError == noErr || osError == fnfErr)
    osError = FSpCreate(&fileSpecTemp,'trsh','trsh',smSystemScript);
  if(osError == noErr)
    osError = FSpOpenDF(&fileSpecTemp,fsRdWrPerm,&tempFileRefNum);
  if(osError == noErr)
  {
    if((*docStrucHdl)->editStrucHdl != NULL)
      osError = doWriteTextData(windowRef,tempFileRefNum);
    else if((*docStrucHdl)->pictureHdl != NULL)
      osError = doWritePictData(windowRef,tempFileRefNum);
  }
  if(osError == noErr)
    osError = FSClose(tempFileRefNum);
  if(osError == noErr)
    osError = FSClose((*docStrucHdl)->fileRefNum);
  if(osError == noErr)
    osError = FSpExchangeFiles(&fileSpecTemp,&fileSpecActual);
  if(osError == noErr)
    osError = FSpDelete(&fileSpecTemp);
  if(osError == noErr)
    osError = FSpOpenDF(&fileSpecActual,fsRdWrPerm,&(*docStrucHdl)->fileRefNum);

  if(osError == noErr)
    osError = doCopyResources(windowRef);

  return osError;
```

```
}

// ************************************************************************** doReadTextFile

OSErr  doReadTextFile(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  TEHandle           textEditHdl;
  SInt32             numberOfBytes;
  Handle             textBuffer;
  OSErr              osError;

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

  textEditHdl = (*docStrucHdl)->editStrucHdl;
  (*textEditHdl)->txSize = 10;
  (*textEditHdl)->lineHeight = 15;

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

  if(numberOfBytes > 32767)
    numberOfBytes = 32767;

  if(!(textBuffer = NewHandle((Size) numberOfBytes)))
    return MemError();

  osError = FSRead(fileRefNum,&numberOfBytes,*textBuffer);
  if(osError == noErr || osError == eofErr)
  {
    HLockHi(textBuffer);
    TESetText(*textBuffer,numberOfBytes,(*docStrucHdl)->editStrucHdl);
    HUnlock(textBuffer);
    DisposeHandle(textBuffer);
  }
  else
    return osError;

  return noErr;
}

// ************************************************************************** doReadPictFile

OSErr  doReadPictFile(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  SInt32             numberOfBytes;
  OSErr              osError;

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

  GetEOF(fileRefNum,&numberOfBytes);
  SetFPos(fileRefNum,fsFromStart,512);
  numberOfBytes -= 512;

  if(!((*docStrucHdl)->pictureHdl = (PicHandle) NewHandle(numberOfBytes)))
    return MemError();

  osError = FSRead(fileRefNum,&numberOfBytes,*(*docStrucHdl)->pictureHdl);
  if(osError == noErr || osError == eofErr)
    return(noErr);
  else
    return osError;
}
```

```
// ************************************************************************ doWriteTextData

OSErr  doWriteTextData(WindowRef windowRef,SInt16 tempFileRefNum)
{
  docStructureHandle docStrucHdl;
  TEHandle           textEditHdl;
  Handle             editText;
  SInt32             numberOfBytes;
  SInt16             volRefNum;
  OSErr              osError;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  textEditHdl = (*docStrucHdl)->editStrucHdl;
  editText = (*textEditHdl)->hText;
  numberOfBytes = (*textEditHdl)->teLength;

  osError = SetFPos(tempFileRefNum,fsFromStart,0);
  if(osError == noErr)
    osError = FSWrite(tempFileRefNum,&numberOfBytes,*editText);
  if(osError == noErr)
    osError = SetEOF(tempFileRefNum,numberOfBytes);
  if(osError == noErr)
    osError = GetVRefNum(tempFileRefNum,&volRefNum);
  if(osError == noErr)
    osError = FlushVol(NULL,volRefNum);

  if(osError == noErr)
    (*docStrucHdl)->windowTouched = false;

  return osError;
}

// ************************************************************************ doWritePictData

OSErr  doWritePictData(WindowRef windowRef,SInt16 tempFileRefNum)
{
  docStructureHandle docStrucHdl;
  PicHandle          pictureHdl;
  SInt32             numberOfBytes, dummyData;
  SInt16             volRefNum;
  OSErr              osError;

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

  numberOfBytes = 512;
  dummyData = 0;

  osError = SetFPos(tempFileRefNum,fsFromStart,0);

  if(osError == noErr)
    osError = FSWrite(tempFileRefNum,&numberOfBytes,&dummyData);

  numberOfBytes = GetHandleSize((Handle) (*docStrucHdl)->pictureHdl);

  if(osError == noErr)
  {
    HLock((Handle) (*docStrucHdl)->pictureHdl);
    osError = FSWrite(tempFileRefNum,&numberOfBytes,*(*docStrucHdl)->pictureHdl);
    HUnlock((Handle) (*docStrucHdl)->pictureHdl);
  }

  if(osError == noErr)
    osError = SetEOF(tempFileRefNum,512 + numberOfBytes);
  if(osError == noErr)
    osError = GetVRefNum(tempFileRefNum,&volRefNum);
  if(osError == noErr)
    osError = FlushVol(NULL,volRefNum);
  if(osError == noErr)
    (*docStrucHdl)->windowTouched = false;

  return osError;
}

// ************************************************************* getFilePutFileEventFunction

void  getFilePutFileEventFunction(NavEventCallbackMessage callBackSelector,
                                  NavCBRecPtr callBackParms,NavCallBackUserData callBackUD)
{
  OSErr              osError;
  NavReplyRecord     navReplyStruc;
  NavUserAction      navUserAction;
  SInt32             count, index;
  AEKeyword          theKeyword;
  DescType           actualType;
  FSSpec             fileSpec;
  Size               actualSize;
  FInfo              fileInfo;
  OSType             documentType;
  WindowRef          windowRef;
  AEDesc             aeDesc;
  AEKeyword          keyWord;
  DescType           typeCode;
  Rect               theRect;
  Str255             theString, numberString;
  docStructureHandle docStrucHdl;

  switch(callBackSelector)
  {
    case kNavCBUserAction:
      osError = NavDialogGetReply(callBackParms->context,&navReplyStruc);
      if(osError == noErr && navReplyStruc.validRecord)
      {
        navUserAction = NavDialogGetUserAction(callBackParms->context);

        switch(navUserAction)
        {
          // ........................................................................ click on Open button in Open dialog

          case kNavUserActionOpen:
            if(gModalToApplicationNavDialogRef != NULL)
            {
              osError = AECountItems(&(navReplyStruc.selection),&count);
              if(osError == noErr)
              {
                for(index=1;index<=count;index++)
                {
                  osError = AEGetNthPtr(&(navReplyStruc.selection),index,typeFSS,
                                        &theKeyword,&actualType,&fileSpec,sizeof(fileSpec),
                                        &actualSize);
                  if((osError = FSpGetFInfo(&fileSpec,&fileInfo)) == noErr)
                  {
                    documentType = fileInfo.fdType;
                    osError = doOpenFile(fileSpec,documentType);
                    if(osError != noErr)
                      doErrorAlert(osError);
                  }
                }
              }
            }
            break;

