17

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 function 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 system-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	When a user opens an application, system software automatically opens the application's resource fork.

	dialog boxes or help balloons.	
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 functions 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

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.

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 functions (such as Get1Resource) which look only in the current resource file's resource map when searching for a specific resource.

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 function 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 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

The system software automatically opens your application's resource fork at application launch. Your application should simply call CurResFile early in its initialisation function 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 structure, 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 structure 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 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, FSpCreateResFile:

    - 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, 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 functions 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 functions which read resources from a resource fork. Depending on which function 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 functions 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 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.

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 functions. However, in general, you should always use Resource Manager functions.

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 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 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 ' 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 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

```
FourCharCode = PACKED ARRAY[1..4] OF CHAR;      { 68K }
FourCharCode = UNSIGNEDLONG;                           { PowerPC }
ResType = FourCharCode;
```

Functions

Initialising the Resource Manager

```
FUNCTION  InitResources: INTEGER;
```

Checking for Errors

```
FUNCTION  ResError: OSErr;
```

Creating an Empty Resource Fork

```
PROCEDURE FSpCreateResFile({CONST}VAR spec: FSSpec; creator: OSType; fileType: OSType;
              scriptTag: ScriptCode);
```

Opening Resource Forks

```
FUNCTION  FSpOpenResFile({CONST}VAR spec: FSSpec; permission: SignedByte): INTEGER;
```

Getting and Setting the Current Resource File

```
PROCEDURE UseResFile(refNum: INTEGER);
FUNCTION  CurResFile: INTEGER;
FUNCTION  HomeResFile(theResource: Handle): INTEGER;
```

Reading Resources Into Memory

```
FUNCTION  GetResource(theType: ResType; theID: INTEGER): Handle;
FUNCTION  Get1Resource(theType: ResType; theID: INTEGER): Handle;
FUNCTION  GetNamedResource(theType: ResType; name: Str255): Handle;
FUNCTION  Get1NamedResource(theType: ResType; name: Str255): Handle;
PROCEDURE SetResLoad(load: BOOLEAN);
PROCEDURE LoadResource(theResource: Handle);
```

Getting and Setting Resource Information

```
PROCEDURE GetResInfo(theResource: Handle; VAR theID: INTEGER; VAR theType: ResType;
              VAR name: Str255);
PROCEDURE SetResInfo(theResource: Handle; theID: INTEGER; name: Str255);
FUNCTION  GetResAttrs(theResource: Handle): INTEGER;
PROCEDURE SetResAttrs(theResource: Handle; attrs: INTEGER);
```

Modifying Resources

```
PROCEDURE ChangedResource(theResource: Handle);
PROCEDURE AddResource(theData: Handle; theType: ResType; theID: INTEGER; name: Str255);
```

Writing to Resource Forks

```
PROCEDURE UpdateResFile(refNum: INTEGER);
PROCEDURE WriteResource(theResource: Handle);
```

Getting a Unique Resource ID

```
FUNCTION  UniqueID(theType: ResType): INTEGER;
FUNCTION  Unique1ID(theType: ResType): INTEGER;
```

Counting and Listing Resource Types

```
FUNCTION  CountResources(theType: ResType): INTEGER;
FUNCTION  Count1Resources(theType: ResType): INTEGER;
FUNCTION  GetIndResource(theType: ResType; index: INTEGER): Handle;
FUNCTION  Get1IndResource(theType: ResType; index: INTEGER): Handle;
FUNCTION  CountTypes: INTEGER;
FUNCTION  Count1Types: INTEGER;
PROCEDURE GetIndType(VAR theType: ResType; index: INTEGER);
PROCEDURE Get1IndType(VAR theType: ResType; index: INTEGER);
```

Getting Resource Sizes

```
FUNCTION  GetResourceSizeOnDisk(theResource: Handle): LONGINT;
FUNCTION  GetMaxResourceSize(theResource: Handle): LONGINT;
```

Disposing of Resources and Closing Resource Forks

```
PROCEDURE ReleaseResource(theResource: Handle);
PROCEDURE DetachResource(theResource: Handle);
PROCEDURE RemoveResource(theResource: Handle);
PROCEDURE CloseResFile(refNum: INTEGER);
```

Getting and Setting Resource Fork Attributes

```
FUNCTION  GetResFileAttrs(refNum: INTEGER): INTEGER;
PROCEDURE SetResFileAttrs(refNum: INTEGER; attrs: INTEGER);
```

