Technical Note TB40

Partial Resource Myths and Legends

CONTENTS

This Technical Note corrects and clarifies *Inside Macintosh: More Macintosh Toolbox* for the `ReadPartialResource` and `WritePartialResource` calls.

[Apr 01 1994]

Three Bogus Error Codes

The documentation for reading and writing partial resources is incorrect in three important ways--it states that the Resource Manager returns error codes to protect you against bad inputs, when it does no such thing. Three of the errors described in Resource Manager documentation are completely erroneous and are never returned by Resource Manager routines. Those codes are:

1. `inputOutOfBounds` does not exist

Inside Macintosh: More Macintosh Toolbox states that the `ReadPartialResource` and `WritePartialResource` calls check your `count` and `offset` input parameters and return `inputOutOfBounds` if they are out of range. This is not true; the only check on your input parameters is the verification of your resource handle. The result code `inputOutOfBounds` is completely erroneous and is not returned by any Resource Manager call.

2. `resourceInMemory` is never returned

The documentation also indicates that the Resource Manager returns the code `resourceInMemory` if the resource handle is not empty. This too is false; the only test performed on your resource handle is whether it is in the current map chain. The result code `resourceInMemory` is completely erroneous and is not returned by any Resource Manager call.

3. `writingPastEnd` is a figment of our imagination

In the case of `WritePartialResource`, if the combination of the `offset` and the `count` would extend the write request past the end of the resource on disk, the documentation indicates that the routine enlarges the resource and returns the result code `writingPastEnd`. This is not true; it does not automatically resize the resource and blindly overwrites any data in the following resource. If it is your intent to enlarge the resource, you must call `SetResourceSize` prior to the write. The result code `writingPastEnd` is completely erroneous and is not returned by any Resource Manager call.

Back to top

Replacement Partial Resource Documentation

The correct text documenting these two routines follows. This supersedes all current documentation concerning partial resources, including: *Inside Macintosh, Volume VI* , pages 13-16-17, 13-21-23; and *Inside Macintosh: More Macintosh Toolbox* , pages 1-111-115.

Back to top

Reading and Writing Partial Resources

You can use the `ReadPartialResource`, `WritePartialResource`, and `SetResourceSize` procedures to work with a portion of a large resource that may not otherwise fit in memory.

When using partial resource routines, you should call the `SetResLoad` procedure, specifying `FALSE` for the load parameter, before you call `GetResource`. Using the `SetResLoad` procedure prevents the Resource Manager from reading the entire resource into memory. Be sure to restore the normal state by calling `SetResLoad` again, with the load parameter set to `TRUE`, immediately after you call `GetResource`. Then use `ReadPartialResource` to read a portion

of the resource into a buffer and WritePartialResource as needed to write a portion of the resource from a buffer to disk.

Note that the partial resources routines work with the data in the memory pointed to by the buffer parameter, not the memory referenced through the resource's handle. Therefore, you may experience problems if you have a copy of a resource in memory when you are using the partial resource routines. If you have modified the copy in memory and then access the resource on disk using the ReadPartialResource procedure, ReadPartialResource reads the data on disk, not the data in memory, which is referenced through the resource's handle. Similarly, WritePartialResource writes data from the specified buffer, not from the data in memory, which is referenced through the resource's handle.

The partial resources routines do not decompress Apple's compressed resource format. It is not really possible to decompress part of a resource, since the current decompression method requires the entire resource to be in memory for the conversion to occur. Therefore, the partial resources routines should not be used when the data is in Apple's proprietary format. You can determine if a resource is compressed by examining bit 0 of the resource's attributes. If this bit is clear, the resource is not compressed and it is safe to use these partial resources routines.

Back to top

ReadPartialResource

You can use the ReadPartialResource procedure to read part of a resource into memory and work with a small subsection of a large resource.

```
PROCEDURE ReadPartialResource (theResource: Handle;
                               offset: LongInt; buffer: UNIV Ptr;
```

theResource	A handle to a resource.
offset	The beginning of the resource subsection to be read, measured in bytes from the beginning of the resource.
buffer	A pointer to the buffer into which the partial resource is to be read.
count	The length of the resource subsection.