          // ............................................................... click on Save button in Save Location dialog

          case kNavUserActionSaveAs:
            windowRef = callBackUD;
            osError = AECoerceDesc(&navReplyStruc.selection,typeFSRef,&aeDesc);
            if(osError == noErr)
            {
              osError = doSaveUsingFSRef(windowRef,&navReplyStruc);
              if(osError != noErr)
                doErrorAlert(osError);
              AEDisposeDesc(&aeDesc);
            }
            else
            {
              osError = doSaveUsingFSSpec(windowRef,&navReplyStruc);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            break;

          // .................................................. click on Choose button in Choose a Folder dialog

          case kNavUserActionChoose:
            if((osError = AEGetNthPtr(&(navReplyStruc.selection),1,typeFSS,&keyWord,&typeCode,
                                      &fileSpec,sizeof(FSSpec),&actualSize)) == noErr)
            {
              FSMakeFSSpec(fileSpec.vRefNum,fileSpec.parID,fileSpec.name,&fileSpec);
            }
            windowRef = callBackUD;
            SetPortWindowPort(windowRef);
            TextSize(10);
            SetRect(&theRect,0,271,600,300);
            EraseRect(&theRect);
            doCopyPString(fileSpec.name,theString);
            doConcatPStrings(theString, "\p   Volume Reference Number: ");
            NumToString((SInt32) fileSpec.vRefNum,numberString);
            doConcatPStrings(theString,numberString);
            doConcatPStrings(theString, "\p   Parent Directory ID: ");
            NumToString((SInt32) fileSpec.parID,numberString);
            doConcatPStrings(theString,numberString);
            MoveTo(10,290);
            DrawString(theString);
            break;
        }

        osError = NavDisposeReply(&navReplyStruc);
      }
      break;

    case kNavCBTerminate:
      if(gModalToApplicationNavDialogRef != NULL)
      {
        NavDialogDispose(gModalToApplicationNavDialogRef);
        gModalToApplicationNavDialogRef = NULL;
      }
      else
      {
        windowRef = callBackUD;
        docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
        if((*docStrucHdl)->modalToWindowNavDialogRef != NULL)
        {
          NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
          (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
        }
      }
      break;

    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;
  }
}

// ************************************************************** askSaveDiscardEventFunction

void askSaveDiscardEventFunction(NavEventCallbackMessage callBackSelector,
                                 NavCBRecPtr callBackParms,NavCallBackUserData callBackUD)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  NavUserAction      navUserAction;
  Rect               portRect;
  OSErr              osError;

  switch(callBackSelector)
  {
    case kNavCBUserAction:
      windowRef = callBackUD;
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
      if((*docStrucHdl)->modalToWindowNavDialogRef != NULL)
      {
        navUserAction = NavDialogGetUserAction(callBackParms->context);
        switch(navUserAction)
        {
          // .................................................. click on Save button in Save Changes dialog

          case kNavUserActionSaveChanges:
            doSaveCommand();

          // ........................................ click on Don't Save button in Save Changes dialog

          case kNavUserActionDontSaveChanges:
            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            if(gRunningOnX)
            {
              osError = doCloseDocWindow(windowRef);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            else
              gCloseDocWindow = true;
            break;

          // ............................................... click on OK button in Discard Changes dialog

          case kNavUserActionDiscardChanges:
            GetWindowPortBounds(windowRef,&portRect);
            SetPortWindowPort(windowRef);
            EraseRect(&portRect);

            if((*docStrucHdl)->editStrucHdl != NULL && (*docStrucHdl)->fileRefNum != 0)
            {
              osError = doReadTextFile(windowRef);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            else if((*docStrucHdl)->pictureHdl != NULL)
            {
              KillPicture((*docStrucHdl)->pictureHdl);
              (*docStrucHdl)->pictureHdl = NULL;

              osError = doReadPictFile(windowRef);
              if(osError != noErr)
                doErrorAlert(osError);
            }
            (*docStrucHdl)->windowTouched = false;
            SetWindowModified(windowRef,false);                                     /////
            InvalWindowRect(windowRef,&portRect);

            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
            break;

          // ............................ click on Cancel button in Save Changes or Discard Changes dialog

          case kNavUserActionCancel:
            if((*docStrucHdl)->isAskSaveChangesDialog == true)
            {
              gQuittingApplication = false;
              (*docStrucHdl)->isAskSaveChangesDialog = false;
            }
            NavDialogDispose((*docStrucHdl)->modalToWindowNavDialogRef);
            (*docStrucHdl)->modalToWindowNavDialogRef = NULL;
            break;
        }
      }
      break;

    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;
  }
}

// ************************************************************************** doCopyResources

OSErr  doCopyResources(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  OSType             fileType;
  OSErr              osError = noErr;
  SInt16             fileRefNum;
  Handle             editTextHdl, textResourceHdl;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

  if((*docStrucHdl)->editStrucHdl)
    fileType = kFileTypeTEXT;
  else if((*docStrucHdl)->pictureHdl)
    fileType = kFileTypePICT;

  FSpCreateResFile(&(*docStrucHdl)->fileFSSpec,kFileCreator,fileType,smSystemScript);

  osError = ResError();
  if(osError == noErr)
    fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);

  if(fileRefNum > 0)
  {
    osError = doCopyAResource('STR ',-16396,gAppResFileRefNum,fileRefNum);

    if(fileType == kFileTypePICT)
    {
      doCopyAResource('pnot',128,gAppResFileRefNum,fileRefNum);
      doCopyAResource('PICT',128,gAppResFileRefNum,fileRefNum);
    }
    if(!gRunningOnX && fileType == kFileTypeTEXT)
    {
      doCopyAResource('pnot',129,gAppResFileRefNum,fileRefNum);

      editTextHdl = (*(*docStrucHdl)->editStrucHdl)->hText;
      textResourceHdl = NewHandleClear(1024);
      BlockMoveData(*editTextHdl,*textResourceHdl,1024);
      UseResFile(fileRefNum);
      AddResource(textResourceHdl,'TEXT',129,"\p");
      if(ResError() == noErr)
        UpdateResFile(fileRefNum);
      ReleaseResource(textResourceHdl);
    }
  }
  else
    osError = ResError();

  if(osError == noErr)
    CloseResFile(fileRefNum);

  osError = ResError();
  return osError;
}

// ************************************************************************ doCopyAResource

OSErr  doCopyAResource(ResType resourceType,SInt16 resourceID,SInt16 sourceFileRefNum,
                      SInt16 destFileRefNum)
{
  Handle  sourceResourceHdl;
  Str255  sourceResourceName;
  ResType ignoredType;
  SInt16  ignoredID;

  UseResFile(sourceFileRefNum);

  sourceResourceHdl = GetResource(resourceType,resourceID);

  if(sourceResourceHdl != NULL)
  {
    GetResInfo(sourceResourceHdl,&ignoredID,&ignoredType,sourceResourceName);
    DetachResource(sourceResourceHdl);
    UseResFile(destFileRefNum);
    AddResource(sourceResourceHdl,resourceType,resourceID,sourceResourceName);
    if(ResError() == noErr)
      UpdateResFile(destFileRefNum);
  }

  ReleaseResource(sourceResourceHdl);

  return ResError();
}

// ***************************************************************************************
// SynchroniseFiles.c
// ***************************************************************************************

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

#include "Files.h"

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

extern SInt16 gCurrentNumberOfWindows;

