Chapter 21
Resource Manager

The Resource Manager is a tool that is loaded when the Apple IIGS® is started. The Resource Manager remains in the system the entire time the system is running.

The Resource Manager manages resources in the resource fork of a file. A resource is one or more bytes of continuous data. The format of the data is defined by an application or by standard resources. The Resource Manager does not know, or need to know, the format of any resource. The Resource Manager reads and writes resources to and from resource files, and tracks them while they are being used.

Resources are referred to by a **resource type and a resource ID number**. The resource type (also referred to as type) defines the structure of the resource. The resource ID number (also referred to as resource ID or just ID) is a number unique to each resource of the same type. Therefore, different types could have the same IDs.

Example: A structure could be defined to hold a screen image, which would be a resource type.

Then there could be many screen images, all with the same defined structure, but containing different data. Each of these would be differentiated by unique IDs.

Note
There is a glossary of Resource Manager terms provided at the end of this document.

A preview of the Resource Manager routines

To introduce you to the capabilities of the Resource Manager, all Resource Manager routines are grouped by function and briefly described in Table 1. These routines are described in detail later in this document, where they are separated into housekeeping routines (discussed in routine number order) and the rest of the Resource Manager routines (discussed in alphabetical order).

Table 1
Resource Manager routines and their functions

Routine	Description
Housekeeping routines	
ResourceBootInit	Called only by the Tool Locator-must not be called by an application
ResourceStartup	Logs an application in with the Resource Manager for use by an application
ResourceShutdown	Logs off an application with the Resource Manager when an application quits
ResourceVersion	Returns the version number of the Resource Manager
ResourceReset	Called only when the system is reset-must not be called by an application
ResourceStatus	Indicates whether the Resource Manager is active
Initialization and termination routines	
CreateResourceFile	Creates and initializes a file for resource use
OpenResourceFile	Opens a resource fork for resource access by the Resource Manager
CloseResourceFile	Close an opened resource fork
AddResource	Creates and adds a new resource to an open resource file
RemoveResource	Deletes a resource from an open resource file
Resource access routines	
LoadResource	Loads a resource into memory from an open resource file
LoadAbsResource	Loads a resource into memory at an absolute address from an open resource file
DetachResource	Makes a loaded resource unknown to the Resource Manager
ReleaseResource	Frees memory used by a loaded resource
WriteResource	Forces the write to disk of a changed resource
ResourceConverter	Installs resource convert routines to allow memory formats different from disk
CountTypes	Returns the number of different resource types in all open resource files
GetIndType	Returns a resource type associated with a given index, use to find every type
CountResources	Returns the number of different resources of a given resource type
GetIndResource	Loads a resource associated with a given index, used to load every resource of a given type
Resource record access routines	
GetResourceAttr	Returns the attributes of a given resource such as locked, fixed, preload, etc...
SetResourceAttr	Changes the attributes of a given resource such as locked, fixed, preload, etc...
GetResourceSize	Returns the number of bytes a resource occupies on disk
MarkResourceChange	Sets a resource to changed or unchanged so it will or will not be written to disk
MatchResourceHandle	Find the resource ID and type given the handle of loaded resource
SetResourceID	Changes the ID of a given resource

Resource file routines

GetCurResourceFile	Returns the ID of the current resource file, where searches start
SetCurResourceFile	Changes the current resource file, which is where searches start
SetResourceFileDepth	Changes the number of files to be searched when searching open resource files
GetOpenFileRefNum	Returns the GS/OS open file reference number of an open resource file
HomeResourceFile	Returns the file ID of the open resource file that contains a given resource
GetMapHandle	Returns the handle of a resource map that has been loaded into memory
UpdateResourceFile	Writes all changed data to disk of a given open resource file

Application switching routines

GetCurResourceApp	Returns the user ID of the current application using the Resource Manager
SetCurResourceApp	Changes the current application using the Resource Manager, used by application switching and desk accessories

Resource global access routines

SetResourceLoad	Tells the Resource Manager to read resources into memory or not
UniqueResourceID	Returns an used resource ID for a given resource, used when creating resources

Structure of Resource Files

GS/OS files have two parts, a data part called the data fork, and a resource part called the resource fork. The data fork is where an application stores information and has a format defined by the application. The resource fork is where the Resource Manager stores information about resources. The format of the resource fork is defined by the Resource Manager. (See GS/OS documentation for more information about data and resource forks.)

An application does not need to know the format of the resource fork to use resources. However, it does know about the format of individual resources contained in the fork. Conversely, the Resource Manager knows the format of the fork, but nothing about the format of individual resources other than they are blocks of data.

Note: References to 'resource file' are actually referring to the resource fork part of a file.

A resource file can be created and edited with the aid of the Resource Editor (**when available for the GS**), or with whatever tools are provided by the development system you are using.

Using the Resource Manager

Tool Loading The Resource Manager is loaded and initialized by the system when it starts up. The is no need for an application to load the Resource Manager. The file RESOURCE.MGR must be placed in the SYSTEM.SETUP directory of the boot disk. A system resource file, if there is one, called SYS.RESOURCES must also be placed in the SYSTEM.SETUP directory.

Startup Call **ResourceStartup** to log in with the Resource Manager.

Open File Call **OpenResourceFile** to open each necessary resource file. If there is a system resource file it will be opened at system startup.

Access Call **LoadResource** to load a resource from disk to memory.

Close Files Call **CloseResourceFile** for each file to be closed, or NIL to close every open file. It may not be necessary for an application to call **CloseResourceFile** because **ResourceShutdown** will close every resource file opened by the application.

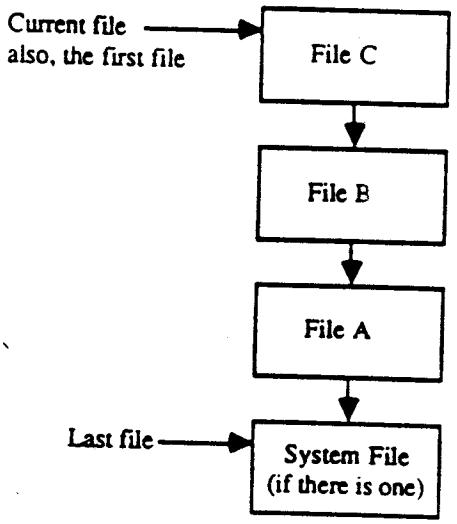
Shutdown Call **ResourceShutdown** before quitting. Do not make any other Resource Manager calls, except for **ResourceStartup**, after **ResourceShutdown**.

Controlling how open resource files are searched

The resource manager searches open resource files to perform many of its functions. The order of the search depends on when files were opened. When files are opened using **OpenResourceFile** they are placed in a list. As files are opened they are placed at the front of the list. After making the following calls in the following order:

OpenResourceFile	File A
OpenResourceFile	File B
OpenResourceFile	File C

the list will look like:



The most recently opened resource file, file C, will be the *current resource file*. It is also considered the *first resource file* (most recently opened). The file that was opened the longest ago is considered the *last resource file*. When asked to find a resource, which happens for almost every call that passes a resource ID and type, the Resource Manager will first search file C, then file B, then file C, then the system file. Generally, the current file will be searched first and continue to the last file. The first occurrence of a resource with the matching ID and type will stop the search. This means the that second occurrence of a resource with the same ID and type, even if in different files, will never be found by the Resource Manager during normal search operations.

