15 Version 1.1

MORE ON RESOURCES

Includes Demonstration Program MoreResources

Introduction

Chapter 1 — System Software, Memory, and Resources 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 file. 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.

Search Path for Resources

Preamble

When your application uses a Resource Manager routine to read, or perform an operation on, a resource, the Resource Manager follows a defined search path to find the 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:

Resource Fork of:	Typical Resources Therein	Comments
System file	Sounds, icons, cursors, and other elements available for use by all applications. Code resources which manage user interface elements such as menus, controls and windows.	On startup, the system software calls `InitResources` to initialise the Resource Manager, which creates a special heap zone within the system heap and builds a resource map which points to ROM-resident resources. The Resource Manager then opens the resource fork of the System file and reads its resource map into memory.
Application file	Descriptions of menus, windows, controls, icons, and other elements. Static data such as text used in dialog boxes or help balloons.	When a user opens an application, system software automatically opens the application's resource fork.
Application's preferences file	Data which encodes 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	Data which defines characteristics specific only to this document, such as its window's last size and location.	When an application opens a document file, it should typically opens the file's resource fork as well as its data fork.

Current Resource File

The first file whose resource fork is searched is called the **current resource file**. Whenever your application opens the resource fork of a file, that file becomes the current resource file.[1] Thus the current resource file usually corresponds to the file whose resource fork was opened most recently.

Most Resource Manager routines assume that the current resource file is the file on which they should operate or, in the case of a search, the file in which to begin the search.

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.

Specifically, when the Resource Manager searches for a resource, it normally looks first in the resource map in memory of the last resource fork your application opened. If the Resource Manager does not find the resource there, it continues to search the resource maps of each resource open to your application in reverse order of opening. After looking in the resource maps of the resource files your application has opened, the Resource Manager searches your application's resource map. If it does not find the resource there, it searches 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 defined in the System file.
- Override resources defined in the System file.
- Override application-defined resources with document-specific resources.
- Share a single resource amongst several files by storing it in the application's resource fork.

Setting the Current Resource File To Dictate the Search Order

Although you can take advantage of the Resource Manager's search order to find a particular resource, your application should generally 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.

`UseResFile` is used to set the current resource file. Note that `UseResFile` takes as its single parameter a **file reference number**, which is a unique number identifying an access path to the resource fork. The Resource Manager assigns a resource file a file reference number when it opens that file. (Your application should keep track of the file reference numbers of all resource files it opens.) `CurResFile` may be used to get the file reference number of the current resource file.

Restricting the Search to the Current Resource File

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

[1]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.

Detaching and Copying Resources

When you have finished using a resource, you typically call `ReleaseResource`, which releases the memory associated with that resource and sets the handle's master pointer to `NULL`, thus making your application's handle to the resource invalid. If the application needs the resource later, it must get a valid handle to the resource by reading the resource into memory again using a routine such as `GetResource`.

Your application can use `DetachResource` to replace a resource's handle in the resource map with `NULL` without releasing the associated memory. `DetachResource` may thus be used when you want your application to access the resource's data directly, without the aid of the Resource Manager, or when you need to pass the handle to a routine which does not accept a resource handle. For example, the `AddResource` routine, 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

The system software automatically opens your application's resource fork at application launch. Your application should simply call `CurResFile` early in its initialisation procedure to save the file reference number for the application's resource fork.

Creating and Opening a Resource Fork

Creating a Resource Fork

To save resources to the resource fork of a file, you must first create the resource fork (if it does not already exist) and obtain a file reference number for it. You use `FSpCreateResFile` to create a resource fork. `FSpCreateResFile` requires four parameters: a file system specification record, the signature of the application creating the file, the file type, and the script code for the file. The effect of `FSpCreateResFile` varies as follows:

- If the file specified by the file system specification record does not already exist (that is, the file has neither a data fork nor a resource fork), `FSpCreateResFile`:
    - 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 record to the specified values.
- If the data fork of the file specified by the file system specification record already exists but the file has a zero-length resource fork, `FSpCreateResFile`:
    - Creates an empty resource fork and resource map.
    - Changes the creator, type, and script code fields of the catalog information record of the file to the specified values.
- If the file specified by the file system specification record already exists and includes a resource fork with a resource map, `FSpCreateResFile` does nothing, and `ResError` returns an appropriate result code.

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[2] which, as previously stated, may be used to change or limit the Resource Manager's search order.

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 routines to write resources to it.[3]

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

The Resource Manager provides a number of routines which read resources from a resource fork. Depending on which routine is used, 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 routines which 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 routines 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 routines will not work properly if resources are not loaded automatically.

Indexing Through Resources

The Resource Manager provides routines 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

After opening a resource fork, you can write resources to it. You can write resources only to the current resource file.

To specify the data for a new resource, you usually use `AddResource`, which creates a new entry for the resource in the resource map in memory (but not on the disk) and sets the entry's location to refer to the resource's data. `UpdateResFile` or `WriteResFile` may then be used to write the resource to disk. Note that `AddResource` always adds the resource to the resource map in memory which corresponds to the

---

[2]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.

[3]It is possible to write to the resource fork using File Manager routines. However, in general, you should always use Resource Manager routines.

current resource file. For this reason, you usually need to set the current resource file to the desired file before calling `AddResource`.

If you change a resource that is referenced through the resource map in memory, you use `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 those resources which have been added or changed to disk. It also writes the entire resource map to disk, overwriting its previous contents.
- `WriteResFile` writes only the resource data of a single resource to disk and does not update the resource's entry in the resource map on disk.

Care with Purgeable Resources

Most applications do not make resources purgeable. However, if you are changing purgeable resources, you should use the Memory Manager routine `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 '` and `'sfnt'` resources, can be too large to fit into available memory. `ReadPartialResource` and `WritePartialResource` allow you to read a portion of the resource into memory or to alter a section of the resource while it is still on disk.

Preferences Files

Many applications allow the user to alter various settings to control the operation or configuration of the application. You can create a preferences file in which to record user preferences, and your application can retrieve the information in that file when the application is launched. Preferences information should be saved as a custom resource to the resource fork of the preferences file.

In deciding how to structure your preferences file, it is important to distinguish document-specific settings from application-specific settings. Some user-specifiable settings affect only a particular document and should, therefore, be saved to the document file's resource fork. Other settings are not specific to a particular document. You could store such settings in the application's resource fork, but it is generally better to store them in a separate preferences file, the main reason being to avoid problems which can arise if the application is located on a server volume.