// ********************************************************************* doSynchroniseFiles

void  doSynchroniseFiles(void)
{
  UInt32             currentTicks;
  WindowRef          windowRef;
  SInt16             trashVRefNum;
  SInt32             trashDirID;
  static UInt32      nextSynchTicks = 0;
  docStructureHandle docStrucHdl;
  Boolean            aliasChanged;
  AliasHandle        aliasHdl;
  FSSpec             newFSSpec;
  OSErr              osError;

  currentTicks = TickCount();
  windowRef     = FrontNonFloatingWindow();

  if(currentTicks > nextSynchTicks)
  {
    while(windowRef != NULL)
    {
      docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

      if(docStrucHdl != NULL)
      {
        if((*docStrucHdl)->aliasHdl == NULL)
          break;

        aliasChanged = false;
        aliasHdl = (*docStrucHdl)->aliasHdl;
        ResolveAlias(NULL,aliasHdl,&newFSSpec,&aliasChanged);

        if(aliasChanged)
        {
          (*docStrucHdl)->fileFSSpec = newFSSpec;
          SetWTitle(windowRef,newFSSpec.name);
        }

        osError = FindFolder(kUserDomain,kTrashFolderType,kDontCreateFolder,
                             &trashVRefNum,&trashDirID);

        if(osError == noErr)
        {
          do
          {
            if(newFSSpec.parID == fsRtParID)
              break;

            if((newFSSpec.vRefNum == trashVRefNum) && (newFSSpec.parID == trashDirID))
            {
              FSClose((*docStrucHdl)->fileRefNum);
              if((*docStrucHdl)->editStrucHdl)
                TEDispose((*docStrucHdl)->editStrucHdl);
              if((*docStrucHdl)->pictureHdl)
                KillPicture((*docStrucHdl)->pictureHdl);
              DisposeHandle((Handle) docStrucHdl);
              DisposeWindow(windowRef);
              gCurrentNumberOfWindows --;
              break;
            }
          } while(FSMakeFSSpec(newFSSpec.vRefNum,newFSSpec.parID,"\p",&newFSSpec) == noErr);
        }
      }

      windowRef = GetNextWindow(windowRef);
    }

    nextSynchTicks = currentTicks + 15;
  }
}

// *******************************************************************************************
// ChooseAFolderDialog.c
// *******************************************************************************************

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

#include "Files.h"

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

extern NavEventUPP  gGetFilePutFileEventFunctionUPP ;
extern NavDialogRef gModalToApplicationNavDialogRef;

// ******************************************************************** doChooseAFolderDialog

OSErr  doChooseAFolderDialog(void)
{
  OSErr                    osError = noErr;
  NavDialogCreationOptions dialogOptions;
  WindowRef                windowRef, parentWindowRef;
  Str255                   message;

  osError = NavGetDefaultDialogCreationOptions(&dialogOptions);
  if(osError == noErr)
  {
    if((osError = GetSheetWindowParent(FrontWindow(),&parentWindowRef)) == noErr)
      windowRef = parentWindowRef;
    else
      windowRef = FrontWindow();

    GetIndString(message,rMiscStrings,sChooseAFolder);
    dialogOptions.message = CFStringCreateWithPascalString(NULL,message,
                                                           CFStringGetSystemEncoding());
    dialogOptions.modality = kWindowModalityAppModal;

    osError = NavCreateChooseFolderDialog(&dialogOptions,gGetFilePutFileEventFunctionUPP ,
                                          NULL,windowRef,&gModalToApplicationNavDialogRef);

    if(osError == noErr && gModalToApplicationNavDialogRef != NULL)
    {
      osError = NavDialogRun(gModalToApplicationNavDialogRef);
      if(osError != noErr)
      {
        NavDialogDispose(gModalToApplicationNavDialogRef);
        gModalToApplicationNavDialogRef = NULL;
      }
    }
  }

  return osError;
}

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

Demonstration Program Files Comments

When the program is run, the user should:

- Exercise the File menu by opening the supplied TEXT and PICT files, saving those files, saving those files under new names, closing files, opening the new files, attempting to open files that are already open, attempting to save files to new files with existing names, making open windows "touched" by choosing the first item in the Demonstration menu, reverting to the saved versions of files associated with "touched" windows, choosing Quit when one "touched" window is open, choosing Quit when two or more "touched" windows are open, and so on.

- Choose, via the Show pop-up menu button, the file types required to be displayed in the Open dialog.

- Choose the Choose a Folder item from the Demonstration menu to display the Choose a Folder dialog, and choose a folder using the Choose button at the bottom of the dialog. (The name of the chosen folder will be drawn in the bottom-left corner of the front window.)

- With either the PICT Document or the TEXT Document open:

  - With the document's Finder icon visible, drag the window proxy icon to the desktop or to another open folder, noting that the Finder icon moves to the latter. Then choose Touch Window from the Demonstration menu to simulate unsaved changes to the document. Note that the proxy icon changes to the disabled state. Then save the file, proving the correct operation of the file synchronisation function. Note that, after the save, the window proxy icon changes back to the enabled state.

  - Command-click the window's title to display the window path pop-up menu, choose a folder from the menu, and note that the Finder is brought to the foreground and the chosen folder opens.

The program may be run from within CodeWarrior to demonstrate responses to the File menu commands and the Choose a Folder dialog.

The built application, together with the supplied 'TEXT' and 'PICT' files, may be used to demonstrate the additional aspect of integrating the receipt of required Apple events with the overall file handling mechanism. To prove the correct handling of the required Apple events, the user should:

- Open the application by double-clicking the application icon, noting that a new document window is opened after the application is launched and the Open Application event is received.

- Double click on a document icon, or select one or more document icons and either drag those icons to the application icon or choose Open from the Finder's File menu, noting that the application is launched and the selected files are opened when the Open Documents event is received.

- Close all windows and double-click the application icon, noting that the application responds to the Re-open Application event by opening a new window.

- With the PICT Document and the TEXT Document open and "touched", and several other windows open, choose Restart or Shut Down from the Mac OS 8/9 Finder's Special menu or the Mac OS X Apple menu (thus invoking a Quit Application event), noting that, for "touched" windows, the Save Changes alert is presented asking the user whether the file should be saved before the shutdown process proceeds. (On Mac OS X, a Review Unsaved alert will be presented at first.)

Note, however, that no printing functions are included. Thus, selecting one or more document icons and choosing Print from the Finder's File menu (Mac OS 8/9) will result in the file/s opening but not printing.

Files.h

defines

After the usual constants relating to menus, windows, and alerts are established, additional constants are established for a 'STR#' resource containing error strings, three specific error conditions, a 'STR#' resource containing the application's name and the message string for the Choose a Folder dialog, and the 'open' resource containing the file types list. kMaxWindows is used to limit the number of windows the user can open.

kFileCreator represents the application's signature and the next two constants represent the file types that are readable and writeable by the application.

typedefs

Each window created by the program will have an associated document structure. The docStructure data type will be used for document structures.

The editStrucHdl field will be assigned a handle to a TextEdit structure ('TEXT' files). The pictureHdl field will be assigned a handle to a Picture structure ('PICT' files). The fileRefNum and fileFSSpec fields will be assigned the file reference number and the file system specification structure of the file associated with the window. When a file is opened, the aliasHdl field will be assigned a handle to a structure of type AliasRecord, which contains the alias data for the file. The windowTouched field will be set to true when a window has been made "touched".

