15 Version 1.1

MORE ON RESOURCES

Includes Demonstration Program MoreResourcesPascal

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 `NIL`, 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 `NIL` 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

```
FourCharCode =  UNSIGNEDLONG;
ResType = FourCharCode;
```

Routines

Initialising the Resource Manager

```
function   InitResources: integer;
```

Checking for Errors

```
function   ResError: integer;
```

Creating an Empty Resource Fork

```
procedure  FSpCreateResFile(VAR spec: FSSpec; creator: OSType; fileType: OSType;
           scriptTag: ScriptCode);
```

Opening Resource Forks

```
function   FSpOpenResFile(VAR spec: FSSpec; permission: ByteParameter): 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: ConstStr255Param): Handle;
function   Get1NamedResource(theType: ResType; name: ConstStr255Param): 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: ConstStr255Param);
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: ConstStr255Param);
```

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

```
{ ############################################################################################
// MoreResourcesPascal.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).
//
// •  An 'ALRT' resource (purgeable) and associated 'DITL' resource (purgeable)
//    associated with the display of error messages.
//
// •  A 'DLOG' resource (purgeable) and associated 'DITL' resource (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 & 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 value) 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 MoreResourcesPascal(input, output);

{ ……… include the following Universal Interfaces }

uses

  Windows, Fonts, Menus, TextEdit, Quickdraw, Dialogs, QuickdrawText, Processes, Types,
  Memory, Events, TextUtils, ToolUtils, OSUtils, Devices, Resources, StandardFile,
  Folders, Printing, Files, Errors, Script, Controls, Segload;

{ ……… define the following constants }

const

mApple = 128;
mFile = 129;
  iOpen = 2;
  iClose = 4;
  iPageSetup = 8;
  iQuit = 11;
mDemonstration = 131;
  iPreferences = 1;

rNewWindow = 128;
rMenubar = 128;
rAlertBox = 128;
rModalDialog = 129;
  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;

kMaxLong = $7FFFFFFF;

{ ……… user defined types }

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 }

