

# 19

## MORE ON RESOURCES, AND CORE FOUNDATION PREFERENCES

*Demonstration Program: MoreResources*

### Introduction

Chapter 1 covered the basics of creating standard resources for an application's resource file and with reading in standard resources from application files and the system. In addition, the demonstration programs in preceding chapters have all involved the reading in of standard resources from those files.

This chapter is concerned with aspects of resources not covered at Chapter 1, including search paths, detaching and copying resources, creating, opening, and closing resource files, and reading from and writing to resource files. In addition, the accompanying demonstration program demonstrates the creation of **custom resources**, together with reading such resources from, and writing them to, the resource forks of files other than application and System files.

This chapter also addresses the matter of storing and retrieving application preferences using Core Foundation Preferences Services.

### Search Path for Resources

#### Preamble

The Resource Manager follows a defined search path to find a resource when you use a Resource Manager function to read, or perform an operation on, a resource,. The different files whose resource forks may constitute the search path are therefore of some relevance. The following summarises the typical locations of resources used by an application:

<i>Resource Fork of:</i>	<i>Typical Resources Therein</i>	<i>Comments</i>
System file	Sounds (Mac OS 8/9), icons, cursors, etc.	On startup, the Resource Manager creates a special zone within the system heap, creates a resource map which points to system-resident resources, opens the resource fork of the System file, and reads its resource map into memory.
Application file	Descriptions of menus, windows, controls, icons, etc. Text used in dialogs, help balloons, etc.	When a user opens an application, system software automatically opens the application's resource fork.
Application's Preferences file	The user's global preferences for the application.	An application should typically open the Preferences file at application launch, and leave it open.

Application's document file	Information specific only to the document, such as its window's last size and location.	When an application opens a document file, it should typically open the file's resource fork as well as its data fork.
-----------------------------	-----------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------

---

## ***Current Resource File***

---

When your application opens the resource fork of a file, that file becomes the **current resource file**.<sup>1</sup> However, your application can change the current resource file, if necessary, using the function `UseResFile`.

The current resource file is always the first file in the search path. It is also the file on which most Resource Manager functions assume they should operate.

---

## ***Default Search Order***

---

During its search for a resource, if the Resource Manager cannot find the resource in the current resource file, it continues searching until it either finds the resource or has searched all files in the search path.

The Resource Manager normally looks first in the resource map in memory of the last resource fork your application opened. If the resource is not found there, the resource maps of each resource file open to your application are searched in reverse order of opening. If the resource is not found there, your application's resource map is searched. If the resource has still not been found, the searches continues in the System file's resource map.

---

## ***Implications of the Default Search Order***

---

The implications of this search order are that it allows your application to:

- Access resources in the System file.
- Override resources in the System file.
- Override your application's resources with document-specific resources.
- Share a resource amongst multiple files by storing it in the application's resource fork.

---

## ***Setting the Current Resource File To Dictate the Search Order***

---

You can, of course, rely on the Resource Manager's search order to find a particular resource; however, it is best to set the current resource file to the file containing the desired resource before reading and writing resource data. This ensures that that file will be searched first, thus possibly obviating unnecessary searches of other files.

As previously stated, the function `UseResFile` is used to set the current resource file. Note that `UseResFile` takes as its single parameter a **file reference number**, which identifies an access path to the resource fork. The Resource Manager assigns a resource file a file reference number when it opens that file.

You can get the file reference number of the current resource file using the function `CurResFile`.

---

## ***Restricting the Search to the Current Resource File***

---

The search path may be restricted to the current resource file by using Resource Manager functions (such as `Get1Resource`) which look only in the current resource file's resource map when searching for a specific resource.

---

## ***Releasing and Detaching Resources***

---

When you have finished using a resource, you typically call `ReleaseResource`, which sets the handle's master pointer to `NULL` and releases the memory associated with the resource.

---

<sup>1</sup> The resource fork of a file is also called the resource file because, in some respects, you can treat it as if it were a separate file.