However, the files searched by the Resource Manager can be controlled by an application. The current file can be set to any file, **SetCurResourceFile**, and the number of files searched can also be set, **SetResourceFileDepth**. From the above example, to search files B, A and the system file the current would be set to file B and the depth to $FFFF. To search only files C, B and A the current would be set to file C and the depth set to 3. To search files in any order the depth would be set to 1 and current file would be set to the desired file repeatedly. To reset to search all files, pass NIL to **SetCurResourceFile** and $FFFF to **SetResourceFileDepth**.

Resource file ID numbers

Each open resource file can be referred to by a file ID. Every open resource file has an unique ID. The ID is returned by **OpenResourceFile** when a file is opened. A file's ID is not the same as its open file reference number. A file's open file reference number can be obtained by calling **GetOpenFileRefNum**.

Resource Types

Resource types are a word in size. Resource types have the following ranges:

Decimal:			
0			Reserved, not a valid resource type.
1	through	32,767	Reserved for application use.
32,768	through	65,535	Reserved for system use.
Hex:			
$0000			Reserved, not a valid resource type.
$0001	through	$7FFF	Reserved for application use.
$8000	through	$FFFF	Reserved for system use.

See Appendix A for information about some resource types that have a public format.

Resource ID Numbers

Resource ID numbers are a long in size. A resource ID must be unique for every resource with the same resource type in the same file. Resources of different types may have the same ID. ID numbers are divided into the following ranges:

Decimal:			
0			Reserved, not a valid ID number.
1	through	134,152,191	Reserved for application use.
134,152,192	through	134,217,727	Reserved for system use.
134,217,728	through	4,294,967,295	Reserved for future expansion.
Hex:			
$00000000			Reserved, not a valid ID number.
$00000001	through	$07FEFFFF	Reserved for application use.
$07FF0000	through	$07FFFFFF	Reserved for system use.
$08000000	through	$FFFFFFFF	Reserved for future expansion.

See Appendix A for information about some resource ID that are defined by the system.

GS Resource Manager Standard Routines

$011E ResourceBootInit

Warning
An application must never make this call.

Does nothing.

Parameters The stack is not affected by this call. There are not input or output parameters.

Errors None

$021E ResourceStartup

Called by applications to log in.

Important
Your application must make this call before it makes any other Resource Manager calls.

Parameters

Stack before call

previous contents	
userID	**Word** - Application's user ID.
	<-- **SP**

Stack after call

previous contents	
	<-- **SP**

Errors Memory Manager errors Returned unchanged

$031E ResourceShutdown

Called by applications to log out. All of the current application's open resource files are, updated, closed and their resources freed.

Important
If your application has call **ResourceStartup**, it must make this call before it quits.

Parameters The stack is not affected by this call. There are not input or output parameters.

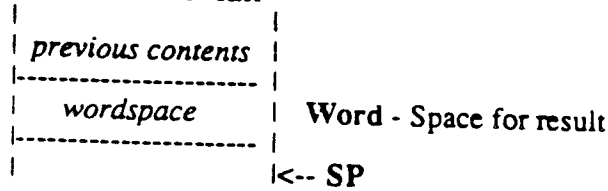
Errors None

$041E ResourceVersion

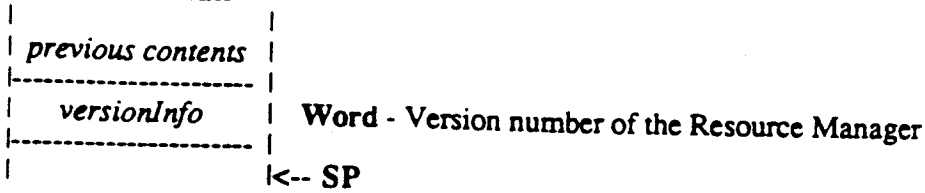
Returns the version number of the Resource Manager.

Parameters

Stack before call



Stack after call



Errors None

$051E ResourceReset

Resets the Resource Manager; called only when the system is reset.

Warning
An application must never make this call.

Parameters The stack is not affected by this call. There are not input or output parameters.

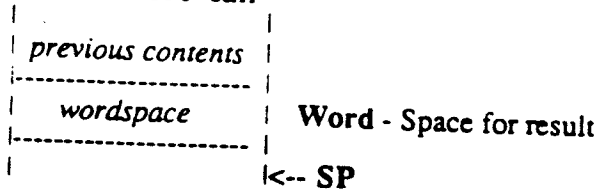
Errors None

$061E ResourceStatus

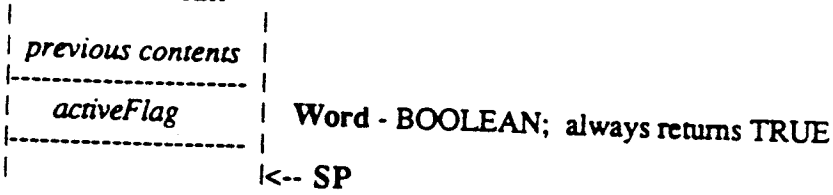
Indicates whether the Resource Manager is active. If the Resource Manager was loaded at system boot time, and was able to initialize, **ResourceStatus** will return TRUE. If one of these items failed the Resource Manager will not install as a tool at all.

Parameters

Stack before call



Stack after call



Errors None

GS Resource Manager Routines

$0C1E AddResource

Adds a resource to the current resource file. The resource will be marked as changed and written to the file when the file is updated.

Parameters

Stack before call

previous contents	
resourceHandle	**Long** - Handle of resource in memory
resourceAttr	**Word** - Attributes of the resource
resourceType	**Word** - Type of resource
resourceID	**Long** - ID of resource
	<-- SP

Stack after call

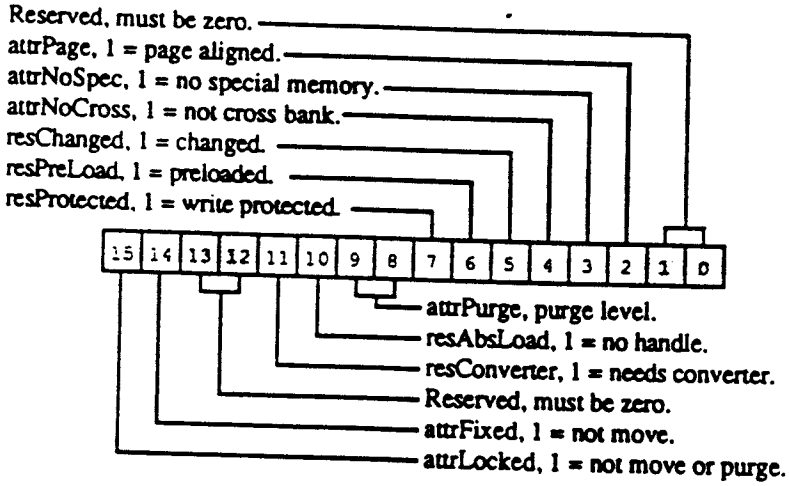
previous contents	
	<-- SP

Errors			
	$1E04	resNoCurFile	There is no current file to add the resource to
	$1E05	resDupID	ID is used by the type in the current file
	Memory Manager errors		Returned unchanged

Parameter description

resourceHandle If the handle is empty a resource of zero length will be written. The size of the resource is the size of the handle. Never pass a handle that was created by the Resource Manager unless the resource has been detached (see **DetachResource**).

resourceAttr Bits that describe some attributes of the resource:



ResChanged will be forced to 1 by the Resource Manager. ResPreload means the resource will be loaded by **OpenResourceFile**. ResProtected means the resource cannot be changed in the file. AttPage, attrNoSpec, attrNoCross, attrPurge, attrFixed, and attrLocked are pass to NewHandle when allocating a handle for the resource. See Memory Manager documentation for more information about these bits.

resourceType Resource type. See **Resource Types** for more information.

resourceID Resource ID. Must be unique for resources of the same type. See **UniqueResourceID** call to obtain a unique ID.

