Using the Resource Manager

The **Resource Manager** in System 7.0 allows you to read a portion of a resource into memory or write a block of data to a resource stored on disk, even if the size of the memory you have to work in is smaller than the entire resource. You can also change the size of a resource on disk.

Using Partial Resources

Some resources, such as the 'snd ' and 'sfnt' resources, can be quite large-larger, in fact, than the memory available. The partial resource routines in System 7.0 allow you to read a portion of the resource into memory or alter a section of the resource while it is still on disk. You can also enlarge or reduce the size of a resource on disk. The **ReadPartialResource** procedure reads a portion of a resource from disk into memory, and the **WritePartialResource** procedure writes a portion of data to a resource on disk. You can also change the size of the resource on disk to any desired size, using the **SetResourceSize** procedure. When you use the partial resource routines, you specify how far into the resource you want to begin reading or writing and how many bytes you actually want to read or write at that spot, so you must be sure of the location of the data.

> **Warning:** Be aware that having a copy of a resource in memory when you are using the partial resource routines may cause problems. If you have modified the copy in memory and then access the resource on disk using either the **ReadPartialResource** or **WritePartialResource** procedure, you will lose changes made to the copy in memory.

To read or write any part of a resource, call the **SetResLoad** procedure specifying FALSE for its load parameter, and then use the **GetResource** function to get an empty handle to the resource. (Because of the call to the **SetResLoad** procedure, the **GetResource** function does not load the entire resource into memory.) Use the **ResError** function to check for errors.

To check for the existence of the new partial resource routines, use the **Gestalt** function with the gestaltResourceMgrAttr selector. This selector returns a range of bits, the meaning of which must be determined by comparison with a list of constants. If the bit defined by the constant gestaltPartialRsrcs is set, the partial resource routines are available.

The Listing below illustrates one way to deal with partial resources. This procedure begins with a call to the **SetResLoad** procedure. A handle to the resource from which you want to read is put into the myResHdl variable. If there is no error on the call to the **GetResource** function, a call is made to the **ReadPartialResource** routine with the handle to the resource. If there are no errors with this call, you exit the procedure. DoError is the name of your routine that handles and processes errors.

Using partial resource calls

```
// Assuming inclusion of MacHeaders

// Prototype routine like this prior to calling it
void ReadAPartial(long, long, Ptr *);

void  ReadAPartial(long start, long count, Ptr *PutItHere)
{
```

```
    ResType myRsrcType;     // resource type passed to GetResource
    short myRsrcID;         // id of resource
    Handle myResHdl;        // handle to resource
    // error variables
    OSErr resErr;
    OSErr RPRErr;
    // prototype error handling procedure
    void DoError(OSErr);

    SetResLoad(FALSE);          // do not load resource

    // Set up myRsrcType and myRsrcID to your liking.
    myResHdl = GetResource(myRsrcType, myRsrcID);
    resErr = ResError();
    SetResLoad(TRUE);           // reset to always load

    if ( !resErr ) {        // GetResource returned successfully
        ReadPartialResource(myResHdl, start, PutItHere, count);
        RPRErr = ResError();
        // Check and report error.
        if ( RPRErr )
            DoError(RPRErr);
    }
    else    // there's an error from GetResource
        DoError(resErr);
}
```

Creating and Opening Resource Files

System 7.0 introduces a simple, standard format for identifying a file or directory, the file system specification (**FSSpec**) record. This record contains the volume reference number of the volume on which a file or directory resides, the directory ID of the parent directory, and the name of the file or directory.

The **FSpOpenResFile** function and **FSpCreateResFile** procedure open and create resource files for files or directories named by the file system specification record.

If you want to open or create resource files under the Hierarchical File System (HFS) but are not using the file system specification record in your application, you can use the **HOpenResFile** function and **HCreateResFile** procedure.

Storing Fonts in a Resource Fork

Storing a font in an application's resource fork can create serious problems for a user who tries to print a document in that font when background printing is on. Never store fonts in a document's resource fork, since this can cause heap corruption. If you feel that a document needs to have a particular font available, you should license it for distribution and let users install it in their System files.

If you use a font as a way to store symbols that your application uses in a palette or for some other special purpose, use a font family ID in the range

assigned for uninterpreted symbols. (For more information about font family ID ranges, see Font Families and Scripts. For more information about scripts and script systems, see the **Worldwide Software Overview**