Demonstration Program

```
{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  MoreResources.p
 ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program uses custom resources to:
//
// •   Store application preferences in the resource fork of a preferences file, and also
//     to assist in the initial creation of the preferences file.
//
// •   Store, in the resource fork of a document file, the user state and current state of
//     the window associated with the document.
//
// •   Store, in the resource fork of a document file, the width and height of the
//     printable area of the paper size chosen in the print Style dialog box.
//
// The program utilises the following standard resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration
//     menus (preload, non-purgeable).
//
// •   A 'WIND' resource (purgeable) (initially invisible).
//
// •   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 and is32BitCompatible flags
//     set.
//
// The program 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 'WiSt' (window state) 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 user state rectangle (a Rect structure) and zoom state (a boolean value).
//
// •   A 'PrAr' (printable area) resource, which is created in the resource fork of the
//     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.
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

program MoreResources;

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

uses

    Appearance, Devices, Folders, Fonts, Printing, Resources, StandardFile, TextUtils,
    ToolUtils;

//
.....................................................................................................................................................................
............................................... constants

const

mApple = 128;
mFile = 129;
iOpen = 2;
iClose = 4;
iPageSetup = 8;
iQuit = 11;
mDemonstration = 131;
iPreferences = 1;
rNewWindow = 128;
rMenubar = 128;
rPrefsDialog = 128;
iSoundOn = 4;
iFullScreenOn = 5;
iAutoScrollOn = 6;
rStringList = 128;
iPrefsFileName = 1;
rTypePrintRect = 'PrAr';
kPrintRectID = 128;
rTypeWinState = 'WiSt';
kWinStateID = 128;
rTypePrefs = 'PrFn';
kPrefsID = 128;
rTypeAppMiss = 'STR ';
kAppMissID = -16397;
MAXLONG = $7FFFFFFF;

//
.....................................................................................................................................................................
.......................... type definitions

type

DocRecord = record
    fileFSSpec : FSSpec;
    end;
DocRecordPointer = ^DocRecord;
DocRecordHandle = ^DocRecordPointer;

AppPrefs = record
    soundOn : boolean;
    fullScreenOn : boolean;
    autoScrollOn : boolean;
    end;
AppPrefsPointer = ^AppPrefs;
```

```
AppPrefsHandle = ^AppPrefsPointer;

WinState = record
   userStateRect : Rect;
   zoomState : boolean;
   end;
WinStatePtr = ^WinState;
WinStateHandle = ^WinStatePtr;

RectHandle = ^RectPtr;

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

var

gDone : boolean;
gInBackground : boolean;
gTPrintHdl : THPrint;
gWindowOpen  : boolean;
gPrintStyleChanged : boolean;
gPrintRect : Rect;
gSoundOn : boolean;
gFullScreenOn : boolean;
gAutoScrollOn : boolean;
gAppResFileRefNum : SInt16;
gPrefsFileRefNum : SInt16;

// ............................................................................................................................................................. main
program block variables

mainMenubarHdl : Handle;
mainMenuHdl : MenuHandle;
mainEvent : EventRecord;

//
.........................................................................................................................................................................
............ routine prototypes

procedure DoInitManagers; forward;
procedure DoEvents({const} var theEvent : EventRecord); forward;
procedure DoUpdateWindow(theWindowPtr : WindowPtr); forward;
procedure DoAdjustMenus; forward;
procedure DoMenuChoice(menuChoice : SInt32); forward;
procedure DoErrorAlert(errorCode : SInt16); forward;
procedure DoOpenCommand; forward;
procedure DoCloseCommand; forward;
procedure DoPreferencesDialog; forward;
procedure DoPrintStyleDialog; forward;
procedure DoGetPreferences; forward;
function  DoCopyResource(theResType : ResType;
            resID, sourceFileRefNum, destFileRefNum : SInt16) : OSErr; forward;
procedure DoSavePreferences; forward;
procedure DoGetandSetWindowPosition(theWindowPtr : WindowPtr); forward;
procedure DoSaveWindowPosition(theWindowPtr : WindowPtr); forward;
procedure DoSetWindowState(theWindowPtr : WindowPtr; userStateRect, stdStateRect : Rect); forward;
procedure DoGetPrintableSize(theWindowPtr : WindowPtr); forward;
procedure DoSavePrintableSize(theWindowPtr : WindowPtr); forward;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoInitManagers

procedure DoInitManagers;
   var
   osError : OSErr;

   begin
   MaxApplZone;
   MoreMasters;

   InitGraf(@qd.thePort);
   InitFonts;
   InitWindows;
   InitMenus;
   TEInit;
   InitDialogs(nil);

   InitCursor;
   FlushEvents(everyEvent, 0);
```

```
    osError := RegisterAppearanceClient;

