Demonstration Program MoreResources Listing

```
// *******************************************************************************************
// MoreResources.c                                                       CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program uses custom resources to:
//
// •  Store application preferences in the resource fork of a Preferences file.
//
// •  Store, in the resource fork of a document file:
//
//    •  The size and position of the window associated with the document.
//
//    •  A flattened PMPageFormat object containing information about how the pages of the
//       document should be printed, for example, on what paper size, in what printable
//       area, and in what orientation (landscape or portrait).
//
// The program also demonstrates setting, storing, and retrieving application preferences
// using Core Foundation Preferences Services.
//
// The program utilises the following standard resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for OS9Apple/Applcation, File, Edit and
//    Demonstration menus (preload, non-purgeable).
//
// •  A 'DLOG' resource (purgeable) and associated 'dlgx', 'DITL' and 'CNTL' resources
//    (purgeable) associated with the display of, and user modification of, current
//    application preferences.
//
// •  A 'STR#' resource (purgeable) containing the required name of the preferences file
//    created by the program.
//
// •  A 'STR ' resource (purgeable) containing the application-missing string, which is copied
//    to the resource fork of the preferences file.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// The program creates and utilises the following custom resources:
//
// •  A 'PrFn' (preferences) resource comprising three boolean values, which is located in the
//    program's resource file, which contains default preference values, and which is copied
//    to the resource fork of a preferences file created when the program is run for the first
//    time.  Thereafter, the 'PrFn' resource in the preferences file is used for the storage
//    and retrieval of application preferences set by the user.
//
// •  A 'WiSp' (window size and position) resource, which is created in the resource fork of
//    the document file used by the program, and which is used to store the associated
//    window's port rectangle converted to global coordinates.
//
// •  A 'PgFt' (page format) resource, which is created in the resource fork of the document
//    file used by the program, and which is used to store a flattened PMPageFormat object.
//
// *******************************************************************************************

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

#include <Carbon.h>

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

#define rMenubar          128
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
```

```
#define  iOpen           2
#define  iClose          4
#define  iPageSetup      9
#define  iQuit           12
#define mEdit            130
#define  iPreferences    10
#define rNewWindow       128
#define rPrefsDialog     128
#define  iSound          4
#define  iFullScreen     5
#define  iAutoScroll     6
#define rStringList      128
#define  iPrefsFileName  1
#define rTypePrefs       'PrFn'
#define  kPrefsID        128
#define rTypeWinState    'WiSt'
#define  kWinStateID     128
#define rPageFormat      'PgFt'
#define  kPageFormatID   128
#define rTypeAppMiss     'STR '
#define  kAppMissID      -16397
#define topLeft(r)       (((Point *) &(r))[0])
#define botRight(r)      (((Point *) &(r))[1])
#define MAX_UINT32       0xFFFFFFFF

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

typedef struct
{
  Boolean sound;
  Boolean fullScreen;
  Boolean autoScroll;
  SInt16  programRunCount;
} appPrefs, **appPrefsHandle;

typedef struct
{
  Rect    userStateRect;
  Boolean zoomedToStandardState;
} windowState, *windowStatePtr,**windowStateHandle;

typedef struct
{
  Handle pageFormatHdl;
  FSSpec fileFSSpec;
} docStructure, **docStructureHandle;

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

Boolean        gRunningOnX = false;
PMPrintSession gPrintSession;
WindowRef      gWindowRef;
SInt16         gAppResFileRefNum;
Boolean        gDone;
Boolean        gSoundPref;
Boolean        gFullScreenPref;
Boolean        gAutoScrollPref;
SInt16         gProgramRunCountPref;
Boolean        gSoundCFPref;
Boolean        gFullScreenCFPref;
Boolean        gAutoScrollCFPref;
SInt16         gProgramRunCountCFPref;
Boolean        gWindowOpen        = false;
Boolean        gPageFormatChanged = false;
SInt16         gPrefsFileRefNum   = 0;

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

void      main                              (void);
void      doPreliminaries                 (void);
OSErr     quitAppEventHandler             (AppleEvent *,AppleEvent *,SInt32);
OSErr     showPrefsEventHandler           (AppleEvent *,AppleEvent *,SInt32);
void      doEvents                        (EventRecord *);
Point     doGetStandardStateHeightAndWidth (void);
void      doUpdateWindow                  (WindowRef);
void      doAdjustMenus                   (void);
void      doMenuChoice                    (long);
void      doErrorAlert                    (SInt16);
void      doOpenCommand                   (void);
void      navEventFunction                (NavEventCallbackMessage,NavCBRecPtr,
                                           NavCallBackUserData);
void      doCloseWindow                   (void);
void      doPreferencesDialog             (void);
Boolean   eventFilter                     (DialogRef,EventRecord *,SInt16 *);
OSStatus  doPageSetupDialog               (void);
void      doGetPreferencesResource        (void);
void      doGetPreferencesCFPrefs         (void);
OSErr     doCopyResource                  (ResType,SInt16,SInt16,SInt16);
void      doSavePreferencesResource       (void);
void      doSavePreferencesCFPrefs        (void);
void      doLoadWindowUserAndZoomState    (WindowRef);
void      doGetFrameWidthAndTitleBarHeight (WindowRef,SInt16 *,SInt16 *);
void      doSaveWindowUserAndZoomState    (WindowRef);
void      doGetPageFormat                 (WindowRef);
void      doSavePageFormat                (WindowRef);

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

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  OSStatus      osStatus;
  EventRecord   eventStructure;

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

  doPreliminaries();

  // .................................. set current resource file to application resource fork

  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);
    }
    menuRef = GetMenuRef(mEdit);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iPreferences);
      DeleteMenuItem(menuRef,iPreferences - 1);
      DisableMenuItem(menuRef,0);
    }
    EnableMenuCommand(NULL,'pref');

    gRunningOnX = true;
  }

  // .................................................................. create printing session