When modal-to-the-window Navigation Services dialogs (Save Location, Save Changes, and Discard Changes alerts) are created, the dialog reference will be assigned to the modalToWindowNavDialogRef field. When Save Changes and Discard Changes alerts are created, a universal procedure pointer to the associated event (callback) function will be assigned to the askSaveDiscardChangesDialog field. When a Save Changes alert is created, the isAskSaveChangesDialog field will be set to true to enable the associated event (callback) function to re-set a "quitting application" flag if the user clicks the Cancel button in a Save Changes alert (but not if the user clicks the Cancel button in a Discard Changes alert).

Files.c

Global Variables

gAppResFileRefNum will be assigned the file reference number of the application's resource fork. gGetFilePutFileEventFunctionUPP will be assigned a universal procedure pointer to the event (callback) function associated with the Open, Save Location, and Choose a Folder dialogs. gQuittingApplication is set to true in certain circumstances within quitAppEventHandler and to false if the Cancel button is clicked in a Save Changes or Review Unsaved alert.

main

The file reference number of the application's resource fork (which is opened automatically at application launch) is assigned to the global variable gAppResFileRefNum.

After the required Apple event handlers are installed, a universal procedure pointer to the event (callback) function associated with the Open, Save Location, and Choose a Folder dialogs is created and assigned to the global variable gGetFilePutFileEventFunctionUPP.

eventLoop

When WaitNextEvent returns 0, the function doSynchroniseFiles is called.

When the applicationis running on Mac OS X, the global variable gQuittingApplication is set to true in certain circumstances in the function quitAppEventHandler, and to false if the user clicks the Cancel button in a Save Changes or Review Unsaved alert. When set to true, gQuittingApplication causes a block of code at the bottom of the function doCloseDocWindow to execute. This code causes all windows with no unsaved changes to be closed and a Save Changes alert to be presented for windows with unsaved changes.

doInstallAEHandlers

doInstallAEHandlers installs handlers for the Open Application, Re-Open Application, Open Documents, Print Documents, and Quit Application events.

doMouseDown

Note that, in the inGoAway case, the constant kNavSaveChangesClosingDocument is passed in the call to doCloseCommand. This affects the text in the Save Changes dialog.

The inProxyIcon and inDrag cases recognise that a mouse-down in the proxy icon region can mean that the user wishes to drag the proxy icon (icon enabled), drag the window (icon disabled) or display the window path pop-up menu (Command key down, icon enabled or disabled).

If the proxy icon is enabled and the mouse-down is in the proxy icon, TrackWindowProxyDrag handles all aspects of the proxy icon drag process while the user drags the icon and returns noErr, in which case the local variable handled is assigned true and execution drops through to the break at the bottom of the inDrag case.

If TrackWindowProxyDrag returns errUserWantsToDragWindow, the user is either dragging the window (proxy icon is disabled) or displaying the window path pop-up menu. In this case, the local variable handled is assigned false and execution falls through to the IsWindowPathSelectClick call inside the inDrag case.

IsWindowPathSelectClick reports whether the mouse-down should activate the window path pop-up menu. If this call returns true, WindowPathSelect displays the pop-up menu, returning the item selected in the third parameter. If the menu item chosen is not the title of the window itself, the function doBringFinderToFront is called to make the Finder the frontmost process, thus ensuring that the window being opened will be visible to the user. The local variable handled is then set to true so that DragWindow will not be called.

If IsWindowPathSelectClick returns false, DragWindow is called to take control of the dragging operation.

doBringFinderToFront

doBringFinderToFront is called from the inDrag case in doMouseDown. It makes the Finder the frontmost process.

The call to the function doFindProcess gets the process serial number of the Finder, which is then passed in a call to SetFrontProcess.

doFindProcess

doFindProcess is called by doBringFinderToFront to get the process serial number of the Finder.

The first two lines initialise the fields of a process serial number structure so that the search starts from kNoProcess. The next block initialises the fields of a process information structure which, amongst other things, contains fields for the signature (creator) and file type of the application file. (The search is for the signature (creator) 'MACS' and the type 'FNDR'.)

Within the while loop, GetNextProcess is called to get the process serial number of the next process. The call to GetProcessInformation gets information about this process into the process information structure. The processSignature and processType fields of process information structure are then examined. If they equal 'MACS' and 'FNDR' respectively, the while loop exits and the process serial number is assigned to the formal parameter outProcSerNo.

doUpdate

doUpdate performs such window updating as is necessary for the satisfactory execution of the demonstration aspects of the program.

doCommand

At the File_Close case, kNavSaveChangesClosingDocument is passed in the call to doCloseCommand. This affects the wording in the Save Changes dialog. At the File_Quit case, kNavSaveChangesQuittingApplication is passed in the call to doQuitCommand. This also affects the wording in the Save Changes dialog. Note that the global gQuittingApplication is set to true at the File_Quit case.

At the Demo_ChooseAFolderDialog case, doChooseAFolderDialog is called to present the Choose a Folder dialog.

doErrorAlert

doErrorAlert handles errors, invoking an appropriate alert (caution or stop) advising of the nature of the problem by error code number or straight text. Note that the program will only be terminated in the case of the memFullErr error (no more space in the application heap).

doTouchWindow

doTouchWindow is called when the user chooses the Touch Window item in the Demonstration menu. Changing the content of the in-memory version of a file is only simulated in this program. The text "WINDOW TOUCHED" is drawn in window and the windowTouched field of the document structure is set to true.

SetWindowModified is called with true passed in the modified parameter. This causes the proxy icon to appear in the disabled state, indicating that the window has unsaved changes.

openAppEventHandler, reopenAppEventHandler, openAndPrintDocsEventHandler

The handlers for the required Apple events are essentially identical to those in the demonstration program AppleEvents. One major difference is that one handler (openAndPrintDocsEventHandler) is used for both the Open Documents and Print Documents events, with a value passed in the handler's handlerRefcon parameter advising the handler which of the two events has been received.

Most programs should simply open a new untitled window on receipt of an Open Application event. Accordingly, openAppEventHandler simply calls the same function (doNewCommand) as is called when the user chooses New from the File menu.

On receipt of a Re-Open Application event, if no windows are currently open, doNewCommand is called to open a window.

The demonstration program supports both 'TEXT' and 'PICT' files. On receipt of an Open Application event, it is thus necessary to determine the type of each file specified in the event. Accordingly, within openAndPrintDocsEventHandler, the call to FSpGetFInfo returns the Finder information from the volume catalog entry for the file relating to the specified FSSpec structure. The fdType field of the FInfo structure "filled-in" by FSpGetFInfo contains the file type. This, together with the FSSpec structure, is then passed in the call to doOpenFile. (doOpenFile is also called when the user chooses Open from the File menu.)

quitAppEventHandler

In the function quitAppEventHandler, if the program is running on Mac OS 8/9, the global variable gQuittingApplication is set to true before doCloseCommand is called with kNavSaveChangesQuittingApplication passed in. The latter affects the wording in the Save Changes alert. This initiates a sequence in which all windows and the program are progressively closed down, a sequence which in interrupted only if the user clicks the Cancel button in a Save Changes alert.

