Patch 4.0.4 File Format

Michael Hecht
AppleLink: SAS.HECHT
Internet: Michael_Hecht@mac.sas.com
April 8, 1993

This document describes the format of the resources contained in version 4.0.4 of a ResCompare self–applying patch.

To edit a patch file, use ResEdit with the supplied ResEdit `Patch templates` file. Simply open this file in ResEdit, along with a patch file, to provide editing templates for the patch resources.

'ZVER' resource

A self-applying patch contains patches for any number of versions of any number of files. There is a 'ZVER' resource for each file that is to be updated by the patch. The resource name is the prompt to be used when asking the user for this file.

NOTE The resource IDs for the 'ZVER' resource are not important and are randomly generated. Use ResEdit's "View by Order in File" command to list these resources in the order they were appended to the patch, from bottom to top.

The 'ZVER' resource contains some header information describing the file to be patched, followed by a list of valid versions. Here is the complete structure of the 'ZVER' resource:

```
/* Information maintained for each file version */

struct {
      ZVerFlags         flags;
      OSType            fdType, fdCreator;
      Size              finalRsrcSize;
      Str31             copyName;
      short             numVersions;

      struct {
            unsigned long     fromVersion;
            unsigned long     toVersion;
            short             zapListID;
            long              totalMunges;
      }                 theZapRef[];
} ZapVersion;
```

`flags`
A bitmask of flags that can be OR-ed together, with the following definitions:

```
/* Zap version flags */
enum {
      kPatchACopy = 0x0001
} ZVerFlags;
```

> `kPatchACopy`
> If set, this flag indicates that the patch application should prompt the user to save a copy of the target file. The patch can provide a default name for the copy. If the user chooses to save the copy over the original, the patch is performed in–place, as though this flag were not set.

`fdType, fdCreator`
These fields are the type and creator of the target file. Only files matching this criteria are listed in the GetFile dialog. You can remove the creator criteria by setting the creator to 0.

`finalRsrcSize`
Final size in bytes of the file's resource fork after patching. This field is used to determine if disk space is available to perform the patch.

`copyName`
If the `kPatchACopy` bit is set in the `flags`, this field is used as the default name for the copy of the target file.

`numVersions`
The number of version records to follow.

`theZapRef[]`
A list of version records. This list is searched to locate a The version records use the following format:

`fromVersion`
The 4–byte version code this patch converts from.

`toVersion`
The 4–byte version code this patch converts to.

`zapListID`
The resource ID of the 'ZAP#' resource for this patch.

`totalMunges`
The total of all the `numMunges` fields in the corresponding 'ZAP#'. This info is used to calculate the goal for the status thermometer.

'ZFIL' resource

Use a 'ZFIL' resource to automatically identify a file's location without prompting the user. The file may be in a folder locatable by `FindFolder`. The resource ID of the 'ZFIL' resource must match the 'ZVER' resource ID. The file must also have the same `fdType` and `fdCreator` as listed in the 'ZVER' resource.

```
/* Will have the same ID as its corresponding ZVER */


typedef struct {
      OSType            folderType;
      short             numFileLocs;

      struct {
            short             region;
            Str31             name;
      }                 theFileLoc[];
} ZapFileLocList;
```

`folderType`
A four–character type code recognized by `FindFolder`. The patch will search this folder on the system disk.

Set the `folderType` to a longword binary 0 (zero) to search for the target file by its full or partial pathname. Partial pathnames are resolved relative to the default volume and directory. The initial default volume and directory is the root directory of the system disk. To set `folderType` to zero with ResEdit, open an existing 'ZFIL' resource with the option key held down and change the first four bytes using the hex editor.

Set the `folderType` to a longword binary 1 to have the same effect as 0, but to reset the default volume and directory to its initial location first.

`numFileLocs`
The number of possible file names to follow.

`theFileLoc[]`
A list of file names. A list is used so multiple international spellings can be included. Each file name is accompanied by its region code, and the name corresponding to the current language of the system script is used.

`region`
The region code that identifies this name, or the constant `kAnyRegion` (–1) to match any region. Region codes are listed in *Inside Macintosh, Volume VI*, p. 14–84, table 14–10, in the <Packages.h> header file, and in ResEdit's "Country Code" pop–up menu located in the 'vers' editor.

`name`

The file name to look for, localized for this language.

The patch will scan the file loc list looking for a region code that matches the region of the system script, or for the wildcard value `kAnyRegion` (–1).