    end;
        { of procedure DoInitManagers }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoEvents

procedure DoEvents({const} var theEvent : EventRecord);
    var
    theWindowPtr : WindowPtr;
    growRect : Rect;
    newSize : longint;
    charCode : SInt8;
    partCode : SInt16;

    begin
    theWindowPtr := WindowPtr(theEvent.message);

    case theEvent.what of
        mouseDown: begin
            partCode := FindWindow(theEvent.where, theWindowPtr);

            case partCode of

                inMenuBar: begin
                    DoAdjustMenus;
                    DoMenuChoice(MenuSelect(theEvent.where));
                    end;

                inContent: begin
                    if (theWindowPtr <> FrontWindow) then
                        begin
                        SelectWindow(theWindowPtr);
                        end;
                    end;

                inDrag: begin
                    DragWindow(theWindowPtr, theEvent.where, qd.screenBits.bounds);
                    end;

                inGoAway: begin
                    if TrackGoAway(theWindowPtr, theEvent.where) then
                        begin
                        DoCloseCommand;
                        end;
                    end;

                inGrow: begin
                    growRect := qd.screenBits.bounds;
                    growRect.top := 145;
                    growRect.left := 345;
                    newSize := GrowWindow(theWindowPtr, theEvent.where, growRect);
                    if (newSize <> 0) then
                        begin
                        SizeWindow(theWindowPtr, LoWord(newSize), HiWord(newSize), true);
                        end;
                    end;

                inZoomIn, inZoomOut: begin
                    if TrackBox(theWindowPtr, theEvent.where, partCode) then
                        begin
                        ZoomWindow(theWindowPtr, partCode, false);
                        end;
                    end;

                otherwise begin
                    end;

                end;
                    { of case statement }
            end;

        keyDown, autoKey: begin
            charCode := SInt8(BAnd(theEvent.message, charCodeMask));
            if (BAnd(theEvent.modifiers, cmdKey) <> 0) then
                begin
                DoAdjustMenus;
                DoMenuChoice(MenuEvent(theEvent));
                end;
            end;
        updateEvt: begin
            BeginUpdate(theWindowPtr);
            DoUpdateWindow(theWindowPtr);
            EndUpdate(theWindowPtr);
            end;

        osEvt: begin
            if (BAnd(BSR(theEvent.message, 24), $000000FF) = suspendResumeMessage) then
                begin
                gInBackground := BAnd(theEvent.message, resumeFlag) = 0;
                end;
            HiliteMenu(0);
            end;

        otherwise begin
            end;

        end;
            { of case statement }
    end;
        { of procedure DoEvents }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoUpdateWindow

procedure DoUpdateWindow(theWindowPtr : WindowPtr);
    var
    theString : Str255;
    whiteColour : RGBColor;
    blueColour : RGBColor;

    begin
    whiteColour.red    := $FFFF;
    whiteColour.green := $FFFF;
    whiteColour.blue := $FFFF;
    blueColour.red := $1818;
    blueColour.green := $4B4B;
    blueColour.blue := $8181;

    RGBForeColor(whiteColour);
    RGBBackColor(blueColour);
    EraseRect(theWindowPtr^.portRect);

    SetPort(theWindowPtr);

    MoveTo(10, 20);
    TextFace([bold]);
    DrawString('Current Application Preferences:');
    TextFace([]);
    MoveTo(10, 35);
    DrawString('Sound On:  ');
    if gSoundOn then
        begin
        DrawString('YES');
        end
    else begin
        DrawString('NO');
        end;

    MoveTo(10, 50);
    DrawString('Full Screen On:  ');
    if gFullScreenOn then
        begin
        DrawString('YES');
        end
    else begin
        DrawString('NO');
        end;

    MoveTo(10, 65);
    DrawString('AutoScroll On:  ');
    if gAutoScrollOn then
        begin
        DrawString('YES');
        end
    else begin
        DrawString('NO');
        end;

    if (gPrintRect.bottom <> 0) then
```

```
        begin
        MoveTo(10, 85);
        TextFace([bold]);
        DrawString('Information From Printable Area (''PrAr'') Resource:');
        TextFace([]);
        NumToString(SInt32(gPrintRect.bottom), theString);
        MoveTo(10, 100);
        DrawString('Page print area height in screen pixels:  ');
        DrawString(theString);
        NumToString(SInt32(gPrintRect.right), theString);
        MoveTo(10, 115);
        DrawString('Page print area width in screen pixels:  ');
        DrawString(theString);
        end
    else begin
        MoveTo(10, 85);
        DrawString('No printable area (''PrAr'') resource saved yet');
        end;
    end;
        { of procedure DoUpdateWindow }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoAdjustMenus

procedure DoAdjustMenus;
    var
    menuHdl : MenuHandle;

    begin
    menuHdl := GetMenuHandle(mFile);