$0B1E CloseResourceFile

Note
Most applications will not have to make this call because **ResourceShutdown** will close all resource files opened by the application.

Updates the resource file, frees memory used by resources in the file and the resource map, and closes the file. If the file being closed is the current resource file the next file in the file list will be made the new current resource file.

System resource files will not be affected by this call.

Parameters

Stack before call

previous contents	
fileID	**Word** - ID of open resource file, NIL to close all files opened by the caller
	<-- **SP**

Stack after call

previous contents	
	<-- **SP**

Errors	GS/OS errors	Returned unchanged

$221E CountResources

Note
Most applications will not have to make this call. This call is a rather slow process. Applications should consider not using this call in time critical procedures.

Returns number of resources of a given type in all open resource files.

Parameters

Stack before call

previous contents	
longspace	**Long** - Space for result
resourceType	**Word** - Resource type
	<-- **SP**

Stack after call

previous contents	
totalResources	**Long** - Total number of resources of the given type in all open files
	<-- **SP**

Errors None

$201E CountTypes

Note
Most applications will not have to make this call. This call is a rather slow process. Applications should consider not using this call in time critical procedures.

Returns number of different resource types in all open resource files.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
	<-- **SP**

Stack after call

previous contents	
totalTypes	**Word** - Total number of different resource types in all open files
	<-- **SP**

Errors	Memory Manager errors	Returned unchanged

$091E CreateResourceFile

Initializes a resource fork with no resources for a file that has an empty resource fork. If the file does not exist it is created with the given aux type, filetype, file access, filename, an empty data fork, and an initialized resource fork. If the file exists and has something in the resource fork an error is returned. If the file exists and has an empty resource fork an initialized resource fork is made.

Parameters

Stack before call

previous contents	
auxType	**Long** - File aux type used to create file (used only if file does not exist)
fileType	**Word** - Filetype used to create file (used only if file does not exist)
fileAccess	**Word** - File access used to create file (used only if file does not exist)
fileName	**Long** - Pointer to GS/OS class 1 input file pathname of resource file
	<-- SP

Stack after call

previous contents	
	<-- SP

Errors

$1E01	resForkUsed	Resource fork not empty. Returned unchanged
GS/OS errors		

$181E DetachResource

Resource Manager will no longer know the resource is in memory. Additional memory will be allocated for the resource if the resource is asked for again. It will be up to the application to dispose of the handle after it is detached. A resource can only be detached if it is marked unchanged, meaning it does not need to be written to disk.

To make a copy of a resource **DetachResource** can be called followed by **AddResource**.

Parameters

Stack before call

previous contents	
resourceType	**Word** - Type of resource
resourceID	**Long** - ID of resource
	<-- SP

Stack after call

previous contents	
	<-- SP

Errors			
	$1E06	resNotFound	The given resource was not found
	$1E0C	resHasChanged	The resource has changed and not been updated

$141E GetCurResourceApp

Note
Most applications will not have to make this call.

Returns the user ID that is currently working with the Resource Manager. User ID of the Resource Manager is returned if there is no current application. This call is used by desk accessories and application switchers. This call should not normally be called by an application. See Appendix C for more information about application switching.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
	<-- **SP**

Stack after call

previous contents	
userID	**Word** - User ID of current application, NIL if no current application
	<-- **SP**

Errors None

$121E GetCurResourceFile

Returns ID of current resource file. Returns NIL if there is no current resource file.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
	<-- SP

Stack after call

previous contents	
activeFlag	**Word** - ID of current resource file, NIL if no current file
	<-- SP

Errors	$1E04	resNoCurFile	No current resource file

$231E GetIndResource

Note
Most applications will not have to make this call. This call is a rather slow process. Applications should consider not using this call in time critical procedures.

Returns the resource ID of a given resource index and type.. This call can be used to find every resource of a given type in all open files by passing indexes of one through *n*. The error resIndexRange is returned when there are no more resources of the given type.

Parameters

Stack before call

previous contents	
longspace	**Long** - Space for result
resourceType	**Word** - Resource type
resourceIndex	**Long** - Index
	<-- **SP**

Stack after call

previous contents	
resourceID	**Long** - ID of resource
	<-- **SP**

Errors	$1E0A	resIndexRange	Index is out of range
	Memory Manager errors		Returned unchanged

$211E GetIndType

Note
Most applications will not have to make this call. This call is a rather slow process. Applications should consider not using this call in time critical procedures.

Returns a unique resource type associated with a given index. This call can be used to find every different resource type in all open files by passing indexes of one through *n*. The error resIndexRange is returned when there are no more types.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
typeIndex	**Word** - Index
	<-- SP

Stack after call

previous contents	
resourceType	**Word** - Resource type associated with the given index
	<-- SP

Errors	$1E0A	resIndexRange	Index is out of range
	Memory Manager errors		Returned unchanged

$261E GetMapHandle

Note
This call is provided for application flexibility. Most applications will not need to use this call.

Returns a handle to an open resource map. All open resource files will be searched starting with the first open file. See Appendix B for information about the format of a map.

Parameters

Stack before call

previous contents	
longspace	**Long** - Space for result
fileID	**Word** - File ID, NIL for current file, $FFFF for system file
	<-- SP

Stack after call

previous contents	
mapHandle	**Long** - Handle of resource map, NIL if none
	<-- SP

Errors	$1E07	resFileNotFound	Invalid file ID passed

$1F1E GetOpenFileRefNum

Note
This call is provided for application flexibility. Most applications will not need to use this call.