The Operating System provides a special folder in the System Folder, called Preferences, where you can store the preferences file.

Main Resource Manager Constants, Data Types and Routines

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;
```

Routines

Initialising the Resource Manager

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

Checking for Errors

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

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);
```

Demonstration Program

```
1   // ##############################################################################################
2   // MoreResources.c
3   // ##############################################################################################
4   //
5   // This program uses custom resources to:
6   //
7   // •  Store application preferences in the resource fork of a preferences file, and also
8   //    to assist in the initial creation of the preferences file.
9   //
10  // •  Store,in the resource fork of a document file, the user state and current state of
11  //    the window associated with the document.
12  //
13  // •  Store, in the resource fork of a document file, the width and height of the
14  //    printable area of the paper size chosen in the print Style dialog box.
15  //
16  // The program utilises the following standard resources:
17  //
18  // •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration
19  //    menus (preload, non-purgeable).
20  //
21  // •  A 'WIND' resource (purgeable) (initially invisible).
22  //
23  // •  An 'ALRT' resource (purgeable) and associated 'DITL' resource (purgeable)
24  //    associated with the display of error messages.
25  //
26  // •  A 'DLOG' resource (purgeable) and associated 'DITL' resource (purgeable) associated
27  //    with the display of, and user modification of, current application preferences.
28  //
29  // •  A 'STR#' resource (purgeable) containing the required name of the preferences file
30  //    created by the program.
31  //
32  // •  A 'STR ' resource (purgeable) containing the application-missing string, which is
33  //    copied to the resource fork of the preferences file.
34  //
35  // •  A 'SIZE' resource with the acceptSuspendResumeEvents and is 32BitCompatible flags
36  //    set
37  //
38  // The program utilises the following custom resources:
39  //
40  // •  A 'PrFn' (preferences) resource comprising three boolean values, which is located
41  //    in the program's resource file, which contains default preference values, and which
42  //    is copied to the resource fork of a preferences file created when the program is
43  //    run for the first time.  Thereafter, the 'PrFn' resource in the preferences file
44  //    is used for the storage and retrieval of application preferences set by the user.
45  //
46  // •  A 'WiSt' (window state) resource, which is created in the resource fork of the
47  //    document file used by the program, and which is used to store the associated
48  //    window's user state rectangle (a Rect value) and zoom state (a Boolean value).
49  //
50  // •  A 'PrAr' (printable area) resource, which is created in the resource fork of the
51  //    document file used by the program, and which is used to store the printable width
```

```
//    and height of the paper size chosen in the print Style dialog box.
//
// ##############################################################################################

// ..... includes

#include <Fonts.h>
#include <Menus.h>
#include <TextEdit.h>
#include <Dialogs.h>
#include <SegLoad.h>
#include <ToolUtils.h>
#include <Devices.h>
#include <Resources.h>
#include <StandardFile.h>
#include <Printing.h>
#include <Folders.h>

// ..... defines

#define mApple          128
#define mFile           129
#define  iOpen          2
#define  iClose         4
#define  iPageSetup     8
#define  iQuit          11
#define mDemonstration  131
#define  iPreferences   1
#define rNewWindow      128
#define rMenubar        128
#define rAlertBox       128
#define rModalDialog    129
#define  iSoundOn       4
#define  iFullScreenOn  5
#define  iAutoScrollOn  6
#define rStringList     128
#define  iPrefsFileName 1
#define rTypePrintRect  'PrAr'
#define  kPrintRectID   128
#define rTypeWinState   'WiSt'
#define  kWinStateID    128
#define rTypePrefs      'PrFn'
#define  kPrefsID       128
#define rTypeAppMiss    'STR '
#define  kAppMissID     -16397
#define MAXLONG         0x7FFFFFFF
#define topLeft(r)      (((Point *) &(r))[0])
#define botRight(r)     (((Point *) &(r))[1])

// ..... typedefs

typedef struct
{
  FSSpec    fileFSSpec;
} docRecord, *docRecordPointer, **docRecordHandle;

typedef struct
{
  Boolean   soundOn;
  Boolean   fullScreenOn;
  Boolean   autoScrollOn;
} appPrefs, *appPrefsPointer, **appPrefsHandle;

typedef struct
{
  Rect      userStateRect;
  Boolean   zoomState;
} winState, *winStatePtr,**winStateHandle;

typedef RectPtr *rectHandle;

// ..... global variables

Boolean   gDone;
Boolean   gInBackground;
THPrint   gTPrintHdl;
WindowPtr gWindowPtr;
```

```
Boolean   gWindowOpen = false;
Boolean   gPrintStyleChanged = false;
Rect      gPrintRect;
Boolean   gSoundOn;
Boolean   gFullScreenOn;
Boolean   gAutoScrollOn;
SInt16    gAppResFileRefNum;
SInt16    gPrefsFileRefNum = 0;

// ……… function prototypes

void  main                      (void);
void  doInitManagers            (void);
void  doEvents                  (EventRecord *);
void  doMouseDown               (EventRecord *);
void  doUpdateWindow            (WindowPtr);
void  doAdjustMenus             (void);
void  doMenuChoice              (SInt32);
void  doFileMenu                (SInt16);
void  invalidateScrollBarArea   (WindowPtr);
void  doError                   (SInt16);
void  doOpenCommand             (void);
void  doCloseCommand            (void);
void  doPreferencesDialog       (void);
void  doPrintStyleDialog        (void);
void  doGetPreferences          (void);
OSErr doCopyResource            (ResType,SInt16,SInt16,SInt16);
void  doSavePreferences         (void);
void  doGetandSetWindowPosition (WindowPtr);
void  doSaveWindowPosition      (WindowPtr);
void  doSetWindowState          (WindowPtr,Rect,Rect);
void  doGetPrintableSize        (WindowPtr);
void  doSavePrintableSize       (WindowPtr);

// ########################################################################### main

void  main(void)
{
  Handle      menubarHdl;
  MenuHandle  menuHdl;
  EventRecord eventRec;

  // ……… initialise managers

  doInitManagers();

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

  gAppResFileRefNum = CurResFile();

  // ……… set up menu bar and menus

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

  menuHdl = GetMenuHandle(mApple);
  if(menuHdl == NULL)
    doError(MemError());
  else
    AppendResMenu(menuHdl,'DRVR');

  // ……… create and initialise a TPrint record

  PrOpen();
  gTPrintHdl = (THPrint) NewHandleClear(sizeof(TPrint));
  PrintDefault(gTPrintHdl);
  PrClose();

  // ……… read in application preferences

  doGetPreferences();

  // ……… enter event loop