Your application can use `DetachResource` to replace a resource's handle in the resource map with `NULL` without releasing the associated memory. You will then be able to access the resource's data directly, without the aid of Resource Manager functions, and you will be able to pass the handle to a function which does not accept a resource handle. For example, the `AddResource` function, which makes arbitrary data in memory into a resource, requires a handle to data, not a handle to a resource.

`DetachResource` is useful when you want to copy a resource. The procedure is to read in the resource using `GetResource`, detach the resource to disassociate it from its resource file, and then copy the resource to a destination file using `AddResource`.

## ***Creating, Opening and Closing Resource Forks***

---

### ***Opening an Application's Resource Fork***

---

As previously stated, the system software automatically opens your application's resource fork at application launch. The only action required by your application at launch is to call `CurResFile` to save the file reference number for the application's resource fork.

### ***Creating and Opening a Resource Fork***

---

#### ***Creating a Resource Fork***

---

If your application needs to save resources to the resource fork of a file, and assuming the resource fork does not already exist, it must first create the resource fork.. A call to `FSpCreateResFile` will create the resource fork. `FSpCreateResFile` requires four parameters: a file system specification structure; the application's signature; the file type; the script code. The effect of `FSpCreateResFile` varies as follows:

- If the file specified by the file system specification structure does not already exist, the function:
  - Creates a file with an empty resource fork and resource map.
  - Sets the creator, type, and script code fields of the file's catalog information structure to the specified values.
- If the data fork of the file specified by the file system specification structure already exists but the file has a zero-length resource fork, the function:
  - Creates an empty resource fork and resource map.
  - Changes the creator, type, and script code fields of the catalog information structure of the file to the specified values.
- If the file specified by the file system specification structure already exists and includes a resource fork with a resource map, the function does nothing.

#### ***Opening a Resource Fork***

---

After creating a resource fork, and before attempting to write to it, you must open it using `FSpOpenResFile`. `FSpOpenResFile` returns a file reference number<sup>2</sup> which, as previously stated, may be used to change or limit the Resource Manager's search order.

As previously stated, when you open a resource fork, the Resource Manager resets the search path so that the file whose resource fork you just opened becomes the current resource file.

After opening a resource fork, you can use Resource Manager functions to write resources to it.<sup>3</sup>

---

<sup>2</sup> Note that, although the file reference number for the data fork and the resource fork usually match, you should not assume that this is always the case.

<sup>3</sup> It is possible to write to the resource fork using File Manager functions. However, in general, you should always use Resource Manager functions.

## ***Closing a Resource Fork***

---

When you are finished using a resource fork that your application explicitly opened, you should close it using `CloseResFile`. Note that the Resource Manager automatically closes any resource forks opened by your application that are still open when your application calls `ExitToShell`.

## ***Reading and Manipulating Resources***

---

Depending on which Resource Manager function is used to read resources from a resource fork, you specify the resource to be read by either its resource type and resource ID or its resource type and resource name.

### ***Reading From the Resource Map Without Loading the Resource***

---

Those Resource Manager functions that return handles to resources normally read the resource data into memory if it is not already there. Sometimes, however, you may want to read, say, resource types and attributes from the resource map without reading the resource data into memory. Calling `SetResLoad` with the `load` parameter set to `false` causes subsequent calls to those functions which return handles to resources to *not* load the resource data to memory. (To read the resource data into memory after a call to `SetResLoad` with the `load` parameter set to `false`, call `LoadResource`.)

If you call `SetResLoad` with the `load` parameter set to `false`, be sure to call it again with the parameter set to `true` as soon as possible. Other parts of the system software that call the Resource Manager rely on the default setting (that is, the `load` parameter set to `true`), and some functions will not work properly if resources are not loaded automatically.

### ***Indexing Through Resources***

---

The Resource Manager provides functions which let you index through all resources of a given type (for example, using `CountResources` and `GetIndResource`). This can be useful when you want to read all resources of a given type.

## ***Writing Resources***

---

When you have opened a resource fork, you can write resources to it (assuming it is the current resource file).