  osStatus = PMCreateSession(&gPrintSession);
  if(osStatus != kPMNoError)
    doErrorAlert(osStatus);

  // ............................ read in application preferences and increment program run count

  doGetPreferencesResource();
  doGetPreferencesCFPrefs();

  gProgramRunCountPref++;
  gProgramRunCountCFPref++;

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

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,MAX_UINT32,NULL))
      doEvents(&eventStructure);
  }
}

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

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(320);
  InitCursor();
  FlushEvents(everyEvent,0);

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

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

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

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

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

  if(osError == errAEDescNotFound)
  {
    if(FrontWindow())
      doCloseWindow();
    doSavePreferencesResource();
    doSavePreferencesCFPrefs();
    PMRelease(&gPrintSession);
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ******************************************************************** showPrefsEventHandler

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

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

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

  return osError;
}

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

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode, zoomPart;
  Rect           constraintRect;
  Point          standardStateHeightAndWidth;

  windowRef = (WindowRef) eventStrucPtr->message;

  switch(eventStrucPtr->what)
  {
    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);

      switch(partCode)
      {
        case kHighLevelEvent:
          AEProcessAppleEvent(eventStrucPtr);
          break;

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

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

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

        case inGrow:
          constraintRect.top   = 190;
          constraintRect.left = 400;
          constraintRect.bottom = constraintRect.right = 32767;
          ResizeWindow(windowRef,eventStrucPtr->where,&constraintRect,NULL);
          break;

        case inZoomIn:
        case inZoomOut:
          standardStateHeightAndWidth = doGetStandardStateHeightAndWidth();
          if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
            zoomPart = inZoomIn;
          else
            zoomPart = inZoomOut;

          if(TrackBox(windowRef,eventStrucPtr->where,partCode))
            ZoomWindowIdeal(windowRef,zoomPart,&standardStateHeightAndWidth);
          break;
      }
      break;

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

    case updateEvt:
      doUpdateWindow(windowRef);
      break;
  }
}

// ********************************************************** doGetStandardStateHeightAndWidth

Point  doGetStandardStateHeightAndWidth(void)
{
  BitMap screenBits;
  Rect   mainScreenRect;
  Point  standardStateHeightAndWidth;

  mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
  standardStateHeightAndWidth.v = mainScreenRect.bottom - 75;
  standardStateHeightAndWidth.h = 600;

  return standardStateHeightAndWidth;
}

// ************************************************************************** doUpdateWindow

void  doUpdateWindow(WindowRef windowRef)
{
  RGBColor           whiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
  RGBColor           blueColour  = { 0x1818, 0x4B4B, 0x8181 };
  Rect               portRect;
  docStructureHandle docStrucHdl;
  PMPageFormat       pageFormat = kPMNoPageFormat;
  Str255             string;
  PMResolution       resolution;
  PMRect             paperRect;
  PMRect             pageRect;
```

```
UInt16              orientation;

BeginUpdate(windowRef);

SetPortWindowPort(windowRef);

RGBForeColor(&whiteColour);
RGBBackColor(&blueColour);
GetWindowPortBounds(windowRef,&portRect);
EraseRect(&portRect);

MoveTo(10,20);
TextFace(bold);
DrawString("\pApplication Preferences:");
MoveTo(10,35);
DrawString("\pResource Fork Prefs File");
MoveTo(170,35);
DrawString("\pCF Preferences Services");
TextFace(normal);
MoveTo(10,50);
DrawString("\pSound On:  ");
if(gSoundPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(10,65);
DrawString("\pFull Screen On:  ");
if(gFullScreenPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(10,80);
DrawString("\pAutoScroll On:  ");
if(gAutoScrollPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(10,95);
DrawString("\pProgram run count: ");
NumToString((SInt32) gProgramRunCountPref,string);
DrawString(string);

MoveTo(170,50);
DrawString("\pSound On:  ");
if(gSoundCFPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(170,65);
DrawString("\pFull Screen On:  ");
if(gFullScreenCFPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(170,80);
DrawString("\pAutoScroll On:  ");
if(gAutoScrollCFPref)  DrawString("\pYES");
else  DrawString("\pNO");
MoveTo(170,95);
DrawString("\pProgram run count: ");
NumToString((SInt32) gProgramRunCountCFPref,string);
DrawString(string);

docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
PMUnflattenPageFormat((*docStrucHdl)->pageFormatHdl,&pageFormat);
PMGetResolution(pageFormat,&resolution);
PMGetAdjustedPaperRect(pageFormat,&paperRect);
PMGetAdjustedPageRect(pageFormat,&pageRect);
PMGetOrientation(pageFormat,&orientation);
if(pageFormat != kPMNoPageFormat)
  PMRelease(&pageFormat);

MoveTo(10,115);
TextFace(bold);
DrawString("\pInformation From Document's 'PgFt' (Page Format) Resource:");
TextFace(normal);

if((*docStrucHdl)->pageFormatHdl != NULL)
{
    MoveTo(10,130);
    DrawString("\pApplication's Drawing Resolution:  ");
    NumToString((long) resolution.hRes,string);
    DrawString(string);
    DrawString("\p dpi horizontal, ");
    NumToString((long) resolution.vRes,string);
    DrawString(string);
    DrawString("\p dpi vertical");

    MoveTo(10,145);
    DrawString("\pPaper Rectangle Size in Drawing Resolution:  ");
    NumToString((long) (paperRect.bottom - paperRect.top),string);
    DrawString(string);
    DrawString("\p by ");
    NumToString((long) (paperRect.right - paperRect.left),string);
    DrawString(string);

    MoveTo(10,160);
    DrawString("\pPage Rectangle Size in Drawing Resolution:  ");
    NumToString((long) (pageRect.bottom - pageRect.top),string);
    DrawString(string);
    DrawString("\p by ");
    NumToString((long) (pageRect.right - pageRect.left),string);
    DrawString(string);

    MoveTo(10,175);
    DrawString("\pOrientation:  ");
    if(orientation == 1)
      DrawString("\pPortrait");
    else if(orientation == 2)
      DrawString("\pLandscape");
  }
  else
  {
    MoveTo(10,130);
    DrawString("\pA page format ('PgFt') resource has not been saved yet.");
    MoveTo(10,145);
    DrawString("\pOpen the Page Setup... dialog before closing the window or quitting.");
  }