```

```
206     gDone = false;
207
208     while(!gDone)
209     {
210       if(WaitNextEvent(everyEvent,&eventRec,MAXLONG,NULL))
211         doEvents(&eventRec);
212     }
213   }
214
215   // ############################################################################ doInitManagers
216
217   void  doInitManagers(void)
218   {
219     MaxApplZone();
220     MoreMasters();
221
222     InitGraf(&qd.thePort);
223     InitFonts();
224     InitWindows();
225     InitMenus();
226     TEInit();
227     InitDialogs(NULL);
228
229     InitCursor();
230     FlushEvents(everyEvent,0);
231   }
232
233   // ############################################################################## doEvents
234
235   void  doEvents(EventRecord *eventRecPtr)
236   {
237     WindowPtr windowPtr;
238     SInt8   charCode;
239
240     windowPtr = (WindowPtr) eventRecPtr->message;
241
242     switch(eventRecPtr->what)
243     {
244       case mouseDown:
245         doMouseDown(eventRecPtr);
246         break;
247
248       case keyDown:
249       case autoKey:
250         charCode = eventRecPtr->message & charCodeMask;
251         if((eventRecPtr->modifiers & cmdKey) != 0)
252         {
253           doAdjustMenus();
254           doMenuChoice(MenuKey(charCode));
255         }
256         break;
257
258       case updateEvt:
259         BeginUpdate(windowPtr);
260         EraseRgn(windowPtr->visRgn);
261         doUpdateWindow(windowPtr);
262         DrawGrowIcon(windowPtr);
263         EndUpdate(windowPtr);
264         break;
265
266       case osEvt:
267         switch((eventRecPtr->message >> 24) & 0x000000FF)
268         {
269           case suspendResumeMessage:
270             gInBackground = (eventRecPtr->message & resumeFlag) == 0;
271             break;
272         }
273         HiliteMenu(0);
274         break;
275     }
276   }
277
278   // ############################################################################ doMouseDown
279
280   void  doMouseDown(EventRecord *eventRecPtr)
281   {
282     WindowPtr windowPtr;
```

```
283     SInt16    partCode;
284     Rect      growRect;
285     SInt32    newSize;
286
287     partCode = FindWindow(eventRecPtr->where,&windowPtr);
288
289     switch(partCode)
290     {
291       case inMenuBar:
292         doAdjustMenus();
293         doMenuChoice(MenuSelect(eventRecPtr->where));
294         break;
295
296       case inSysWindow:
297         SystemClick(eventRecPtr,windowPtr);
298         break;
299
300       case inContent:
301         if(windowPtr != FrontWindow())
302           SelectWindow(windowPtr);
303         break;
304
305       case inDrag:
306         DragWindow(windowPtr,eventRecPtr->where,&qd.screenBits.bounds);
307         break;
308
309       case inGoAway:
310         if(TrackGoAway(windowPtr,eventRecPtr->where) == true)
311           doCloseCommand();
312         break;
313
314       case inGrow:
315         growRect = qd.screenBits.bounds;
316         growRect.top  = 145;
317         growRect.left = 345;
318         newSize = GrowWindow(windowPtr,eventRecPtr->where,&growRect);
319         if(newSize != 0)
320         {
321           invalidateScrollBarArea(windowPtr);
322           SizeWindow(windowPtr,LoWord(newSize),HiWord(newSize),true);
323           invalidateScrollBarArea(windowPtr);
324         }
325         break;
326
327       case inZoomIn:
328       case inZoomOut:
329         if(TrackBox(windowPtr,eventRecPtr->where,partCode))
330         {
331           SetPort(windowPtr);
332           EraseRect(&windowPtr->portRect);
333           ZoomWindow(windowPtr,partCode,false);
334           InvalRect(&windowPtr->portRect);
335         }
336         break;
337     }
338   }
339
340   // ############################################################################ doUpdateWindow
341
342   void  doUpdateWindow(WindowPtr windowPtr)
343   {
344     Str255 string;
345
346     SetPort(windowPtr);
347
348     MoveTo(10,20);
349     DrawString("\pCurrent Application Preferences:");
350     MoveTo(10,35);
351     DrawString("\pSound On: ");
352     if(gSoundOn)  DrawString("\pYES");
353     else  DrawString("\pNO");
354     MoveTo(10,50);
355     DrawString("\pFull Screen On: ");
356     if(gFullScreenOn)  DrawString("\pYES");
357     else  DrawString("\pNO");
358     MoveTo(10,65);
359     DrawString("\pAutoScroll On: ");
```

```
  if(gAutoScrollOn)  DrawString("\pYES");
  else  DrawString("\pNO");

  if(gPrintRect.bottom != 0)
  {
    MoveTo(10,85);
    DrawString("\pInformation from printable area ('PrAr') resource:");
    NumToString((SInt32) gPrintRect.bottom,string);
    MoveTo(10,100);
    DrawString("\pPage print area height in screen pixels: ");
    DrawString(string);
    NumToString((SInt32) gPrintRect.right,string);
    MoveTo(10,115);
    DrawString("\pPage print area width in screen pixels:  ");
    DrawString(string);
  }
  else
  {
    MoveTo(10,85);
    DrawString("\pNo printable area ('PrAr') resource saved yet");
  }
}

// ########################################################################## doAdjustMenus

void  doAdjustMenus(void)
{
  MenuHandle      menuHdl;

  if(gWindowOpen)
  {
    menuHdl = GetMenuHandle(mFile);
    DisableItem(menuHdl,iOpen);
    EnableItem(menuHdl,iClose);
    EnableItem(menuHdl,iPageSetup);
  }
  else
  {
    menuHdl = GetMenuHandle(mFile);
    EnableItem(menuHdl,iOpen);
    DisableItem(menuHdl,iClose);
    DisableItem(menuHdl,iPageSetup);
  }

  DrawMenuBar();
}

// ########################################################################## doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  SInt16  menuID, menuItem;
  Str255  itemName;
  SInt16  daDriverRefNum;

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

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mApple:
      GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
      daDriverRefNum = OpenDeskAcc(itemName);
      break;

    case mFile:
      doFileMenu(menuItem);
      break;

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