A call to `AddResource` creates a new entry for a resource in the resource map in memory (but not on the disk) and sets that entry's location to refer to that resource's data. A call to either `UpdateResFile` or `WriteResFile` will then write the resource to disk. Note that you usually need to set the current resource file to the desired file before calling `AddResource` because `AddResource` always adds the resource to the resource map in memory which corresponds to the current resource file.

When you change a resource referenced through the resource map in memory, you should call `ChangedResource` to set the `resChanged` attribute of that resource's resource map entry. `ChangedResource` reserves enough disk space to contain the changed resource. Immediately after calling `ChangedResource`, you should call `UpdateResFile` or `WriteResFile` to write the changed resource data to disk.

The difference between `UpdateResFile` and `WriteResFile` is as follows:

- `UpdateResFile` writes to disk only those resources that have been added or changed. It also writes the entire resource map to disk, overwriting its previous contents.
- `WriteResFile` writes only a single resource to disk and does not update the resource's entry in the resource map on disk.

## *Care with Purgeable Resources*

---

If you are changing purgeable resources, you should use the Memory Manager function `HNoPurge` to ensure that the Resource Manager does not purge the resource while your application is in the process of changing it.

## *Partial Resources*

---

Some resources, such as 'snd ' (Mac OS 8/9) and 'sfnt' resources, can be too large to fit into available memory. In these cases, you can read a portion of the resource into memory, or alter a section of the resource while it is still on disk, using the functions `ReadPartialResource` and `WritePartialResource`.

## *Application Preferences*

---

Many applications allow the user to set application preferences, which the application stores in a preferences file and retrieves when the application is launched. On Mac OS 8/9, your Preferences file should be located in the special folder titled Preferences provided by the operating system. The Preferences folder is located in the System folder. On Mac OS X, preferences are stored in the Preferences folder in the Library folder in each user's home directory (`~/Library/Preferences`).

You can save your application's preferences as a custom resource to the resource fork of your preferences file. Alternatively, and more correctly in the Mac OS X era, you can use the methodology provided by Core Foundation Preferences Services.

## *Using Core Foundation Preferences Services*

---

Using Core Foundation Preferences Services involves storing values associated with a key, the key being used to later retrieve the value. The concept is similar to that applying to the key/value pairs used in information property lists (see Chapter 9).

## *Application ID*

---

The name of the file in which Preference Services stores preferences information is specified by an **application ID**. For Mac OS X, this should be the same as the name associated with the `CFBundleIdentifier` key in the information property list in your application's 'plist' resource. It thus takes the form of a Java package name, that is, a unique domain name followed by the application's name (for example, `com.MyCompany.MyApplication`). For Mac OS 8/9, it is sufficient to simply use the application's name. (Indeed, this abbreviation will be essential if the application ID in Java package form exceeds the 31-character limit for Mac OS 8/9 file names.)

## *Setting, Storing and Retrieving a Preference*

---

You use the function `CFPreferencesSetAppValue` to set a preferences value. The following example sets a value which specifies that the application should use a full screen display:

```
CFStringRef applicationID = CFSTR("com.MyCompany.MyApplication");
CFStringRef fullScreenKey = CFSTR("fullScreen");
CFStringRef yes           = CFSTR("yes");
CFStringRef no            = CFSTR("no");
```

```
CFPreferencesSetAppValue(fullScreenKey, yes, applicationID);
```

`CFPreferencesSetAppValue` stores the value in a cache owned by your application. To flush the cache to permanent storage, you must call the function `CFPreferencesAppSynchronize`, passing in the application ID.

You would use the function `CFPreferencesGetAppBooleanValue` to retrieve the value, as in the following example:

```
CFStringRef applicationID = CFSTR("com.MyCompany.MyApplication");
CFStringRef fullScreenKey = CFSTR("fullScreen");
Boolean      booleanValue, success;

booleanValue = CFPREFERENCESGetAppBooleanValue(fullScreenKey, applicationID, &success);
```

You can use `CFPREFERENCESGetAppIntegerValue` to retrieve integer values (which are stored as strings) in the same way.