```

```
127  var
128
129  gDone : boolean;
130  gInBackground : boolean;
131  gTPrintHdl : THPrint;
132  gWindowPtr : WindowPtr;
133  gWindowOpen : boolean;
134  gPrintStyleChanged : boolean;
135  gPrintRect : Rect;
136  gSoundOn : boolean;
137  gFullScreenOn : boolean;
138  gAutoScrollOn : boolean;
139  gAppResFileRefNum : integer;
140  gPrefsFileRefNum : integer;
141  menubarHdl : Handle;
142  menuHdl : MenuHandle;
143  eventRec : EventRecord;
144
145  { ############################################################################ DoInitManagers }
146
147  procedure DoInitManagers;
148
149    begin
150    MaxApplZone;
151    MoreMasters;
152
153    InitGraf(@qd.thePort);
154    InitFonts;
155    InitWindows;
156    InitMenus;
157    TEInit;
158    InitDialogs(nil);
159
160    InitCursor;
161    FlushEvents(everyEvent, 0);
162    end;
163      {of procedure DoInitManagers}
164
165  { ################################################################################ DoError }
166
167  procedure DoError(errorCode : integer);
168
169    var
170    errorString : string;
171    ignored : OSErr;
172
173    begin
174    NumToString(errorCode, errorString);
175    ParamText(errorString, '', '', '');
176
177    if (errorCode = memFullErr) then
178      begin
179      ignored := StopAlert(rAlertBox, nil);
180      ExitToShell;
181      end
182    else
183      ignored := CautionAlert(rAlertBox, nil);
184    end;
185      {of procedure DoError}
186
187  { ##################################################################### DoSavePrintableSize }
188
189  procedure DoSavePrintableSize(myWindowPtr : WindowPtr);
190
191    var
192    docRecHdl : DocRecordHandle;
193    fileRefNum : integer;
194    printRectHdl : RectHandle;
195    osError : OSErr;
196
197    begin
198    docRecHdl := DocRecordHandle(GetWRefCon(myWindowPtr));
199
200    fileRefNum := FSpOpenResFile(docRecHdl^^.fileFSSpec, fsRdWrPerm);
201    if (fileRefNum < 0) then
202      begin
203      osError := ResError;
```

```
      DoError(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
        DoError(osError);
      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
        DoError(osError);

      ReleaseResource(Handle(printRectHdl));
      end;

    gPrintStyleChanged := false;

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

  { ############################################################################ DoGetPrintableSize }

  procedure DoGetPrintableSize(myWindowPtr : WindowPtr);

    var
    docRecHdl : DocRecordHandle;
    fileRefNum : integer;
    osError : OSErr;
    printRectHdl : RectHandle;

    begin
    docRecHdl := DocRecordHandle(GetWRefCon(myWindowPtr));

    fileRefNum := FSpOpenResFile(docRecHdl^^.fileFSSpec, fsRdWrPerm);
    if (fileRefNum < 0) then
      begin
      osError := ResError;
      DoError(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}

  { ############################################################################ DoSetWindowState }

  procedure DoSetWindowState(myWindowPtr : WindowPtr; userStateRect, stdStateRect : Rect);

    var
```

```
281     windowRecPtr : WindowPeek;
282     winStateDataPtr : WStateDataPtr;
283
284     begin
285     windowRecPtr := WindowPeek(myWindowPtr);
286     winStateDataPtr := WStateDataPtr(windowRecPtr^.dataHandle^);
287     winStateDataPtr^.userState := userStateRect;
288     winStateDataPtr^.stdState := stdStateRect;
289     end;
290       {of procedure DoSetWindowState}
291
292   { ################################################################# DoSaveWindowPosition }
293
294   procedure DoSaveWindowPosition(myWindowPtr : WindowPtr);
295
296     var
297     docRecHdl : DocRecordHandle;
298     fileRefNum : integer;
299     windowRecPtr : WindowPeek;
300     winStateDataPtr : WStateDataPtr;
301     stdRect, userRect : Rect;
302     contentRgnHdl : RgnHandle;
303     userRectAndZoomState : WinState;
304     winStateHdl : WinStateHandle;
305     osError : OSErr;
306
307     begin
308     docRecHdl := DocRecordHandle(GetWRefCon(myWindowPtr));
309
310     fileRefNum := FSpOpenResFile(docRecHdl^^.fileFSSpec, fsRdWrPerm);
311     if (fileRefNum < 0) then
312       begin
313       osError := ResError;
314       DoError(osError);
315       Exit(DoSaveWindowPosition);
316       end;
317
318     windowRecPtr := WindowPeek(myWindowPtr);
319     winStateDataPtr := WStateDataPtr(windowRecPtr^.dataHandle^);
320     stdRect := winStateDataPtr^.stdState;
321     userRect := winStateDataPtr^.userState;
322
323     contentRgnHdl := windowRecPtr^.contRgn;
324     userRectAndZoomState.userStateRect := contentRgnHdl^^.rgnBBox;
325     userRectAndZoomState.zoomState := EqualRect(userRectAndZoomState.userStateRect, stdRect);
326     if (userRectAndZoomState.zoomState) then
327       userRectAndZoomState.userStateRect := userRect;
328
329     winStateHdl := WinStateHandle(Get1Resource(rTypeWinState, kWinStateID));
330     if (winStateHdl <> nil) then
331       begin
332       winStateHdl^^ := userRectAndZoomState;
333       ChangedResource(Handle(winStateHdl));
334       osError := ResError;
335       if (osError <> noErr) then
336         DoError(osError);
337       end
338     else
339       begin
340       winStateHdl := WinStateHandle(NewHandle(sizeof(WinState)));
341       if (winStateHdl <> nil) then
342         begin
343         winStateHdl^^ := userRectAndZoomState;
344         AddResource(Handle(winStateHdl), rTypeWinState, kWinStateID,
345                     'Last window state');