```
437
438     HiliteMenu(0);
439   }
440
441   // ############################################################################## doFileMenu
442
443   void  doFileMenu(SInt16 menuItem)
444   {
445     switch(menuItem)
446     {
447       case iClose:
448         doCloseCommand();
449         break;
450
451       case iOpen:
452         doOpenCommand();
453         break;
454
455       case iPageSetup:
456         doPrintStyleDialog();
457         break;
458
459       case iQuit:
460         while(FrontWindow())
461           doCloseCommand();
462         gDone = true;
463         break;
464     }
465   }
466
467   // ################################################################ invalidateScrollBarArea
468
469   void  invalidateScrollBarArea(WindowPtr windowPtr)
470   {
471     Rect  tempRect;
472
473     SetPort(windowPtr);
474
475     tempRect = windowPtr->portRect;
476     tempRect.left = tempRect.right - 15;
477     InvalRect(&tempRect);
478
479     tempRect = windowPtr->portRect;
480     tempRect.top = tempRect.bottom - 15;
481     InvalRect(&tempRect);
482   }
483
484   // ################################################################################ doError
485
486   void  doError(SInt16 errorCode)
487   {
488     Str255  errorString;
489
490     NumToString((SInt32) errorCode,errorString);
491     ParamText(errorString,NULL,NULL,NULL);
492
493     if(errorCode == memFullErr)
494     {
495       StopAlert(rAlertBox,NULL);
496       ExitToShell();
497     }
498     else
499       CautionAlert(rAlertBox,NULL);
500   }
501
502   // ############################################################################ doOpenCommand
503
504   void  doOpenCommand(void)
505   {
506     SFTypeList        fileTypes;
507     StandardFileReply fileReply;
508     docRecordHandle   docRecHdl;
509     OSErr             osError = 0;
510
511     fileTypes[0] = 'TEXT';
512
513     StandardGetFile(NULL,1,fileTypes,&fileReply);
```

```
514     if(!(fileReply.sfGood))
515       return;
516
517     if(!(gWindowPtr = GetNewWindow(rNewWindow,NULL,(WindowPtr)-1)))
518       return;
519
520     if(!(docRecHdl = (docRecordHandle) NewHandle(sizeof(docRecord))))
521     {
522       DisposeWindow(gWindowPtr);
523       return;
524     }
525
526     gWindowOpen = true;
527     SetPort(gWindowPtr);
528
529     SetWRefCon(gWindowPtr,(SInt32) docRecHdl);
530     (*docRecHdl)->fileFSSpec = fileReply.sfFile;
531     SetWTitle(gWindowPtr,(*docRecHdl)->fileFSSpec.name);
532
533     doGetandSetWindowPosition(gWindowPtr);
534     doGetPrintableSize(gWindowPtr);
535
536     ShowWindow(gWindowPtr);
537   }
538
539   // ############################################################################ doCloseCommand
540
541   void  doCloseCommand(void)
542   {
543     WindowPtr       windowPtr;
544     docRecordHandle docRecHdl;
545     OSErr           osError = 0;
546
547     windowPtr = FrontWindow();
548     docRecHdl = (docRecordHandle) GetWRefCon(windowPtr);
549
550     doSaveWindowPosition(windowPtr);
551
552     if(gPrintStyleChanged)
553       doSavePrintableSize(gWindowPtr);
554
555     DisposeHandle((Handle) docRecHdl);
556     DisposeWindow(windowPtr);
557     gWindowOpen = false;
558   }
559
560   // ######################################################################## doPreferencesDialog
561
562   void  doPreferencesDialog(void)
563   {
564     DialogPtr   modalDlgPtr;
565     GrafPtr     oldPort;
566     PenState    oldPenState;
567     SInt16      buttonOval, itemHit, itemType;
568     Handle      itemHdl;
569     Rect        itemRect;
570
571     if(!(modalDlgPtr = GetNewDialog(rModalDialog,NULL,(WindowPtr) -1)))
572       return;
573
574     GetDialogItem(modalDlgPtr,iSoundOn,&itemType,&itemHdl,&itemRect);
575     SetControlValue((ControlHandle) itemHdl,gSoundOn);
576     GetDialogItem(modalDlgPtr,iFullScreenOn,&itemType,&itemHdl,&itemRect);
577     SetControlValue((ControlHandle) itemHdl,gFullScreenOn);
578     GetDialogItem(modalDlgPtr,iAutoScrollOn,&itemType,&itemHdl,&itemRect);
579     SetControlValue((ControlHandle) itemHdl,gAutoScrollOn);
580
581     ShowWindow(modalDlgPtr);
582
583     GetPort(&oldPort);
584     GetPenState(&oldPenState);
585     GetDialogItem(modalDlgPtr,1,&itemType,&itemHdl,&itemRect);
586     SetPort((*(ControlHandle) itemHdl)->contrlOwner);
587     InsetRect(&itemRect,-4,-4);
588     PenPat(&qd.black);
589     PenSize(3,3);
590     buttonOval = (itemRect.bottom - itemRect.top) / 2 + 2;
```