If the program is running on Mac OS X, the following occurs:

- GetFrontWindowOfClass is called to determine whether any window has a sheet. If so, that window is brought to the front and activated and the handler returns immediately, keeping the application alive.

- The do-while loop walks the window list counting the number of document windows with unsaved changes (that is, "touched" windows)) and, at the same time, bringing those windows to the front. At the next block, if there are no touched document windows, gDone is set to true to close the application down.

- If there is only one touched window open, the flag gQuittingApplication is set to true. As will be see, this results in a sequence involving doCloseCommand and doCloseDocWindow whereby all untouched windows in front of the touched window are disposed of and a Save Changes alert is presented for the touched window. In this sequence, if the event handler for the Save Changes alert detects a Cancel button click, gQuittingApplication will be set to false, an action which will cause the process of closing down the remaining windows and the application to be terminated. If the Save or Don't Save buttons are clicked, all remaining windows will be closed down, and gDone will be set to true within the function doCloseDocWindow, causing the program to be closed down.

- If more than one window has been touched, doReviewChangesAlert is called to create, display and handle a Review Changes alert. If the Review Changes… button is hit, the flag gQuittingApplication is set to true, causing doCloseCommand to be called from within doEvents and resulting in the same general process of close-down, and possible termination of that close-down process, described above. If the Cancel button is hit, the flag gQuittingApplication is set to false (which defeats the execution of the last block of code in doCloseDocWindow) and quitAppEventHandler simply returns. If the Discard Changes button is hit, gDone is set to true to terminate the program.

NewOpenCloseSave.c

Global Variables

gModalToApplicationNavDialogRef will be assigned the dialog reference for the Open dialog, which is made application-modal. gCurrentNumberOfWindows keeps a count of the number of windows opened. gDestRect and gViewRect are used to set the destination and view rectangles for the TextEdit structures associated with 'TEXT' files.

doNewCommand

doNewCommand is called when the user chooses New from the File menu and when an Open Application or Re-Open Application event is received.

Since this demonstration does not support the actual entry of text or the drawing of graphics, the document type passed to doNewDocWindow is immaterial. The document type 'TEXT' is passed in this instance simply to keep doNewDocWindow happy.

If doNewDocWindow returns no error, SetWindowProxyCreatorAndType is called to set the proxy icon for the window. (A new, untitled window, even though it has no associated file, needs a proxy icon to maintain visual consistency with other windows which have associated files.) The proxy icon will display in the disabled state, indicating, in this particular case, that the window has no associated file rather than unsaved changes. The creator code and file type passed in the second and third parameters of SetWindowProxyCreatorAndType determine the icon to be displayed.)

doOpenCommand

doOpenCommand, which is called when the user chooses Open from the File menu, uses Navigation Services 3.0 functions to create and display an application-modal Open dialog.

NavGetDefaultDialogCreationOptions initialises the specified NavDialogCreationOptions structure with the defaults.

GetIndString retrieves the application's name and assigns it to an Str255 variable. This is then converted to a CFString and assigned to the clientName field of the NavDialogCreationOptions structure. This will cause the application's name to appear in the dialog's title bar.

The next line assigns a value to the modality field of the NavDialogCreationOptions structure which will cause the dialog to be application-modal. An 'open' resource containing the file type list is then read in and the handle assigned a variable of type NavTypeListHandle. (The 'open' resource specifies that 'TEXT' and 'PICT' file types are supported.)

The call to NavCreateGetFileDialog creates the dialog. Since the default options are being used, multiple file selection is allowed. A universal procedure pointer to the event function getFilePutFileEventFunction, which will respond to button clicks in the dialog, is passed in the third parameter. No preview function or filter function is used, and no user data is passed in. The last parameter (a global variable) receives the dialog reference.

The call to NavDialogRun displays the dialog.

doCloseCommand

doCloseCommand is called when the user chooses Close from the File menu or clicks in the window's go-away box. It is also called from doEvents when the global variable gQuittingApplication is set to true.

The first two lines get a reference to the front window and establish whether the front window is a document window or a modeless dialog.

If the front window is a document window, the handle to the window's document structure is retrieved from the window's window object, allowing a check of whether the window is touched (that is, has unsaved changes). If it does, doCreateAskSaveChangesDialog is called to create and display a Save Changes alert and the function returns, otherwise doCloseDocWindow is called. Prior to the call to doCreateAskSaveChangesDialog, if the window is collapsed (Mac OS 8/9) or minimized in the dock (Mac OS X) it is first uncollapsed or brought out of the Dock.

No modeless dialogs are used by this program. However, if the front window was a modeless dialog, the appropriate action would be taken at the second case.

doSaveCommand

doSaveCommand is called when the user chooses Save from the File menu or clicks the Save button in a Save Changes alert.

The first two lines get the WindowRef for the front window and retrieve the handle to that window's document structure. If a file currently exists for the document in this window, the function doWriteFile is called. The next four lines are incidental to the demonstration; they simply remove the words "WINDOW TOUCHED" from the window.

SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doSaveAsCommand

doSaveAsCommand uses Navigation Services 3.0 functions to create and display a window-modal Save Location dialog. It is called when the user chooses Save As… from the File menu. It is also called by doSaveCommand if the user chooses Save when the front window contains a document for which no file currently exists.

NavGetDefaultDialogCreationOptions initialises the specified NavDialogCreationOptions structure with the defaults. The first line in the if block unsets the "allow saving of stationery files" bit (one of the defaults). On Mac OS 8/9, this means that the dialog will not contain the Format: pop-up menu.

GetWTitle gets the front window's title into an Str255 variable. This is then converted to a CFString and assigned to the saveFileName field of the NavDialogCreationOptions structure. This will be the default name for the saved file and will appear in the Name (OS 8/9) and Save As (OS X) edit text fields in the Save Location dialog.

The next two lines assign the application's name to the clientName field of the NavDialogCreationOptions structure. This will then appear in the dialog's title bar.

The next two lines assign the window reference to the parentWindow field of the NavDialogCreationOptions structure and assign a value to the modality field which will cause the dialog to be window-modal.

The next block gets the file type from the window's document structure into a local variable.

The call to NavCreatePutFileDialog creates the dialog. A universal procedure pointer to the event function getFilePutFileEventFunction, which will respond to button clicks in the dialog, is passed in the fourth parameter. The window reference is passed in the fifth (user data) parameter. This will be passed to the event function. The dialog reference is assigned to a field of the window's document structure.

The call to NavDialogRun displays the dialog.

doRevertCommand

doRevertCommand, which is called when the user chooses Revert from the File menu, uses Navigation Services 3.0 functions to create and display a window-modal Discard Changes alert. The general approach is similar to that used to create and display the Save Location dialog, the main difference being that a universal procedure pointer to the event function askSaveDiscardEventFunction is stored in the askSaveDiscardEventFunctionUPP field of the window's document structure.

doNewDocWindow

doNewDocWindow is called by doNewCommand, doOpenFile and the Open Application event handler. It creates a new window and associated document structure.

If the current number of open windows is the maximum allowable by this program, the function immediately exits, returning an error code which will cause an advisory error alert to be displayed.

GetNewCWindow creates a new window. The call to NewHandle allocates memory for the window's document structure. If this call is not successful, the window is disposed of and the function returns with the error code returned by MemError. The call to SetWRefCon assigns the handle to the document structure to the window structure's refCon field. The next block initialises fields of the document structure.

