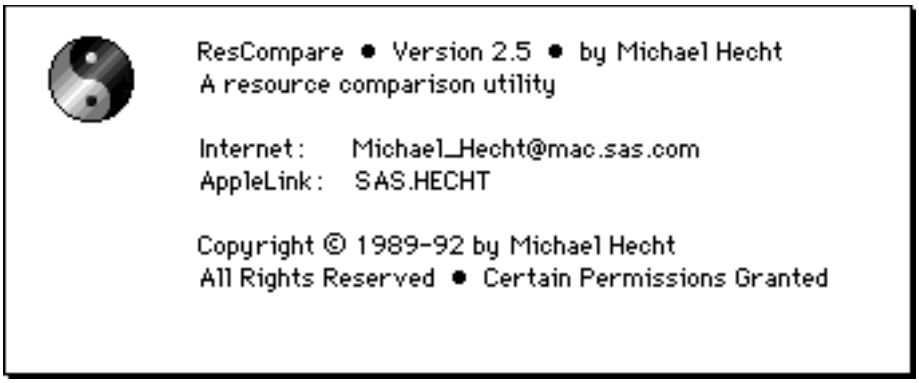
About ResCompare…



What is it?

ResCompare is a programmer's utility that compares resource files. Use it to

- find out what resources you changed while working at home late last night,
- find out what resources the rest of the team changed in the project you're all working on while you were on vacation,
- see what resources have changed between versions 1.0 and 1.0.1 of your favorite application,
- create a Patch application that will apply changes to applications and other files, which you can use to distribute upgrades of your software without sending a new copy of the entire application,
- or just get a quick list of the resources in any file.

Legal Stuff

You should use this program with care. If you don't understand what a resource is for, learn first; then use this program. As with ResEdit, you can do **irreparable damage** to a resource file by changing resources that shouldn't be changed. This software is a powerful tool; use it with caution. This software comes with no warranty and I am not liable if you damage your resource

file with it. Although I have tested the software on several configurations and believe it to be bug–free (or at least bug–light), you, the user, are responsible for any crashes or equipment failure resulting directly or indirectly from its use. Make lots of backups!

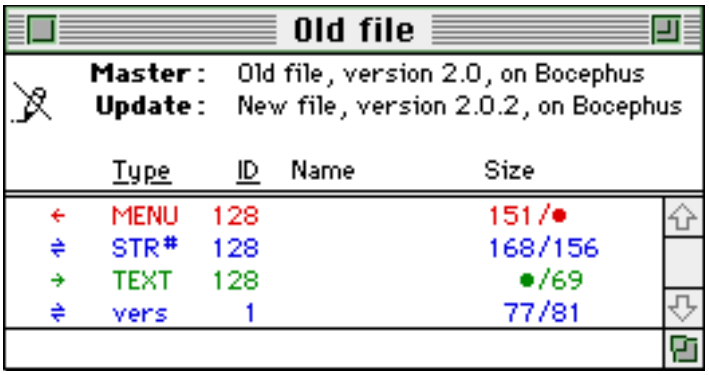
How do I use it?

ResCompare is easy to use. Just start it up and choose **Open…** from the **File** menu. Pick a resource file to look at and ResCompare gives you a list of all the resources within that file. This list is pretty handy because it's sorted by resource type and ID, and it also contains the name and size of each resource. You can print out the entire list, or you can select some resources and copy the formatted list of information to the clipboard.

But the real fun starts when you choose the **Compare with…** command (⌘K) from the **File** menu. This command lets you open a second resource file, and it compares each resource in the first file (the *master* file) with the resource having the same type and ID in the second file (the *update* file).

You can open the update file while the master file is being read. This generally speeds up the comparison process for larger resource files. Under System 7, you can drag any resource file onto ResCompare. (*Note*: If you can't drag files onto ResCompare, try rebuilding your Desktop by restarting with the *option* and *command* keys held down; or, use Mike Engber's free Save a BNDL utility.) If the front most window is a resource list that hasn't been compared to anything yet, ResCompare will compare the new file to it. Otherwise, ResCompare will open the new file.