Returns GS/OS open file reference number of an open resource file. All open resource files will be searched starting with the first open file. The reference number is for the resource fork of the file. The reference number should not be used to close the file, only the Resource Manager should close files it has opened. It is OK to use the reference number to read data from the file, but writing to the file could destroy the resource fork format unless the structure of the fork is maintained.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
fileID	**Word** - ID of open resource file
	<-- SP

Stack after call

previous contents	
openRefNum	**Word** - GS/OS open file reference number
	<-- SP

Errors	$1E07	resFileNotFound	Invalid file ID passed

$1B1E GetResourceAttr

Returns attributes of resource.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
resourceType	**Word** - Resource type
resourceID	**Long** - Resource ID
	<-- SP

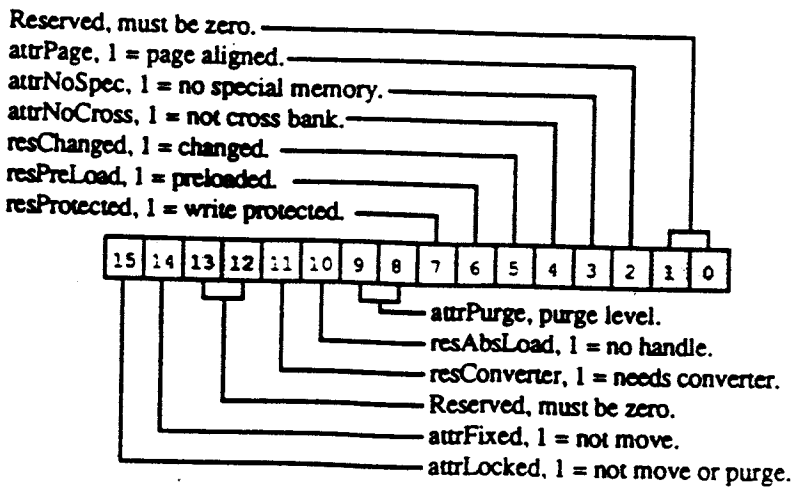
Stack after call

previous contents	
resourceAttr	**Word** - Attributes of resource
	<-- SP

Errors

$1E06	resNotFound	Resource not found

Resource attributes word



$1D1E GetResourceSize

The size of the given resource is returned. The size is the number of bytes the resource occupies on disk.

Parameters

Stack before call

previous contents	
longspace	Long - Space for result
resourceType	Word - Type of resource
resourceID	Long - ID of resource
	<-- SP

Stack after call

previous contents	
resourceSize	Long - Size of resource
	<-- SP

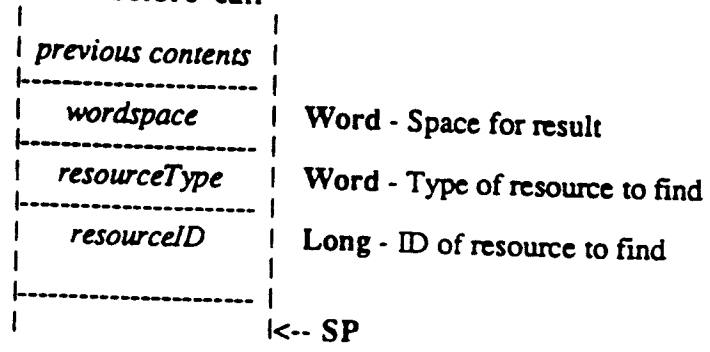
Errors	$1E06	resNotFound	Resource not found

$151E HomeResourceFile

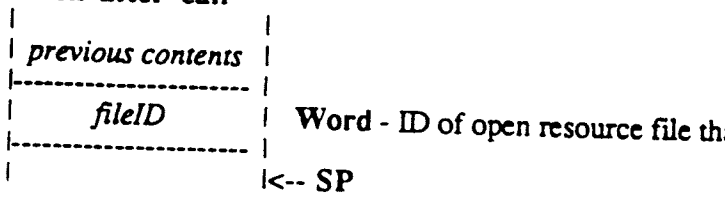
Returns the file ID of the file containing the given resource. If the resource is not found in any open resource file, NIL is returned on the stack and error is returned.

Parameters

Stack before call



Stack after call



Errors	$1E06	resNotFound	Resource was not found.

$271E LoadAbsResource

Note
Most applications will not have to make this call. Using this call requires a understanding of using absolute memory and how it can corrupt the system. The programmer must accept a great deal of responsibility when using this call.

Reads a resource into an absolute memory location. The resource will be read from disk into the given memory address, but no more than the given maximum size will be read. In order for a resource to be loaded to an absolute address it must have its resAbsLoad bit set. If NIL is passed as the address the value stored in the resHandle field of the resource's entry in the resource index is used as the absolute address. If that address is NIL, or some other inappropriate value, the resource will be loaded there and corrupt the system.

The size of the resource on disk is provided if the programmer would like to compare it to the maximum size passed to determine how much of the buffer was used.

Parameters

Stack before call

previous contents	
longspace	**Long** - Space for result
loadAddress	**Long** - Address to load resource at, NIL to load at preassigned address
maxSize	**Long** - Maximum number of bytes to load
resourceType	**Word** - Type of resource to find
resourceID	**Long** - ID of resource to find
	<-- **SP**

Stack after call

previous contents	
resourceSize	**Long** - Size of resource on disk
	<-- **SP**

Errors	$1E06	resNotFound	Resource was not found.
	GS/OS errors		Returned unchanged

$0E1E LoadResource

Reads a resource into memory. **LoadResource** can be called repeatedly whenever a resource is needed without having to wonder if it has already been loaded. If the resource has not been load, it will be and a handle returned. If the resource has been loaded, but its handle is empty (purged), the resource will be loaded again and the handle returned. If the resource has been loaded and its handle is not empty, the handle is returned and no action is performed. In any case, other than memory error, a handle to the resource will be returned.

Handles to resources should not be disposed of by an application (unless **DetachResource** is used) unless the application wants to mess up memory in which case a jump into a random memory location would be faster. The handle can be resized to any size other than zero. The handle data can be changed and move around in memory. If the resource changes, and the application wants the change to occur in the file, pass TRUE to **MarkResourceChange**. The next time the file is updated the new information will be written to disk. The resource can be forced to be written immediately after the call to **MarkResourceChange** by calling **WriteResource** or **UpdateResourceFile**.

Parameters

Stack before call

previous contents	
longspace	**Long** - Space for result
resourceType	**Word** - Type of resource to find
resourceID	**Long** - ID of resource to find
	<-- **SP**

Stack after call

previous contents	
resourceHandle	**Long** - Handle of resource in memory
	<-- **SP**

Errors			
	$1E06	resNotFound	Resource was not found.
	Memory Manager errors		Returned unchanged
	GS/OS errors		Returned unchanged

$101E MarkResourceChange

Tells Resource Manager to write a given resource to the resource file the next time the file is updated.

Parameters

Stack before call

previous contents	
changeFlag	Word - BOOLEAN; TRUE marks as changed, FALSE marks as not changed
resourceType	Word - Type of resource to find
resourceID	Long - ID of resource to find
	<-- SP

Stack after call

previous contents	
	<-- SP

Errors			
	$1E06	resNotFound	Resource was not found.

$1E1E MatchResourceHandle

Returns the resource ID and type of the resource that owns the given resource handle. The handle passed must be a valid handle. All open resource files will be searched starting with the first open file.

Note

The Resource Manager is designed to use resource ID and types in an efficient manner no matter how many resources are in a file. For small number of resources (less than 100) **MatchResourceHandle** can work very well. But, for files with a large number of resources, this call can be very slow if it is used often. A faster method is to include the resource's ID and type inside the resource structure. This way the ID and type can be pulled directly from the resource rather than making a **MatchResourceHandle** call.

Parameters

Stack before call

previous contents	
foundRec	**Long** - POINTER to space to store type and ID of resource
resourceHandle	**Long** - Handle of resource
	<-- **SP**

Stack after call

previous contents	
	<-- **SP**

Errors

$1E06	resNotFound	Resource not found

$0A1E OpenResourceFile

Opens a resource file and makes it the current resource file. The resource map for the file is loaded into memory as well as any resources marked resPreLoad. The Resource Manager will allow an application to open a maximum of 4,095 resource files.