### ***Preference Domains***

---

Preference Services uses the concept of **domains** when creating or searching for a preference. User name, host name, and application ID identify a domain. `CFPREFERENCESSetAppValue` uses the current user and "any host" domain qualifiers as the default, which is why only the application ID is passed in the function call. Alternative low-level Preferences Services functions are available to specify an exact domain.

# ***Main Resource Manager Constants, Data Types and Functions***

---

## ***Constants***

---

### ***Resource Attributes***

```
resSysHeap    = 64   System or application heap?
resPurgeable  = 32   Purgeable resource?
resLocked     = 16   Load it in locked?
resProtected  = 8    Protected?
resPreload    = 4    Load in on OpenResFile?
resChanged    = 2    Resource changed?
```

## ***Data Types***

---

```
typedef unsigned long  FourCharCode;
typedef FourCharCode  ResType;
```

## ***Functions***

---

### ***Initialising the Resource Manager***

```
short  InitResources(void);
```

### ***Checking for Errors***

```
short  ResError(void);
```

### ***Creating an Empty Resource Fork***

```
void    FSpCreateResFile(const FSSpec *spec, OSType creator, OSType fileType, ScriptCode
scriptTag);
```

### ***Opening Resource Forks***

```
short  FSpOpenResFile(const FSSpec *spec, SignedByte permission);
```

### ***Getting and Setting the Current Resource File***

```
void    UseResFile(short refNum);
short  CurResFile(void);
short  HomeResFile(Handle theResource);
```

### ***Reading Resources Into Memory***

```
Handle  GetResource(ResType theType, short theID);
Handle  Get1Resource(ResType theType, short theID);
Handle  GetNamedResource(ResType theType, ConstStr255Param name);
Handle  Get1NamedResource(ResType theType, ConstStr255Param name);
void    SetResLoad(Boolean load);
void    LoadResource(Handle theResource);
```

### ***Getting and Setting Resource Information***

```
void    GetResInfo(Handle theResource, short *theID, ResType *theType, Str255 name);
void    SetResInfo(Handle theResource, short theID, ConstStr255Param name);
short  GetResAttrs(Handle theResource);
void    SetResAttrs(Handle theResource, short attrs);
```

### ***Modifying Resources***

```
void    ChangedResource(Handle theResource);
void    AddResource(Handle theResource, ResType theType, short theID, ConstStr255Param name);
```

### ***Writing to Resource Forks***

```
void    UpdateResFile(short refNum);
void    WriteResource(Handle theResource);
```

### ***Getting a Unique Resource ID***

```
short  UniqueID(ResType theType);
short  Unique1ID(ResType theType);
```

## ***Counting and Listing Resource Types***

```
short    CountResources(ResType theType);
short    Count1Resources(ResType theType);
Handle   GetIndResource(ResType theType,short index);
Handle   Get1IndResource(ResType theType,short index);
short    CountTypes(void);
short    Count1Types(void);
void     GetIndType(ResType *theType,short index);
void     Get1IndType(ResType *theType,short index);
```

## ***Getting Resource Sizes***

```
long     GetResourceSizeOnDisk(Handle theResource);
long     GetMaxResourceSize(Handle theResource);
```

## ***Disposing of Resources and Closing Resource Forks***

```
void     ReleaseResource(Handle theResource);
void     DetachResource(Handle theResource);
void     RemoveResource(Handle theResource);
void     CloseResFile(short refNum);
```

## ***Getting and Setting Resource Fork Attributes***

```
short    GetResFileAttrs(short refNum);
void     SetResFileAttrs(short refNum,short attrs);
```