After comparing, the resources that are identical are removed from the list (it doesn't alter the files). All resources that were deleted from the master file are shown in red, with the "deleted" icon (
) on the left margin. Any resources that were added to the update file are shown in green, and get the "added" icon (
). Any resources whose contents, attributes or names have changed get the "changed" icon (
), and are shown in blue. Here's what a window looks like after comparing two resource files:



You can select resources from this list by clicking them. Use the *shift* key to extend your selection and the *command* key to form a discontinuous selection. You can also type a resource's type and ID to scroll to the closest match and select it. Use **Select All** (⌘A) from the **Edit** menu to select all resources at once.

At the top of the window, the names of the master and update files are shown, along with their version numbers and the names of the volumes they reside on. In the Size column, you can see the “before/after” sizes of each resource. The little “no–write” icon () is next to the update file to remind you that it cannot be changed. However, the master file doesn’t have a no–write icon. You can change its resources using the **Update** command in the **Change** menu.

If you select one or more resources in the list and choose **Update** (⌘U), the update is applied to the selected resources. What does that mean? It means that if a resource was added to the update file, it gets added to the master file. A resource that was deleted gets deleted from the master file. A changed resource is copied from the update file to the master file, along with its attributes and resource name. In other words, the selected resources are changed in the master file so that they are identical to the update file. After updating, ResCompare removes the selected resources from the list (since they’re now identical).

If you click the 
icon, ResCompare swaps the master and update files. This is useful if you want to get some changed resources into your master file, and also put some resources you’ve changed into the update file.

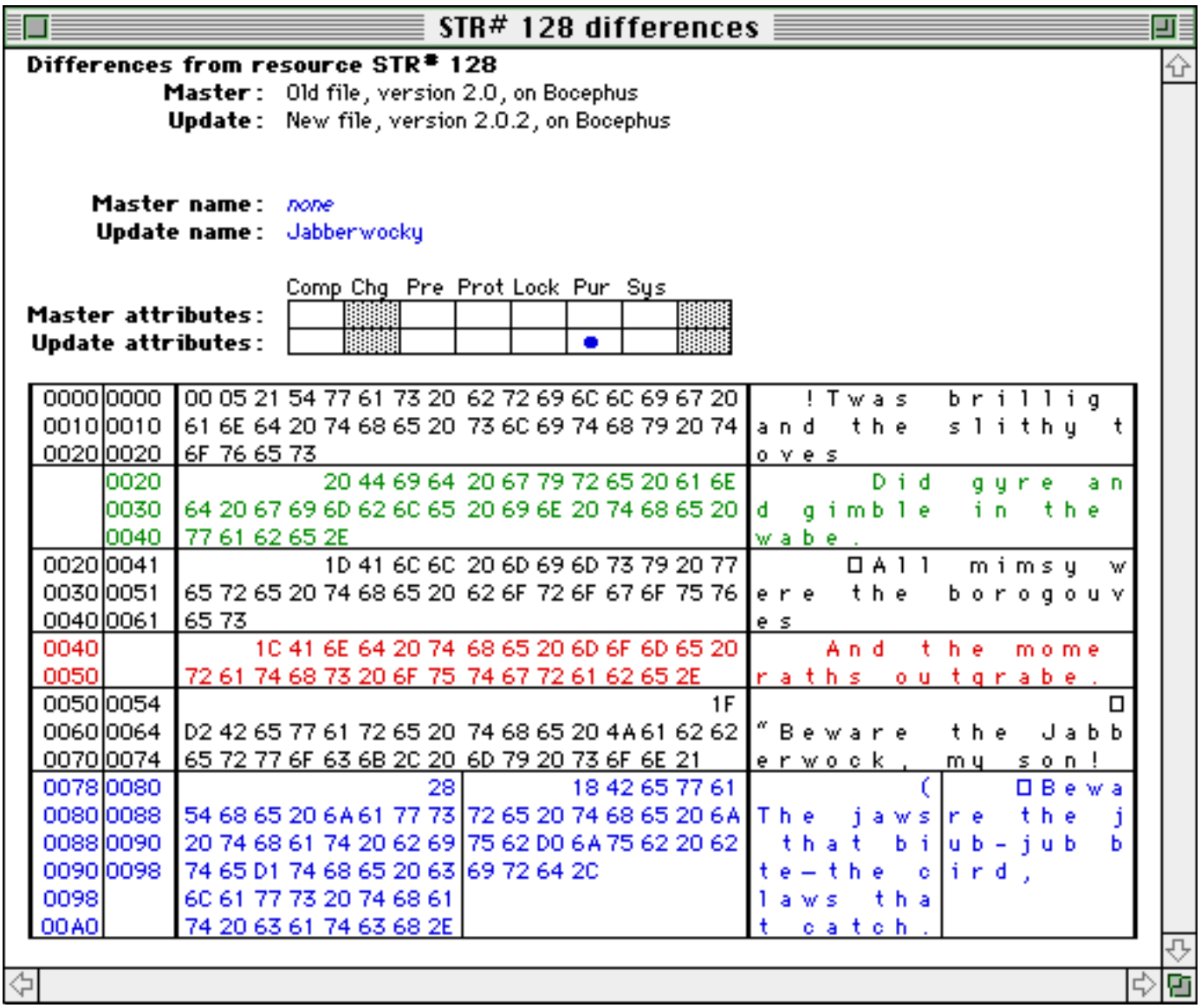
You can also use the **Font** and **Size** menus in the usual manner to adjust the font used. You should probably switch to Helvetica before printing to a LaserWriter.