```
591     FrameRoundRect(&itemRect,buttonOval,buttonOval);
592     SetPenState(&oldPenState);
593     SetPort(oldPort);
594
595     do
596     {
597       ModalDialog(NULL,&itemHit);
598       GetDialogItem(modalDlgPtr,itemHit,&itemType,&itemHdl,&itemRect);
599       SetControlValue((ControlHandle) itemHdl,!GetControlValue((ControlHandle) itemHdl));
600     } while((itemHit != 1) && (itemHit != 2));
601
602     if(itemHit == 1)
603     {
604       GetDialogItem(modalDlgPtr,iSoundOn,&itemType,&itemHdl,&itemRect);
605       gSoundOn = GetControlValue((ControlHandle) itemHdl);
606       GetDialogItem(modalDlgPtr,iFullScreenOn,&itemType,&itemHdl,&itemRect);
607       gFullScreenOn = GetControlValue((ControlHandle) itemHdl);
608       GetDialogItem(modalDlgPtr,iAutoScrollOn,&itemType,&itemHdl,&itemRect);
609       gAutoScrollOn = GetControlValue((ControlHandle) itemHdl);
610     }
611
612     DisposeDialog(modalDlgPtr);
613
614     if(gWindowPtr)
615       InvalRect(&gWindowPtr->portRect);
616
617     doSavePreferences();
618   }
619
620   // ######################################################################## doPrintStyleDialog
621
622   void  doPrintStyleDialog(void)
623   {
624     Boolean clickedOK;
625
626     PrOpen();
627
628     if(clickedOK = PrStlDialog(gTPrintHdl))
629     {
630       gPrintStyleChanged = true;
631       gPrintRect = (*gTPrintHdl)->prInfo.rPage;
632       InvalRect(&gWindowPtr->portRect);
633     }
634
635     PrClose();
636   }
637
638   // ######################################################################## doGetPreferences
639
640   void  doGetPreferences(void)
641   {
642     Str255          prefsFileName;
643     OSErr           osError;
644     SInt16          volRefNum;
645     SInt32          directoryID;
646     FSSpec          fileSSpec;
647     SInt16          fileRefNum;
648     appPrefsHandle  appPrefsHdl;
649
650     GetIndString(prefsFileName,rStringList,iPrefsFileName);
651
652     osError = FindFolder(kOnSystemDisk,kPreferencesFolderType,kDontCreateFolder,&volRefNum,
653                          &directoryID);
654
655     if(osError == noErr)
656       osError = FSMakeFSSpec(volRefNum,directoryID,prefsFileName,&fileSSpec);
657     if(osError == noErr || osError == fnfErr)
658       fileRefNum = FSpOpenResFile(&fileSSpec,fsCurPerm);
659
660     if(fileRefNum == -1)
661     {
662       FSpCreateResFile(&fileSSpec,'PpPp','pref',smSystemScript);
663       osError = ResError();
664
665       if(osError == noErr)
666       {
667         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;

    gSoundOn      = (*appPrefsHdl)->soundOn;
    gFullScreenOn = (*appPrefsHdl)->fullScreenOn;
    gAutoScrollOn = (*appPrefsHdl)->autoScrollOn;

    gPrefsFileRefNum = fileRefNum;

    UseResFile(gAppResFileRefNum);
  }
}

// ############################################################################ 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);
}
```

```
// ######################################################################## doSavePreferences

void  doSavePreferences(void)
{
  appPrefsHandle  appPrefsHdl;
  Handle          existingResHdl;
  Str255          resourceName = "\pPreferences";

  if(gPrefsFileRefNum == -1)
    return;

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

  HLock((Handle) appPrefsHdl);

  (*appPrefsHdl)->soundOn = gSoundOn;
  (*appPrefsHdl)->fullScreenOn = gFullScreenOn;
  (*appPrefsHdl)->autoScrollOn = gAutoScrollOn;

  UseResFile(gPrefsFileRefNum);

  existingResHdl = Get1Resource(rTypePrefs,kPrefsID);
  if(existingResHdl != NULL)
  {
    RemoveResource(existingResHdl);
    if(ResError() == noErr)
      AddResource((Handle) appPrefsHdl,rTypePrefs,kPrefsID,resourceName);
    if(ResError() == noErr)
      WriteResource((Handle) appPrefsHdl);
  }

  HUnlock((Handle) appPrefsHdl);

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

// ############################################################ doGetandSetWindowPosition

void  doGetandSetWindowPosition(WindowPtr windowPtr)
{
  Rect            userStateRect, stdStateRect, displayRect;
  docRecordHandle docRecHdl;
  SInt16          fileRefNum;
  winStateHandle  winStateHdl;
  Boolean         gotResource;
  OSErr           osError;

  userStateRect = qd.screenBits.bounds;
  SetRect(&userStateRect,userStateRect.left + 3,userStateRect.top + 42,
          userStateRect.right - 40,userStateRect.bottom - 6);

  stdStateRect = qd.screenBits.bounds;
  SetRect(&stdStateRect,stdStateRect.left + 3,stdStateRect.top + 42,
          stdStateRect.right - 3,stdStateRect.bottom - 6);

  docRecHdl = (docRecordHandle) GetWRefCon(windowPtr);

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

  winStateHdl = (winStateHandle) Get1Resource(rTypeWinState,kWinStateID);
  if(winStateHdl != NULL )
  {
    gotResource = true;
    userStateRect = (*winStateHdl)->userStateRect;
  }
  else
    gotResource = false;

  if(gotResource)
  {
```

```
822       if((*winStateHdl)->zoomState)
823         displayRect = stdStateRect;
824       else
825         displayRect = userStateRect;
826     }
827     else
828     {
829       displayRect = userStateRect;
830     }
831
832     MoveWindow(windowPtr,displayRect.left,displayRect.top,false);
833
834     GlobalToLocal(&topLeft(displayRect));
835     GlobalToLocal(&botRight(displayRect));
836     SizeWindow(windowPtr,displayRect.right,displayRect.bottom,true);
837
838     doSetWindowState(windowPtr,userStateRect,stdStateRect);
839
840     ReleaseResource((Handle) winStateHdl);
841     CloseResFile(fileRefNum);
842   }
843
844   // ############################################################## doSaveWindowPosition
845
846   void  doSaveWindowPosition(WindowPtr windowPtr)
847   {
848     docRecordHandle docRecHdl;
849     SInt16          fileRefNum;
850     WindowPeek      windowRecPtr;
851     WStateData      *winStateDataPtr;
852     Rect            stdRect, userRect;
853     RgnHandle       contentRgnHdl;
854     winState        userRectAndZoomState;
855     winStateHandle  winStateHdl;
856     OSErr           osError;
857
858     docRecHdl = (docRecordHandle) GetWRefCon(windowPtr);
859
860     fileRefNum = FSpOpenResFile(&(*docRecHdl)->fileFSSpec,fsRdWrPerm);
861     if(fileRefNum < 0)
862     {
863       osError = ResError();
864       doError(osError);
865       return;
866     }
867
868     windowRecPtr = (WindowPeek) windowPtr;
869     winStateDataPtr = (WStateData *) *(windowRecPtr->dataHandle);
870     stdRect = winStateDataPtr->stdState;
871     userRect = winStateDataPtr->userState;
872
873     contentRgnHdl = windowRecPtr->contRgn;
874     userRectAndZoomState.userStateRect = (*contentRgnHdl)->rgnBBox;
875     userRectAndZoomState.zoomState = EqualRect(&userRectAndZoomState.userStateRect,&stdRect);
876     if(userRectAndZoomState.zoomState)
877       userRectAndZoomState.userStateRect = userRect;
878
879     winStateHdl = (winStateHandle) Get1Resource(rTypeWinState,kWinStateID);
880     if(winStateHdl != NULL)
881     {
882       **winStateHdl = userRectAndZoomState;
883       ChangedResource((Handle) winStateHdl);
884       osError = ResError();
885       if(osError != noErr)
886         doError(osError);
887     }
888     else
889     {
890       winStateHdl = (winStateHandle) NewHandle(sizeof(winState));
891       if(winStateHdl != NULL)
892       {
893         **winStateHdl = userRectAndZoomState;
894         AddResource((Handle) winStateHdl,rTypeWinState,kWinStateID,"\pLast window state");
895       }
896     }
897
898     if(winStateHdl != NULL)
```