## ***Main Core Foundation Preferences Services Functions***

---

```
void     CFPreferencesSetAppValue(CFStringRef key,CFPropertyListRef value,
    CFStringRef applicationID);
void     CFPreferencesSetValue(CFStringRef key,CFPropertyListRef value,
    CFStringRef applicationID,CFStringRef userName,CFStringRef hostName);
Boolean  CFPreferencesAppSynchronize(CFStringRef applicationID);
Boolean  CFPreferencesSynchronize(CFStringRef applicationID,CFStringRef userName,
    CFStringRef hostName);
Boolean  CFPreferencesGetAppBooleanValue(CFStringRef key,CFStringRef applicationID,
    Boolean *keyExistsAndHasValidFormat);
CFIndex  CFPreferencesGetAppIntegerValue(CFStringRef key,CFStringRef applicationID,
    Boolean *keyExistsAndHasValidFormat);
```



## Demonstration Program MoreResources Listing

```
// *****
// MoreResources.c CARBON 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/Application, 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 rPrefsDialog   128
#define iSound         4
#define iFullScreen    5
#define iAutoScroll    6
#define rStringList    128
#define iPrefsFileName 1
#define rTypePrefs     'PrFn'
#define kPrefsID       128
#define rTypeWinSizePos 'WiSp'
#define kWinSizePosID  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])

// ..... typedefs

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

typedef struct
{
    Rect windowRect;
} windowSizePos, **windowSizePosHandle;

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

// ..... global variables

Boolean      gRunningOnX = false;
PMPrintSession gPrintSession;
WindowRef    gWindowRef;
SInt16       gAppResFileRefNum;
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);
OSStatus appEventHandler                    (EventHandlerCallRef, EventRef, void *);
OSStatus windowEventHandler                 (EventHandlerCallRef, EventRef, void *);
void    doDrawContent                       (WindowRef);

```

```

void      doAdjustMenus                (void);
void      doMenuChoice                 (MenuID,MenuItemIndex);
void      doErrorAlert                 (SInt16);
void      doOpenCommand                (void);
void      navEventFunction              (NavEventCallbackMessage,NavCBRecPtr,
                                       NavCallBackUserData);

void      doOpenWindow                 (FSSpec);
void      doCloseWindow                (void);
void      doPreferencesDialog           (void);
OSStatus  doPageSetupDialog            (void);
void      doGetPreferencesResource      (void);
void      doGetPreferencesCFPrefs       (void);
OSErr     doCopyResource                (ResType,SInt16,SInt16,SInt16);
void      doSavePreferencesResource     (void);
void      doSavePreferencesCFPrefs      (void);
void      doLoadAndSetWindowSizeAndPosition (WindowRef);
void      doGetFrameWidthAndTitleBarHeight (WindowRef,SInt16 *,SInt16 *);
void      doSaveWindowSizeAndPosition  (WindowRef);
void      doGetPageFormat               (WindowRef);
void      doSavePageFormat              (WindowRef);

// ***** main

void main(void)
{
    MenuBarHandle menubarHdl;
    SInt32         response;
    MenuRef        menuRef;
    OSStatus       osStatus;
    EventTypeSpec  applicationEvents[] = { { kEventClassCommand, kEventProcessCommand },
                                           { kEventClassMenu,    kEventMenuEnableItems } };

    // ..... 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,kHICCommandPreferences);

        gRunningOnX = true;
    }
}

```

```

else
{
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
        SetMenuItemCommandID(menuRef,iQuit,kHICommandQuit);

    menuRef = GetMenuRef(mEdit);
    if(menuRef != NULL)
        SetMenuItemCommandID(menuRef,iPreferences,kHICommandPreferences);
}

// ..... create printing session

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

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

doGetPreferencesResource();
doGetPreferencesCFPrefs();

gProgramRunCountPref++;
gProgramRunCountCFPref++;

// ..... install application event handler

InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                             GetEventTypeCount(applicationEvents),applicationEvents,
                             0,NULL);

// ..... run application event loop

RunApplicationEventLoop();
}

// ***** doPreliminaries

void doPreliminaries(void)
{
    MoreMasterPointers(320);
    InitCursor();
}

// ***** appEventHandler

OSStatus appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                        void * userData)
{
    OSStatus      result = eventNotHandledErr;
    UInt32        eventClass;
    UInt32        eventKind;
    HICommand      hiCommand;
    MenuID         menuID;
    MenuItemIndex  menuItem;

    eventClass = GetEventClass(eventRef);
    eventKind  = GetEventKind(eventRef);

    switch(eventClass)
    {
    case kEventClassCommand:
        if(eventKind == kEventProcessCommand)
        {
            GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                             sizeof(HICommand),NULL,&hiCommand);
            menuID = GetMenuID(hiCommand.menu.menuRef);
            menuItem = hiCommand.menu.menuItemIndex;
            if(hiCommand.commandID == kHICommandPreferences)