If the document type is 'TEXT', the if block executes, creating a TextEdit structure and assigning a handle to that structure to the editStrucHdl field of the document structure. (Note that the processes here are not explained in detail because TextEdit and TextEdit structures are not central to the demonstration. For the purposes of the demonstration, it is sufficient to understand that the text data retrieved from, and saved to, disk is stored in a TextEdit structure. (TextEdit is addressed in detail at Chapter 21.))

If the Boolean value passed to doNewDocWindow was set to true, the call to ShowWindow makes the window visible, otherwise the window is left invisible. The penultimate line increments the global variable which keeps track of the number of open windows.

doCloseDocWindow

doCloseDocWindow is called from doCloseCommand when the subject window is not touched and from the Save Changes alert event handler askSaveDiscardEventFunction when the user clicks the Save or Don't Save buttons in a Save Changes alert.

The FSClose call closes the file, and FlushVol stores to disk all unwritten data currently in the volume buffer.

If the document is a text document, the TextEdit structure is disposed of. If it is a picture document, the Picture structure is disposed of. Finally, the document structure and window are disposed of and the global variable which keeps track of the number of open windows is decremented.

The last block executes only if gQuittingApplication has been set to true in the function quitAppEventHandler. If all windows have been closed, QuitApplicationEventLoop has been closed to terminate the program; otherwise doCloseCommand is called. This repetitive calling of doCloseCommand and doCloseDocWindow will continue until no windows remain or until gQuittingApplication is set to false by a click in the Cancel button in a Save Changes or, on Mac OS X only, a Review Unsaved alert.

doOpenFile

doOpenFile opens a new document window and calls the functions which read in the file. It is called by the event function getFilePutFileEventFunction when an Open button click occurs in an Open dialog. The event function passes the file system specification structure and document type to doOpen file.

The call to doNewDocWindow opens a new window and creates an associated document structure. SetWTitle sets the window's title using information in the file system specification structure. FSpOpenDF opens the file's data fork. If this call is not successful, the window is disposed of and the function returns.

The next three lines assign the file reference number and file system specification structure to the relevant fields of the document structure.

The next block calls the appropriate function for reading in the file, depending on whether the file type is of type 'TEXT' or 'PICT'. If the file is read in successfully, ShowWindow makes the window visible.

Just before the call to ShowWindow, SetWindowProxyFSSpec is called to establish a proxy icon for the window and associate the file with the window. (The creator code and file type of the file determine the icon to be displayed.) GetWindowProxyAlias assigns a copy of the alias data for the file to the aliasHdl field of the window's document structure. (This is used by the file synchronisation function.) SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doCreateAskSaveChangesDialog

doCreateAskSaveChangesDialog, which is called from doCloseCommand, uses Navigation Services 3.0 functions to create and display a window-modal Save Changes alert. The general approach is similar to that used to create and display the Discard Changes alert, but note that in this case that the isAskSaveChangesDialog field of the window's document structure is set to true.

Note also that, if the program is running on Mac OS 8/9, and if gCloseDocWindow is true, doCloseDocWindow is called to close the file, flush the volume, and close down the window. (gCloseDocWindow is set to true in the callback function askSaveDiscardEventFunction if the user clicks the Don't Save push button button in the Save Changes alert.)

doSaveUsingFSSPec

As will be seen in the event function getFilePutFileEventFunction, when the user clicks on the Save button in a Save Location dialog, AECoerceDesc is called on the descriptor structure in the selection field of the NavReplyRecord structure in an attempt to coerce it to type FSRef. If the call is successful (meaning that the program is running on Mac OS X), doSaveUsingFSRef is called to perform the save using the HFS+ API. If the call is not successful (meaning that the program is running on Mac OS 8/9) this function (doSaveUsingFSSpec) is called.

A descriptor structure is returned in the selection field of the NavReplyRecord structure. AEGetNthPtr coerces the descriptor structure to typeFSS and stores the result in the local variable fileSpec.

The name field of fileSpec will be empty at this stage. Accordingly, the Navigation Services 3.0 function NavDialogGetSaveFileName is called to get a CFStringRef to the filename from the dialog object, which is converted to a Pascal string and assigned to the name field of fileSpec.

If the value in the replacing field of the NavReplyRecord structure indicates that the file is not being replaced, FSpCreate is called to create a new file of the specified type and with the application's signature as the specified creator. If this call is not successful, the NavReplyRecord structure is disposed of and the function returns.

The file system specification structure returned by the FSpCreate call is assigned to the fileFSSpec field of the window's document structure. If a file currently exists for the document, that file is closed by the call to FSClose. The data fork of the newly created file is then opened by a call to FSpOpenDF, the fileRefNum field of the document structure is assigned the file reference number returned by FSpOpenDF, the window's title is set to the new file's name, and the function doWriteFile is called to write the document to the new file. NavCompleteSave is called to complete the save operation.

Just before the call to doWriteFile, SetWindowProxyFSSpec is called to establish a proxy icon for the window and associate the file with the window. (The creator code and file type of the file determine the icon to be displayed.) GetWindowProxyAlias assigns a copy of the alias data for the file to the aliasHdl field of the window's document structure. (This is used by the file synchronisation function.) SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doSaveUsingFSRef

doSaveUsingFSRef, which is called from the event function getFilePutFileEventFunction, performs the save using the HFS+ API. The main if block executes only if the call to AECoerceDesc is successful in coercing the descriptor structure in the selection field of the NavReplyRecord to type FSRef.

In Carbon, the dataHandle field of descriptor structures is opaque. Thus AEGetDescData is used to extract the data in this field, which is assigned to the local variable fsRefParent. This is the FSRef for the parent directory.

At the next block, CFStringGetLength is called to get the number of 16-bit Unicode characters in the saveFileName field of the NavReplyRecord structure. This facilitates the call to CFStringGetCharacters, which extracts the contents of the string into a buffer.

If the value in the replacing field of the NavReplyRecord structure indicates that the file is being replaced, the existing file is first deleted. FSMakeFSRefUnicode, given a parent directory and Unicode file name, creates an FSRef for the file. This is passed in the call to FSDeleteObject, which deletes the file.

The call to FSCreateFileUnicode creates a new file with the Unicode name. On return, the last parameter contains a file system specification structure for the new file. (Although the file is created with a Unicode name, it can be written using a file system specification structure.)

The call to FSpGetFInfo gets the Finder information from the volume catalog entry for the file. The file type extracted from the window's document structure is then assigned to the fdType field of the returned FInfo structure, following which FSpSetFInfo is called to set the new Finder information in the file's volume catalog entry.

The file system specification structure is assigned to the fileFSSpec field of the window's document structure.

The data fork of the newly created file is then opened by a call to FSpOpenDF, the fileRefNum field of the document structure is assigned the file reference number returned by FSpOpenDF, the window's title is set to the new file's name, and the function doWriteFile is called to write the document to the new file. NavCompleteSave is called to complete the save operation.

Just before the call to doWriteFile, SetWindowProxyFSSpec is called to establish a proxy icon for the window and associate the file with the window. (The creator code and file type of the file determine the icon to be displayed.) GetWindowProxyAlias assigns a copy of the alias data for the file to the aliasHdl field of the window's document structure. (This is used by the file synchronisation function.) SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes.

doWriteFile

doWriteFile is called by doSaveCommand, doSaveUsingFSSPec, and doSaveUsingFSRef. In conjunction with two supporting functions, it writes the document to disk using the "safe-save" procedure.