```
899     {
900       UpdateResFile(fileRefNum);
901       osError = ResError();
902       if(osError != noErr)
903         doError(osError);
904 
905       ReleaseResource((Handle) winStateHdl);
906     }
907 
908     CloseResFile(fileRefNum);
909   }
910 
911   // ########################################################################## doSetWindowState
912 
913   void  doSetWindowState(WindowPtr windowPtr,Rect userStateRect,Rect stdStateRect)
914   {
915     WindowPeek  windowRecPtr;
916     WStateData  *winStateDataPtr;
917 
918     windowRecPtr = (WindowPeek) windowPtr;
919     winStateDataPtr = (WStateData *) *(windowRecPtr->dataHandle);
920     winStateDataPtr->userState = userStateRect;
921     winStateDataPtr->stdState = stdStateRect;
922   }
923 
924   // ########################################################################## doGetPrintableSize
925 
926   void  doGetPrintableSize(WindowPtr windowPtr)
927   {
928     docRecordHandle   docRecHdl;
929     SInt16            fileRefNum;
930     OSErr             osError;
931     rectHandle        printRectHdl;
932 
933     docRecHdl = (docRecordHandle) GetWRefCon(windowPtr);
934 
935     fileRefNum = FSpOpenResFile(&(*docRecHdl)->fileFSSpec,fsRdWrPerm);
936     if(fileRefNum < 0)
937     {
938       osError = ResError();
939       doError(osError);
940       return;
941     }
942 
943     printRectHdl = (rectHandle) Get1Resource(rTypePrintRect,kPrintRectID);
944     if(printRectHdl != NULL)
945     {
946       gPrintRect = **printRectHdl;
947       ReleaseResource((Handle) printRectHdl);
948     }
949 
950     CloseResFile(fileRefNum);
951   }
952 
953   // ########################################################################## doSavePrintableSize
954 
955   void  doSavePrintableSize(WindowPtr windowPtr)
956   {
957     docRecordHandle docRecHdl;
958     SInt16          fileRefNum;
959     rectHandle      printRectHdl;
960     OSErr           osError;
961 
962     docRecHdl = (docRecordHandle) GetWRefCon(windowPtr);
963 
964     fileRefNum = FSpOpenResFile(&(*docRecHdl)->fileFSSpec,fsRdWrPerm);
965     if(fileRefNum < 0)
966     {
967       osError = ResError();
968       doError(osError);
969       return;
970     }
971 
972     printRectHdl = (rectHandle) Get1Resource(rTypePrintRect,kPrintRectID);
973     if(printRectHdl != NULL)
974     {
975       **printRectHdl = (*gTPrintHdl)->prInfo.rPage;
```

```
      ChangedResource((Handle) printRectHdl);
      osError = ResError();
      if(osError != noErr)
        doError(osError);
    }
    else
    {
      printRectHdl = (rectHandle) NewHandle(sizeof(Rect));
      if(printRectHdl != NULL)
      {
        **printRectHdl = (*gTPrintHdl)->prInfo.rPage;
        AddResource((Handle) printRectHdl,rTypePrintRect,kPrintRectID,"\pPrint rectangle");
      }
    }

    if(printRectHdl != NULL)
    {
      UpdateResFile(fileRefNum);
      osError = ResError();
      if(osError != noErr)
        doError(osError);

      ReleaseResource((Handle) printRectHdl);
    }

    gPrintStyleChanged = false;