Editing Resources

Select one or more resources and choose **Edit resources** (⌘E) from the **Edit** menu. This command opens the master and update resource files in a resource editor (either ResEdit or Resorceror™) and displays the selected resources. If the resource editor isn’t running, ResCompare will launch it. Use the **Choose editor…** command from the **Edit** menu to select your preferred resource editor. These commands only work under System 7.

Viewing Resource Differences

ResCompare can show you the exact changes to a resource. After comparing files, select a resource and choose **Differences** from the **Change** menu. It may take ResCompare a few moments to compute the resource differences; but when it’s done, ResCompare will display a window like this.



The top portion of the window displays any differences found in the resource name and attributes. Differences in the resource contents are displayed in a hexadecimal format. Identical, inserted, deleted, and changed sections are shown in their respective colors. If you aren't using a color monitor, you can refer to the byte offsets listed in the two left–hand columns. The left column is the offset within the master file, and the right column is the offset within the update file. A section has been deleted if it has no update offset, and it has been added if it has no master offset. If the section has both offsets, it has been changed if the hex dump is split into two sections. Otherwise, the section is identical in both files. The same information is also displayed in an ASCII dump in the right–hand column.

ResCompare will also show differences if you double–click a resource or select a resource and press *return*. You can use the **Change** menu's **Hide same parts** command to suppress the display of the identical (black) parts. You can also select any portion of the differences and copy it to the clipboard as text.

Make Patch…

ResCompare can be used to create a self–applying patch application. You can use the patch to distribute upgrades of your software without sending a new copy of the entire application. Patches are typically 40% of the original file, so you can save disk space and communications upload/download time by sending the patch instead of the entire application. Also, the patch is

only useful to owners of the original version, so you can distribute patches to your commercial software over public bulletin boards.

How to Apply a Patch

Patches are easy to apply. Just open a self–applying patch and it displays some notes, then prompts you for the file to patch. A bar graph displays the progress. When done, it tells you if it was successful. It's that simple—a real no–brainer.

Creating Self–Applying Patches

Self–applying patches are created with ResCompare's **Make Patch…** command. Select the resources that you want included in the patch and choose **Make Patch**. ResCompare asks if you want to create a new patch or append to an existing patch. If you're making a new patch, click **New**. A second dialog lets you set various patch options.

