

#101: CreateResFile and the Poor Man's Search Path

See also: The File Manager
 The Resource Manager
 Technical Note #77—HFS Ruminations

Written by: Jim Friedlander January 12, 1987
Updated: March 1, 1988

`CreateResFile` checks to see if a resource file with a given name exists, and if it does, returns a `dupFNErr` (–48) error. Unfortunately, to do this check, `CreateResFile` uses a call that follows the Poor Man's Search Path (PMSP).

`CreateResFile` checks to see if a resource file with a given name exists, and if it does, returns a `dupFNErr` (–48) error. Unfortunately, to do the check, `CreateResFile` calls `PBOpenRF`, which uses the Poor Man's Search Path (PMSP). For example, if we have a resource file in the System folder named 'MyFile' (and no file with that name in the current directory) and we call `CreateResFile('MyFile')`, `ResError` will return a `dupFNErr`, since `PBOpenRF` will search the current directory first, then search the blessed folder on the same volume. This makes it impossible to use `CreateResFile` to create the resource file 'MyFile' in the current directory if a file with the same name already exists in a directory that's in the PMSP.

To make sure that `CreateResFile` will create a resource file in the current directory whether or not a resource file with the same name already exists further down the PMSP, call `_Create` (`PBCreate` or `Create`) before calling `CreateResFile`:

```
err := Create('MyFile',0,myCreator,myType);
      {0 for VRefNum means current volume/directory}
CreateResFile('MyFile');
err := ResError; {check for error}
```

In MPW C:

```
err = Create("\pMyFile",0,myCreator,myType);
CreateResFile("\pMyFile");
err = ResError();
```

This works because `_Create` does **not** use the PMSP. If we already have 'MyFile' in the current directory, `_Create` will fail with a `dupFNErr`, then, if 'MyFile' has an empty resource fork, `CreateResFile` will write a resource map, otherwise, `CreateResFile` will return `dupFNErr`. If there is no file named 'MyFile' in the current directory, `_Create` will create one and then `CreateResFile` will write the resource map.

Notice that we are intentionally ignoring the error from `_Create`, since we are calling it only to assure that a file named 'MyFile' does exist in the current directory.

Please note that `SFPutFile` does **not** use the PMSP, but that `FSDelete` does. `SFPutFile` returns the `vRefNum`/`WDRRefNum` of the volume/folder that the user selected. If your program deletes a resource file before creating one with the same name based on information returned from `SFPutFile`, you can use the following strategy to avoid deleting the wrong file, that is, a file that is not in the directory specified by the `vRefNum`/`WDRRefNum` returned by `SFPutFile`, but in some other directory in the PMSP:

```

VAR
    wher    : Point;
    reply   : SFReply;
    err     : OSErr;
    oldVol  : Integer;

...

wher.h := 80; wher.v := 90;
SFPutFile(wher, '', '', NIL, reply);
IF reply.good THEN BEGIN
    err := GetVol(NIL, oldVol); {So we can restore it later}
    err := SetVol(NIL, reply.vRefNum); {for the CreateResFile call}

{Now for the Create/CreateResFile calls to create a resource file that
we know is in the current directory}

    err := Create(reply.fName, reply.vRefNum, myCreator, myType);
    CreateResFile(reply.fName); {we'll use the ResError from this ...}

CASE ResError OF
    noErr: {the create succeeded, go ahead and work with the new
           resource file -- NOTE: at this point, we don't know
           what's in the data fork of the file!!} ;
    dupFNErr: BEGIN {duplicate file name error}
        {the file already existed, so, let's delete it. We're now
        sure that we're deleting the file in the current directory}

        err:= FSDelete(reply.fName, reply.vRefNum);

        {now that we've deleted the file, let's create the new one,
        again, we know this will be in the current directory}

        err:= Create(reply.fName, reply.vRefNum, myCreator, myType);
        CreateResFile(reply.fName);
    END; {CASE dupFNErr}
    OTHERWISE {handle other errors} ;
END; {CASE ResError}
err := SetVol(NIL, oldVol); {restore the default directory}
END; {If reply.good}
...

```

In MPW C:

```
Point      wher;
SFReply    reply;
OSError    err;
short      oldVol;

wher.h = 80; wher.v = 90;
SFPutFile(wher, "", "", nil, &reply);
if (reply.good )
{
    err = GetVol(nil, &oldVol);
    /*So we can restore it later*/
    err = SetVol(nil, reply.vRefNum); /*for the CreateResFile call*/

    /*Now for the Create/CreateResFile calls to create a resource file
    that we know is in the current directory*/

    err = Create(&reply.fName, reply.vRefNum, myCreator, myType);
    CreateResFile(&reply.fName);
    /*we'll use the ResError from this ...*/

    switch (ResError())
    {
        case noErr: /*the create succeeded, go ahead and work with the
                     new resource file -- NOTE: at this point, we don't
                     know what's in the data fork of the file!!*/
            break; /* case noErr*/
        case dupFNErr: /*duplicate file name error*/
            /*the file already existed, so, let's delete it.
            We're now sure that we're deleting the file in the
            current directory*/

            err= FSDelete(&reply.fName, reply.vRefNum);

            /*now that we've deleted the file, let's create the
            new one, again, we know this will be in the current
            directory*/

            err= Create(&reply.fName, reply.vRefNum,
                       myCreator, myType);
            CreateResFile(&reply.fName);
            break; /*case dupFNErr*/
        default:; /*handle other errors*/
    } /* switch */
    err = SetVol(nil, oldVol); /*restore the default directory*/
} /*if reply.good*/
```

Note: OpenResFile uses the PMSP too, so you may have to adopt similar strategies to make sure that you are opening the desired resource file and not some other file further down the PMSP. This is normally not a problem if you use SFGetFile, since SFGetFile does not use the PMSP, in fact, SFGetFile does not open or close files, so it doesn't run into this problem.