If a file name cannot be found for the current region, the folder cannot be found, the requested file does not exist, or the file exists but its type and creator don't match, the patch will prompt the user for the file. If the patch was able to determine a file name, it places the name as dialog substitution parameter "^0", so you can use a prompt string of "Where is ^0?" to include the localized name in the prompt.

You can also provide a partial pathname, such as ":MyApp Folder:MyApp" to search a folder at the root level of the system disk named "MyApp Folder" for the "MyApp" application.

'ZAP#' resource

The 'ZAP#' resource contains a list of all resources that need to be patched, for a given version transition of a given file. The 'ZVER' contains a list of 'ZAP#' resource IDs. The resource name of the 'ZAP#' will be the name of the master file from which the patch was created. This name is not used by the patch and is for reference purposes only.

```
struct {
      short       numZaps;

      /* Information maintained for each resource to be zapped */
      struct {
            short       zapID;
            ZapFlags    flags;
            ResType     resType;
            short       resID;

            /* These fields are used to verify the resources */
            short       resAttrs;
            Size        resSize;
            Str255      resName;
      }           theZap[];
} ZapList;
```

`numZaps`
The number of zap records to follow. There is one for each resource to be patched.

`zapID`
The resource ID of the 'ZAP ' resource that holds the munge data for patching this resource. There is also a corresponding 'ZIS#' or 'ZIL#' resource with this ID, based on the setting of the `shortMunges` flag.

`flags`

A bitmask of flags that can be OR-ed together, with the following definitions:

```
/* Flags for use in the ZapRecord */
enum {
      shortMunges             = 0x0001,
      needNotVerify           = 0x0002,
      verifyFailed            = 0x0004,
      alwaysUpdate            = 0x0008
} ZapFlags;
```

`shortMunges`
If set, this flag indicates that both the master and update resource are less than 32K, and 16–bit integers can be used for their offsets and lengths. Otherwise, 32–bit integers must be used. Most resources are less than 32K.

`needNotVerify`
If set, this flag indicates that, should resource verification fail, the patch application should perform the operation anyway in certain cases. If adding a resource that already exists, no change is made. If deleting a resource, the resource can be of any length if it exists. If changing a resource, no change is made.

`verifyFailed`
This flag is used internally during the patching process and should be set to 0.

`alwaysUpdate`
If set, this flag indicates that, should resource verification fail, the patch application should perform the operation anyway in all cases. If adding a resource that already exists, the entire resource is replaced. If deleting a resource, the resource can be of any length if it exists. If changing a resource, the resource's size is adjusted to its size in the master file and the patch is applied anyway.

The `needNotVerify` bit must be set for `alwaysUpdate` to take effect.

`resType`, `resID`
This is the resource type and ID of the resource to be patched.

`resAttrs`
These are the expected resource attributes of the master resource. These attributes must match during resource verification.

`resSize`
This is the expected size of the master resource. If the resource is being added, this size will be 0. The sizes must match during resource verification.

`resName`

This is the expected name of the master resource. The names must match during resource verification.

'ZIL#' and 'ZIS#' resources

The 'ZIL#' and 'ZIS#' resources contain a list of *munges*, or patch operations to be performed on a resource. Their formats are identical, except the 'ZIL#' uses 32–bit offsets and lengths, while the 'ZIS#' uses 16–bit offsets and lengths. 'ZIS#' is typically used, except for extremely large (>32K) resources.

If `masterLength` is the same as `updateLength` (a typical occurrence), the `updateLength` is suppressed, and the high bit of `masterLength` is set by masking in the constant `SameLengthFlag` or `ShortSameLengthFlag`. For this reason, the supplied ResEdit templates for viewing these resources simply dumps the munge instructions as hex data.

```
/* 'ZIL#' */
struct {
      short       numMunges;
      Size        sizeofMunges;

      struct {
            long        masterOffset;
            long        masterLength;
            long        updateLength;
      }           theMunge[];
} MungeList;

/* This flag is set on masterLength when updateLength is the same */
#define SameLengthFlag              0x80000000


/* 'ZIS#' */
struct {
      short       numMunges;
      Size        sizeofMunges;

      struct {
            short       masterOffset;
            short       masterLength;
            short       updateLength;
      }           theMunge[];
} ShortMungeList;

/* This flag is set on masterLength when updateLength is the same */
#define ShortSameLengthFlag         0x8000
```

'ZAP ' resource

The 'ZAP ' resource (note trailing space) contain the actual*munge data* that's used for insert or replace patch operations.

```
/* ZAP  resource--the munge data to insert/replace */
#define ZapMungeType          'ZAP '
```