- You can choose the prompt that the patch will use when asking for the original file. By default, the prompt will be "Where is '*original file*?", where *original file* is the name of your master file.

- You can also choose whether you want the original file to be patched in place, or whether you want the patch application to make a copy first. If you would prefer the patch to make a copy, you can choose a default name for the copy. After opening the original file, the patch will prompt the user to save a copy. If the user replaces the original file, the patch application does not make a copy, but patches the original file in place anyway.

When you click **OK**, ResCompare asks you to save the patch. It then finds the differences between the selected resources and saves them in the self–applying patch.

One Patch for Many Versions

If you have released several versions of your application, you can create a patch that will work with all released versions. I'll go through an example that describes the process. Let's say you have three released versions of your application named Sample: version 1.0, version 1.0.1 and version 1.2. You want to release version 1.3 as a self–applying patch that will upgrade all existing versions.

- Create the patch "Sample 1.3 patch," as described above, that patches version 1.0 to 1.0.1.

- Using ResCompare, compare version 1.0.1 to version 1.2. Select the changed resources and choose **Make Patch…** When asked if you want to create a new patch, click the **Append** button and select "Sample 1.3 patch," which you created in step 1. After opening the patch, you will have an opportunity to adjust the patch options.

- Repeat step 2 for the changes from version 1.2 to 1.3. You now have a self–applying patch that can upgrade versions 1.0, 1.0.1 and 1.2 of Sample to version 1.3.

An easy way to do this incrementally is:

- Create the first patch for the first upgrade.

- When the time comes to release the second upgrade, duplicate the patch for the first upgrade, rename it, and append the new patch information to it.

- For each subsequent upgrade, always duplicate and rename the last patch you created, and append your changes to it.

One Patch for Many Files

A single patch can also contain the information needed to patch multiple files. Simply compare each additional file and append the patch information, as describe above. The patch options can be set individually for each file you want patched.

Adding Notes to the Patch

Open the patch with a resource editor and edit TEXT/styl resource 128. This resource contains the patch notes that are displayed when the patch is first opened. You can add your own notes here if you like. Remember, leave the paragraph that begins “This patch application was created…” at the end of the notes. Do not alter this paragraph.

Localizing a Patch

You can localize an individual patch by changing its dialog, alert, and string resources. If you would like to create a version of ResCompare that will always generate your localized self–applying patch, follow these steps:

- Make a small patch and localize all text, except for the resources 'ZVER', 'ZAP#', 'ZIL#', 'ZIS#', and 'ZAP '. You may localize the default patch notes, including the ResCompare notice at the end. Just don’t remove the notice.

- Open the patch in ResCompare and select all resources except those listed above.

- Choose the **Encapsulate…** command from the **File** menu. This command creates a new resource file and saves the selected resources into it, stripped of their resource types, IDs, and attributes. All selected resources are written to the encapsulated file with the type 'RES '. Also, a resource of type 'RES#' is written to the file, which contains a mapping of each encapsulated resource to its original resource. ResCompare contains a ResEdit template for the 'RES#' resource type, so you can view its contents. See Appendix 1 for a source code listing that will expand an encapsulated resource file back into its original form.

- Make a copy of ResCompare and open it and the encapsulated patch in ResEdit.

- Select all 'RES ' and 'RES#' resources from the ResCompare copy and delete them. Select all 'RES ' and 'RES#' resources from the encapsulated patch and copy them to ResCompare. Close and save the ResCompare copy.

ResCompare will now use your localized version each time you create a new patch.

Limitations

Always patch a copy of your application, not the original. Never patch your master disks. Use

the “patch a copy” option if your users might not follow this advice.

ResCompare allows you to append patches whose version numbers are not in strict ascending order, but it warns you when it detects this situation. Be sure to test such patches thoroughly, on all intended versions of your application, since this disables version checking.

Self–applying patches work on applications the same way that some viruses do (by modifying their CODE resources). Because of this, some virus detectors may get upset when a patch starts doing its thing. Your best bet is to turn off virus protectors and restart before patching. Because of this limitation, **please remember to check your floppies for viruses** before distributing a patch on them.