  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

  EndUpdate(windowRef);
}

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

void  doAdjustMenus(void)
{
  MenuRef menuRef;

  if(gWindowOpen)
  {
    menuRef = GetMenuRef(mFile);
    DisableMenuItem(menuRef,iOpen);
    EnableMenuItem(menuRef,iClose);
    EnableMenuItem(menuRef,iPageSetup);
  }
  else
  {
    menuRef = GetMenuRef(mFile);
    EnableMenuItem(menuRef,iOpen);
    DisableMenuItem(menuRef,iClose);
    DisableMenuItem(menuRef,iPageSetup);
  }

  DrawMenuBar();
}
```

```
// ****************************************************************************** doMenuChoice

void  doMenuChoice(long menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;
  OSStatus      osStatus;
  Rect          portRect;

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

  if(menuID == 0)
    return;

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

    case mFile:
      switch(menuItem)
      {
        case iClose:
          doCloseWindow();
          break;

        case iOpen:
          doOpenCommand();
          break;

        case iPageSetup:
          osStatus = doPageSetupDialog();
          if(osStatus != kPMNoError && osStatus != kPMCancel)
            doErrorAlert(osStatus);
          if(FrontWindow())
          {
            GetWindowPortBounds(FrontWindow(),&portRect);
            InvalWindowRect(FrontWindow(),&portRect);
          }
          break;

        case iQuit:
          if(FrontWindow())
            doCloseWindow();
          PMRelease(&gPrintSession);
          doSavePreferencesResource();
          doSavePreferencesCFPrefs();
          gDone = true;
          break;
      }
      break;

    case mEdit:
      if(menuItem == iPreferences)
        doPreferencesDialog();
      break;
  }

  HiliteMenu(0);
}

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

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

  NumToString((SInt32) errorCode,errorString);

  if(errorCode != memFullErr)
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  else
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}

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

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

  osError = NavGetDefaultDialogOptions(&dialogOptions);

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

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

          if(!(gWindowRef = GetNewCWindow(rNewWindow,NULL,(WindowRef)-1)))
            return;

          if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
          {
            DisposeWindow(gWindowRef);
            return;
          }

          SetWRefCon(gWindowRef,(SInt32) docStrucHdl);
          (*docStrucHdl)->fileFSSpec = fileSpec;
          SetWTitle(gWindowRef,(*docStrucHdl)->fileFSSpec.name);
          (*docStrucHdl)->pageFormatHdl = NULL;

          SetPortWindowPort(gWindowRef);
          UseThemeFont(kThemeSmallSystemFont,smSystemScript);

          doLoadWindowUserAndZoomState(gWindowRef);
          doGetPageFormat(gWindowRef);
          ShowWindow(gWindowRef);
          gWindowOpen = true;
        }
      }

      NavDisposeReply(&navReplyStruc);
    }
  }
}

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

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

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

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

void  doCloseWindow(void)
{
  WindowRef          windowRef;
  docStructureHandle docStrucHdl;
  OSErr              osError = 0;

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

  doSaveWindowUserAndZoomState(windowRef);

  if(gPageFormatChanged)
    doSavePageFormat(windowRef);

  DisposeHandle((Handle) docStrucHdl);
  DisposeWindow(windowRef);
  gWindowOpen = false;
}

// *********************************************************************** doPreferencesDialog

void  doPreferencesDialog(void)
{
  DialogRef      modalDlgRef;
  ModalFilterUPP eventFilterUPP;
  ControlRef     controlHdl;
  SInt16         itemHit;
  Rect           portRect;

  if(!(modalDlgRef = GetNewDialog(rPrefsDialog,NULL,(WindowRef) -1)))
    return;

  SetDialogDefaultItem(modalDlgRef,kStdOkItemIndex);
  SetDialogCancelItem(modalDlgRef,kStdCancelItemIndex);
  GetDialogItemAsControl(modalDlgRef,iSound,&controlHdl);
  SetControlValue(controlHdl,gSoundPref);
  GetDialogItemAsControl(modalDlgRef,iFullScreen,&controlHdl);
  SetControlValue(controlHdl,gFullScreenPref);
  GetDialogItemAsControl(modalDlgRef,iAutoScroll,&controlHdl);
  SetControlValue(controlHdl,gAutoScrollPref);

  ShowWindow(GetDialogWindow(modalDlgRef));

  eventFilterUPP = NewModalFilterUPP((ModalFilterProcPtr) eventFilter);

  do
  {
    ModalDialog(eventFilterUPP,&itemHit);
    GetDialogItemAsControl(modalDlgRef,itemHit,&controlHdl);
    SetControlValue(controlHdl,!GetControlValue(controlHdl));
  } while((itemHit != kStdOkItemIndex) && (itemHit != kStdCancelItemIndex));

  if(itemHit == kStdOkItemIndex)
  {
    GetDialogItemAsControl(modalDlgRef,iSound,&controlHdl);
    gSoundPref = gSoundCFPref = GetControlValue(controlHdl);
    GetDialogItemAsControl(modalDlgRef,iFullScreen,&controlHdl);
    gFullScreenPref = gFullScreenCFPref = GetControlValue(controlHdl);
    GetDialogItemAsControl(modalDlgRef,iAutoScroll,&controlHdl);
    gAutoScrollPref = gAutoScrollCFPref = GetControlValue(controlHdl);
  }

  DisposeDialog(modalDlgRef);
  DisposeModalFilterUPP(eventFilterUPP);

  if(gWindowRef)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    InvalWindowRect(gWindowRef,&portRect);
  }

  doSavePreferencesResource();
  doSavePreferencesCFPrefs();
}

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

Boolean  eventFilter(DialogRef dialogRef,EventRecord *eventStrucPtr,SInt16 *itemHit)
{
  Boolean handledEvent;
  GrafPtr oldPort;

  handledEvent = false;

  if((eventStrucPtr->what == updateEvt) &&
     ((WindowRef) eventStrucPtr->message != GetDialogWindow(dialogRef)))
  {
    doUpdateWindow((WindowRef) eventStrucPtr->message);
  }
  else
  {
    GetPort(&oldPort);
    SetPortDialogPort(dialogRef);

    handledEvent = StdFilterProc(dialogRef,eventStrucPtr,itemHit);

    SetPort(oldPort);
  }

  return handledEvent;
}

// ************************************************************************* doPageSetupDialog
```

```
OSStatus  doPageSetupDialog(void)
{
  docStructureHandle docStrucHdl;
  OSStatus           osStatus    = kPMNoError;
  PMPageFormat       pageFormat  = kPMNoPageFormat;
  Boolean            userClickedOKButton;

  docStrucHdl = (docStructureHandle) GetWRefCon(gWindowRef);
  HLock((Handle) docStrucHdl);

  if((*docStrucHdl)->pageFormatHdl == NULL)
  {
    osStatus = PMCreatePageFormat(&pageFormat);
    if((osStatus == kPMNoError) && (pageFormat != kPMNoPageFormat))
      osStatus = PMSessionDefaultPageFormat(gPrintSession,pageFormat);
    if(osStatus == kPMNoError)
      osStatus = PMFlattenPageFormat(pageFormat,&(*docStrucHdl)->pageFormatHdl);
  }
  else
  {
    osStatus = PMUnflattenPageFormat((*docStrucHdl)->pageFormatHdl,&pageFormat);