    if gWindowOpen then
        begin
        DisableItem(menuHdl, iOpen);
        EnableItem(menuHdl, iClose);
        EnableItem(menuHdl, iPageSetup);
        end
    else begin
        EnableItem(menuHdl, iOpen);
        DisableItem(menuHdl, iClose);
        DisableItem(menuHdl, iPageSetup);
        end;

    DrawMenuBar;
    end;
        { of procedure DoAdjustMenus }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoMenuChoice

procedure DoMenuChoice(menuChoice : SInt32);
    var
    menuID, menuItem : SInt16;
    itemName : Str255;
    DaDriverRefNum : SInt16;

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

    if (menuID = 0) then
        begin
        Exit(DoMenuChoice);
        end;

    case menuID of

        mApple: begin
            GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
            DaDriverRefNum := OpenDeskAcc(itemName);
            end;

        mFile: begin
            case menuItem of

                iClose: begin
                    DoCloseCommand;
                    end;

                iOpen: begin
                    DoOpenCommand;
                    end;
                iPageSetup: begin
                    DoPrintStyleDialog;
                    end;

                iQuit: begin
                    while (FrontWindow <> nil) do
                        begin
                        DoCloseCommand;
                        end;
                    gDone := true;
                    end;

                otherwise begin
                    end;

                end;
                    { of case statement }
            end;

        mDemonstration: begin
            if (menuItem = iPreferences) then
                begin
                DoPreferencesDialog;
                end;
            end;

        otherwise begin
            end;

        end;
            { of case statement }

    HiliteMenu(0);
    end;
        { of procedure DoMenuChoice }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoErrorAlert

procedure DoErrorAlert(errorCode : SInt16);
    var
    paramRec : AlertStdAlertParamRec;
    errorString : Str255;
    itemHit : SInt16;
    ignoredErr : OSErr;

    begin
    paramRec.movable := true;
    paramRec.helpButton := false;
    paramRec.filterProc := nil;
    paramRec.defaultText := StringPtr(kAlertDefaultOKText);
    paramRec.cancelText := nil;
    paramRec.otherText := nil;
    paramRec.defaultButton := kAlertStdAlertOKButton;
    paramRec.cancelButton := 0;
    paramRec.position := kWindowDefaultPosition;

    NumToString(errorCode, errorString);

    if (errorCode <> memFullErr) then
        begin
        ignoredErr := StandardAlert(kAlertCautionAlert, @errorString, nil, @paramRec, itemHit);
        end
    else begin
        ignoredErr := StandardAlert(kAlertStopAlert, @errorString, nil, @paramRec, itemHit);
        ExitToShell;
        end;
    end;
        { of procedure DoErrorAlert }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOpenCommand

procedure DoOpenCommand;
    var
    fileTypes : SFTypeList;
    fileReply : StandardFileReply;
    docRecordHdl : DocRecordHandle;
    osError : OSErr;
    theWindowPtr : WindowPtr;

    begin
    osError := 0;
    fileTypes[0] := 'TEXT';

    StandardGetFile(nil, 1, @fileTypes[0], fileReply);
    if not fileReply.sfGood then
        begin
        Exit(DoOpenCommand);
        end;

    theWindowPtr := GetNewCWindow(rNewWindow, nil, WindowPtr(-1));
    if (theWindowPtr = nil) then
        begin
        Exit(DoOpenCommand);
        end;

    docRecordHdl := DocRecordHandle(NewHandle(sizeof(DocRecord)));
    if (docRecordHdl = nil) then
        begin
        DisposeWindow(theWindowPtr);
        Exit(DoOpenCommand);
        end;

    gWindowOpen := true;
    SetPort(theWindowPtr);
    TextSize(10);

    SetWRefCon(theWindowPtr, SInt32(docRecordHdl));
    docRecordHdl^^.fileFSSpec := fileReply.sfFile;
    SetWTitle(theWindowPtr, docRecordHdl^^.fileFSSpec.name);

    DoGetandSetWindowPosition(theWindowPtr);
    DoGetPrintableSize(theWindowPtr);

    ShowWindow(theWindowPtr);
    end;
        { of procedure DoOpenCommand }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCloseCommand

procedure DoCloseCommand;
    var
    theWindowPtr : WindowPtr;
    docRecordHdl : DocRecordHandle;
    osError : OSErr;

    begin
    osError := 0;
    theWindowPtr := FrontWindow;
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    DoSaveWindowPosition(theWindowPtr);

    if gPrintStyleChanged then
        begin
        DoSavePrintableSize(theWindowPtr);
        end;

    DisposeHandle(Handle(docRecordHdl));
    DisposeWindow(theWindowPtr);
    gWindowOpen := false;
    end;
        { of procedure DoCloseCommand }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoPreferencesDialog

procedure DoPreferencesDialog;
    var
    modalDlgPtr : DialogPtr;
    controlHdl : ControlHandle;
    itemHit : SInt16;
    ignoredErr : OSErr;
    theWindowPtr : WindowPtr;

    begin
    modalDlgPtr := GetNewDialog(rPrefsDialog, nil, WindowPtr(-1));
    if (modalDlgPtr = nil) then
        begin
        Exit(DoPreferencesDialog);