346         end;
347       end;
348
349     if (winStateHdl <> nil) then
350       begin
351       UpdateResFile(fileRefNum);
352       osError := ResError;
353       if (osError <> noErr) then
354         DoError(osError);
355
356       ReleaseResource(Handle(winStateHdl));
357       end;
```

```
358
359     CloseResFile(fileRefNum);
360     end;
361       {of procedure DoSaveWindowPosition}
362
363   { ############################################################ DoGetandSetWindowPosition }
364
365   procedure DoGetandSetWindowPosition(myWindowPtr : WindowPtr);
366
367     var
368     userStateRect, stdStateRect, displayRect : Rect;
369     docRecHdl : DocRecordHandle;
370     fileRefNum : integer;
371     winStateHdl : WinStateHandle;
372     gotResource : boolean;
373     osError : OSErr;
374
375     begin
376     userStateRect := qd.screenBits.bounds;
377     SetRect(userStateRect, userStateRect.left + 3, userStateRect.top + 42,
378             userStateRect.right - 40, userStateRect.bottom - 6);
379
380     stdStateRect := qd.screenBits.bounds;
381     SetRect(stdStateRect, stdStateRect.left + 3, stdStateRect.top + 42,
382             stdStateRect.right - 3, stdStateRect.bottom - 6);
383
384     docRecHdl := DocRecordHandle(GetWRefCon(myWindowPtr));
385
386     fileRefNum := FSpOpenResFile(docRecHdl^^.fileFSSpec, fsRdWrPerm);
387     if (fileRefNum < 0) then
388       begin
389       osError := ResError;
390       DoError(osError);
391       Exit(DoGetandSetWindowPosition);
392       end;
393
394     winStateHdl := WinStateHandle(Get1Resource(rTypeWinState, kWinStateID));
395     if (winStateHdl <> nil ) then
396       begin
397       gotResource := true;
398       userStateRect := winStateHdl^^.userStateRect;
399       end
400     else
401       gotResource := false;
402
403     if (gotResource) then
404       begin
405       if (winStateHdl^^.zoomState) then
406         displayRect := stdStateRect
407       else
408         displayRect := userStateRect;
409       end
410     else
411       begin
412       displayRect := userStateRect;
413       end;
414
415     MoveWindow(myWindowPtr, displayRect.left, displayRect.top, false);
416
417     GlobalToLocal(displayRect.topLeft);
418     GlobalToLocal(displayRect.botRight);
419     SizeWindow(myWindowPtr, displayRect.right, displayRect.bottom, true);
420
421     DoSetWindowState(myWindowPtr, userStateRect, stdStateRect);
422
423     ReleaseResource(Handle(winStateHdl));
424     CloseResFile(fileRefNum);
425     end;
426       {of procedure DoGetandSetWindowPosition}
427
428   { ################################################################## DoSavePreferences }
429
430   procedure DoSavePreferences;
431
432     var
433     appPrefsHdl : AppPrefsHandle;
434     existingResHdl : Handle;
```

```
435     resourceName : string;
436
437     begin
438     resourceName := 'Preferences';
439     if (gPrefsFileRefNum = -1) then
440       Exit(DoSavePreferences);
441
442     appPrefsHdl := AppPrefsHandle(NewHandleClear(sizeof(AppPrefs)));
443
444     HLock(Handle(appPrefsHdl));
445
446     appPrefsHdl^^.soundOn := gSoundOn;
447     appPrefsHdl^^.fullScreenOn := gFullScreenOn;
448     appPrefsHdl^^.autoScrollOn := gAutoScrollOn;
449
450     UseResFile(gPrefsFileRefNum);
451
452     existingResHdl := Get1Resource(rTypePrefs, kPrefsID);
453     if (existingResHdl <> nil) then
454       begin
455       RemoveResource(existingResHdl);
456       if (ResError = noErr) then
457         AddResource(Handle(appPrefsHdl), rTypePrefs, kPrefsID, resourceName);
458       if (ResError = noErr) then
459         WriteResource(Handle(appPrefsHdl));
460       end;
461
462     HUnlock(Handle(appPrefsHdl));
463
464     ReleaseResource(Handle(appPrefsHdl));
465     UseResFile(gAppResFileRefNum);
466     end;
467       {of procedure DoSavePreferences}
468
469   { ############################################################################ DoCopyResource }
470
471   function DoCopyResource(rsrcType : ResType; resID, sourceFileRefNum,
472                         destFileRefNum : integer) : OSErr;
473
474     var
475     oldResFileRefNum : integer;
476     sourceResourceHdl : Handle;
477     ignoredType : ResType;
478     ignoredID : integer;
479     resourceName : string;
480     resAttributes : integer;
481     osError : OSErr;
482
483     begin
484     oldResFileRefNum := CurResFile;
485     UseResFile(sourceFileRefNum);
486
487     sourceResourceHdl := Get1Resource(rsrcType, resID);
488
489     if (sourceResourceHdl <> nil) then
490       begin
491       GetResInfo(sourceResourceHdl, ignoredID, ignoredType, resourceName);
492       resAttributes := GetResAttrs(sourceResourceHdl);
493       DetachResource(sourceResourceHdl);
494       UseResFile(destFileRefNum);
495       if (ResError = noErr) then
496         AddResource(sourceResourceHdl, rsrcType, resID, resourceName);
497       if (ResError = noErr) then
498         SetResAttrs(sourceResourceHdl, resAttributes);
499       if (ResError = noErr) then
500         ChangedResource(sourceResourceHdl);
501       if (ResError = noErr) then
502         WriteResource(sourceResourceHdl);
503       end;
504
505     osError := ResError;
506
507     ReleaseResource(sourceResourceHdl);
508     UseResFile(oldResFileRefNum);
509
510     DoCopyResource := osError;
511     end;
```

```
    {of function DoCopyResource}

{ ############################################################################ DoGetPreferences }

