Technote 1120

Opening Resource Files Twice Considered Hard?

CONTENTS

Most Mac OS programmers do not consider `FSpOpenResFile` (or it's older cousins `OpenResFile`, `OpenRFPerm`, and `HOpenResFile`) to be difficult to call. However, the behavior of these calls when the resource file is already open has never been documented properly.

This Note describes the exact behavior of `FSpOpenResFile` when the resource file is already open, and describes some cookbook solutions to avoid problems in this case.

All Mac OS programmers who use the Resource Manager should read the Summary section of this Note, just to familiarize themselves with the problem. In addition, programmers who are writing non-application code should carefully read the entire Note to ensure maximum compatibility for their code.

Describing the Problem

Inside Macintosh: More Macintosh Toolbox (p1-60) has the following to say about the behavior of `FSpOpenResFile` when the resource file is already open:

> If you attempt to use `FSpOpenResFile` to open a resource fork that is already open, `FSpOpenResFile` returns the existing file reference number or a new one, depending on the access permission for the existing access path. For example, your application receives a new file reference number after a successful request for read-only access to a file previously opened with write access, whereas it receives the same file reference number in response to a second request for write access to the same file. In this case, `FSpOpenResFile` doesn't make that file the current resource file.

This statement is mostly true, but it certainly is not the whole truth. The exact situation is a lot more complex.

Complicating Factors

The complicating factors when opening a resource file include:

- the exact call used to open the file, i.e. `OpenResFile`, `OpenRFPerm`, `HOpenResFile`, and `FSpOpenResFile`,
- the permissions supplied to the call,
- whether the resource file is already open,
- if the resource file is already open (whether by this process or by another)
- if the resource file is already open by another process (whether this process is on the local machine or on another machine accessing the file via File Sharing)

- if the resource file is already open (the permissions previously used to open the file).

These complicating factors combine to make opening a resource file potentially much more complicated than it seems on first examination.

Simplifying the Problem

Fortunately, not all of the above factors actually complicate the problem. We can simplify the analysis by noting the following:

- `OpenResFile`, `OpenRFPerm`, `HOpenResFile`, and `FSpOpenResFile` all behave in the same way. While you might think this is obvious, DTS engineers take nothing for granted and I hope you feel reassured to know that I actually tested this.

NOTE:
`OpenResFile` does not take a permission parameter. When you call `OpenResFile`, it acts as if you had supplied the permission `fsCurPerm`.

- There are only three outcomes from calling `FSpOpenResFile`:
    1. **Complete Success** -- A new resource map is created and added to the top of the resource chain. The routine makes the new resource map the current resource map and returns its resource file reference number.
    2. **Partial Success** -- The resource file reference number of an existing resource map is returned. That existing map will become the current resource map.
    3. **Failure** -- `FSpOpenResFile` returns -1 and the resource chain remains unchanged. You can call `ResError` to get the real error number.

NOTE:
In the Partial Success outcome above, `CurResFile` is changed, which directly contradicts the last sentence of the quote from *Inside Macintosh: More Macintosh Toolbox* . This Technote is correct and Inside Macintosh is in error.

- In all cases, asking for `fsWrPerm` and asking for `fsRdWrPerm` have the same effect.

These points combine to make it much easier to describe the exact situation. What remains is to describe the behavior in each of the remaining cases.

`FSpOpenResFile` Explained!

The following table show the exact behavior of `FSpOpenResFile` in the various interesting cases.

Permissions Used to Open First	Permissions Used to Open Second	Same Process	Same Machine	Different Machines
`fsCurPerm`	`fsCurPerm`	Success R-O	Success R-O	Failure -54
`fsCurPerm`	`fsRdPerm`	Success R-O	Success R-O	Failure -54
`fsCurPerm`	`fsWrPerm` `fsRdWrPerm`	Partial R/W	Failure -49	Failure -54
`fsRdPerm`	`fsCurPerm`	Success R-O	Success R-O	Success R-O
`fsRdPerm`	`fsRdPerm`	Success R-O	Success R-O	Success R-O
`fsRdPerm`	`fsWrPerm` `fsRdWrPerm`	Partial R-O	Failure -49	Success R-O
`fsWrPerm` `fsRdWrPerm`	`fsCurPerm`	Success R-O	Success R-O	Failure -54
`fsWrPerm` `fsRdWrPerm`	`fsRdPerm`	Success R-O	Success R-O	Failure -54
`fsWrPerm` `fsRdWrPerm`	`fsWrPerm` `fsRdWrPerm`	Partial R/W	Failure -49	Failure -54

The first column gives the permission value used the first time the file was opened. The second column gives the permission value for this call to `FSpOpenResFile`.