        end;
    ignoredErr := GetDialogItemAsControl(modalDlgPtr, iSoundOn, controlHdl);
    SetControlValue(controlHdl, SInt32(gSoundOn));
    ignoredErr := GetDialogItemAsControl(modalDlgPtr, iFullScreenOn, controlHdl);
    SetControlValue(controlHdl, SInt32(gFullScreenOn));
    ignoredErr := GetDialogItemAsControl(modalDlgPtr, iAutoScrollOn, controlHdl);
    SetControlValue(controlHdl, SInt32(gAutoScrollOn));

    ShowWindow(modalDlgPtr);

    repeat
        begin
        ModalDialog(nil, itemHit);
        ignoredErr := GetDialogItemAsControl(modalDlgPtr, itemHit, controlHdl);
        SetControlValue(controlHdl, (GetControlValue(ControlHandle(controlHdl)) + 1) mod 2);
        end;
    until ((itemHit = kStdOkItemIndex) or (itemHit = kStdCancelItemIndex));

    if (itemHit = kStdOkItemIndex) then
        begin
        ignoredErr := GetDialogItemAsControl(modalDlgPtr, iSoundOn, controlHdl);
        gSoundOn := boolean(GetControlValue(controlHdl));

        ignoredErr := GetDialogItemAsControl(modalDlgPtr, iFullScreenOn, controlHdl);
        gFullScreenOn := boolean(GetControlValue(controlHdl));

        ignoredErr := GetDialogItemAsControl(modalDlgPtr, iAutoScrollOn, controlHdl);
        gAutoScrollOn := boolean(GetControlValue(controlHdl));
        end;

    DisposeDialog(modalDlgPtr);

    theWindowPtr := FrontWindow;
    if (theWindowPtr <> nil) then
        begin
        InvalRect(theWindowPtr^.portRect);
        end;

    DoSavePreferences;
    end;
        { of procedure DoPreferencesDialog }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoPrintStyleDialog

procedure DoPrintStyleDialog;
    var
    clickedOK : boolean;
    theWindowPtr : WindowPtr;

    begin
    PrOpen;

    clickedOK := PrStlDialog(gTPrintHdl);
    if clickedOK then
        begin
        gPrintStyleChanged := true;
        gPrintRect := gTPrintHdl^^.prInfo.rPage;

        theWindowPtr := FrontWindow;
        if (theWindowPtr <> nil) then
            begin
            InvalRect(theWindowPtr^.portRect);
            end;
        end;

    PrClose;
    end;
        { of procedure DoPrintStyleDialog }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGetPreferences

procedure DoGetPreferences;
    var
    prefsFileName : Str255;
    osError : OSErr;
    volRefNum : SInt16;
    directoryID : longint;
    fileSSpec : FSSpec;
    fileRefNum : SInt16;
    appPrefsHdl : AppPrefsHandle;
    begin
    GetIndString(prefsFileName, rStringList, iPrefsFileName);

    osError := FindFolder(kOnSystemDisk, kPreferencesFolderType, kDontCreateFolder,
                                volRefNum, directoryID);

    if (osError = noErr) then
        begin
        osError := FSMakeFSSpec(volRefNum, directoryID, prefsFileName, fileSSpec);
        end;

    if ((osError = noErr) or (osError = fnfErr)) then
        begin
        fileRefNum := FSpOpenResFile(fileSSpec, fsCurPerm);
        end;

    if (fileRefNum = -1) then
        begin
        FSpCreateResFile(fileSSpec, 'PpPp', 'pref', smSystemScript);
        osError := ResError;

        if (osError = noErr) then
            begin
            fileRefNum := FSpOpenResFile(fileSSpec, fsCurPerm);
            if (fileRefNum <> -1) then
                begin
                UseResFile(gAppResFileRefNum);

                osError := DoCopyResource(rTypePrefs, kPrefsID, gAppResFileRefNum, fileRefNum);
                if (osError = noErr) then
                    begin
                    osError := DoCopyResource(rTypeAppMiss, kAppMissID, gAppResFileRefNum, fileRefNum);
                    end;

                if (osError <> noErr) then
                    begin
                    CloseResFile(fileRefNum);
                    osError := FSpDelete(fileSSpec);
                    fileRefNum := -1;
                    end;
                end;
            end;
        end;

    if (fileRefNum <> -1) then
        begin
        UseResFile(fileRefNum);

        appPrefsHdl := AppPrefsHandle(Get1Resource(rTypePrefs, kPrefsID));
        if (appPrefsHdl = nil) then
            begin
            Exit(DoGetPreferences);
            end;

        gSoundOn := appPrefsHdl^^.soundOn;
        gFullScreenOn := appPrefsHdl^^.fullScreenOn;
        gAutoScrollOn := appPrefsHdl^^.autoScrollOn;

        gPrefsFileRefNum := fileRefNum;

        UseResFile(gAppResFileRefNum);
        end;
    end;
        { of procedure DoGetPreferences }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCopyResource

function  DoCopyResource(theResType : ResType;
                resID, sourceFileRefNum, destFileRefNum : SInt16) : OSErr;
    var
    oldResFileRefNum : SInt16;
    sourceResourceHdl : Handle;
    ignoredType : ResType;
    ignoredID : SInt16;
    resourceName : Str255;
    resAttributes : SInt16;
    osError : OSErr;

    begin
    oldResFileRefNum := CurResFile;
    UseResFile(sourceFileRefNum);

    sourceResourceHdl := Get1Resource(theResType, resID);

    if (sourceResourceHdl <> nil) then
        begin
        GetResInfo(sourceResourceHdl, ignoredID, ignoredType, resourceName);
        resAttributes := GetResAttrs(sourceResourceHdl);