```

```

    {
        doPreferencesDialog();
        result = noErr;
    }
    else if(hiCommand.commandID == kHICommandQuit)
    {
        while(FrontWindow())
            doCloseWindow();
        PMRelease(&gPrintSession);
        doSavePreferencesResource();
        doSavePreferencesCFPrefs();
    }
    else if((menuID >= mAppleApplication && menuID <= mEdit))
    {
        doMenuChoice(menuID,menuItem);
        result = noErr;
    }
}
break;

case kEventClassMenu:
    if(eventKind == kEventMenuEnableItems)
    {
        doAdjustMenus();
        result = noErr;
    }
    break;
}

return result;
}

// ***** windowEventHandler

OSStatus windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                           void* userData)
{
    OSStatus result = eventNotHandledErr;
    UInt32 eventClass;
    UInt32 eventKind;
    WindowRef windowRef;
    Rect mainScreenRect;
    BitMap screenBits;
    Point idealHeightAndWidth, minimumHeightAndWidth;

    eventClass = GetEventClass(eventRef);
    eventKind = GetEventKind(eventRef);

    switch(eventClass)
    {
        case kEventClassWindow:
            GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                             NULL,&windowRef);
            switch(eventKind)
            {
                case kEventWindowDrawContent:
                    doDrawContent(windowRef);
                    result = noErr;
                    break;

                case kEventWindowGetIdealSize:
                    mainScreenRect = GetQDGlobalsScreenBits(&screenBits)->bounds;
                    idealHeightAndWidth.v = mainScreenRect.bottom - 75;
                    idealHeightAndWidth.h = 600;
                    SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
                                     sizeof(idealHeightAndWidth),&idealHeightAndWidth);
                    result = noErr;
                    break;
            }
        }
    }
}

```

```

        case kEventWindowGetMinimumSize:
            minimumHeightAndWidth.v = 190;
            minimumHeightAndWidth.h = 400;
            SetEventParameter(eventRef, kEventParamDimensions, typeQDPoint,
                             sizeof(minimumHeightAndWidth), &minimumHeightAndWidth);
            result = noErr;
            break;

        case kEventWindowClose:
            doCloseWindow();
            result = noErr;
            break;
    }
    break;
}

return result;
}

// ***** doDrawContent

void doDrawContent(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;

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

    SetPortWindowPort(windowRef);

    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);
}

// ***** 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(MenuID menuID,MenuItemIndex menuItem)
{
    OSStatus osStatus;
    Rect portRect;

    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;
            }
            break;
    }
}

```



```

}

// ***** 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;
    NavDialogOptions dialogOptions;
    NavEventUPP     navEventFunctionUPP;
    NavReplyRecord   navReplyStruc;
    SInt32           index, count;
    AEKeyword        theKeyword;
    DescType         actualType;
    FSSpec           fileSpec;
    Size             actualSize;

    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);

                    doOpenWindow(fileSpec);
                }
            }

            NavDisposeReply(&navReplyStruc);
        }
    }
}

// ***** navEventFunction

void navEventFunction(NavEventCallbackMessage callBackSelector,NavCBRecPtr callBackParms,
                    NavCallBackUserData callBackUD)
{
}