The remaining three columns describe the behavior in each of the three important cases:

- **Same Process** -- The file has already been opened by the same process.
- **Same Machine** -- The file has already been opened by another process on the same machine.
- **Different Machines** -- The file has already been opened by another process on another machine connected via File Sharing.

Each cell is labelled with a pair of items. The first item is the outcome of the call, as defined above. The second item is either the permissions associated with the returned resource file reference number (for outcomes Success and Partial) or the error code returned (for outcome Failure).

Gotchas and Observations

There are a number of important observations to be made about the above table:

- Asking for write permissions to a file if it has already been opened by your process will yield Partial Success, i.e. `FSpOpenResFile` will return the resource file reference number of an existing resource map. If you're not prepared for this (and you close the resource map after using it, say), you will find yourself in a world of pain. For example, it's common for developers to accidentally close the resource map of their own application, or the system resource map.
- It is possible to get lesser permissions than you asked for. For example, in Same Process column in the table, if the file is already open with `fsRdPerm` and you open it with `fsRdWrPerm`, the resource file reference number returned has read-only permissions. You can check for this situation using the code shown below.
- Asking for `fsRdWrPerm` permissions to a file which has already been opened with `fsRdPerm` by another machine will succeed (!), but will give you a read-only resource file reference number.
- The exact error code you get back when a resource file is already opened by another process depends on whether that process is running on the local machine or not.

One Final Gotcha

The last problem with opening the same resource file twice is intrinsic to the design of the Resource Manager and very hard to guard against. When you open a resource file, the Resource Manager loads a catalog of all the resources (the **resource map**) into your heap, and uses that map to locate the data for each resource resides in the resource file.

When changing the resource map, the Resource Manager does not coordinate between the various processes that might have the resource file open. While the Resource Manager prevents you from opening two read/write resource file reference numbers to the same resource file, it does not stop you from having a read-only and a read/write resource file reference number simultaneously.

This can cause serious problems in the following situation:

1. Process A opens the resource file read/write. The Resource Manager reads the resource map for the file into Process's A's heap.
2. Process B opens the resource read-only. The Resource Manager reads the resource map for the file into Process's B's heap.
3. Process A then changes the resource file and calls `UpdateResFile` to write those changes back to the disk. This can cause the position of various resources in the resource file to change quite dramatically.
4. Process B then calls the Resource Manager to read a resource from the resource file. Because Process B's copy of the resource map no longer matches the actual file, the Resource Manager returns bogus resource data.

The restriction is described quite well in the Special Considerations section of the description of FSpOpenResFile in *Inside Macintosh: More Macintosh Toolbox* , but that description is worth reiterating while we're on the subject of opening resource files twice.

Cookbook Solutions

This section describes some useful techniques you can employ to ensure that the weirdnesses of `FSpOpenResFile` does not bite your software. These are listed with the most recommended (and easiest to implement) first.

Open Resource Files Once

If you're writing normal application-level code, it's easy to remember whether your process has already opened a resource file and avoid opening it twice. The following snippet shows a simple example of this.

```
static SInt16 gResourceResFile = 0;
static UInt32 gResourceUsageCount = 0;

static OSErr StartUsingResources(ConstFSSpecPtr fss)
{
    OSErr  err;
    SInt16 tmpResFile;

    err = noErr;
    if (gResourceUsageCount == 0) {
        tmpResFile = FSpOpenResFile(fss, fsRdWrPerm);
        err = ResError();
        if (err == noErr) {
            gResourceResFile = tmpResFile;
        }
    }
    if (err == noErr) {
        gResourceUsageCount += 1;
    }

    return err;
}

static void StopUsingResources(void)
{
    gResourceUsageCount -= 1;
    if (gResourceUsageCount == 0) {
        CloseResFile(gResourceResFile);
        gResourceResFile = 0;
    }
}
```

Extending this technique for more than one resource file is left as an exercise to the developer.