        DetachResource(sourceResourceHdl);
        UseResFile(destFileRefNum);
        if (ResError = noErr) then
            begin
            AddResource(sourceResourceHdl, theResType, resID, resourceName);
            end;

        if (ResError = noErr) then
            begin
            SetResAttrs(sourceResourceHdl, resAttributes);
            end;

        if (ResError = noErr) then
            begin
            ChangedResource(sourceResourceHdl);
            end;

        if (ResError = noErr) then
            begin
            WriteResource(sourceResourceHdl);
            end;
        end;

    osError := ResError;

    ReleaseResource(sourceResourceHdl);
    UseResFile(oldResFileRefNum);

    DoCopyResource := osError;
    end;
        { of procedure DoCopyResource }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSavePreferences

procedure DoSavePreferences;
    var
    appPrefsHdl : AppPrefsHandle;
    existingResHdl : Handle;
    resourceName : Str255;

    begin
    resourceName := 'Preferences';

    if (gPrefsFileRefNum = -1) then
        begin
        Exit(DoSavePreferences);
        end;

    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 <> nil) then
        begin
        RemoveResource(existingResHdl);
        if (ResError = noErr) then
            begin
            AddResource(Handle(appPrefsHdl), rTypePrefs, kPrefsID, resourceName);
            end;

        if (ResError = noErr) then
            begin
            WriteResource(Handle(appPrefsHdl));
            end;
        end;
    HUnlock(Handle(appPrefsHdl));

    ReleaseResource(Handle(appPrefsHdl));
    UseResFile(gAppResFileRefNum);
    end;
        { of procedure DoSavePreferences }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGetandSetWindowPosition

procedure DoGetandSetWindowPosition(theWindowPtr : WindowPtr);
    var
    userStateRect, stdStateRect, displayRect : Rect;
    docRecordHdl : DocRecordHandle;
    fileRefNum : SInt16;
    winStateHdl : WinStateHandle;
    gotResource : boolean;
    osError : OSErr;

    begin
    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);

    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    fileRefNum := FSpOpenResFile(docRecordHdl^^.fileFSSpec, fsRdWrPerm);
    if (fileRefNum < 0) then
        begin
        osError := ResError;
        DoErrorAlert(osError);
        Exit(DoGetandSetWindowPosition);
        end;

    winStateHdl := WinStateHandle(Get1Resource(rTypeWinState, kWinStateID));
    if (winStateHdl <> nil ) then
        begin
        gotResource := true;
        userStateRect := winStateHdl^^.userStateRect;
        end
    else begin
        gotResource := false;
        end;

    if gotResource then
        begin
        if (winStateHdl^^.zoomState) then
            begin
            displayRect := stdStateRect;
            end
        else begin
            displayRect := userStateRect;
            end;
        end
    else begin
        displayRect := userStateRect;
        end;

    MoveWindow(theWindowPtr, displayRect.left, displayRect.top, false);

    GlobalToLocal(displayRect.topLeft);
    GlobalToLocal(displayRect.botRight);
    SizeWindow(theWindowPtr, displayRect.right, displayRect.bottom, true);

    DoSetWindowState(theWindowPtr, userStateRect, stdStateRect);

    ReleaseResource(Handle(winStateHdl));
    CloseResFile(fileRefNum);
    end;
        { of procedure DoGetandSetWindowPosition }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSaveWindowPosition

procedure DoSaveWindowPosition(theWindowPtr : WindowPtr);
    var
    docRecordHdl : DocRecordHandle;
```

```
   fileRefNum : SInt16;
   windowRecPtr : WindowPeek;
   winStateDataPtr : WStateDataPtr;
   stdRect, userRect : Rect;
   contentRgnHdl : RgnHandle;
   userRectAndZoomState : WinState;
   winStateHdl : WinStateHandle;
   osError : OSErr;

   begin
   docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

   fileRefNum := FSpOpenResFile(docRecordHdl^^.fileFSSpec, fsRdWrPerm);
   if (fileRefNum < 0) then
      begin
      osError := ResError;
      DoErrorAlert(osError);
      Exit(DoSaveWindowPosition);
      end;

   windowRecPtr := WindowPeek(theWindowPtr);
   winStateDataPtr := WStateDataPtr(windowRecPtr^.dataHandle^);
   stdRect := winStateDataPtr^.stdState;
   userRect := winStateDataPtr^.userState;

   contentRgnHdl := windowRecPtr^.contRgn;
   userRectAndZoomState.userStateRect := contentRgnHdl^^.rgnBBox;
   userRectAndZoomState.zoomState := EqualRect(userRectAndZoomState.userStateRect, stdRect);
   if userRectAndZoomState.zoomState then
      begin
      userRectAndZoomState.userStateRect := userRect;
      end;

   winStateHdl := WinStateHandle(Get1Resource(rTypeWinState, kWinStateID));
   if (winStateHdl <> nil) then
      begin
      winStateHdl^^ := userRectAndZoomState;
      ChangedResource(Handle(winStateHdl));
      osError := ResError;
      if (osError <> noErr) then
         begin
         DoErrorAlert(osError);
         end;
      end
   else begin
      winStateHdl := WinStateHandle(NewHandle(sizeof(WinState)));
      if (winStateHdl <> nil) then
         begin