The order the files are opened may be important depending on how an application structures resource use. If files A, B and C are opened in this order, C will be considered the first resource file (most recently opened), B the second, and A the last. Also, C will be the current resource file. When performing many search operations the Resource Manager will start with the current file and look through to the last. The system file, if there is one, will be the last (end of search list) of every application's resource file list.

All open resource files will be searched starting with the first open file when finding a unique file ID for the file.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
mapAddress	**Long** - Address of map in memory, NIL if map not in memory
fileName	**Long** - Pointer to GS/OS class 1 input file pathname of resource file
	<-- **SP**

Stack after call

previous contents	
fileID	**Word** - ID of open resource file
	<-- **SP**

Errors

$1E02	resBadFormat	Fork has an unknown format
$1E09	resNoUniqueID	No ID available for file, too many files open
GS/OS errors		Returned unchanged
Memory Manager errors		Returned unchanged

$171E ReleaseResource

Frees memory used by a resource. Memory used by the resource is freed by setting its purge level or disposed of if negative is passed as a purge level. See Memory Manager documentation for more information about purge levels and purging. If the resource is needed again it will be unpurged or loaded from disk.

Parameters

Stack before call

previous contents	
purgeLevel	**Word** - Purge level of 0 through 3 or negative to dispose of handle
resourceType	**Word** - Type of resource
resourceID	**Long** - ID of resource
	<-- **SP**

Stack after call

previous contents	
	<-- **SP**

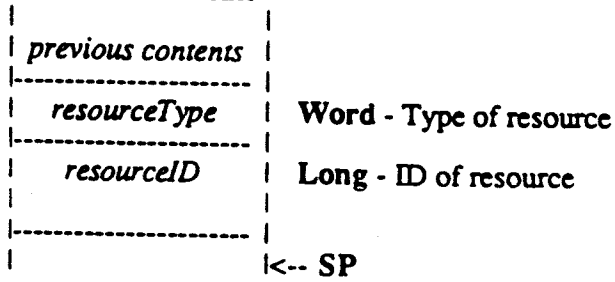
Errors			
	$1E06	resNotFound	The given resource was not found
	$1E0C	resHasChanged	The resource has changed and not been updated

$0F1E RemoveResource

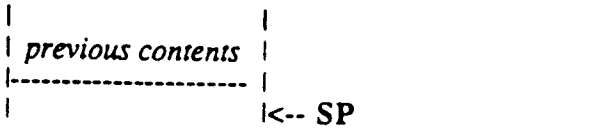
Deletes a resource from a resource file and releases any memory the resource occupied. The resource will no longer be available.

Parameters

Stack before call



Stack after call



Errors	$1E06	resNotFound	The given resource was not found

$281E ResourceConverter

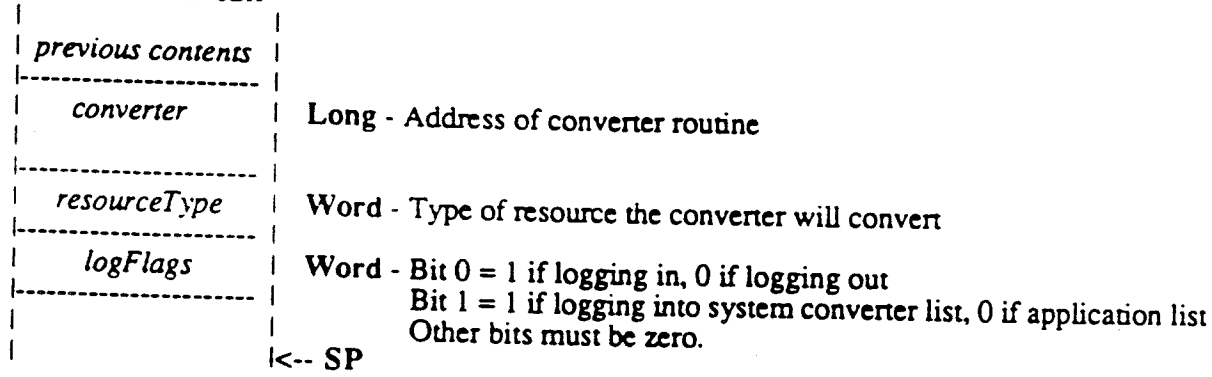
Adds or deletes a routines from a converter list. Converters perform conversions on resources for reading and writing to disk. The converter can perform operations such as compaction and expansion on resources. The application will only see the resource in its true form, not its compacted form. Converters could also be used to convert data used in 640 mode to 320 mode data.

There are two kinds of converter lists the Resource Manager maintains. One is the application converter list. Each application has its own converter list that it can add and delete converter from. The other kind is the system converter list. Converters added to this list can be used by all applications. When the Resource Manager looks for a converter to read or write a resource it will first search the application's list then the system list. This way an application can install a converter that overrides a converter installed in the system list. Applications should not install or delete converters in the system list.

Up to 10,922 converters can be logged in by any one application. This limit is not checked by the Resource Manager and no error is returned. The same can converter can be logged in more than once for different resource types. Logging in the same converter for the same resource type more than once does nothing. Logging in or out a different converter for a resource type already in the list results in a resDiffConverter error.

Parameters

Stack before call



Stack after call

Errors		
	$1E0D resDiffConverter	Different converter logged in for the type
	Memory Manager errors	Returned unchanged

Converter Routines

Converter routines are called by the Resource Manager to read and write resources to disk. Converters allow one format to reside on disk and another in memory. With a converter an application can deal with just the memory format of resources and let the converter worry about the disk format.

Some uses for converters might include the following. Code resources can have a relocatable format on disk and be relocated into memory by a converter. Graphic images and sound data can be packed on disk and expanded in memory. Dialog windows can be stored on disk in 640 mode format and converted for 320 mode if read in while in 320 mode. Hopefully there will be many more uses created in the future.

Converters must take a great deal of responsibility for system integrity. First, the converter must deal with the flexibility of the resource reference record and the GS/OS parameter blocks. A resource can be configured many ways, such as absolute memory or handle. GS/OS parameter blocks can contain a variable number of parameters. The converter must understand all these parameters to properly accomplish their task. Second, the converter must deal with the system context. The Resource Manager is called by an application which controls the state of the system.

Parameters to Converter Routines

Stack before call

previous contents	
longspace	**Long** - Space for result
convertCommand	**Word** - Command the converter should perform
convertParam	**Long** - Parameter defined by *convertCommand*
resPointer	**Long** - Pointer to resource reference record
	<-- SP

Stack after call

previous contents	
result	**Long** - Result returned is defined by the *convertCommand*
	<-- SP

Parameter description for Converter Routine

Direct page and data bank pointers are undefined when the converter is called. If the converter changes either of these values it must restore the original values before returning to the caller.

convertCommand

Command number that tells the converter what operation to perform. It also defines the format of other parameters. The defined commands follow, but a converter must make sure the command passed is in the range that it can handle. If a command is out of range the converter should return an error (any error code) and a NIL result.