    if(osStatus == kPMNoError)
      osStatus = PMSessionValidatePageFormat(gPrintSession,pageFormat,kPMDontWantBoolean);
  }

  if((osStatus == kPMNoError) && (pageFormat != kPMNoPageFormat))
  {
    SetThemeCursor(kThemeArrowCursor);

    osStatus = PMSessionPageSetupDialog(gPrintSession,pageFormat,&userClickedOKButton);
    if(!userClickedOKButton)
      osStatus = kPMCancel;
  }

  if(osStatus == kPMNoError && userClickedOKButton)
  {
    DisposeHandle((*docStrucHdl)->pageFormatHdl);
    (*docStrucHdl)->pageFormatHdl = NULL;
    osStatus = PMFlattenPageFormat(pageFormat,&(*docStrucHdl)->pageFormatHdl);

    gPageFormatChanged = true;
  }

  if(pageFormat != kPMNoPageFormat)
    PMRelease(&pageFormat);

  HUnlock((Handle) docStrucHdl);

  return osStatus;
}

// ******************************************************************** doGetPreferencesResource

void  doGetPreferencesResource(void)
{
  Str255         prefsFileName;
  OSErr          osError;
  SInt16         volRefNum;
  long           directoryID;
  FSSpec         fileSSpec;
  SInt16         fileRefNum;
  appPrefsHandle appPrefsHdl;

  GetIndString(prefsFileName,rStringList,iPrefsFileName);

  osError = FindFolder(kUserDomain,kPreferencesFolderType,kDontCreateFolder,&volRefNum,
                       &directoryID);
  if(osError == noErr)
```

```
    osError = FSMakeFSSpec(volRefNum,directoryID,prefsFileName,&fileSSpec);
  if(osError == noErr || osError == fnfErr)
    fileRefNum = FSpOpenResFile(&fileSSpec,fsCurPerm);

  if(fileRefNum == -1)
  {
    FSpCreateResFile(&fileSSpec,'PpPp','pref',smSystemScript);
    osError = ResError();

    if(osError == noErr)
    {
      fileRefNum = FSpOpenResFile(&fileSSpec,fsCurPerm);
      if(fileRefNum != -1 )
      {
        UseResFile(gAppResFileRefNum);

        osError = doCopyResource(rTypePrefs,kPrefsID,gAppResFileRefNum,fileRefNum);
        if(osError == noErr)
          osError = doCopyResource(rTypeAppMiss,kAppMissID,gAppResFileRefNum,fileRefNum);
        if(osError != noErr)
        {
          CloseResFile(fileRefNum);
          osError = FSpDelete(&fileSSpec);
          fileRefNum = -1;
        }
      }
    }
  }

  if(fileRefNum != -1)
  {
    UseResFile(fileRefNum);

    appPrefsHdl = (appPrefsHandle) Get1Resource(rTypePrefs,kPrefsID);
    if(appPrefsHdl == NULL)
      return;

    gSoundPref = (*appPrefsHdl)->sound;
    gFullScreenPref = (*appPrefsHdl)->fullScreen;
    gAutoScrollPref = (*appPrefsHdl)->autoScroll;
    gProgramRunCountPref = (*appPrefsHdl)->programRunCount;

    gPrefsFileRefNum = fileRefNum;

    UseResFile(gAppResFileRefNum);
  }
}

// ******************************************************************* doGetPreferencesCFPrefs

void  doGetPreferencesCFPrefs(void)
{
  CFStringRef applicationID9 = CFSTR("MoreResources CFPrefs");
  CFStringRef applicationIDX = CFSTR("com.Windmill Software.MoreResources");
  CFStringRef applicationID;
  CFStringRef soundOnKey     = CFSTR("sound");
  CFStringRef fullScreenKey  = CFSTR("fullScreen");
  CFStringRef autoScrollKey  = CFSTR("autoScroll");
  CFStringRef runCountKey    = CFSTR("runCount");
  Boolean     booleanValue, success;

  if(gRunningOnX)
    applicationID = applicationIDX;
  else
    applicationID = applicationID9;

  booleanValue = CFPreferencesGetAppBooleanValue(soundOnKey,applicationID,&success);
  if(success)
    gSoundCFPref = booleanValue;
  else
  {
    gSoundCFPref       = true;
    gFullScreenCFPref  = true;
    gAutoScrollCFPref  = true;
    doSavePreferencesCFPrefs();
    return;
  }

  booleanValue = CFPreferencesGetAppBooleanValue(fullScreenKey,applicationID,&success);
  if(success)
    gFullScreenCFPref = booleanValue;

  booleanValue = CFPreferencesGetAppBooleanValue(autoScrollKey,applicationID,&success);
  if(success)
    gAutoScrollCFPref = booleanValue;

  gProgramRunCountCFPref = CFPreferencesGetAppIntegerValue(runCountKey,applicationID,
                                                           &success);
}

// *************************************************************************** doCopyResource

OSErr  doCopyResource(ResType resType,SInt16 resID,SInt16 sourceFileRefNum,
                      SInt16 destFileRefNum)
{
  SInt16  oldResFileRefNum;
  Handle  sourceResourceHdl;
  ResType ignoredType;
  SInt16  ignoredID;
  Str255  resourceName;
  SInt16  resAttributes;
  OSErr   osError;

  oldResFileRefNum = CurResFile();
  UseResFile(sourceFileRefNum);

  sourceResourceHdl = Get1Resource(resType,resID);

  if(sourceResourceHdl != NULL)
  {
    GetResInfo(sourceResourceHdl,&ignoredID,&ignoredType,resourceName);
    resAttributes = GetResAttrs(sourceResourceHdl);
    DetachResource(sourceResourceHdl);
    UseResFile(destFileRefNum);
    if(ResError() == noErr)
      AddResource(sourceResourceHdl,resType,resID,resourceName);
    if(ResError() == noErr)
      SetResAttrs(sourceResourceHdl,resAttributes);
    if(ResError() == noErr)
      ChangedResource(sourceResourceHdl);
    if(ResError() == noErr)
      WriteResource(sourceResourceHdl);
  }

  osError = ResError();

  ReleaseResource(sourceResourceHdl);
  UseResFile(oldResFileRefNum);

  return osError;
}

// ****************************************************************** doSavePreferencesResource

void  doSavePreferencesResource(void)
{
  SInt16         currentResFile;
  appPrefsHandle appPrefsHdl;
  Handle         existingResHdl;
  Str255         resourceName = "\pPreferences";

  if(gPrefsFileRefNum == -1)
    return;

  currentResFile = CurResFile();

  appPrefsHdl = (appPrefsHandle) NewHandleClear(sizeof(appPrefs));

  HLock((Handle) appPrefsHdl);

  (*appPrefsHdl)->sound = gSoundPref;
  (*appPrefsHdl)->fullScreen = gFullScreenPref;
  (*appPrefsHdl)->autoScroll = gAutoScrollPref;
  (*appPrefsHdl)->programRunCount = gProgramRunCountPref;

  UseResFile(gPrefsFileRefNum);

  existingResHdl = Get1Resource(rTypePrefs,kPrefsID);

  if(existingResHdl != NULL)
  {
    RemoveResource(existingResHdl);
    DisposeHandle(existingResHdl);
    if(ResError() == noErr)
      AddResource((Handle) appPrefsHdl,rTypePrefs,kPrefsID,resourceName);
    if(ResError() == noErr)
      WriteResource((Handle) appPrefsHdl);
  }

  HUnlock((Handle) appPrefsHdl);

  ReleaseResource((Handle) appPrefsHdl);
  UseResFile(currentResFile);
}

// ***************************************************************** doSavePreferencesCFPrefs

void  doSavePreferencesCFPrefs(void)
{
  CFStringRef applicationID9 = CFSTR("MoreResources CFPrefs");
  CFStringRef applicationIDX = CFSTR("com.Windmill Software.MoreResources");
  CFStringRef applicationID;
  CFStringRef soundOnKey     = CFSTR("sound");
  CFStringRef fullScreenKey  = CFSTR("fullScreen");
  CFStringRef autoScrollKey  = CFSTR("autoScroll");
  CFStringRef yes            = CFSTR("yes");
  CFStringRef no             = CFSTR("no");
  CFStringRef runCountKey    = CFSTR("runCount");
  Str255      runCountPascalString;
  CFStringRef runCountCFString;

  if(gRunningOnX)
    applicationID = applicationIDX;