         winStateHdl^^ := userRectAndZoomState;
         AddResource(Handle(winStateHdl), rTypeWinState, kWinStateID, 'Last window state');
         end;
      end;

   if (winStateHdl <> nil) then
      begin
      UpdateResFile(fileRefNum);
      osError := ResError;
      if (osError <> noErr) then
         begin
         DoErrorAlert(osError);
         end;

      ReleaseResource(Handle(winStateHdl));
      end;

   CloseResFile(fileRefNum);
   end;
      { of procedure DoSaveWindowPosition }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSetWindowState

procedure DoSetWindowState(theWindowPtr : WindowPtr; userStateRect, stdStateRect : Rect);
   var
   windowRecPtr : WindowPeek;
   winStateDataPtr : WStateDataPtr;

   begin
   windowRecPtr := WindowPeek(theWindowPtr);
   winStateDataPtr := WStateDataPtr(windowRecPtr^.dataHandle^);
   winStateDataPtr^.userState := userStateRect;
```

```
    winStateDataPtr^.stdState := stdStateRect;
    end;
        { of procedure DoSetWindowState }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGetPrintableSize

procedure DoGetPrintableSize(theWindowPtr : WindowPtr);
    var
    docRecordHdl : DocRecordHandle;
    fileRefNum : SInt16;
    osError : OSErr;
    printRectHdl : RectHandle;

    begin
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    fileRefNum := FSpOpenResFile(docRecordHdl^^.fileFSSpec, fsRdWrPerm);
    if (fileRefNum < 0) then
        begin
        osError := ResError;
        DoErrorAlert(osError);
        Exit(DoGetPrintableSize);
        end;

    printRectHdl := RectHandle(Get1Resource(rTypePrintRect, kPrintRectID));
    if (printRectHdl <> nil) then
        begin
        gPrintRect := printRectHdl^^;
        ReleaseResource(Handle(printRectHdl));
        end;

    CloseResFile(fileRefNum);
    end;
        { of procedure DoGetPrintableSize }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSavePrintableSize

procedure DoSavePrintableSize(theWindowPtr : WindowPtr);
    var
    docRecordHdl : DocRecordHandle;
    fileRefNum : SInt16;
    printRectHdl : RectHandle;
    osError : OSErr;

    begin
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    fileRefNum := FSpOpenResFile(docRecordHdl^^.fileFSSpec, fsRdWrPerm);
    if (fileRefNum < 0) then
        begin
        osError := ResError;
        DoErrorAlert(osError);
        Exit(DoSavePrintableSize);
        end;

    printRectHdl := RectHandle(Get1Resource(rTypePrintRect, kPrintRectID));
    if (printRectHdl <> nil) then
        begin
        printRectHdl^^ := gTPrintHdl^^.prInfo.rPage;
        ChangedResource(Handle(printRectHdl));
        osError := ResError;
        if (osError <> noErr) then
            begin
            DoErrorAlert(osError);
            end;
        end
    else begin
        printRectHdl := RectHandle(NewHandle(sizeof(Rect)));
        if (printRectHdl <> nil) then
            begin
            printRectHdl^^ := gTPrintHdl^^.prInfo.rPage;
            AddResource(Handle(printRectHdl), rTypePrintRect, kPrintRectID, 'Print rectangle');
            end;
        end;

    if (printRectHdl <> nil) then
        begin
        UpdateResFile(fileRefNum);
        osError := ResError;
        if (osError <> noErr) then
```

```
            begin
            DoErrorAlert(osError);
            end;

        ReleaseResource(Handle(printRectHdl));
        end;

    gPrintStyleChanged := false;

    CloseResFile(fileRefNum);
    end;
        { of procedure DoSavePrintableSize }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ main

begin

    gWindowOpen := false;
    gPrintStyleChanged := false;
    gPrefsFileRefNum := 0;

    //
.........................................................................................................................................................................
...... initialise managers

    DoInitManagers;

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

    gAppResFileRefNum := CurResFile;

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

    mainMenubarHdl := GetNewMBar(rMenubar);
    if (mainMenubarHdl = nil) then
        begin
        DoErrorAlert(MemError);
        end;
    SetMenuBar(mainMenubarHdl);
    DrawMenuBar;

    mainMenuHdl := GetMenuHandle(mApple);
    if (mainMenuHdl = nil) then
        begin
        DoErrorAlert(MemError);
        end
    else begin
        AppendResMenu(mainMenuHdl, 'DRVR');
        end;

    // ...................................................................................................................................... create and initialise a
TPrint record

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

    // ....................................................................................................................................................... read in
application preferences

    DoGetPreferences;

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

    gDone := false;

    while not gDone do
        begin
        if WaitNextEvent(everyEvent, mainEvent, MAXLONG, nil) then
            begin
            DoEvents(mainEvent);
            end;
        end;

end.
    { of main program block }
```