The first two lines retrieve a handle to the document structure and the file system specification from the document structure.

The next two lines create a temporary file name which is bound to be unique. FindFolder finds the temporary folder on the file's volume, or creates a temporary folder if necessary. FSMakeFSSpec makes a file system specification structure for the temporary file, using the volume reference number and parent directory ID returned by the FindFolder call. FSpCreate creates the temporary file in that directory on that volume, and FSpOpenDF opens the file's data fork.

Within the next if block, the appropriate function is called to write the document's data to the temporary file.

The two calls to FSClose close both the temporary and existing files prior to the call to FSpExchangeFiles, which swaps the files' data. The temporary file is then deleted and the data fork of the existing file is re-opened.

The function doCopyResources is called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new document file. If the file type is 'PICT', a 'pnot' resource and associated 'PICT' resource is also copied to the resource fork. If the program is running on Mac OS 8/9 and the file type is 'TEXT', a 'pnot' resource is copied and a 'TEXT' resource is created and copied so as to provide a a preview for 'TEXT' files in the Open dialog.

doReadTextFile

doReadTextFile is called by doOpenFile and the event function askSaveDiscardEventFunction to read in data from an open file of type 'TEXT'.

The first two lines retrieve the file reference number from the document structure.

The next three lines retrieve the handle to the TextEdit structure from the document structure and modify the text size and line height fields of the edit structure.

SetFPos sets the file mark to the beginning of the file. GetEOF gets the number of bytes in the file. If the number of bytes exceeds that which can be stored in a TextEdit edit structure (32,767), the number of bytes which will be read from the file is restricted to 32,767.

NewHandle allocates a buffer equal to the size of the file (or 32,767 bytes if the preceding if statement executed). FSRead reads the data from the file into the buffer. MoveHHi and HLockHi move the buffer high in the heap and lock it preparatory to the call to TESetText. TESetText copies the text in the buffer into the existing hText handle of the TextEdit edit structure. The buffer is then unlocked and disposed of.

doReadPictFile

doReadPictFile is called by doOpenFile and the event function askSaveDiscardEventFunction to read in data from an open file of type 'PICT'.

The first two lines retrieve the file reference number from the document structure. GetEOF gets the number of bytes in the file. SetFPos sets the file mark 512 bytes (the size of a 'PICT' file's header) past the beginning of the file, and the next line subtracts the header size from the total size of the file. NewHandle allocates memory for the Picture structure and FSRead reads in the file's data.

doWriteTextData

doWriteTextData is called by doWriteFile to write text data to the specified file.

The first two lines retrieve the handle to the TextEdit structure from the document structure. The number of bytes of text is then retrieved from the teLength field of the TextEdit structure.

SetFPos sets the file mark to the beginning of the file. FSWrite writes the specified number of bytes to the file. SetEOF adjusts the file's size. FlushVol stores to disk all unwritten data currently in the volume buffer.

The penultimate line sets the windowTouched field of the document structure to indicate that the document data on disk equates to the document data in memory.

doWritePictData

doWritePictData is called by doWriteFile to write picture data to the specified file.

The first two lines retrieve the handle to the relevant Picture structure from the document structure. SetFPos sets the file mark to the start of the file. FSWrite writes zeros in the first 512 bytes (the size of a 'PICT' file's header). GetHandleSize gets the size of the Picture structure and FSWrite writes the bytes in the Picture structure to the file. SetEOF adjusts the file's size and FlushVol stores to disk all unwritten data currently in the volume buffer.

The penultimate line sets the windowTouched field of the document structure to indicate that the document data on disk equates to the document data in memory.

getFilePutFileEventFunction

getFilePutFileEventFunction is the event (callback) function pertaining to the Open, Save Location, and Choose a Folder dialogs. It responds to button clicks in those dialogs.

When the user has clicked one of the dialog's buttons, the kNavCBUserAction message is received in the callBackSelector formal parameter. When this message is received, the first action is to call NavDialogGetReply to get a NavReplyRecord structure containing information about the dialog session. NavDialogGetUserAction is then called to get the user action which dismissed the dialog.

If the user clicked the Open button in an Open dialog, AECountItems is called to count the number of descriptor structures in the descriptor list returned in the selection field of the NavReplyRecord structure, and which is created from FSSpec references to items selected in the Open dialog. The for loop repeats for each of the descriptor structures. AEGetNthPtr gets the file system specification into a local variable of type FSSpec. This file system specification is then passed in the first parameter of a call to FSpGetFInfo, allowing the file type to be ascertained. The file system specification and file type are then passed in a call to the function doOpenFile, which creates a new window and reads in the file.

If the user clicked the Save button in a Save Location dialog, the window reference received in the callBackUD formal parameter is assigned to the local variable windowRef. (Recall that the window reference for the front window was passed in the fifth parameter of the call to NavCreatePutFileDialog.) The next task is to determine which of the two file saving functions (doSaveUsingFSSpec or doSaveUsingFSRef) should be called to save the file. Accordingly, AECoerceDesc is called in an attempt to coerce the descriptor structure in the selection field of the NavReplyRecord structure to type FSRef. If the call is successful, doSaveUsingFSRef is called; if not, doSaveUsingFSSpec is called.

If the user clicked the Choose button in a Choose a Folder dialog, AEGetNthPtr is called to get the file system specification into a local variable of type FSSpec. When a file system specification describes a directory, as it does in this case, the name field is empty and the parID field contains the directory ID of that directory, not the ID of the parent directory. In this demonstration, the volume reference number and directory ID are passed in a call to FSMakeFSSpec, which fills in the fields of the FSSpec record pointed to by the fourth parameter. The contents of the fields of this FSSpec structure (the directory name, its parent directory ID, and the volume reference number) are then drawn in the bottom of the front window.

Before exit from the kNavCBUserAction case, NavDisposeReply is called to release the memory allocated for the NavReplyRecord structure.

When the user has clicked a dialog's Cancel button, the kNavCBTerminate message is received in the callBackSelector formal parameter. When this message is received, if a dialog reference has been assigned to the global variable gModalToApplicationNavDialogRef (as it will be in the case of the application-modal Open and Choose a Folder dialogs), the dialog is disposed of and the global variable is assigned NULL. If gModalToApplicationNavDialogRef contains NULL, the window reference received in the callBackUD formal parameter is assigned to the local variable windowRef. (Recall that the window reference for the front window was passed in the fifth parameter of the call to NavCreatePutFileDialog.) A handle to the window's document structure is then retrieved, allowing access to the dialog reference stored in that structure. The dialog is disposed of and the relevant field of the document structure is assigned NULL.

The kNavCBEvent message indicates that an event has occurred. callBackParms is a pointer to a structure of type NavCBRec. The event's event structure resides in the eventData field of the NavCBRec structure. The event type is extracted from the event structure's what field. If the event is an update event, the relevant window's WindowRef is retrieved from the event structure's message field and the application's window updating function doUpdate is called.

askSaveDiscardEventFunction

askSaveDiscardEventFunction is the event (callback) function pertaining to the Save Changes and Discard Changes alerts. It responds to button clicks in those dialogs.