Commands

readResource 0 - Read resource from disk. *ConvertParam* is a pointer to a GS/OS read file parameter block. The file mark is set to the beginning of the resource on disk and the block is set up to read the entire resource from disk. To simply read the resource from disk perform the following instructions.

```
push  convertParam    Pointer to read parameter block.
push  $2012           GS/OS read command.
jsl   $E100B0         Call GS/OS.
                      Return any errors.
```

This will ready the resource into memory. For conversion you can find the address of where the data is in memory as well as its size from the read parameter block. The address to read the data into is either the resource handle dereferenced or the resource's absolute address (check the resource reference record to see which). If it is a handle it was locked by the Resource Manager and should be locked when the converter returns. *Result* returned must be NIL.

writeResource 2 - Write resource to disk. *ConvertParam* is a pointer to a GS/OS write file parameter block. The file mark is set to the beginning of the resource on disk and the block is set up to write the entire resource to disk. To simply write the resource to disk perform the following instructions.

```
push  convertParam    Pointer to write parameter block.
push  $2013           GS/OS write command.
jsl   $E100B0         Call GS/OS.
                      Return any errors.
```

The size of the resource on disk is the same as the converter returned from the returnDiskSize command (defined next). *Result* returned must be NIL.

returnDiskSize 4 - Return amount of disk space needed on disk for resource. *ConvertParam* is undefined. *Result* is the size the resource will need on disk. If the size is different from the amount of disk space the resource currently uses the disk space will be freed and new space allocated. This command is not made for resources loaded into absolute memory as their sizes cannot change.

convertParam

Defined by *convertCommand*. Pointer to GS/OS read parameter block if readResource command. Pointer to GS/OS write parameter block if writeResource command. Undefined if returnDiskSize command.

resPointer

Pointer to resource reference record. See **Resource Manager Summary** for a description of a resource reference record. The record contain information that many be needed by the converter.

result

Defined by *convertCommand*. NIL if readResource or writeResource command. Size resource needs on disk if returnDiskSize command.

$131E SetCurResourceApp

Note
Most applications will not have to make this call.

Tells the Resource Manager that a different application will now be making Resource Manager calls. This call is used by desk accessories and application switchers. See Appendix C for more information about application switching.

Parameters

Stack before call

previous contents	
userID	**Word** - User ID of application, NIL is OK to pass, it has no effect
	<-- SP

Stack after call

previous contents	
	<-- SP

Errors			
	$1E08	resBadAppID	Application has not called ResourceStartup

$111E SetCurResourceFile

Makes a given resource file current. All open files are available starting with the first file. Most searches will start with the given file and continue to resource file the application opened first. This call can be used to control which files are searched for resources. Also see **SetResourceFileDepth** to control also control the number of files searched.

Parameters

Stack before call

previous contents	
fileID	Word - ID of open resource file
	<-- SP

Stack after call

previous contents	
	<-- SP

Errors

$1E07	resFileNotFound	Resource file not open

$1C1E SetResourceAttr

The attributes of the given resource are changed to the given attributes. Attribute changes will only affect future Resource Manager calls. For example, the resource handle for the given resource will not be locked by **SetResourceAttr** if the attrLocked bit is passed as 1. However, the resource will locked the next time the Resource Manager allocates a handle for the given resource.

Parameters

Stack before call

previous contents	
resourceAttr	**Word** - New attributes of resource
resourceType	**Word** - Type of resource
resourceID	**Long** - ID of resource
	<-- SP

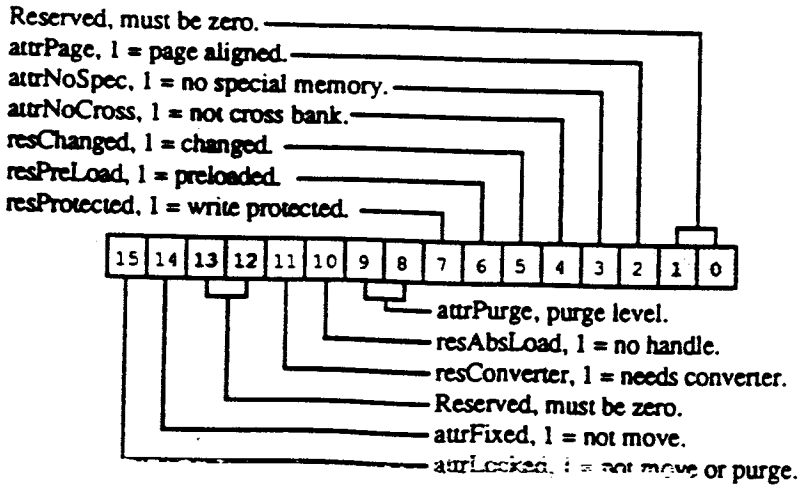
Stack after call

previous contents	
	<-- SP

Errors

$1E06	resNotFound	Resource not found

Resource attributes word



$251E SetResourceFileDepth

Sets the number of open resource files the Resource Manager will search during file search operations. The number will be used by all Resource Manager calls unless the call notes otherwise.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
searchDepth	**Word** - Number of files to search, $FFFF to search to very last file, NIL to just return current depth
	<-- **SP**

Stack after call

previous contents	
originalDepth	**Word** - Search file depth before call
	<-- **SP**

Errors None

$1A1E SetResourceID

The resource's ID is changed to a new ID.

Parameters

Stack before call

previous contents	
newID	**Long** - Resource's new ID
resourceType	**Word** - Type of resource
currentID	**Long** - Resource's current ID
	<-- **SP**

Stack after call

previous contents	
	<-- **SP**

Errors			
	$1E06	resNotFound	Resource not found
	$1E05	resDupID	New ID already used for the resource type

$241E SetResourceLoad

Note
Most applications will not have to make this call.

Disables and enables reading of resources from disk. When loading is set to FALSE the Resource Manager will not read resources from disk, but will allocate handles for the resource. For example a call to **LoadResource** will return an empty handle if the resource had not yet been read into memory. This is true for any Resource Manager call that reads resources from disk unless noted otherwise by a Resource Manager call.

Parameters

Stack before call

previous contents	
wordspace	**Word** - Space for result
readFlag	**Word** - NIL to not read resources, 1 to read, negative to just return current
	<-- **SP**

Stack after call

previous contents	
originalFlag	**Word** - Read flag before call, 0 if not reading, 1 if reading
	<-- **SP**

Errors None

$191E UniqueResourceID

Returns a resource ID for a given type which is not used by any resource of that type in any of the application's open resource files.

Parameters

Stack before call

previous contents	
longspace	**Long** - Space for result
IDrange	**Word** - Range of ID to return, $FFFF for any range of application reserved ID
resourceType	**Word** - Type of resource
	<-- SP

Stack after call

previous contents	
resourceID	**Long** - Unique resource ID
	<-- SP

Errors

$1E09	resNoUniqueID	No unique ID found.
$1E04	resNoCurFile	There is no current file

ID range

UniqueResourceID can be limited to finding an ID with a 64K boundary by passing an *IDrange* parameter from $0000 to $7FFF (applications should never pass a range greater than $07FE). The purpose of the range is to enable an application to place resources into logical groups in addition to resource types. The following table show some examples of *IDrange* parameters and their result.