  else
    applicationID = applicationID9;

  if(gSoundCFPref)
    CFPreferencesSetAppValue(soundOnKey,yes,applicationID);
  else
    CFPreferencesSetAppValue(soundOnKey,no,applicationID);

  if(gFullScreenCFPref)
    CFPreferencesSetAppValue(fullScreenKey,yes,applicationID);
  else
    CFPreferencesSetAppValue(fullScreenKey,no,applicationID);

  if(gAutoScrollCFPref)
```

```
    CFPreferencesSetAppValue(autoScrollKey,yes,applicationID);
  else
    CFPreferencesSetAppValue(autoScrollKey,no,applicationID);

  NumToString((SInt32) gProgramRunCountCFPref,runCountPascalString);
  runCountCFString = CFStringCreateWithPascalString(NULL,runCountPascalString,
                                                     CFStringGetSystemEncoding());
  CFPreferencesSetAppValue(runCountKey,runCountCFString,applicationID);

  CFPreferencesAppSynchronize(applicationID);

  if(runCountCFString != NULL)
    CFRelease(runCountCFString);
}

// ************************************************************** doLoadWindowUserAndZoomState

void  doLoadWindowUserAndZoomState(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  OSErr              osError;
  windowStateHandle  windowStateHdl;
  Rect               userStateRect, standardStateRect, displayRect;
  Point              standardStateHeightAndWidth;
  SInt16             frameWidth, titleBarHeight, menuBarHeight;
  BitMap             screenBits;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  windowStateHdl = (windowStateHandle) Get1Resource(rTypeWinState,kWinStateID);

  if(windowStateHdl != NULL )
  {
    userStateRect = (*windowStateHdl)->userStateRect;

    if((*windowStateHdl)->zoomedToStandardState)
    {
      standardStateHeightAndWidth = doGetStandardStateHeightAndWidth();
      doGetFrameWidthAndTitleBarHeight(windowRef,&frameWidth,&titleBarHeight);
      GetThemeMenuBarHeight(&menuBarHeight);

      SetRect(&standardStateRect,frameWidth + 1,titleBarHeight + menuBarHeight + 1,
              frameWidth + standardStateHeightAndWidth.h,
              titleBarHeight + menuBarHeight + standardStateHeightAndWidth.v);

      displayRect = standardStateRect;
    }
    else
      displayRect = userStateRect;
  }
  else
  {
    doGetFrameWidthAndTitleBarHeight(windowRef,&frameWidth,&titleBarHeight);
    GetThemeMenuBarHeight(&menuBarHeight);

    userStateRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
    SetRect(&userStateRect,frameWidth + 1,titleBarHeight + menuBarHeight + 1,
            frameWidth + 400,titleBarHeight + menuBarHeight + 175);

    displayRect = userStateRect;
  }
  MoveWindow(windowRef,displayRect.left,displayRect.top,false);

  SetPortWindowPort(windowRef);
  GlobalToLocal(&topLeft(displayRect));
  GlobalToLocal(&botRight(displayRect));
  SizeWindow(windowRef,displayRect.right,displayRect.bottom,true);

  SetWindowIdealUserState(windowRef,&userStateRect);

  ReleaseResource((Handle) windowStateHdl);
  CloseResFile(fileRefNum);
}

// ********************************************************** doGetFrameWidthAndTitleBarHeight

void  doGetFrameWidthAndTitleBarHeight(WindowRef windowRef,SInt16 *frameWidth,
                                       SInt16 *titleBarHeight)
{
  RgnHandle structureRegionHdl = NewRgn();
  RgnHandle contentRegionHdl   = NewRgn();
  Rect      structureRect, contentRect;

  GetWindowRegion(windowRef,kWindowStructureRgn,structureRegionHdl);
  GetRegionBounds(structureRegionHdl,&structureRect);
  GetWindowRegion(windowRef,kWindowContentRgn,contentRegionHdl);
  GetRegionBounds(contentRegionHdl,&contentRect);

  *frameWidth = contentRect.left - structureRect.left - 1;
  *titleBarHeight = contentRect.top - structureRect.top - 1;

  DisposeRgn(structureRegionHdl);
  DisposeRgn(contentRegionHdl);
}

// ************************************************************** doSaveWindowUserAndZoomState

void  doSaveWindowUserAndZoomState(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  OSErr              osError;
  Point              standardStateHeightAndWidth;
  windowState        userStateRectAndZoomState;
  windowStateHandle  windowStateHdl;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  standardStateHeightAndWidth = doGetStandardStateHeightAndWidth();
  if(IsWindowInStandardState(windowRef,&standardStateHeightAndWidth,NULL))
    userStateRectAndZoomState.zoomedToStandardState = true;
  else
    userStateRectAndZoomState.zoomedToStandardState = false;

  GetWindowUserState(windowRef,&userStateRectAndZoomState.userStateRect);

  windowStateHdl = (windowStateHandle) Get1Resource(rTypeWinState,kWinStateID);
  if(windowStateHdl != NULL)
  {
    **windowStateHdl = userStateRectAndZoomState;
    ChangedResource((Handle) windowStateHdl);
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);
  }
  else
  {
    windowStateHdl = (windowStateHandle) NewHandle(sizeof(windowState));
    if(windowStateHdl != NULL)
    {
      **windowStateHdl = userStateRectAndZoomState;
      AddResource((Handle) windowStateHdl,rTypeWinState,kWinStateID,"\pLast window state");
    }
  }

  if(windowStateHdl != NULL)
  {
    UpdateResFile(fileRefNum);
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);

    ReleaseResource((Handle) windowStateHdl);
  }

  CloseResFile(fileRefNum);
}

// ************************************************************************** doGetPageFormat

void  doGetPageFormat(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  SInt16             fileRefNum;
  OSErr              osError;
  Handle             pageFormatResourceHdl = NULL;
  PMPageFormat       pageFormat  = kPMNoPageFormat;
  Boolean            settingsChanged;

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

  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  pageFormatResourceHdl = Get1Resource(rPageFormat,kPageFormatID);

  if(pageFormatResourceHdl != NULL)
  {
    PMUnflattenPageFormat(pageFormatResourceHdl,&pageFormat);
    PMSessionValidatePageFormat(gPrintSession,pageFormat,&settingsChanged);

    DisposeHandle((*docStrucHdl)->pageFormatHdl);
    PMFlattenPageFormat(pageFormat,&((*docStrucHdl)->pageFormatHdl));

    if(pageFormat != kPMNoPageFormat)
      PMRelease(&pageFormat);
    ReleaseResource(pageFormatResourceHdl);
  }

  CloseResFile(fileRefNum);

  HUnlock((Handle) docStrucHdl);
}

// ************************************************************************* doSavePageFormat
```

```
void  doSavePageFormat(WindowRef windowRef)
{
  docStructureHandle docStrucHdl;
  Handle             pageFormatHdl;
  SInt16             fileRefNum;
  OSErr              osError;
  Size               handleSize;

  docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);
  handleSize = GetHandleSize((*docStrucHdl)->pageFormatHdl);

  fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec,fsRdWrPerm);
  if(fileRefNum < 0)
  {
    osError = ResError();
    doErrorAlert(osError);
    return;
  }

  pageFormatHdl = Get1Resource(rPageFormat,kPageFormatID);

  if(pageFormatHdl != NULL)
  {
    RemoveResource(pageFormatHdl);
    DisposeHandle(pageFormatHdl);