procedure DoGetPreferences;

  var
  prefsFileName : string;
  osError : OSErr;
  volRefNum : integer;
  directoryID : longint;
  fileSSpec : FSSpec;
  fileRefNum : integer;
  appPrefsHdl : AppPrefsHandle;

  begin
  GetIndString(prefsFileName, rStringList, iPrefsFileName);

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

  if (osError = noErr) then
    osError := FSMakeFSSpec(volRefNum, directoryID, prefsFileName, fileSSpec);
  if ((osError = noErr) or (osError = fnfErr)) then
    fileRefNum := FSpOpenResFile(fileSSpec, fsCurPerm);

  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
          osError := DoCopyResource(rTypeAppMiss, kAppMissID, gAppResFileRefNum,
                            fileRefNum);
        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
      Exit(DoGetPreferences);

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

    gPrefsFileRefNum := fileRefNum;

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

{ ######################################################################## DoPrintStyleDialog }

procedure DoPrintStyleDialog;

  begin
```

```
  PrOpen;

  if (PrStlDialog(gTPrintHdl)) then
    begin
    gPrintStyleChanged := true;
    gPrintRect := gTPrintHdl^^.prInfo.rPage;
    InvalRect(gWindowPtr^.portRect);
    end;

  PrClose;
  end;
    {of procedure DoPrintStyleDialog}

{ ###################################################################### DoPreferencesDialog }

procedure DoPreferencesDialog;

  var
  modalDlgPtr : DialogPtr;
  oldPort : GrafPtr;
  oldPenState : PenState;
  buttonOval, itemHit, itemType, temp : integer;
  itemHdl : Handle;
  itemRect : Rect;

  begin
  modalDlgPtr := GetNewDialog(rModalDialog, nil, WindowPtr(-1));
  if (modalDlgPtr = nil) then
    Exit(DoPreferencesDialog);

  GetDialogItem(modalDlgPtr, iSoundOn, itemType, itemHdl, itemRect);
  if gSoundOn
    then temp := 1
    else temp := 0;
  SetControlValue(ControlRef(itemHdl), temp);
  GetDialogItem(modalDlgPtr, iFullScreenOn, itemType, itemHdl, itemRect);
  if gFullScreenOn
    then temp := 1
    else temp := 0;
  SetControlValue(ControlRef(itemHdl), temp);
  GetDialogItem(modalDlgPtr, iAutoScrollOn, itemType, itemHdl, itemRect);
  if gAutoScrollOn
    then temp := 1
    else temp := 0;
  SetControlValue(ControlRef(itemHdl), temp);

  ShowWindow(modalDlgPtr);

  GetPort(oldPort);
  GetPenState(oldPenState);
  GetDialogItem(modalDlgPtr, 1, itemType, itemHdl, itemRect);
  SetPort(ControlHandle(itemHdl)^^.contrlOwner);
  InsetRect(itemRect, -4, -4);
  PenPat(qd.black);
  PenSize(3, 3);
  buttonOval := trunc((itemRect.bottom - itemRect.top) / 2) + 2;
  FrameRoundRect(itemRect, buttonOval, buttonOval);
  SetPenState(oldPenState);
  SetPort(oldPort);

  repeat
  ModalDialog(nil, itemHit);
  GetDialogItem(modalDlgPtr, itemHit, itemType, itemHdl, itemRect);
  if GetControlValue(ControlRef(itemHdl)) = 0
    then temp := 1
    else temp := 0;
  SetControlValue(ControlRef(itemHdl), temp);
  until ((itemHit = 1) or (itemHit = 2));

  if (itemHit = 1) then
    begin
    GetDialogItem(modalDlgPtr, iSoundOn, itemType, itemHdl, itemRect);
    if GetControlValue(ControlRef(itemHdl)) = 1
      then gSoundOn := true
      else gSoundOn := false;

    GetDialogItem(modalDlgPtr, iFullScreenOn, itemType, itemHdl, itemRect);
```

```
666       if GetControlValue(ControlRef(itemHdl)) = 1
667         then gFullScreenOn := true
668         else gFullScreenOn := false;
669
670       GetDialogItem(modalDlgPtr, iAutoScrollOn, itemType, itemHdl, itemRect);
671       if GetControlValue(ControlRef(itemHdl)) = 1
672         then gAutoScrollOn := true
673         else gAutoScrollOn := false;
674       end;
675
676     DisposeDialog(modalDlgPtr);
677
678     if (gWindowPtr <> nil) then
679       InvalRect(gWindowPtr^.portRect);
680
681     DoSavePreferences;
682     end;
683       {of procedure DoPreferencesDialog}
684
685   { ############################################################################ DoCloseCommand }
686
687   procedure DoCloseCommand;
688
689     var
690     myWindowPtr : WindowPtr;
691     docRecHdl : DocRecordHandle;
692     osError : OSErr;
693
694     begin
695     osError := 0;
696     myWindowPtr := FrontWindow;
697     docRecHdl := DocRecordHandle(GetWRefCon(myWindowPtr));
698
699     DoSaveWindowPosition(myWindowPtr);
700
701     if (gPrintStyleChanged) then
702       DoSavePrintableSize(gWindowPtr);
703
704     DisposeHandle(Handle(docRecHdl));
705     DisposeWindow(myWindowPtr);
706     gWindowOpen := false;
707     end;
708       {of procedure DoCloseCommand}
709
710   { ############################################################################# DoOpenCommand }
711
712   procedure DoOpenCommand;
713
714     var
715     fileTypes : SFTypeList;
716     fileReply : StandardFileReply;
717     docRecHdl : DocRecordHandle;
718     osError : OSErr;
719
720     begin
721     osError := 0;
722     fileTypes[0] := 'TEXT';
723
724     StandardGetFile(nil, 1, @fileTypes, fileReply);
725     if not (fileReply.sfGood) then
726       Exit(DoOpenCommand);
727
728     gWindowPtr := GetNewWindow(rNewWindow, nil, WindowPtr(-1));
729     if (gWindowPtr = nil) then
730       Exit(DoOpenCommand);
731
732     docRecHdl := DocRecordHandle(NewHandle(sizeof(DocRecord)));
733     if (docRecHdl = nil) then
734       begin
735       DisposeWindow(gWindowPtr);
736       Exit(DoOpenCommand);
737       end;
738
739     gWindowOpen := true;
740     SetPort(gWindowPtr);
741
742     SetWRefCon(gWindowPtr, longint(docRecHdl));