    CloseResFile(fileRefNum);
  }

  // ############################################################################################
```

Demonstration Program Comments

When this program is run for the first time, a preferences file (titled "MoreResources Preferences") is created in the Preferences folder in the System folder and two resources are copied to the resource fork of that file from the program's resource file. These two resources are a custom preferences ('PrFn') resource and a "application missing" 'STR ' resource. Thereafter, the preferences resource will be read in from the preferences file every time the program is run and replaced whenever the user invokes the Preferences dialog box to change the application preferences settings. In addition, if the user double clicks on the preferences file's icon, an alert box is invoked displaying the text contained in the "application missing" 'STR ' resource. (Note that this latter will not occur when the program is run under system software version 7.5 or later and automatic document translation is selected to on in the Macintosh Easy Open control panel.)

After the program is launched, the user should choose Open from the File menu to open the included demonstration document file titled "Document"). The resource fork of this file contains two custom resources, namely, a 'WiSt' resource containing the last saved window user state and zoom state, and a 'PrAr' resource containing the last saved printable area rectangle of the currently chosen paper size. These two resources are read in whenever the document file is opened and written to whenever the file is closed. (Actually, the 'PrAr' resource is written to only if the user invoked the print Style dialog box 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 the current printable area (that is, page rectangle) values.

The user should choose different paper size, scaling and orientation settings in the print style dialog box, resize or zoom the window, close the file, re-open the file, and note that, firstly, the saved printable area values are correctly retrieved and, secondly, the window is re-opened in the size and zoom state in which is was closed. The user should also change the application preferences settings via the Preferences dialog box (which is invoked when the single item in the Demonstration 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 'WiSt' and 'PrAr' resources from the document file, run the program, force a 'PrAr' resource to be created and written to by invoking the print Style dialog box while the document file is open, quit the program, and re-run the program, noting that 'WiSt' and 'PrAr' resources are created in the document file's resource fork if they do not already exist.

When done, the user should remove the preferences file from the Preferences folder in the System folder.

#define

Lines 72-79 establish constants relating to menu IDs and menu item numbers. Lines 80-82 establish constants relating to window, menubar and alert resource IDs.

The constants at Lines 83-86 relate to the Preferences dialog box resource and associated checkbox item numbers. Lines 87-88 represent the resource ID and index for the string containing the name of the application's preferences file. Lines 89-96 represent resource types and IDs for the custom printable area resource, the custom window state resource, the custom preferences resource, and the application missing string resource.

Line 97 defines MAXLONG as the maximum possible long value. Lines 98-99 define two common macros. The first converts the top and left fields of a Rect to a Point. The second converts the bottom and right field of a Rect to a Point.

#typedef

The docRecord data type (Lines 103-106) is for the document record. In this demonstration, the only field required is that for a file system specification.

The appPrefs data type (Lines 108-113) is for the application preferences settings. The three Boolean values are set by checkboxes in the Preferences dialog box.

The winState data type (Lines 115-119) 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 rectHandle data type (Line 121) will be used in the functions related to the getting and saving of the printable area width and height.

Global Variables

gDone controls exit from the main event loop and thus program termination. gInBackground relates to foreground/background switching. gTPrintHdl will be assigned a handle to a TPrint record, the latter being required because of the use by the program of the print style dialog. gWindowPtr will be assigned the pointer to the window's graphics port. gWindowOpen is used to control File menu item enabling/disabling according to whether the window is open or closed.

gPrintStyleChanged is set to true when the print style dialog is invoked, and determines whether a new printable area resource will be written to the document file when the file is closed. gPrintRect will be assigned the rectangle representing the printable area.

gSoundOn, gFullScreenOn, and gAutoScrollOn will hold the application preferences settings.

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

main

The main function initialises the system software managers (Line 173), sets the application's resource fork as the current resource file (Line 177), sets up the menus (Lines 181-191), and creates and initialises a TPrint record (195-198). Then, before the main loop (Lines 206-212) is entered, main calls the application-defined function which reads in the application preferences settings from the preferences file. (As will be seen, if the preferences file 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.)

doEvents, doMouseDown, doUpdateWindow

doEvents and doMouseDown perform such event processing as is necessary for the satisfactory execution of the demonstration aspects of the program.

doUpdateWindow simply prints the current preferences and printable area information in the window for the information of the user.

doAdjustMenus, doMenuChoice, doFileMenu

doAdjustMenus controls File menu item enabling and disabling according to whether the document window is opened or closed. doMenuChoice and doFileMenu handle menu choices from the Apple, File and Demonstration menus.

InvalidateScrollBarArea, doError

InvalidateScrollBarArea invalidates the areas occupied by the scroll bars whenever the window is resized.

doError presents an alert box displaying the error code passed to it. In the case of a memFullErr code, a stop alert is presented and the program is terminated when the user clicks the OK button. In all other cases, a caution alert is presented and the program continues when the user clicks the OK button.

doOpenCommand

doOpenCommand is a much simplified version of the actions normally taken when a user chooses the Open command from a File menu.

The standard Open dialog box is presented (Line 513) and, if the user clicks the Cancel button, the function simply returns. If the user clicks the OK button, a window is opened (Line 517), a document record is created (Line 520), a flag is set to indicate that the window is open (Line 526), the window's graphics port is set as the current port for drawing (Line 527), the document record is connected to the window record (Line 529), the file system specification for the chosen file is assigned to the document record's file system specification field (Line 530), and the window's title is set (Line 531).

At that point, the application-defined function which reads in the window state resource from the document's resource fork, and positions and sizes the window accordingly, is called (Line 533). In addition, the application-defined function which reads in the printable area resource from the document's resource fork is called (Line 534).

With the window positioned and sized, ShowWindow is called (Line 536) to make the window visible. (The window's 'WIND' resource specifies that the window is to be initially invisible.)

doCloseCommand

doCloseCommand is a much simplified version of the actions normally taken when a user chooses the Close command from a File menu.

At Lines 547-548, a pointer to the front window, and a handle to the associated document record, are retrieved.

Line 550 calls the application-defined function which 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 (Line 552), a call is made to the application-defined function which saves the printable area rectangle to the printable area resource in the document file's resource fork (Line 553).

Line 555 disposes of the document record, Line 556 disposes of the window record, and Line 557 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 in the Demonstration menu. The function presents the Preferences dialog box 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 Line 617, a call is made to the application-defined function which saves the dialog box's preference settings to the resource fork of the preferences file.

doPrintStyleDialog

doPrintStyleDialog is called when the user chooses the Page Setup… item in the File menu. It presents the print style dialog box (Line 628).

If the user dismisses the dialog with a click on the OK button, the flag which indicates that a print style change has been made is set to true (Line 630), and the global variable which holds the printable rectangle is assigned the value in the rPage (printable page size) field of the TPrInfo record, a handle to which is at the prInfo field of the TPrint record (Line 631). In addition, the window's port rectangle is invalidated (Line 632) to force an update of the window, thus ensuring that the new printable area values are displayed immediately.

doGetPreferences

doGetPreferences, which is called from the main function immediately after program launch, is the first of those application-defined functions which are 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 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.

Line 650 retrieves from the application's resource file the resource containing the required name of the preferences file ("MoreResources Preferences").