```

```
// ***** doOpenWindow

void doOpenWindow(FSSpec fileSpec)
{
    OSStatus      osError;
    docStructureHandle docStrucHdl;
    Rect          contentRect = { 100,100,290,500 };
    WindowAttributes attributes = { kWindowStandardHandlerAttribute |
                                    kWindowStandardDocumentAttributes };
    EventTypeSpec  windowEvents[] = { { kEventClassWindow, kEventWindowDrawContent },
                                       { kEventClassWindow, kEventWindowGetIdealSize },
                                       { kEventClassWindow, kEventWindowGetMinimumSize },
                                       { kEventClassWindow, kEventWindowClose } };

    osError = CreateNewWindow(kDocumentWindowClass,attributes,&contentRect,&gWindowRef);
    if(osError != noErr)
        QuitApplicationEventLoop();

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

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

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

    InstallWindowEventHandler(gWindowRef,
                             NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                             GetEventTypeCount(windowEvents),windowEvents,0,NULL);

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

// ***** doCloseWindow

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

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

    doSaveWindowSizeAndPosition(windowRef);

    if(gPageFormatChanged)
        doSavePageFormat(windowRef);

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

// ***** doPreferencesDialog

void doPreferencesDialog(void)
{
    DialogRef modalDlgRef;
    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));

do
{
    ModalDialog(NULL,&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);

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

doSavePreferencesResource();
doSavePreferencesCFPrefs();
}

// ***** 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 = FMakeFSSpec(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.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.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);
}

// ***** doLoadAndSetWindowSizeAndPosition

void doLoadAndSetWindowSizeAndPosition(WindowRef windowRef)
{
    docStructureHandle docStruchdl;
    SInt16 fileRefNum;

```

```

OSError      osError;
windowSizePosHandle windowSizePosHdl;
Rect         windowRect;
SInt16       frameWidth, titleBarHeight, menuBarHeight;

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

windowSizePosHdl = (windowSizePosHandle) Get1Resource(rTypeWinSizePos,kWinSizePosID);

if(windowSizePosHdl != NULL )
{
    windowRect = (*windowSizePosHdl)->windowRect;
}
else
{
    doGetFrameWidthAndTitleBarHeight(windowRef,&frameWidth,&titleBarHeight);
    GetThemeMenuBarHeight(&menuBarHeight);
    SetRect(&windowRect,frameWidth + 1,titleBarHeight + menuBarHeight + 1,
        frameWidth + 400,titleBarHeight + menuBarHeight + 190);
}

MoveWindow(windowRef,windowRect.left,windowRect.top,false);
SizeWindow(windowRef,windowRect.right - windowRect.left,windowRect.bottom - windowRect.top,
    true);

ReleaseResource((Handle) windowSizePosHdl);
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);
}

// ***** doSaveWindowSizeAndPosition

void doSaveWindowSizeAndPosition(WindowRef windowRef)
{
    docStructureHandle docStrucHdl;
    Rect              portRect;
    SInt16            fileRefNum;
    OSError           osError;
    windowSizePos      windowSizePosStruct;
    windowSizePosHandle windowSizePosHdl;

    docStrucHdl = (docStructureHandle) GetWRefCon(windowRef);