IDrange	Lowest possible ID returned	Highest possible ID returned
$0000	$00000001 (NIL is invalid)	$0000FFFF
$0001	$00010000	$0001FFFF
$0002	$00020000	$0002FFFF
etc...		
$07FE	$07FE0000	$07FEFFFF (end of application reserved IDs)
$07FF	$07FF0000 (system reserved)	$07FFFFFF

$0800-$FFFE are invalid ranges

$FFFF	$00000001	$07FEFFFF (end of application reserved IDs)

$0D1E UpdateResourceFile

Note
Most applications will not have to make this call because **ResourceShutdown** will update all resource files opened by an application.

Does any changes, adds, or deletes of resources to the resource file and writes out the resource map if needed. All open resources files will be searched starting with the first file when searching for the given file ID.

Parameters

Stack before call

previous contents	
fileID	**Word** - ID of open resource file
	<-- SP

Stack after call

previous contents	
	<-- SP

Errors			
	$1E07	resFileNotFound	The given file ID is invalid
	GS/OS errors		Returned unchanged

$161E WriteResource

Note
Most applications will not have to make this call because **ResourceShutdown** will write all changed resources to disk.

Writes a resource to the resource file if it has been changed or added. **WriteResource** will only write the resource to disk if it is marked as changed. A resource is marked as changed only if **AddResource**, **MarkResourceChange**, or **SetResourceAttr** have been called.

Parameters

Stack before call

previous contents	
resourceType	**Word** - Type of resource to find
resourceID	**Long** - ID of resource to find
	<-- **SP**

Stack after call

previous contents	
	<-- **SP**

Errors			
	$1E06	resNotFound	Resource was not found.
	GS/OS errors		Returned unchanged

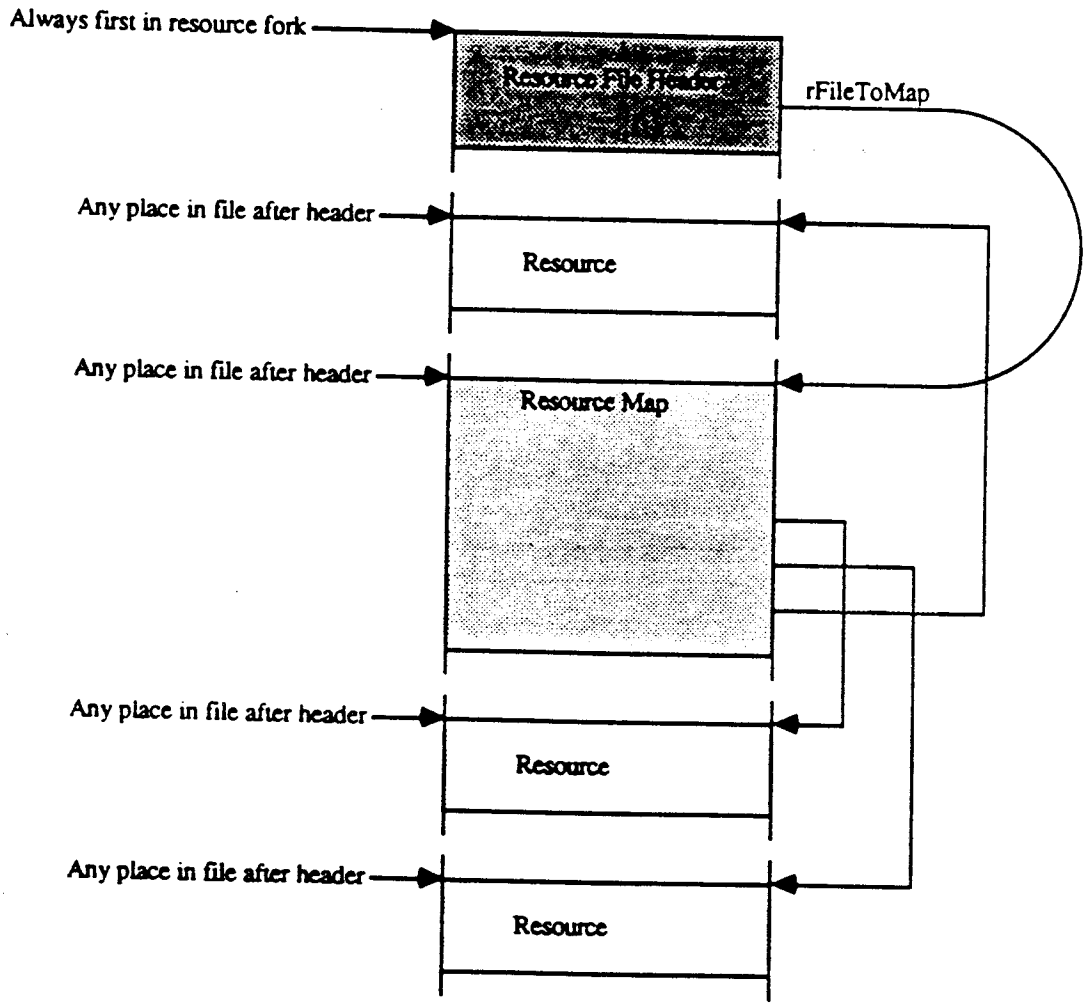
Appendix A: Standard Resource Types

There are currently no standard types.

Appendix B: Resource File Structure

This appendix is intended only for application programmers writing tools used to create, delete and edit resources in the resource fork. Other applications should not need to know this information. Applications that access the data fork or its map directly are probably just making things hard for themselves. It will be just another place that bugs and incompatibility will occur.

The format of the resource fork is diagramed as:



The header is the only block of information in the fork that is at an absolute location, first data in the fork. The header then contains an file offset to the map. The map then contains file offsets for every resource. The map and resource parts of the fork are allocated and moved within the file as the Resource Manager requires. Therefore, the position of any resource, or map can never be at an absolute location. Loading a resource, or map, from an absolute location in the file is a for sure way to be incompatible.

The first LONG in the resource fork is the version of the file's format. The version defined in this document is version 0. Versions 0-127 are GS Resource Manager formats, versions greater than 127 a Macintosh formats (although the Macintosh really doesn't have a version number as the first LONG, but

the LONG is always greater than 127 as the structure is currently defined). The format version defines the format of the entire resource fork. The format described in this document is version 0. Not checking the version and assuming the format of the resource fork will help your application to first destroy the user's file then crash in the future.

Version 0 Format:

The file header is a fixed size with the following format:

rFileVersion	LONG	Version of file's format, 0.
rFileToMap	LONG	File position of resource map, if format version is 0.
rFileMapSize	LONG	Size of map, if format version is 0.
rFileMemo	BYTE[128]	Reserved for application use, if format version is 0.

The memo space is reserved for application use. There are no Resource Manager calls to read or write to this area. Reads and writes must be done through GS/OS calls. Be sure not to write more than 128 bytes into the memo area, it will damage the fork's structure.

The format the resource map for format version 0 is:

mapNext	LONG	Space for handle of next resource map, NIL terminates list.
mapFlag	WORD	0 = application map, 1 = system map.
mapOffset	LONG	Map's file position.
mapSize	LONG	Size of map in file (size can change when in memory).
mapToIndex	LONG	Offset to index in map from start of map.
mapFileNum	WORD	Space for open resource file reference number (file ID).
mapID	WORD	File ID.
mapIndexSize	LONG	Total number of RESREF records in index.
mapIndexUsed	LONG	Number of used RESREF records in index.
mapFreeListSize	WORD	Number of RESBLK records in mapFreeList array.
mapFreeListUsed	WORD	Number of RESBLK records used in mapFreeList array.
mapFreeList	RESBLK[n]	Array of free blocks in file (RESBLK defined below).
	RESREF[n]	Resource index (RESREF defined below).