```

```
743     docRecHdl^^.fileFSSpec := fileReply.sfFile;
744     SetWTitle(gWindowPtr, docRecHdl^^.fileFSSpec.name);
745
746     DoGetandSetWindowPosition(gWindowPtr);
747     DoGetPrintableSize(gWindowPtr);
748
749     ShowWindow(gWindowPtr);
750     end;
751       {of procedure DoOpenCommand}
752
753   { ############################################################ InvalidateScrollBarArea }
754
755   procedure InvalidateScrollBarArea(myWindowPtr : WindowPtr);
756
757     var
758     tempRect : Rect;
759
760     begin
761     SetPort(myWindowPtr);
762
763     tempRect := myWindowPtr^.portRect;
764     tempRect.left := tempRect.right - 15;
765     InvalRect(tempRect);
766
767     tempRect := myWindowPtr^.portRect;
768     tempRect.top := tempRect.bottom - 15;
769     InvalRect(tempRect);
770     end;
771       {of procedure InvalidateScrollBarArea}
772
773   { ################################################################################ DoFileMenu }
774
775   procedure DoFileMenu(menuItem : integer);
776
777     begin
778     case (menuItem) of
779
780       iClose:
781         begin
782         DoCloseCommand;
783         end;
784
785       iOpen:
786         begin
787         DoOpenCommand;
788         end;
789
790       iPageSetup:
791         begin
792         DoPrintStyleDialog;
793         end;
794
795       iQuit:
796         begin
797         while (FrontWindow <> nil) do
798           DoCloseCommand;
799         gDone := true;
800         end;
801       end;
802         {of case statement}
803     end;
804       {of procedure DoFileMenu}
805
806   { ############################################################################ DoMenuChoice }
807
808   procedure DoMenuChoice(menuChoice : longint);
809
810     var
811     menuID, menuItem : integer;
812     itemName : string;
813     daDriverRefNum : integer;
814
815     begin
816     menuID := HiWord(menuChoice);
817     menuItem := LoWord(menuChoice);
818
819     if (menuID = 0) then
```

```
820       Exit(DoMenuChoice);
821
822     case (menuID) of
823
824       mApple:
825         begin
826         GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
827         daDriverRefNum := OpenDeskAcc(itemName);
828         end;
829
830       mFile:
831         begin
832         DoFileMenu(menuItem);
833         end;
834
835       mDemonstration:
836         begin
837         if (menuItem = iPreferences) then
838           DoPreferencesDialog;
839         end;
840       end;
841         {of case statement}
842
843     HiliteMenu(0);
844     end;
845       {of procedure DoMenuChoice}
846
847   { ############################################################################ DoAdjustMenus }
848
849   procedure DoAdjustMenus;
850
851     var
852     menuHdl : MenuHandle;
853
854     begin
855     if (gWindowOpen) then
856       begin
857       menuHdl := GetMenuHandle(mFile);
858       DisableItem(menuHdl, iOpen);
859       EnableItem(menuHdl, iClose);
860       EnableItem(menuHdl, iPageSetup);
861       end
862     else
863       begin
864       menuHdl := GetMenuHandle(mFile);
865       EnableItem(menuHdl, iOpen);
866       DisableItem(menuHdl, iClose);
867       DisableItem(menuHdl, iPageSetup);
868       end;
869
870     DrawMenuBar;
871     end;
872       {of procedure DoAdjustMenus}
873
874   { ########################################################################## DoUpdateWindow }
875
876   procedure DoUpdateWindow(myWindowPtr : WindowPtr);
877
878     var
879     str : string;
880     oldPort : GrafPtr;
881
882     begin
883     SetPort(myWindowPtr);
884
885     MoveTo(10, 20);
886     DrawString('Current Application Preferences:');
887     MoveTo(10, 35);
888     DrawString('Sound On: ');
889     if (gSoundOn) then
890       DrawString('YES')
891     else
892       DrawString('NO');
893
894     MoveTo(10, 50);
895     DrawString('Full Screen On: ');
896     if (gFullScreenOn) then
```

```
      DrawString('YES')
    else
      DrawString('NO');

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

    if (gPrintRect.bottom <> 0) then
      begin
      MoveTo(10, 85);
      DrawString('Information from printable area ("PrAr") resource:');
      NumToString(longint(gPrintRect.bottom), str);
      MoveTo(10, 100);
      DrawString('Page print area height in screen pixels: ');
      DrawString(str);
      NumToString(longint(gPrintRect.right), str);
      MoveTo(10, 115);
      DrawString('Page print area width in screen pixels:  ');
      DrawString(str);
      end
    else
      begin
      MoveTo(10, 85);
      DrawString('No printable area ("PrAr") resource saved yet');
      end;

    end;
      {of procedure DoUpdateWindow}

{ ############################################################################## DoMouseDown }

procedure DoMouseDown(var eventRec : EventRecord);

    var
    myWindowPtr : WindowPtr;
    partCode : integer;
    growRect : Rect;
    newSize : longint;

    begin
    partCode := FindWindow(eventRec.where, myWindowPtr);

    case (partCode) of

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

      inSysWindow:
        begin
        SystemClick(eventRec, myWindowPtr);
        end;

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

      inDrag:
        begin
        DragWindow(myWindowPtr, eventRec.where, qd.screenBits.bounds);
        end;

      inGoAway:
        begin
        if (TrackGoAway(myWindowPtr, eventRec.where)) then
          DoCloseCommand;
        end;

      inGrow:
```