```



```

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

GetWindowPortBounds(windowRef,&portRect);
SetPortWindowPort(windowRef);
LocalToGlobal(&topLeft(portRect));
LocalToGlobal(&botRight(portRect));
windowSizePosStruct.windowRect = portRect;

windowSizePosHdl = (windowSizePosHandle) Get1Resource(rTypeWinSizePos,kWinSizePosID);
if(windowSizePosHdl != NULL)
{
    **windowSizePosHdl = windowSizePosStruct;
    ChangedResource((Handle) windowSizePosHdl);
    osError = ResError();
    if(osError != noErr)
        doErrorAlert(osError);
}
else
{
    windowSizePosHdl = (windowSizePosHandle) NewHandle(sizeof(windowSizePos));
    if(windowSizePosHdl != NULL)
    {
        **windowSizePosHdl = windowSizePosStruct;
        AddResource((Handle) windowSizePosHdl,rTypeWinSizePos,kWinSizePosID,
            "\pLast window size and position");
    }
}

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

    ReleaseResource((Handle) windowSizePosHdl);
}

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;
    OSError            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.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 it 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 windowSizePos data type is for the window size and position. The docStructure data type is for the window's document structure.

### ***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. (On Mac OS X, the Preferences command is in the Application menu.) Since, on Mac OS X, the Preferences item is disabled by default, EnableMenuCommand is called to enable that 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 preferences file will be created by Core Foundation Preferences Services and default preferences settings will be copied to it.

## ***appEventHandler***

When the kEventProcessCommand event type is received, if the commandID field of the HICCommand structure contains 'pref', the function doPreferencesDialog is called to display the Preferences dialog.

## ***windowEventHandler***

When the kWindowEventClose event type is received, the function doCloseWindow is called. Amongst other things, doCloseWindow calls the function doSaveWindowSizeAndPosition, which saves the window's position and size to the resource fork of the document file.

## ***doDrawContent***

doDrawContent 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.

## ***doOpenWindow***

doOpenWindow creates a new window, creates a block for a document structure, associates the document structure with the window object, assigns the file system specification for the chosen file to the document structure's file system specification field, sets the window's title, assigns NULL to the pageFormatHdl field of the document structure, and installs the window's event handler.

At that point, doLoadAndSetWindowSizeAndPosition is called to read in the window size and position ('Wisp') resource from the document's resource fork, and to position and size the window accordingly. In addition, doGetPageFormat is called to read in the page format ('PgFt') resource, and assign a handle to it to the pageFormatHdl field of the document structure.

With the window sized and positioned, 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. However, in a Carbon event model application this function may be empty if its only purpose would be to call the window's updating function, and if the window receives, and responds to, the kEventWindowDrawContent event type.

## ***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 `doSaveWindowSizeAndPosition` saves the window's port rectangle coordinates, converted to global coordinates, to the window size and position ('WiSp') resource in the document's resource fork. If the Page Setup 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 `PMPageFormat` object to the page format ('PgFt') resource in the document file's resource fork.

`DisposeHandle` disposes of the document structure, `DisposeWindow` disposes of the window structure, and the last line sets the "window is open" flag to indicate that the window is not open.

### ***doPreferencesDialog***

`doPreferencesDialog` is called when the user chooses the Preferences item. 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 `FSOpenResFile` call to open the resource fork of the preferences file with exclusive read/write permission.

If the specified file does not exist, `FSOpenResFile` returns -1. In that event `FSCreateResFile` creates the preferences file. The call to `FSCreateResFile` 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 (Mac OS 8/9). 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 `FSOpenResFile` 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` are called to 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.

### ***doLoadAndSetWindowSizeAndPosition***

`doLoadAndSetWindowSizeAndPosition` gets the window size and position ('WiSp') resource from the resource fork of the document file and moves and sizes the window according to retrieved size and position 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 'WiSp' resource. If the `Get1Resource` call is successful, the retrieved data is assigned to a local variable of type `Rect`. (Recall that this is the window's port rectangle in global coordinates.) If `Get1Resource` call is not successful (as it will be if no 'WiSp' resource currently exists in the document file's resource fork), a default size and position for the window is established.

The calls to `MoveWindow` and `SizeWindow` set the position and size of the window to either the retrieved or default position and size.

### ***doSaveWindowSizeAndPosition***

`doSaveWindowSizeAndPosition` saves the current window size and position (the window's port rectangle in global coordinates) to the document file's resource fork, and is called when the associated window is closed by the user or the Quit item is chosen.

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 next block gets the window's port rectangle, converts it to global coordinates, and assigns it to the `windowRect` field of a `windowSizePosStruct` structure.

`Get1Resource` attempts to read the 'WiSp' 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" `windowSizePosStruct` 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 'WiSp' resource), the else block creates a new `windowSizePosStruct` structure, makes this structure equal to the previously "filled-in" `windowSizePosStruct` structure, and makes this data in memory into a 'WiSp' resource.

If an existing 'WiSp' resource was successfully read in, or if a new 'WiSp' resource was successfully created in memory, `UpdateResFile` writes the resource data and map 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.

If the resource fork is successfully opened, 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.