Line 652 finds the location of the Preferences folder, returning the volume reference number and directory ID in the last two parameters. Line 656 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 (Line 658) 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 this case, Line 662 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 box. 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 (Line 667) and the master preferences ('PrFn') and application missing 'STR ' resources are copied to the resource fork from the application's resource file (Lines 672 and 674). If the resources are not successfully copied (Line 675), the resource fork of the new file is closed (Line 677), the file is deleted (Line 678), and the fileRefNum variable is set to indicate that the file does not exist (line 679).

If the preferences file exists (either previously or newly-created) (Line 685), the resource fork of that file is set as the current resource file (Line 687), the preferences resource is read in from the resource fork (Line 689) and, if the read was successful, the three Boolean values are assigned to the global variables which store those values (Lines 693-695). (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.)

Line 697 assigns the file reference number for the open preferences file resource fork to a global variable (the fork is left open), and Line 699 resets the application's resource fork as the current resource file.

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.

Line 716 saves the current resource file's file reference number and Line 717 sets the application's resource fork as the current resource file. This will be the "source" file.

The Get1Resource call at Line 719 reads the specified resource into memory. Line 723 gets the resource's name and Line 724 gets the resource's attributes. The call to DetachResource at Line 725 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.

Line 726 sets the preferences file's resource fork as the current resource file. The AddResource call (Line 728) makes the arbitrary data in memory into a resource, assigning it the specified type, ID and name. Line 730 sets the resource attributes in the resource map. The ChangedResource call (Line 732) tags the resource for update and pre-allocates the required disk space. The WriteResource call (Line 734) then writes the resource to disk.

With the resource written to disk, Line 739 discards the resource in memory and Line 740 resets the resource file saved at Line 716 as the current resource file.

doSavePreferences

doSavePreferences is called when the user dismisses the preferences dialog box to save the new preference settings to the preferences file. It assumes that the preferences file is already open.

If doGetPreferences was not successful in opening the preferences file at program launch, the function simply returns (Lines 753-754).

Lines 756-762 create a new preferences record and assign to its fields the values in the global variables which store the current preference settings. Line 764 makes the preferences file's resource fork the current resource file. The Get1Resource call at Line 766 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 (Line 769), AddResource is called to make the preferences record in memory into a resource (Line 771), and WriteResource is called to write the resource to disk (Line 773).

With the resource written to disk, Line 778 disposes of the preferences record in memory and Line 779 resets the application's resource fork as the current resource file.

doGetandSetWindowPosition

doGetandSetWindowPosition 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.

Lines 793-795 establish a default user state rectangle to cater for the possibility that the document file may not yet have a 'WiSt' resource in its resource fork. Lines 797-799 establish the standard state rectangle as desired by the application.

Line 801 gets a handle to the window's document record so that the file system specification can be retrieved and used in the FSpOpenResFile call (Line 803) to open the document file's resource fork.

Line 811 attempts to read in the 'WiSt' resource. If the Get1Resource call is successful (Line 812), a "success" flag is set and the user state rectangle is set to that retrieved from the resource (Lines 814-815). If the call is not successful, the "success" flag is unset (Lines 817-818) and the user state rectangle remains as the default rectangle defined at Lines 793-795.

If the Get1Resource call was successful, the zoom state is also retrieved from the resource (Line 822). If the zoom state is "zoomed out" to the standard state, the rectangle to be used to display the window is set to the standard state (Line 823). If the zoom state is "zoomed in" to the user state, the rectangle to be used to display the window is set to the user state (Line 825). If the Get1Resource call was not successful (Line 827) the display rectangle is set to the user state rectangle, which will be the default defined at Lines 793-795.

Line 832 moves the window to the specified coordinates, keeping it inactive. Lines 834-836 size the window to the specified size, adding any area added to the content region to the update region.

Line 838 calls an application-defined function which assigns the specified rectangles to the userState and stdState fields of the WStateData record for the window. With this action completed, Line 840 discards the 'WiSt' resource in memory. Line 841 then closes the document file's resource fork.

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.

Line 858 gets a handle to the window's document record so that the document file's file system specification can be retrieved and used in the FSpOpenResFile call at Line 860. If the resource fork cannot be opened, an error alert is presented and the function simply returns (Lines 861-866).

Line 868 gets a pointer to the window record, allowing Line 869 to get a pointer to the WStateData record. Lines 870-871 retrieve the current standard state and user state rectangles from the WStateData record.

The next step is to determine whether the window is currently in the "zoomed out" (standard) state or the "zoomed in" (user) state. Lines 873-874 get a rectangle equal to the content region of the window. Line 875 sets up a forthcoming test by assigning this rectangle to the userStateRect field of a window state record. The test is at the next line: If the content region rectangle equals the current standard state rectangle, the call to EqualRect at Line 875 will return true, in which case:

- The zoomstate field of the window state record is assigned a value indicating that the window is in the standard state.

- The userStateRect field of the window state record is assigned the current user state rectangle.

If, on the other hand, the content region rectangle does not equal the current standard state rectangle, the call to EqualRect at Line 875 will return false, in which case:

- The zoomstate field of the window state record is assigned a value indicating that the window is in the user state.

- The userStateRect field of the window state record retains the rectangle it was assigned at Line 874 which, not being equal to the standard state rectangle (Line 875), must be equal to the current user state rectangle.

Line 879 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" window state record (Line 882) and the resource is tagged as changed (Line 883). If the Get1Resource call is not successful (that is, the document file's resource fork does not yet contain a 'WiSt' resource), Line 890 creates a new window state record, Line 893 makes this record equal to the previously "filled-in" window state record, and Line 894 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 (Line 898), Line 900 writes the resource map and data to disk, and Line 905 discards the resource in memory. The document file's resource fork is then closed (Line 908).

doSetWindowState

doSetWindowState is called by doGetandSetWindowPosition to assign the user and standard state rectangles defined by that function to the userState and stdState fields of the window's WStateData record.

doGetPrintableSize

doGetPrintableSize gets the rectangle representing the printable area of the chosen page size from the 'PrAr' resource in the document file's resource fork. The function is called when the document is opened.

Line 933 gets a handle to the window's document record so that the document file's file system specification can be retrieved and used in the call to FSpOpenResFile at Line 935. If the call is not successful, an error alert box is presented and the function simply returns (Lines 936-941).

If the resource fork is successfully opened, the call to Get1Resource at Line 943 attempts to read in the resource. If the call is successful (Line 944), Line 946 assigns the data in the resource in memory to the global variable which stores the current printable area rectangle. The resource in memory is then discarded (Line 947) and the document file's resource fork is closed (Line 950).

doSavePrintableSize

doSavePrintableSize saves the printable area rectangle for the currently chosen paper size to a 'PrAr' resource in the document file's resource fork. The function is called when the file is closed if the user invoked the print Style dialog while the document was open and dismissed the dialog by clicking the OK button.

Line 962 gets a handle to the window's document record so that the document file's file system specification can be retrieved and used in the call to FSpOpenResFile at Line 964. If the call is not successful, an error alert box is presented and the function simply returns (Lines 965-970).

Line 972 attempts to read the 'PrAr' resource from the document's resource fork into memory. If the Get1Resource call is successful, the resource in memory is made equal to the rectangle in the prPage field of the prInfo record, which is itself part of the TPrint record, and the resource is tagged as changed (Lines 973-976). If the Get1Resource call is not successful

(that is, the document file's resource fork does not yet contain a 'PrAr' resource), Line 983 allocates a block of memory for a Rect, Line 986 copies the rectangle in the prPage field of the prInfo record to this block, and Line 987 makes this data in memory into a 'PrAr' resource.

If an existing 'PrAr' resource was successfully read in, or if a new 'PrAr' resource was successfully created in memory (Line 991), Line 993 writes the resource map and data to disk, and Line 998 discards the resource in memory. The document file's resource fork is then closed (Line 1003).