Open Resource Files Read Only

In situations where you don't know whether a resource file has already been opened by the current process (in system extension code, for example), the easiest solution is to always open the resource file read-only, i.e. using `fsRdPerm`. If you do this, you will always get a new resource reference number that you can safely close.

A further refinement of this solution is to open, read, and close the resource file quickly, without yielding time to other processes in between. This helps prevent another process from modifying the file while you're reading it, and minimizes your vulnerability to the trickiest gotcha described above. This refinement is only useful in certain situations, but is definitely one to keep in your 'cookbook'.

Check `TopMapHndl`

In situations where you don't know whether a resource file has already been opened by the current process and you must open the resource file read/write, the best technique is to monitor the `TopMapHndl` low memory global to see if it changes around your call to `FSpOpenResFile`. If this global changes, a new resource map has been added to the top of the resource chain, and you are responsible for closing it. If the global does not change, an existing resource map reference number was returned and you should not close it.

The following snippet illustrates this technique.

```
static void SafeOpenResFileReadWrite(ConstFSSpecPtr fss)
{
    OSErr   err;
    SInt16  oldResFile;
    Handle  oldTopMap;
    SInt16  resFile;
    Boolean shouldClose;

    oldResFile = CurResFile();

    oldTopMap = LMGetTopMapHndl();
    resFile = FSpOpenResFile(fss, fsRdWrPerm);
    err = ResError();

    if (err == noErr) {
        shouldClose = (LMGetTopMapHndl() != oldTopMap);

        // do the stuff with the resource file

        if (shouldClose) {
            CloseResFile(resFile);
        }
    }

    UseResFile(oldResFile);
}
```

It's important to remember that this technique is only necessary if you need to open the file read/write and you don't know whether the file is already open by the current process. As such, this technique is needed most by non-application code -- such as system extensions, shared libraries, application plug-ins, etc -- but it may also be useful for application code is running in strange environments, such as a Standard File filter function.

Always Preserve `CurResFile`

Regardless of which of above techniques you use, it's always a good idea to bracket your calls to `FSpOpenResFile` with calls to `CurResFile` and `UseResFile` to ensure that your code does not accidentally change the current resource map. The above snippets also illustrates this technique.

Check Permissions

If you need to write to a resource file and you are not sure whether that file has already been opened, it pays to examine the resource file reference number to ensure that it supports read/write access. While having a read/write resource file reference number is not a guarantee that writing to the file will succeed, it's a good idea to check this as the first step.

You can check whether a resource file reference number is read/write by calling the File Manager routine `PBGetFCBInfoSync` and looking at bit 8 of `ioFCBFlags`. The following snippet demonstrates this technique.

```
static Boolean IsResourceFileRefNumWritable(SInt16 rsrcRefNum)
{
    Boolean result;
    FCBPBRec fcbPB;

    fcbPB.ioNamePtr = nil;
    fcbPB.ioVRefNum = 0;
    fcbPB.ioRefNum = rsrcRefNum;
    fcbPB.ioFCBIndx = 0;
    if ( PBGetFCBInfoSync(&fcbPB) == noErr ) {
        result = ((fcbPB.ioFCBFlags & (1 << 8)) != 0);
    } else {
        result = false;
    }

    return result;
}
```

Summary

If you open the same resource file twice, you are vulnerable to a number of strange behaviors of the Resource Manager, including:

- `FSpOpenResFile` returning an existing resource file reference number rather than opening a new resource file reference number.
- `FSpOpenResFile` returning a read-only resource file reference number, even though you explicitly asked for read/write access.
- Possible corrupt resource data if you read a resource file through a read-only resource file reference number while simultaneously modifying it through another read/write resource file reference number.

The best way to guard against these problems is to avoid opening a resource file twice. If this is unavoidable, this Note suggests a number of approaches you can use to minimize your vulnerability.

Further References

- *Inside Macintosh: More Macintosh Toolbox* , Chapter 1 Resource Manager
- DTS Technote FL 37 You Want Permission to do What?!!

Downloadables

Acrobat version of this Note (K)

To contact us, please use the Contact Us page.
Updated: 30-January-98

Technotes
Previous Technote | Contents | Next Technote