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

      inZoomIn, inZoomOut:
        begin
        if (TrackBox(myWindowPtr, eventRec.where, partCode)) then
          begin
          SetPort(myWindowPtr);
          EraseRect(myWindowPtr^.portRect);
          ZoomWindow(myWindowPtr, partCode, false);
          InvalRect(myWindowPtr^.portRect);
          end;
        end;
      end;
        {of case statement}
    end;
      {of procedure DoMouseDown}

{ ********************************************************************************* DoEvents }

procedure DoEvents(var eventRec : EventRecord);

    var
    myWindowPtr : WindowPtr;
    charCode : char;

    begin
    myWindowPtr := WindowPtr(eventRec.message);

    case (eventRec.what) of

      mouseDown:
        begin
        DoMouseDown(eventRec);
        end;

      keyDown, autoKey:
        begin
        charCode := chr(BAnd(eventRec.message, charCodeMask));
        if (BAnd(eventRec.modifiers, cmdKey) <> 0) then
          begin
          DoAdjustMenus;
          DoMenuChoice(MenuKey(charCode));
          end;
        end;

      updateEvt:
        begin
        BeginUpdate(myWindowPtr);
        EraseRgn(myWindowPtr^.visRgn);
        DoUpdateWindow(myWindowPtr);
        DrawGrowIcon(myWindowPtr);
        EndUpdate(myWindowPtr);
        end;