{ ◊◊◊ }

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 automatic document translation is selected to off in the Macintosh Easy Open control panel, and 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.

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. The 'WiSt' resource is written to whenever the file is closed. The 'PrAr' resource is written to when the file is closed 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, re-size 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 MoreResources preferences file from the Preferences folder in the System folder.

constants

rPrefsDialog represents the 'DLOG' resource ID for the Preferences dialog box 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. The next eight constants 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.

type definitions

The DocRecord data type is for the document record. In this demonstration, the only field required is that for a file system specification.

The AppPrefs data type is for the application preferences settings. The three Boolean values are set by checkboxes in the Preferences dialog box.

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

The RectHandle data type 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 structure, the latter being required because of the use by the program of the print style dialog. 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 program block

The call to CurResFile sets the application's resource fork as the current resource file. The block beginning with the call to PrOpen creates and initialises a TPrint structure. The call to DoGetPreferences 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.)

DoUpdateWindow

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

DoErrorAlert

DoErrorAlert 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.

StandardGetFile presents the standard Open dialog box. If the user clicks the Cancel button, the function simply returns. If the user clicks the OK button, a new window is created, a document structure is created, a flag is set to indicate that the window is open, the window's graphics port is set as the current port for drawing, the text size is set to 10pt, the document structure is connected to the window structure, the file system specification for the chosen file is assigned to the document structure's file system specification field, and the window's title is set.

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

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

DoCloseCommand

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

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

The call to DoSaveWindowPosition saves the window's user state and zoom state to the window state resource in the document's resource fork. If the print Style dialog was invoked while the window was open, and if the user dismissed the dialog by clicking the OK button, a call is made to DoSavePrintableSize to save the printable area rectangle to the printable area 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 in the Demonstration menu. The routine 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 the last line, a call is made to the application-defined routine 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.

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, and the global variable which holds the printable rectangle is assigned the value in the rPage (printable page size) field of the TPrInfo structure, a handle to which is at the prInfo field of the TPrint structure. In addition, the window's port rectangle is invalidated to force an update of the window, thus ensuring that the new printable area values are displayed immediately.

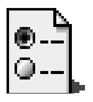
DoGetPreferences

DoGetPreferences, which is called from the main function immediately after program launch, is the first of those application-defined routines 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.

GetIndString retrieves from the application's resource file 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. FSMakeFSSpec makes a file system specification from the preferences file name, volume reference number and directory ID. This file system specification is used in the FSpOpenResFile call to open the resource fork of the preferences file with exclusive read/write permission.

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



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

If the preferences file exists (either previously or newly-created), UseResFile sets the resource fork of that file as the current resource file, the preferences resource is read in from the resource fork by Get1Resource and, if the read was successful, the three Boolean values 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.

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.

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.

At the first two lines, if DoGetPreferences was not successful in opening the preferences file at program launch, the routine simply returns.

The next five 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 to disk.

With the resource written to disk, ReleaseResource disposes of the preferences structure in memory and UseResFile 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.

The first three lines 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. The next three lines establish the standard state rectangle as desired by the application.

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

Get1Resource attempts to read in the 'WiSt' resource. If the Get1Resource call is successful, a "success" flag is set and the user state rectangle is set to that retrieved from the resource. If the call is not successful, the "success" flag is unset and the user state rectangle remains as the default rectangle defined earlier.

If the Get1Resource call was successful, the zoom state is also retrieved from the resource. 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. 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. If the Get1Resource call was not successful, the display rectangle is set to the user state rectangle, which will be the default rectangle defined earlier.

MoveWindow moves the window to the specified coordinates, keeping it inactive. The next block re-sizes the window to the specified size, adding any area added to the content region to the update region.

DoSetWindowState assigns the specified rectangles to the userState and stdState fields of the WStateData structure for the window. With this action completed, ReleaseResource discards the 'WiSt' resource in memory and CloseResFile 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 routine is called when the associated window is closed by the user.

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

At the next block, a pointer to the window structure is retrieved, allowing a pointer to the WStateData structure to be retrieved. The last two lines in this block retrieve the current standard state and user state rectangles from the WStateData structure.

The next step is to determine whether the window is currently in the "zoomed out" (standard) state or the "zoomed in" (user) state. The first two lines of the next block get a rectangle equal to the content region of the window and set up a forthcoming test by assigning this rectangle to the userStateRect field of a window state structure. The test is at the next line: If the content region rectangle equals the current standard state rectangle, the call to EqualRect will return true, in which case:

- The zoomstate field of the window state structure is assigned a value indicating that the window is in the standard state.
- The userStateRect field of the window state structure 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 will return false, in which case:

- The zoomstate field of the window state structure is assigned a value indicating that the window is in the user state.
- The userStateRect field of the window state structure retains the rectangle it was earlier assigned which, not being equal to the standard state rectangle, must be equal to the current user state rectangle.

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

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

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 structure.

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.

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 box 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 data in the resource in memory is assigned to the global variable which stores the current printable area rectangle. The resource in memory is then discarded and the document file's resource fork is closed.

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.

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 box is presented and the function simply returns.

Get1Resource 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 structure, which is itself part of the TPrint 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 'PrAr' resource), a block of memory is allocated for a Rect, the rectangle in the prPage field of the prInfo structure is copied to this block, and AddResource 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, 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.