    pageFormatHdl = NewHandle(handleSize);
    BlockMove(*((*docStrucHdl)->pageFormatHdl),*pageFormatHdl,handleSize);
    AddResource(pageFormatHdl,rPageFormat,kPageFormatID,"\pPage Format");
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);
  }
  else
  {
    pageFormatHdl = NewHandle(handleSize);
    if(pageFormatHdl != NULL)
    {
      BlockMove(*((*docStrucHdl)->pageFormatHdl),*pageFormatHdl,handleSize);
      AddResource(pageFormatHdl,rPageFormat,kPageFormatID,"\pPage Format");
      osError = ResError();
      if(osError != noErr)
        doErrorAlert(osError);
    }
  }

  if(pageFormatHdl != NULL)
  {
    UpdateResFile(fileRefNum);
    osError = ResError();
    if(osError != noErr)
      doErrorAlert(osError);

    ReleaseResource(pageFormatHdl);
  }

  gPageFormatChanged = false;

  CloseResFile(fileRefNum);
}

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

Demonstration Program MoreResources Comments

This program uses two methods for saving and retrieving application preferences. The first method involves a custom preferences ('PrFn') resource, which is saved to the resource fork of a preferences file named MoreResources Preferences. The second method involves the use of Core Foundation Preferences Services, which creates, and saves preferences to, a file named com.Windmill Software.MoreResources.plist on Mac OS X and MoreResources CFPrefs.plist on Mac OS 8/9. These files are created when the program is first run. Thereafter, the preferences will be read in from the preferences files every time the program is run and replaced whenever the user invokes the Preferences dialog to change the preferences settings.

When the application is first run, an "application missing" 'STR ' resource is also copied to the resource fork of the preferences file created by the first method. On Mac OS 8/9, if the user double clicks on the icon for this preferences file, an alert is invoked displaying the text contained in the "application missing" resource.

After the program is launched, the user should choose Open from the File menu to open the included demonstration document file (titled "MoreResources Document"). The resource fork of this file contains two custom resources, namely, a 'WiSp' resource containing the last saved window user state and zoom state, and a 'PgFt' resource containing a flattened PMPageFormat object. These two resources are read in whenever the document file is opened. The 'WiSp' resource is written to whenever the file is closed. The 'PgFt' resource is written to when the file is closed only if the user invoked the Page Setup dialog while the document was open.

No data is read in from the document's data fork. Instead, the window is used to display the current preferences settings and information extracted from the document's 'PgFt' resource. This is simply to keep the code not directly related to resources to a minimum.

The user should choose different paper size, scaling, and orientation settings in the Page Setup dialog, re-size or zoom the window, close the file, re-open the file, and note that, firstly, the saved page format values are correctly retrieved and, secondly, the window is re-opened in the size and position in which is was closed. The user should also change the application preferences settings via the Preferences dialog (which is invoked when the Preference item in the Mac OS 8/9 Edit menu or Mac OS X Application menu is chosen), quit the program, re-launch the program, and note that the last saved preferences settings are retrieved at program launch.

The user may also care to remove the 'WiSp' and 'PgFt' resources from the document file's resource fork, run the program, force a 'PgFt' resource to be created and written to by invoking the Page Setup dialog while the document file is open, quit the program, and re-run the program, noting that 'WiSp' and 'PgFt' resources are created in the document file's resource fork if they do not already exist.

When done, the user may want to remove the preferences files from the Preferences folder.

defines

rPrefsDialog represents the 'DLOG' resource ID for the Preferences dialog and the following three constants represent the item numbers of the dialog's checkboxes. rStringList and iPrefsFileName represent the 'STR#' resource ID and index for the string containing the name of the application's preferences file created by the first method. The next eight constants represent resource types and IDs for the custom preferences resource, the custom window state resource, the custom page format resource, and the application missing string resource.

typedefs

The appPrefs data type is for the application preferences settings. The three Boolean fields are set by checkboxes in the Preferences dialog. The SInt16 field is incremented each time the program is run.

The windowState data type is for the window user state (a rectangle) and zoom state (a Boolean value indicating whether the window is in the standard (zoomed out) or user (zoomed in) state).

The docStructure data type is for the window's document structure. The fields are for a flattened PMPageFortmat object and a a file system specification.

Global Variables

gAppResFileRefNum will be assigned the file reference number for the application file's resource fork.

gSoundPref, gFullScreenPref, and gAutoScrollPref will hold the application preferences settings, as will gSoundCFPref gFullScreenCFPref and gAutoScrollCFPref. gProgramRunCountPref will hold the number of times the application has been run, as will gProgramRunCountCFPref.

gPageFormatChanged is set to true when the Page Setup dialog is invoked, and determines whether a new page format resource will be written to the document file when the file is closed.

gPrefsFileRefNum will be assigned the file reference number for the preferences file's resource fork.

main

The call to CurResFile sets the application's resource fork as the current resource file.

If the application is running on Mac OS X, DeleteMenuItem is called to delete the Preferences item and its preceding divider from the Edit menu. (In Mac OS X, the Preferences command is in the Application menu.) Since, on Mac OS X, the Preferences command is disabled by default, EnableMenuCommand is called to enable the item.

The call to PMCreateSession creates a printing session.

The calls to doGetPreferencesResource and doGetPreferencesCFPrefs read in the application preferences settings from the preferences files. As will be seen:

- If the preferences file to which the doGetPreferencesResource call pertains does not exist, a preferences file will be created, default preferences settings will be copied to it from the application file, and these default settings will then be read in from the newly-created file.

- If the preferences file to which the doGetPreferencesCFPrefs call pertains does not exist, a a preferences file will be created by Core Foundation Preferences Services and default preferences settings will be copied to it.

