Technical Note TN1132

Version Territory

CONTENTS

The Settings Library (`SettingsLib`), introduced in LaserWriter 8.5.1, allows applications to access and change the information created by LaserWriter 8 when a desktop printer is created. `SettingsLib` prevents contention over the printer database between different parts of a printer driver, different printer drivers, and other printer database clients such as the Desktop Print Monitor. The printer database is stored in the LaserWriter 8 preferences file. The Settings Library is especially useful for applications that previously relied on the format of the `PAPA` resource in the LaserWriter 8 driver. With the introduction of LaserWriter 8.5.1, the size of the `PAPA` resource changed, and applications that depended upon its size broke. Since the resource size may change again in the future, Apple has introduced this library to prevent such problems. This Technote overviews the Settings Library APIs.

Updated: [Jun 01 1998]

Finder 6.1 introduced version ('vers') resources as a way to allow the creator of a file to identify the version of a file, as well as the version of a set of files which includes this file. The format of this resource is described in Inside Macintosh: Macintosh Toolbox Essentials.

This Note, originally *Technote OV 12- Version Territory* , clarifies the format of data in the `NumVersion` structure used in a version resource, and provides guidelines for the use of version resources based on the version numbering scheme used at Apple.

All Mac OS programmers who distribute program files of any type should include version resources in their files.

Back to top

Version Number Contents

While the version resource structure is described in Inside Macintosh, it does not clearly describe the format for the data in all the fields of the `NumVersion` structure, which is the data type for the `numericVersion` field in a version resource. The comments in `MacTypes.h` from Universal Interfaces 3 aren't any more helpful, which has lead to different interpretation of the content of the fields in the `NumVersion` structure.

A `NumVersion` structure is defined as containing four `UInt8` values. This allows a `NumVersion` value to be used as a structure to access individual fields, or cast to an unsigned long for the purpose of comparing version numbers. There is a problem you should be aware of when comparing version number as unsigned long values.

The values in the `majorRev` and `minorAndBugRev` fields are stored in binary-coded decimal (BCD) format where each digit has a range of 0 - 9 (normal binary digits have a range of 0 - 15). The `majorRev` field contains two BCD digits for the major revision level. The `MinorAndBugRev` field contains two values, each stored as a single BCD digit -- the minor revision level and the bug revision level. This means that a version number can range from 0.0.0 to 99.9.9.

The value in the `nonRelRev` field is stored as an unsigned binary integer value. This give the `nonRelRev` field a range of 0 - 255. This is the field that is most often interpreted incorrectly, that is, as a BCD value rather than as a binary value.

Consequences of Using BCD for `nonRelRev` Values

As long as version resources are consistently created the same way the only consequence to using BCD values is that there are fewer `nonRelRev` values available: 100 using BCD and 256 using binary. Consistency is important, since BCD and binary values won't compare as equal even though they represent the same value. For example, a BCD revision 10 is `0x10` and a binary revision 10 is `0x0A`.

Since most comparisons of version numbers are done by first casting the `numericVersion` field to an unsigned long, the use of BCD values for the `nonRelRev` field does not affect this comparison if the versions resource being compared were created the same way, whether as BCD or binary values.

There have been some applications, notably ResEdit, which have interpreted the `nonRelRev` as BCD. Resorcerer was changed in version 2.0 to correctly interpret the `nonRelRev` field as an unsigned binary value.

Back to top

Apple's Version NumberScheme

Apple uses a version numbering scheme for its software products which you might want to adopt. Table 1 summarizes the scheme, which involves three numbers, each separated by periods.

Event	Version
First released version	1.0
First revision	1.1
First bug fix to the first revision	1.1.1
First major revision or rewrite	2.0

Table 1-Apple's Version Numbering Scheme

Note that Apple increments the first number when it releases a major revision, the second number when it releases a minor revision, and the third number when it releases a version to address bugs (the third number is omitted if it is zero).

During product development, Apple uses a version number followed by a suffix which indicates the stage of development. Table 2 presents a few examples.