      osEvt:
        begin
        case (BAnd(BSR(eventRec.message, 24), $000000FF)) of

          suspendResumeMessage:
            begin
            gInBackground := (BAnd(eventRec.message, resumeFlag) = 0);
            end;
          end;
            {of inner case statement}
        HiliteMenu(0);
        end;
```

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

{ ################################################################## start of main program }

begin

  { ……… initialise managers }

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

  { ……… set current resource file to application resource fork }

  gAppResFileRefNum := CurResFile;

  { ……… set up menu bar and menus }

  menubarHdl := GetNewMBar(rMenubar);
  if (menubarHdl = nil) then
    DoError(MemError);
  SetMenuBar(menubarHdl);
  DrawMenuBar;

  menuHdl := GetMenuHandle(mApple);
  if (menuHdl = nil) then
    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 }

  gDone := false;

  while not (gDone) do
    begin
    if (WaitNextEvent(everyEvent, eventRec, kMaxLong, nil)) then
      DoEvents(eventRec);
    end;
end.

{ ############################################################################### }
```

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.

The constant declaration block

Lines 69-76 establish constants relating to menu IDs and menu item numbers. Lines 78-80 establish constants relating to window, menubar and alert resource IDs.

The constants at Lines 81-84 relate to the Preferences dialog box resource and associated checkbox item numbers. Lines 85-86 represent the resource ID and index for the string containing the name of the application's preferences file. Lines 87-94 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 96 defines kMaxLong as the maximum possible long value.

The type declaration block

The DocRecord data type (Lines 102-106) is for the document record. In this demonstration, the only field required is that for a file system specification.

The AppPrefs data type (Lines 108-114) is for the application preferences settings. The three Boolean values are set by checkboxes in the Preferences dialog box.

The WinState data type (Lines 116-121) 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 123) will be used in the functions related to the getting and saving of the printable area width and height.

The variable declaration block

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.

The procedure DoError

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.

The procedure 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 198 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 200. If the call is not successful, an error alert box is presented and the function simply returns (Lines 203-205).

Line 208 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 211-215). If the Get1Resource call is not successful (that is, the document file's resource fork does not yet contain a 'PrAr' resource), Line 219 allocates a block of memory for a Rect, Line 222 copies the rectangle in the prPage field of the prInfo record to this block, and Line 223 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 228), Line 230 writes the resource map and data to disk, and Line 235 discards the resource in memory. The document file's resource fork is then closed (Line 240).

The procedure 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 255 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 257. If the call is not successful, an error alert box is presented and the function simply returns (Lines 260-262).

If the resource fork is successfully opened, the call to Get1Resource at Line 265 attempts to read in the resource. If the call is successful (Line 266), Line 268 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 269) and the document file's resource fork is closed (Line 272).

The procedure 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.

The procedure 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 308 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 310. If the resource fork cannot be opened, an error alert is presented and the function simply returns (Lines 313-315).

Line 318 gets a pointer to the window record, allowing Line 319 to get a pointer to the WStateData record. Lines 320-321 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 323-324 get a rectangle equal to the content region of the window. Line 324 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 325 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 325 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 324 which, not being equal to the standard state rectangle (Line 326), must be equal to the current user state rectangle.

Line 329 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 332) and the resource is tagged as changed (Line 333). If the Get1Resource call is not successful (that is, the document file's resource fork does not yet contain a 'WiSt' resource), Line 340 creates a new window state record, Line 343 makes this record equal to the previously "filled-in" window state record, and Line 344 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 349), Line 351 writes the resource map and data to disk, and Line 356 discards the resource in memory. The document file's resource fork is then closed (Line 359).

The procedure 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 376-377 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 380-381 establish the standard state rectangle as desired by the application.

Line 384 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 386) to open the document file's resource fork.

Line 394 attempts to read in the 'WiSt' resource. If the Get1Resource call is successful (Line 395), a "success" flag is set and the user state rectangle is set to that retrieved from the resource (Lines 397-398). If the call is not successful, the "success" flag is unset (Lines 401-820) and the user state rectangle remains as the default rectangle defined at Lines 376-377.

If the Get1Resource call was successful, the zoom state is also retrieved from the resource (Lines 405). 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 406). 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 408). If the Get1Resource call was not successful (Line 412) the display rectangle is set to the user state rectangle, which will be the default defined at Lines 376-377.

Line 415 moves the window to the specified coordinates, keeping it inactive. Lines 417-419 size the window to the specified size, adding any area added to the content region to the update region.