The self–applying patch requires some free space on the volume, and it makes every possible attempt to apply the patch within the space provided. However, I have had some reports of problems patching files on a nearly full floppy. I therefore suggest copying the file to your hard disk before patching.

Revision History

Version 2.5

- New features: difference windows, encapsulation, editor support.
- You can use the page up, page down, home, and end keys to scroll a resource list window.
- Comparing and differencing have been sped up dramatically.
- Resources are now sorted alphabetically in a case-insensitive fashion, identical to ResEdit. That is, 'acur' now sorts before 'ALRT'.
- The **Other…** command on the **Font** menu now does something. **Up/Down one point** have been removed, in accordance with Apple’s Human Interface guidelines.
- ⌘P now means **Print**, not **Make Patch**.
- The error messages have been improved.
- You can now create patches for files where the version numbers are out of order.
- The Patch Notes window now has a **Copy** button for copying the notes to the clipboard.
- Patch files can contain “optional” resources, which are only patched if they verify correctly. Otherwise, they’re ignored. Sorry, no ResCompare user interface for setting the “needNotVerify” bit yet—use ResEdit.
- The “missing”sizes for added or deleted resources are now represented by a bullet (•) instead of an ellipsis (…).
- You can now start comparing the update file while the master file is being read. This will dramatically improve the speed of comparison, since identical resources are removed from the resource list almost as soon as they get added. Keeping the resource list small in this manner greatly reduces the time needed to build it.
- Printing a resource list from the Finder now works (I think).
- ResCompare now has a preferences file which contains an alias to your preferred resource editor application.
- Resources no longer appear out-of-order in the resource list.
- Version numbers with components greater than 8 no longer appear negative.
- ResCompare no longer changes the modified date of a file it’s comparing.
- **Open** would sometimes crash under System 7 in the presence of SuperBoomerang 3.0.1 or DFaultD 2.22. This was caused by bugs in these extensions. The latest version of

SuperBoomerang is 3.0.2, which fixes its bug. I don't know of a version of DFaultD that fixes its problem. For those who don't have the latest versions of these extensions, I placed a work around kludge into ResCompare that prevents it from crashing. I will remove the kludge when fixed versions of these extensions appear.

- Compressed resources now compare correctly. ResCompare warns you if you ask it to show differences between a compressed and an uncompressed resource.
- Patch can now delete protected resources. It also no longer strips resource attributes, and it now allows you to delete zero-sized resources without complaint.
- ResCompare plays a custom sound when it finishes a task in the background.

Not Yet Implemented

- Saving resource lists and differences as text files.
- Printing difference windows.
- Changing the sorting order of a resource list. Currently, it's always Type/ID.
- Comparing data forks.
- ⌘-. to interrupt Encapsulate and Update commands.
- Cascading and tiling windows.
- Add interface for setting individual resource patching options: needNotVerify, etc.
- Improve differencing routine using the Myers, et. al. O(*PD*) algorithm.
- Saving fonts, view, hide same parts, window locations, etc. in ResCompare Preferences.

Known Bugs

- Despite my best efforts, resources with the "System" attribute get loaded into the system heap.

To contact me…

I hope you enjoy ResCompare. If you like it and find it useful, let me hear from you. If you have any suggestions, questions, or bug reports, send me e–mail at:

Internet: Michael_Hecht@mac.sas.com
AppleLink: SAS.HECHT

Cheers!
— Michael Hecht

Appendix 1

The following code snippet will convert an encapsulated resource file back into its original form. Encapsulation is a useful technique for embedding one resource file within another. You can encapsulate selected resources from a file using the **Encapsulate…** command on ResCompare's **File** menu.

```
/* Main routine; place this in an H file */
OSErr MakeResourceFile( FSSpecPtr spec, short mapListID );

/* The following structs define the 'RES#' resource. */
#define ResMapListType        'RES#'
#define ResStoredType         'RES '
```

```
typedef struct {
      short             resID;
      short             resAttrs;
} ResMap;

typedef struct {
      ResType           realType;

      short             numIDs;
      ResMap            realRes[];
} ResTypeMap, *ResTypeMapPtr;

typedef struct {
      ResType           fdType, fdCreator;
      short             fdFlags;
      ResType           storedType;
      short             storedFirstID;

      short             numTypes;
      ResTypeMap        typeMap[];
} ResMapList, **ResMapListHandle;


/* A useful struct for keeping track of a resource's information */
typedef struct {
      Handle            resHandle;
      ResType           resType;
      short             resID;
      short             resAttrs;
      long              resSize;
      Str255            resName;
} ResInfo, *ResInfoPtr;


/* Get all available information about a resource */
static OSErr GetResourceInfo( ResInfoPtr info )
{
      OSErr             err;


      /* Get its ID and name */
      GetResInfo( info->resHandle, &info->resID, &info->resType,
                        info->resName );
      err = ResError();
      if( err != noErr )
            return err;

      /* Get its attributes */
      info->resAttrs = GetResAttrs( info->resHandle );
      info->resAttrs &= 0x00FF;     /* High-order byte is ignored */
      err = ResError();
      if( err != noErr )
            return err;

      info->resSize = SizeResource( info->resHandle );
      if( info->resSize < 0 )
            return ResError();
```

```
	return noErr;
}


/* Load a resource into the application heap with no attributes */
static OSErr LoadRes( ResInfoPtr info )
{
	OSErr		err, loadErr;


	/* Clear resource attributes */
	SetResAttrs( info->resHandle, 0 );
	err = ResError();
	if( err != noErr )
		return err;

	/* Load it (non-purgeable and unlocked) into application heap */
	LoadResource( info->resHandle );
	loadErr = ResError();

	/* Restore attributes in resource map */
	SetResAttrs( info->resHandle, info->resAttrs );
	err = ResError();
	if( err != noErr )
		return err;

	/* Return any errors encountered while loading
		(out-of-memory, etc.) */
	return loadErr;
}


/* Create a resource as described by the given information
	(resSize is ignored) */
static OSErr CreateResource( ResInfoPtr info )
{
	OSErr		err;
	Handle		resHandle;


	/* Grab a copy of the resource */
	resHandle = info->resHandle;

	/* Does this handle already belong to a resource? */
	if( HomeResFile( resHandle ) != -1 ) {

		/* Load the resource data */
		err = LoadRes( info );
		if( err != noErr )
			return err;

		/* Duplicate it */
		err = HandToHand( &resHandle );
		if( err != noErr )
			return err;

		/* Throw out the real one */
		EmptyHandle( info->resHandle );
	}
```

```
	/* Add it to the output file */
	AddResource( resHandle, info->resType, info->resID,
			info->resName );
	err = ResError();
	if( err != noErr )
		return err;

	/* Write it out */
	WriteResource( resHandle );
	err = ResError();
	if( err != noErr )
		return err;

	/* Change its attributes */
	SetResAttrs( resHandle, info->resAttrs );
	err = ResError();
	if( err != noErr )
		return err;

	/* Release memory */
	ReleaseResource( resHandle );

	return noErr;
}


static OSErr OpenNewResFile( FSSpecPtr spec,
		ResType fdType, ResType fdCreator,
		Boolean dupOK, short *refNum )
{
	OSErr		err;


	/* Create output file */
	err = HCreate( spec->vRefNum, spec->parID, spec->name, fdCreator,
			fdType );
	if( err != noErr ) {