RESBLK record definition:

blkOffset	LONG	Offset to block from start of fork, NIL terminates the array.
blkSize	LONG	Size of the block in bytes.

RESREF record definition:

resType	WORD	Resource type, NIL terminates index.
resID	LONG	Resource ID.
resOffset	LONG	Offset to resource in file from start of file.
resAttr	WORD	File's attributes (see **GetResourceAttr** for bit definition).
resSize	LONG	Size, in bytes, of resource in file.
resHandle	LONG	Handle of resource in memory, NIL if not loaded.

Appendix C: Desk Accessory and Application Switching

Switching between applications is the responsibility of the code doing the switch. The current application ID must be saved before switching to a different application and restored when switching back to the original application.

Desk Accessories must handle this switching themselves. This can be done by:

```
                (On entry to desk accessory task handler.)
;
                pha                         Space for result.
                _GetCurResourceApp          Get original app, save on stack.
;
                pei     <myUserID           Pass my user ID.
                _SetCurResourceApp          Switch to my resource files and current file.
;
;               (Perform task including resource calls.)
;
;
                                            Pass result from GetCurResourceApp.
                _SetCurResourceApp          Restore original app resource file list.
;
                (Return to caller.)
```

The only exception to the above code is when **ResourceStartup** is called. In this case the code shoi be:

```
                (On entry to desk accessory task handler.)
;
                pha                         Space for result.
                _GetCurResourceApp          Get original app, save on stack.
;
                pei     <myUserID           Pass my user ID.
                _ResourceStartup
;
;               (Perform task including resource calls.)
;
;
                                            Pass result from GetCurResourceApp.
                _SetCurResourceApp          Restore original app resource file list.
;
                (Return to caller.)
```

The only difference is that the first call to **SetCurResourceApp** is not made. Instead a call to **ResourceStartup** is made. **ResourceStartup** will actually perform the same function as **SetCurResourceApp** with the addition of some initialization..

Resource Manager summary

This section briefly summarizes the constants, data structures, and tool set error codes contained in the Resource Manager.

Resource Manager data structures

Name	Offset	Type	Definition
ResHeaderRec (resource file header record)			
rFileVersion	$0000	LongWord	Format version of resource fork
rFileToMap	$0004	LongWord	Offset from start of fork to resource map record
rFileMapSize	$0008	LongWord	Number of bytes the map occupies in file
rFileMemo	$000C	128 bytes	Reserved space for application
rFileRecSize	$008C		Size of ResHeaderRec record
MapRec (resource map record)			
mapNext	$0000	Handle	Handle of next resource map
mapFlag	$0004	Word	Bit flags
mapOffset	$0006	LongWord	Map's file position
mapSize	$000A	LongWord	Number of bytes the map occupies in file
mapToIndex	$000E	Word	Offset from start of map to resource index
mapFileNum	$0010	Word	GS/OS open file reference number of resource file
mapID	$0012	Word	ID assigned to this resource map
mapIndexSize	$0014	LongWord	Total resource records allocated in the resource index
mapIndexUsed	$0018	LongWord	Number of resource records used in the resource index
mapFreeListSize	$001C	Word	Total free block records allocated in the free list
mapFreeListUsed	$001E	Word	Number of free block records used in the free list
mapFreeList	$0020	n bytes	Array of free block records
FreeBlockRec (free block record)			
blkOffset	$0000	LongWord	Offset from start of file to a free space in the file
blkSize	$0004	LongWord	Number of bytes free in the free space
blkRecSize	$0008		Size of free block record
ResRefRec (resource reference record)			
resType	$0000	Word	Resource type
resID	$0002	LongWord	Resource ID
resOffset	$0006	LongWord	Offset from start of file to the resource
resAttr	$000A	Word	Bit flags, attributes of the resource
resSize	$000C	LongWord	Number of bytes the resource occupies in the file
resHandle	$0010	Handle	Handle of resource loaded into memory
resRecSize	$0014		Size of resource reference record

Resource Manager constants

Name	Value	Description
Map flag values		
mapChanged	$0002	TRUE if the map has changed and needs to be written to disk
romMap	$0004	TRUE if the resource file is in ROM
		All other bits are reserved and must be zero.
Resource flag values		
resChanged	$0020	TRUE if the resource has changed and needs to be written to disk
resPreLoad	$0040	TRUE if the resource should be loaded by **OpenResourceFile**
resProtected	$0080	TRUE if the resource should never be written to disk
resAbsLoad	$0400	TRUE if the resource should be load at absolute address
resConverter	$0800	TRUE if the resource requires a converter for loading and writing
resMemAttr	$C31C	Bits passed to **NewHandle** when allocating memory for a resource
		All other bits are reserved and must be zero.

Resource Manager error codes

Code	Name	Description
$1E01	resForkUsed	Resource fork not empty
$1E02	resBadFormat	Format of the resource fork is unknown
$1E03	resNoConverter	No converter logged in for resource
$1E04	resNoCurFile	There are no current (open) resource files
$1E05	resDupID	ID is already used
$1E06	resNotFound	Resource was not found
$1E07	resFileNotFound	Resource file was not found
$1E08	resBadAppID	User ID was not found, caller has not called **ResourceStartup**
$1E09	resNoUniqueID	A unique ID was not found
$1E0A	resIndexRange	Index is out of range
$1E0C	resHasChanged	Resource is marked as changed, operation cannot be performed
$1E0D	resDiffConverter	Different converter logged in for the given resource type.

Glossary

application	Any code that has a unique user ID and wants to keep its resource files separate from other applications that may run at the same time. Applications include programs, parts of the same program, and desk accessories.
current file	The file the Resource Manager starts with when searching open resource files. **OpenResourceFile** makes the opened file the current file. The current file can be set by the application with **SetCurResourceFile**.
file list	A list of files in the order they were opened. The most recently opened file is considered the first file in the list, the file that was opened the longest ago is considered the last resource file.
first file	The most recently opened resource file.
last file	The resource file that was opened the longest ago. If there is a system resource file it will always be the last file because it was opened when the Resource Manager first started. If there is no system resource file the first file opened by the application is considered the last file.
map	The resource map contained in a resource file that has information about all the resources in a resource file. The resource map is read into memory from disk when the resource file is opened. Map usually refers to the resource map in memory rather than on disk.
resource	Zero or more bytes of continuous data. Although the format of the data is defined by an application or by standards, the Resource Manager does not know, or need to know, the format of any resource.
resource file	The resource fork of a file. Sometimes refers to the resource map in memory.
resource ID	A unique number given to a resource of a specific resource type.
resource type	A number that specifies a unique a resource format.

Change History

01 Feb Harry Yee

Changed description of UpdateResourceFile (BRC #39758).
LoadAbsResource call number changed from $1B1E to $271E.
Changed definition of resDupID error in AddResource call (BRC #40117)
Changed definition of resBadFormat error in OpenResourceFile call (BRC #39759)
Minor grammatical errors fixed (BRC #40474)
Removed all mention of "resNoCurApp" errors.