Description

The ReadPartialResource procedure reads the resource subsection identified by the theResource, offset, and count parameters into a buffer specified by the buffer parameter. Your application is responsible for the buffer's memory management. You cannot use the ReleaseResource procedure to release the memory the buffer occupies.

The ReadPartialResource procedure always tries to read resources from disk. If the handle in the parameter theResource doesn't refer to a resource in an open resource fork, ResError returns the result code resNotFound.

When using partial resource routines, you should call the SetResLoad procedure, specifying FALSE for the load parameter, before you call GetResource. Using the SetResLoad procedure prevents the Resource Manager from reading the entire resource into memory. Be sure to restore the normal state by calling SetResLoad again, with the load parameter set to TRUE, immediately after you call GetResource. Then use ReadPartialResource to read a portion of the resource into a buffer.

Note:
If the entire resource is in memory and you want only part of its data, it's faster to use the Memory Manager procedure BlockMove instead of the ReadPartialResource procedure. If you read a partial resource into memory and then change its size, you can use SetResourceSize to change the entire resource's size on disk as necessary.

Special Considerations

The ReadPartialResource procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

Assembly-Language Information

The trap macro and routine selector for ReadPartialResource are:

Trap macro	Selector
_ResourceDispatch	$7001

Result Codes		
noErr	0	No error
resNotFound	-192	Resource not found

Back to top

WritePartialResource

You can use the WritePartialResource procedure to write part of a resource to disk when working with a small subsection of a large resource.

```
PROCEDURE WritePartialResource (theResource: Handle;
                                offset: LongInt; buffer: UNIV Ptr;
```

theResource	A handle to a resource.
offset	The beginning of the resource subsection to write, measured in bytes from the beginning of the resource.
buffer	A pointer to the buffer containing the data to write.
count	The length of the resource subsection to write.

Description

The WritePartialResource procedure writes the data specified by the buffer parameter to the resource subsection identified by the theResource, offset, and count parameters. Your application is responsible for the buffer's memory management.

If the disk or the file is locked, the ResError function returns an appropriate File Manager result code.

The WritePartialResource procedure tries to write the data from the buffer to disk. If the attempt is successful and the resource data (referenced through the resource's handle) is in memory, be aware that the data of the resource subsection on disk matches the data from the buffer, not the resource data referenced through the resource's handle. If the attempt to write the data from the buffer to the disk fails, ResError returns an appropriate error.

If the handle in the parameter theResource does not refer to a resource in an open resource fork, ResError returns the result code resNotFound.

The WritePartialResource procedure checks that the information in the resource map is internally consistent. If it isn't, the ResError function returns the result code mapReadErr.

When using partial resource routines, you should call the SetResLoad procedure, specifying FALSE for the load parameter, before you call GetResource. Doing so prevents the Resource Manager from reading the entire resource into memory. Be sure to restore the normal state by calling SetResLoad again, with the load parameter set to TRUE, immediately after you call GetResource.

Note:
If you read a partial resource into memory and then change its size, you must use SetResourceSize to change the entire resource's size on disk as necessary before you write the partial resource.

Special Considerations

The WritePartialResource procedure may move or purge memory blocks in the application heap. Your application should not call this procedure at interrupt time.

Assembly-Language Information

The trap macro and routine selector for WritePartialResource are:

Trap macro	Selector_ResourceDispatch	$7002

Result Codes

noErr	0
No errordskFulErr	-34
Disk fullresNotFound	-192
Resource not foundmapReadErr	-199
Map inconsistent with operation	

Back to top

References

Inside Macintosh: More Macintosh Toolbox

Technical Note TB 555- Resource Manager Q&As

Back to top

Downloadables

Acrobat version of this Note (K). Download

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code