Line 421 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 423 discards the 'WiSt' resource in memory. Line 424 then closes the document file's resource fork.

The procedure 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 439-440).

Lines 442-448 create a new preferences record and assign to its fields the values in the global variables which store the current preference settings. Line 450 makes the preferences file's resource fork the current resource file. The Get1Resource call at Line 452 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 455), AddResource is called to make the preferences record in memory into a resource (Line 457), and WriteResource is called to write the resource to disk (Line 459).

With the resource written to disk, Line 464 disposes of the preferences record in memory and Line 465 resets the application's resource fork as the current resource file.

The function 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 484 saves the current resource file's file reference number and Line 485 sets the application's resource fork as the current resource file. This will be the "source" file.

The Get1Resource call at Line 487 reads the specified resource into memory. Line 491 gets the resource's name and Line 492 gets the resource's attributes. The call to DetachResource at Line 493 replaces the resource's handle with NULL without releasing the associated memory. The resource data is now simply arbitrary data in memory.

Line 494 sets the preferences file's resource fork as the current resource file. The AddResource call (Line 496) makes the arbitrary data in memory into a resource, assigning it the specified type, ID and name. Line 498 sets the resource attributes in the resource map. The ChangedResource call (Line 500) tags the resource for update and pre-allocates the required disk space. The WriteResource call (Line 502) then writes the resource to disk.

With the resource written to disk, Line 507 discards the resource in memory and Line 508 resets the resource file saved at Line 484 as the current resource file.

The procedure 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 528 retrieves from the application's resource file the resource containing the required name of the preferences file ("MoreResources Preferences"). Line 530 finds the location of the Preferences folder, returning the volume reference number and directory ID in the last two parameters.

Line 534 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 536) 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, Lines 540-541 create the preferences file. The call to FSpCreateResFile (Line 540) 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 (unless you've hacked your System file... /kdg):

If the file is created successfully, the resource fork of the file is opened (Line 545) and the master preferences ('PrFn') and application missing 'STR ' resources are copied to the resource fork from the application's resource file (Lines 550 and 552). If the resources are not successfully copied (Line 555), the resource fork of the new file is closed (Line 557), the file is deleted (Line 558), and the fileRefNum variable is set to indicate that the file does not exist (line 559).

If the preferences file exists (either previously or newly-created) (Line 565), the resource fork of that file is set as the current resource file (Line 567), the preferences resource is read in from the resource fork (Line 569) and, if the read was successful, the three Boolean values are assigned to the global variables which store those values (Lines 573-575). (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 577 assigns the file reference number for the open preferences file resource fork to a global variable (the fork is left open), and Line 579 resets the application's resource fork as the current resource file.

The procedure DoPrintStyleDialog

DoPrintStyleDialog is called when the user chooses the Page Setup… item in the File menu. It presents the print style dialog box (Line 591).

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 593), 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 594). In addition, the window's port rectangle is invalidated (Line 595) to force an update of the window, thus ensuring that the new printable area values are displayed immediately.

The procedure 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 681, 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.

The procedure 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 696-697, a pointer to the front window, and a handle to the associated document record, are retrieved.

Line 699 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 701), 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 702).

Line 704 disposes of the document record, Line 705 disposes of the window record, and Line 706 sets the "window is open" flag to indicate that the window is not open.

The procedure 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 724) and, if the user clicks the Cancel button, the function simply returns. If the user clicks the OK button, a window is opened (Line 728), a document record is created (Line 732), a flag is set to indicate that the window is open (Line 739), the window's graphics port is set as the current port for drawing (Line 740), the document record is connected to the window record (Line 742), the file system specification for the chosen file is assigned to the document record's file system specification field (Line 743), and the window's title is set (Line 744).

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

746). In addition, the application-defined function which reads in the printable area resource from the document's resource fork is called (Line 747).

With the window positioned and sized, ShowWindow is called (Line 749) to make the window visible. (The window's 'WIND' resource specifies that the window is to be initially invisible.)

The procedure InvalidateScrollBarArea

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

The procedures DoFileMenu, DoMenuChoice, and DoAdjustMenus

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.

The procedures DoUpdateWindow, DoMouseDown, and DoEvents

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.

The main program block

The main function initialises the system software managers (Line 1062), sets the application's resource fork as the current resource file (Line 1069), sets up the menus (Lines 1073-1083), and creates and initialises a TPrint record (1087-1090). Then, before the main loop (Lines 1100-1104) is entered, main calls the application-defined function which reads in the application preferences settings from the preferences file (Line 1094). (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.)