doPreliminaries

The second call to AEInstallEventHandler installs an Apple event handler for Mac OS X's Preferences command.

showPrefsEventHandler

showPrefsEventHandlerHandler is the handler for the Mac OS X Preferences command. If the user chooses the Preferences command from the Application menu when the application is running on Mac OS X, doPreferencesDialog is called.

doUpdateWindow

doUpdateWindow simply prints the current preferences and page format information in the window for the information of the user.

doOpenCommand

doOpenCommand is a much simplified version of the actions normally taken when a user chooses the Open command from a File menu. Note that, in this program, NavGetFile, rather than the Navigation Services 3.0 functions NavCreateGetFileDialog and NavDialogRun, is used to create and display the Open dialog.

NavGetFile presents the Open dialog. If the user clicks the Open button, doOpenWindow is called to create a new window, create a block for a document structure, associate the document structure with the window object, assign the file system specification for the chosen file to the document structure's file system specification field, set the window's title, and assign NULL to the pageFormatHdl field of the document structure.

At that point, doLoadWindowUserAndZoomState is called to read in the window state resource from the document's resource fork, and to position and size the window accordingly. In addition, doGetPagePormat is called to read in the page format resource and assign a handle to it to the pageFormatHdl field of the document structure.

With the window positioned and sized, ShowWindow is called to make the window visible (the window's 'WIND' resource specifies that the window is to be initially invisible) and a flag is set to indicate that the window is open.

navEventFunction

A universal procedure pointer to a navigation event (callback) function must be passed in the eventProc parameter of the NavGetFile call if the Open dialog is to be movable and resizable.

The formal parameter callBackSelector is a constant indicating which type of call Navigation Services is making to navEventFunction. One such constant is kNavCBEvent, which 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 eventDataParms field of the NavCBRec structure.

At the kNavCBEvent case, the event type is extracted from the event structure's what field. If it is an update event, the window's WindowRef is retrieved from the event structure's message field. If the event is not for a Navigation Services dialog (this application does not open any other dialogs), the application's window updating function doUpdateWindow is called.

doCloseWindow

doCloseWindow is called when the user chooses the Quit item, chooses Close from the File menu, or clicks the window's close box/button.

At the first two lines, a reference to the front window, and a handle to the associated document structure, are retrieved.

The call to doSaveWindowUserAndZoomState saves the window's user state and zoom state to the window state resource in the document's resource fork.

If the print Style dialog was invoked while the window was open, and if the user dismissed the dialog by clicking the OK button, a call is made to doSavePageFormat to save the flattened PMPageFormat object to the page format resource in the document file's resource fork.

DisposeHandle disposes of the document structure and DisposeWindow disposes of the window structure.

doPreferencesDialog

doPreferencesDialog is called when the user chooses the Preferences item in the Demonstration menu. The function presents the Preferences dialog and sets the values in the global variables which hold the current application preferences according to the settings of the dialog's checkboxes.

Note that, at the last two lines, calls are made to doSavePreferencesResource and doSavePreferencesCFPrefs.

doPageSetupDialog

doPageSetupDialog is called when the user chooses the Page Setup… item in the File menu. It presents the Page Setup dialog.

If the user dismisses the dialog with a click on the OK button, the flag which indicates that a page format change has been made is set to true.

doGetPreferencesResource

doGetPreferencesResource, which is called from the main function immediately after program launch, is the first of those functions central to the demonstration aspects of the program. Its purpose is to create the preferences file if it does not already exist, copying the default preferences resource and the missing application string resource to that file as part of the creation process, and to read in the preferences resource from the previously existing or newly-created preferences file.

GetIndString retrieves from the application's resource fork the resource containing the required name of the preferences file ("MoreResources Preferences").

FindFolder finds the location of the Preferences folder, returning the volume reference number and directory ID in the last two parameters. (Note that kUserDomain is passed in the vRefNum parameter. On Mac OS 8/9, this is mapped to kOnSystemDisk.) FSMakeFSSpec makes a file system specification from the preferences file name, volume reference number and directory ID. This file system specification is used in the FSpOpenResFile call to open the resource fork of the preferences file with exclusive read/write permission.

If the specified file does not exist, FSpOpenResFile returns -1. In that event FSpCreateResFile creates the preferences file. The call to FSpCreateResFile creates the file of the specified type on the specified volume in the specified directory and with the specified name and creator. (Note that the creator is set to an arbitrary signature which no other application known to the Finder is likely to have. This is so that a double click on the preferences file icon will cause the Finder to immediately display the missing application alert. Note also that, if 'pref' is used as the fileType parameter, the icon used for the file will be the system-supplied preferences document icon, which looks like this:

If the file is created successfully, the resource fork of the file is opened by FSpOpenResFile and the master preferences ('PrFn') and application missing 'STR ' resources are copied to the resource fork from the application's resource file. If the resources are not successfully copied, CloseResFile closes the resource fork of the new file, FspDelete deletes the file, and the fileRefNum variable is set to indicate that the file does not exist.

If the preferences file exists (either previously or newly-created), UseResFile sets the resource fork of that file as the current resource file, the preferences resource is read in from the resource fork by Get1Resource and, if the read was successful, the three Boolean values and one SInt16 value that constitute the application's preference settings are assigned to the global variables which store those values. (Note that, in this program, the function Get1Resource is used to read in resources so as to restrict the Resource Manager's search for the specified resource to the current resource file.)

The penultimate line assigns the file reference number for the open preferences file resource fork to a global variable. (The fork is left open). The last line resets the application's resource fork as the current resource file.

doGetPreferencesCFPrefs

doGetPreferencesCFPrefs is called from main to retrieve the application's preferences using Core Foundation Preferences Services routines.

Since the application ID in Java package name format (with ".plst" appended) exceeds the 31-character limit for Mac OS 8/9 filenames, an abbreviated ID is used if the program is running on Mac OS 8/9.

If the first call to CFPreferencesGetAppBooleanValue is successful, the preferences file exists and the Boolean value retrieved using the "sound" key is assigned to the relevant global variable. If this call is not successful, default values are assigned to the three global variables which hold the Boolean preferences values, and the function returns.

Two more calls to CFPreferencesGetAppBooleanValue with the appropriate keys retrieve the remaining two Boolean values and assign them to the relevant global variables. The call to CFPreferencesGetAppIntegerValue with the "run count" key retrieves the integer value representing the number of times the program has been run and assigns it to the relevant global variable.

doCopyResource

doCopyResource is called by doGetPreferences to copy the default preferences and application missing string to the newly-created preferences file from the application file.

The first two lines save the current resource file's file reference number and set the application's resource fork as the current resource file. This will be the "source" file.

The Get1Resource call reads the specified resource into memory. GetResInfo gets the resource's name and GetResAttrs gets the resource's attributes. DetachResource replaces the resource's handle in the resource map with NULL without releasing the associated memory. The resource data is now simply arbitrary data in memory.