		/* All errors except duplicate files indicate failure */
		if( err != dupFNErr )
			return err;

		/* Are duplicates allowed? */
		if( !dupOK ) {

			/* It already exists, but duplicates are not allowed.
				Delete it... */
			err = HDelete( spec->vRefNum, spec->parID,
					spec->name );
			if( err != noErr )
				return err;

			/* ...then try again--this time, with no recourse */
			err = HCreate( spec->vRefNum, spec->parID, spec->name,
					fdCreator, fdType );
			if( err != noErr )
				return err;
		}
```

```
	}

	/* Add a resource fork to the file if it doesn't
		already have one */
	HCreateResFile( spec->vRefNum, spec->parID, spec->name );
	err = ResError();
	if( err != noErr && ( err != dupFNErr || !dupOK ))
		return err;

	/* Open output file */
	SetResLoad( FALSE );
	*refNum = HOpenResFile( spec->vRefNum, spec->parID, spec->name,
				fsRdWrPerm );

	/*
	 *	OpenResFile might return an invalid refNum even though
	 *	ResError says there's no problem.
	 */
	err = ResError();
	if( *refNum == -1 && err == noErr )
		err = paramErr;

	SetResLoad( TRUE );

	return err;
}

/* Recreate the input file from the given ResMapList */
OSErr MakeResourceFile( FSSpecPtr spec, short mapListID )
{
	OSErr			err;
	short			curRefNum, outRefNum;
	ResMapListHandle	theMapList;
	ResTypeMapPtr		theTypeMap;
	long			offset;
	short			iType, iRes, nRes;
	ResType			storedType;
	short			storedID;
	ResType			realType;
	ResInfo			info;
	FInfo			fndrInfo;


	/* Save our place */
	curRefNum = CurResFile();

	/* Get resource map list */
	theMapList = ( ResMapListHandle )GetResource( ResMapListType,
							mapListID );
	if( !theMapList ) {
		err = ResError();
		return err == noErr ? resNotFound : err;
	}

	/* Move it out of the way */
	HLockHi(( Handle )theMapList );

	/* Create/open the resource file */
	err = OpenNewResFile( spec, ( *theMapList )->fdType,
```

```
                        ( *theMapList )->fdCreator,
                              FALSE, &outRefNum );
if( err != noErr )
      return err;

/* Switch back to starting point */
UseResFile( curRefNum );

/* Set Finder flags */
HGetFInfo( spec->vRefNum, spec->parID, spec->name, &fndrInfo );
fndrInfo.fdFlags = ( *theMapList )->fdFlags;
HSetFInfo( spec->vRefNum, spec->parID, spec->name, &fndrInfo );

/* Get stored type and starting id */
storedType = ( *theMapList )->storedType;
storedID = ( *theMapList )->storedFirstID;

/* Dereference the resource map list */
theTypeMap = ( *theMapList )->typeMap;

/* For all types do */
for( iType = 0; iType < ( *theMapList )->numTypes; iType++ ) {

      realType = theTypeMap->realType;
      nRes = theTypeMap->numIDs;

      for( iRes = 0; iRes < nRes; iRes++ ) {

            /* Access the stored resource */
            SetResLoad( FALSE );
            info.resHandle =
                  GetResource( storedType, storedID++ );
            SetResLoad( TRUE );

            /* Handle errors */
            if( !info.resHandle ) {
                  err = ResError();
                  if( err == noErr )
                        err = resNotFound;
                  goto exit;
            }

            /* Get its information */
            err = GetResourceInfo( &info );
            if( err != noErr )
                  goto exit;

            /* Change the type and ID */
            info.resType      = realType;
            info.resID        = theTypeMap->realRes[ iRes ].resID;
            info.resAttrs     =
                  theTypeMap->realRes[ iRes ].resAttrs;

            /* Write it back out */
            UseResFile( outRefNum );
            err = CreateResource( &info );
            if( err != noErr )
                  goto exit;
            UseResFile( curRefNum );
```

```
            }

            /* Move to next type map */
            theTypeMap = ( ResTypeMapPtr )(( Ptr )theTypeMap +
                                           sizeof( ResTypeMap ) +
                                           nRes * sizeof( ResMap ));
      }

exit:
      HUnlock(( Handle )theMapList );

      /* Leave us in the newly-created resource file */
      UseResFile( outRefNum );
      return err;
}
```