When the user has clicked one of the dialog's buttons, the kNavCBUserAction message is received in the callBackSelector formal parameter. When this message is received, the first action is to get a handle to the front window's document structure. (Recall that the reference to the front window was passed in the third parameter of the NavCreateAskSaveChangesDialog and NavCreateAskDiscardChangesDialog calls.) The main if block executes only if the modalToWindowNavDialogRef field of the document structure contains a dialog reference.

If the user clicked the Save button in a Save Changes alert, doSaveCommand is called to save the file and execution falls through to the kNavUserActionDontSaveChanges case where doCloseDocWindow is called to close the file, flush the volume, and close down the window.

If the user clicked the Don't Save button in a Save Changes alert, and if the program is running on Mac OS X, doCloseDocWindow is called to close the file, flush the volume, and close down the window. If the program is running on Mac OS 9, the global variable gCloseDocWindow is set to true, causing the doCloseDocWindow call to occur in the function doCreateAskSaveChangesDialog. Before all this occurs, NavDialogDispose is called to dispose of the alert before the window is closed by the call to doCloseDocWindow.

If the user clicked the OK button in a Discard Changes alert, the window's content area is erased and the appropriate function (doReadTextFile or doReadPictFile) is called depending on whether the file type is 'TEXT' or 'PICT'. In addition, the window's "touched" field in the document structure is set to false and InvalWindowRect is called to force a redraw of the window's content region. Just before the InvalWindowRect call, SetWindowModified is called with false passed in the modified parameter. This causes the window proxy icon to appear in the enabled state, indicating no unsaved changes. The Discard Changes alert is then disposed of.

If the user clicked the Cancel button in a Save Changes or Discard Changes alert, and if it is a Save Changes alert, the flag gQuittingApplication is set to false. This has the effect of defeating the execution of the last block of code in the function doCloseDocWindow. (Recall that the isAskSaveChangesDialog field of the window's document structure is set to true when such alerts are created.) The alert is then disposed of.

doCopyResources

doCopyResources is called by doWriteFile. It copies the missing application name string resource from the resource fork of the application file to the resource fork of the new file. If the file type is PICT, a 'pnot' resource and associated 'PICT' resource is also copied. If the file type is TEXT, a 'pnot' resource, together with a 'TEXT' resource created within the function, are also copied

The first line retrieves a handle to the file's document structure. The next four lines establish the file type involved. FSpCreateResFile creates the resource fork in the new file and FSpOpenResFile opens the resource fork. The function for copying specified resources between specified files (doCopyAResource) is then called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new file.

If the file type is 'PICT', a 'pnot' resource and associated 'PICT' resource is copied so as to provide a preview for 'PICT' files in the Open dialog. (Of course, in a real application, the 'pnot' and 'PICT' resource would be created by the application for each separate 'PICT' file.)

If the program is running on Mac OS 9 and the file type is 'TEXT', a 'pnot' resource is copied and a 'TEXT' resource is created and copied so as to provide a a preview for 'TEXT' files in the Open dialog. After the 'pnot' resource is copied, a relocatable block is created and the text in the TextEdit structure is copied to that block. AddResource turns that arbitrary data in memory into a 'TEXT' resource, assigns a resource type, ID, and name to that resource, and inserts an entry in the resource map for the current resource file (in this case, the resource fork of the TEXT file). UpdateResFile then writes the resource map and data to disk. (Note that all this is not required for Mac OS X because, on that system, a preview for 'TEXT' files is created automatically by the Finder.

CloseResFile closes the resource fork of the new file.

doCopyAResource

doCopyAResource copies specified resources between specified files. In this program, it is called only by doCopyResources.

UseResFile sets the application's resource fork as the current resource file. GetResource reads the specified resource into memory.

GetResInfo, given a handle, gets the resource type, ID and name. (Note that this line is included only because of the generic nature of doCopyResource. The calling function has passed doCopyResource the type and ID in this instance.)

DetachResource removes the resource's handle from the resource map without removing the resource from memory, and converts the resource handle into a generic handle. UseResFile makes the new file's resource fork the current resource file. AddResource makes the now arbitrary data in memory into a resource, assigns a resource ID, type and name to that resource, and inserts an entry in the resource map for the current resource file. UpdateResFile then writes the resource map and data to disk.

SynchroniseFiles.c

doSynchroniseFiles

doSynchroniseFiles is called from the main event loop whenever a null event is received.

currentTicks is assigned the number of ticks since system startup. As will be seen, currentTicks will be used to ensure that synchronisations only occur every quarter second (15 ticks). windowRef is assigned a reference to the front non-floating window.

If 15 ticks have elapsed since the last synchronisation, the outer if block executes. The while loop walks the window list (see the call to GetNextWindow at the bottom of the loop) looking for associated files whose locations have changed. When the last window in the list has been examined, the loop exits.

Within the while loop, GetWRefCon is called to retrieve the handle to the window's document structure.

If the aliasHdl field of the window's document structure contains NULL, the window does not yet have a file associated with it, in which case execution falls through to the next iteration of the while loop and the next window is examined.

If the window has an associated file, the handle to the associated alias structure, which contains the alias data for the file, is retrieved. ResolveAlias is then called to perform a search for the target of the alias, returning the file system specification for the target file in the third parameter. After identifying the target, ResolveAlias compares some key information about the target with the information in the alias structure. If the information differs, ResolveAlias updates the alias structure to match the target and sets the aliasChanged parameter to true.

If the aliasChanged parameter is set to true, meaning that the location of the file has changed, the fileFSSpec field of the window's document structure is assigned the file system specification structure returned by ResolveAlias. Since it is also possible that the user has renamed the file, SetWTitle is called to set the window's title to the filename contained in the name field of the file system specification structure returned by ResolveAlias.

The next task is to determine whether the user has moved the file to the trash or to a folder in the trash, in which case the document must be closed.

FindFolder is called to get the volume reference number and parent directory ID of the trash folder.

The do/while loop walks up the parent folder hierarchy to the root folder. At the first line in the do/while loop, if the root folder has been reached (fsRtParID is the parent ID of the root directory), the file is not in the trash, in which case the loop exits at that point. At the next if statement, the volume reference number and parent directory ID of the file are compared with the volume reference number and directory ID of the trash. If they match, the file is closed, its associated memory is disposed of, and the window is disposed of.

The bottom line of the do/while loop effects the walk up the parent directory hierarchy, FSMakeFSSpec creates a file system specification structure from the current contents of the vRefNum and parID fields of newFSSPec. Since newFSSpec is also the target, the parID field is "filled in" again, at every iteration of the loop, with the parent ID of the directory passed in the second parameter of the FSMakeFSSpec call.

ChooseAFolderDialog.c

doChooseAFolderDialog

doChooseAFolderDialog, which is called when the user chooses the Choose a Folder Dialog item in the demonstration menu, creates and displays a Choose a Folder dialog.

NavGetDefaultDialogCreationOptions initialises the specified NavDialogCreation Options structure with the defaults. GetIndString retrieves a Pascal string, which is converted to a CFString and assigned to the message field of a NavDialogOptions structure. This will appear immediately below the browser list in the OS 8/9 dialog and above the browser list in the OS X dialog.

The next line ensures that the dialog will be application-modal.

NavCreateChooseFolderDialog creates the dialog and NavDialogRun displays it.