UseResFile sets the preferences file's resource fork as the current resource file. AddResource makes the arbitrary data in memory into a resource, assigning it the specified type, ID and name. SetResAttrs sets the resource attributes in the resource map. ChangedResource tags the resource for update and pre-allocates the required disk space. WriteResource then writes the resource to disk.

With the resource written to disk, ReleaseResource discards the resource in memory and UseResFile resets the resource file saved at the first line as the current resource file.

doSavePreferencesResource

doSavePreferencesResource is called when the user dismisses the preferences dialog to save the new preference settings to the preferences file created by doGetPreferencesResource. It assumes that that preferences file is already open.

At the first two lines, if doGetPreferences was not successful in opening the preferences file at program launch, the function simply returns. The call to CurResFile saves the file reference number of the current resource file for later restoration.

The next six lines create a new preferences structure and assign to its fields the values in the global variables which store the current preference settings. UseResFile makes the preferences file's resource fork the current resource file. Get1Resource gets a handle to the existing preferences resource. Assuming the call is successful (that is, the preferences resource exists), RemoveResource is called to remove the resource from the resource map, AddResource is called to make the preferences structure in memory into a resource, and WriteResource is called to write the resource data to disk.

ReleaseResource disposes of the preferences structure in memory and UseResFile makes the previously saved resource file the current resource file.

doSavePreferencesCFPrefs

doSavePreferencesCFPrefs is called when the user dismisses the preferences dialog to set and store the new preference settings using Core Foundation Preferences Services functions.

Since the application ID in Java package name format (with ".plst" appended) exceeds the 31-character limit for Mac OS 8/9 filenames, an abbreviated ID is used if the program is running on Mac OS 8/9.

For the three Boolean preferences values, CFPreferencesSetAppValue is called with the appropriate key to set the values in the application's preferences cache. For the program run count (integer) value, NumToString and CFStringCreateWithPascalString convert the integer value to a CFString before CFPreferencesSetAppValue is called.

The call to CFPreferencesAppSynchronize flushes the cache to permanent storage, creating the preferences file if it does not already exist.

doLoadWindowUserAndZoomState

doLoadWindowUserAndZoomState gets the window state ('WiSt') resource from the resource fork of the document file and moves and sizes the window according to retrieved user state and zoom state data.

GetWRefCon gets a handle to the window's document structure so that the file system specification can be retrieved and used in the FSpOpenResFile call to open the document file's resource fork.

Get1Resource attempts to read in the 'WiSt' resource. If the call is successful, meaning that the resource exists, the following occurs. Casting is used to impose a windowState structure on the read-in data. The user state rectangle is extracted from the userStateRect field of that structure. If the zoomedToStandardState field of the resource is set to true, doGetStandardStateHeightAndWidth is called to get a rectangle 600 pixels wide and 75 less than the height of the main screen, a rectangle which is reduced in size according the window frame width, menu bar height and window title bar height, and then assigned to the variable displayRect. If the zoomedToStandardState field of the resource is set to false, displayRect is assigned the user state rectangle read in from the 'WiSt' resource.

If, however, the call to Get1Resource is not successful, meaning that the document does not yet have a 'WiSt' resource, a default user state rectangle is defined and assigned to the variable displayRect.

MoveWindow (which takes global coordinates) moves the window to the specified coordinates, keeping it inactive. After the coordinates are converted to local coordinates, SizeWindow is called to size the window to the height and width represented by the bottom and right fields of displayRect.

The call to SetWindowIdealUserState sets the ideal user state to be used by ZoomWindowIdeal.

doSaveWindowPosition

doSaveWindowPosition saves the current user state rectangle and zoom state to the document file's resource fork. The function is called when the associated window is closed by the user.

The first line gets a handle to the window's document structure so that the document file's file system specification can be retrieved and used in the FSpOpenResFile call. If the resource fork cannot be opened, an error alert is presented and the function simply returns.

The calls to IsWindowInStandardState determines whether the current height and width of the window equates to that defined by the program as the standard state. If so, the zoomedToStandardState field of a windowState structure is set to true, otherwise it is set to false.

The call to GetWindowUserState gets the current user state rectangle into the userStateRect field of the windowState structure.

Get1Resource attempts to read the 'WiSt' resource from the document's resource fork into memory. If the Get1Resource call is successful, the resource in memory is made equal to the previously "filled-in" windowState structure and the resource is tagged as changed. If the Get1Resource call is not successful (that is, the document file's resource fork does not yet contain a 'WiSt' resource), the else block creates a new windowState structure, makes this structure equal to the previously "filled-in" window state structure, and makes this data in memory into a 'WiSt' resource.

If an existing 'WiSt' resource was successfully read in, or if a new 'WiSt' resource was successfully created in memory, UpdateResFile writes the resource map and data to disk, and ReleaseResource discards the resource in memory. The document file's resource fork is then closed by CloseResFile.

doGetPageFormat

doGetPageFormat reads in the 'PgFt' resource, which contains a flattened PMPageFormat object, from the document file's resource fork. The function is called when the document is opened.

The first line gets a handle to the window's document structure so that the document file's file system specification can be retrieved and used in the call to FSpOpenResFile. If the call is not successful, an error alert is presented and the function simply returns.

The call to Get1Resource attempts to read in the resource. If the call is successful, the resource is unflattened into a PMPageFormat object, PMValidatePageFormat is called to ensure that the page format information is compatible with the driver for the current printer, and PMFlattenPageFormat is called to flatten the PMPageFormat object. When PMFlattenPageFormat returns, the second parameter (the relevant field of the window's document structure) contains a handle to the flattened data. The resource is then released and the document file's resource fork is closed.

doSavePageFormat

doSavePageFormat saves the flattened PMPageFormat object, whose handle is stored in the window's document structure, to a 'PgFt' resource in the document file's resource fork. The function is called when the file is closed if the user invoked the PageSetup dialog while the document was open and dismissed the dialog by clicking the OK button.

The first line gets a handle to the window's document structure so that the document file's file system specification can be retrieved and used in the call to FSpOpenResFile. If the call is not successful, an error alert is presented and the function simply returns.

Get1Resource attempts to read the 'PgFt' resource from the document's resource fork into memory. If the Get1Resource call is successful, the contents of the handle in the document structure's pageFormatHdl field are copied to the resource in memory, and the resource is tagged as changed. If the Get1Resource call is not successful (that is, the document file's resource fork does not yet contain a 'PgFt' resource), a block of memory the size of a 'PgFt' resource is allocated, the contents of the handle in the document structure's pageFormatHdl field are copied to that block, and AddResource makes that block into a 'PgFt' resource.

If an existing 'PgFt' resource was successfully read in, or if a new 'PgFt' resource was successfully created in memory, UpdateResFile writes the resource map and data to disk. ReleaseResource then discards the resource in memory. At the last line, the document file's resource fork is then closed.