Event	Version	Stage
First version	1.0d1, 1.0d2, ...	development
Product feature defined (begin testing)	1.0a1, 1.0a2, ...	alpha
Product is stable (begin final testing)	1.0b1, 1.0b2, ...	beta
Final candidate (almost ready to ship)	1.0fc1, 1.0fc2, ...	final
First revision shipped	1.0	final
First revision	1.1d1,...,1.1a1,...,1.1b1,...,1.1	
First bug fix to first revision	1.1.1d1,...,1.1.1a1,...,1.1.1b1,...,1.1.1	
First major revision	2.0d1,...,2.0a1,...,2.0b1,...,2.0	

Table 2-Development Version Numbering

Back to top

Comparing Version Numbers

A problem can arise when comparing version numbers by casting them to unsigned longs. When compared this way, Golden Master (GM) version numbers will compare as being older than any of the final candidate versions.

For the GM release of a file, the version resource will have the `stage` field set to `final` and the `nonRelRev` field set to zero. Most final candidate releases will contain a version resource, which has the `stage` field set to `final` and the `nonRelRev` field set to some value greater than zero. The problem here is that when the version numbers are cast to unsigned longs, the nonzero value in the `nonRelRev` field of final candidate version resources causes it to compare as greater than--and thus newer than--the GM version, which is in fact the newest version available.

In the past, this is most often a problem during installations when installing the GM version of a package over a perviously installed final candidate version of the same package. The installer would complain that you are trying to replace newer versions of the files in the package when this is clearly not the case. The Apple installer (and most other installers) avoid this problem by comparing the individual fields of version resources.

The following function will properly compare two `NumVersion` values:

```
pascal SInt16 CompareVersions( NumVersion *vers1, NumVersion *vers2 )
{
    UInt16 nonRelRev1, nonRelRev2;

    if (vers1->majorRev        > vers2->majorRev)        return  1;
    if (vers1->majorRev        < vers2->majorRev)        return -1;
    if (vers1->minorAndBugRev  > vers2->minorAndBugRev)  return  1;
    if (vers1->minorAndBugRev  < vers2->minorAndBugRev)  return -1;
    if (vers1->stage           > vers2->stage)           return  1;
    if (vers1->stage           < vers2->stage)           return -1;

    nonRelRev1 = vers1->nonRelRev;
    nonRelRev2 = vers2->nonRelRev;

    if (vers1->stage == finalStage) {
        if (vers1->nonRelRev == 0)            nonRelRev1 = 0xFFFF;
        if (vers2->nonRelRev == 0)            nonRelRev2 = 0xFFFF;
    }

    if (nonRelRev1 > nonRelRev2)                          return  1;
    if (nonRelRev1 < nonRelRev2)                          return -1;

    return 0;
}
```

Back to top

'vers' Resource Structure

The structure of a 'vers' resource is defined in `MacTypes.r` (from Universal Interfaces 3.1) as:

```
type 'vers' {
    hex byte;                       /* Major revision in BCD*/
    hex byte;                       /* Minor revision in BCD*/
    hex byte    development = 0x20, /* Release stage        */
                alpha = 0x40,
                beta = 0x60,
                final = 0x80, /* or */ release = 0x80;
    hex byte;                       /* Non-final release #  */
    integer;                        /* Region code          */
    pstring;                        /* Short version number */
    pstring;                        /* Long version number  */
};
```

The structure of the corresponding `VersRec` type is defined in `MacTypes.h` (from Universal Interfaces 3.1) as:

```
struct VersRec {
                                    /* 'vers' resource format */
    NumVersion  numericVersion;     /* encoded version number */
    short       countryCode;        /* country code from intl utilities */
    Str255      shortVersion;       /* version number string
                                       - worst case */
    Str255      reserved;           /* longMessage string packed
                                       after shortVersion*/
};
typedef struct VersRec          VersRec;
typedef VersRec *               VersRecPtr;
typedef VersRecPtr *            VersRecHndl;
```

The structure of the `NumVersion` type is defined in `MacTypes.h` (from Universal Interfaces 3.1) as:

```
struct NumVersion {
               /* Numeric version part of 'vers' resource */
    UInt8      majorRev;       /* 1st part of version number in BCD*/
    UInt8      minorAndBugRev; /* 2nd & 3rd part of version number
                                  share a byte*/
    UInt8      stage;          /* stage code:
                                  dev, alpha, beta, final*/
    UInt8      nonRelRev;      /* revision level of non-released
                                  version*/
};
typedef struct NumVersion      NumVersion;
```

The structure of the `NumVersionVariant` type is defined in `MacTypes.h` (from Universal Interfaces 3.1) as:

```
union NumVersionVariant {
                /* NumVersionVariant is a wrapper so
                   NumVersion can be accessed as a 32-bit value */
    NumVersion      parts;
    unsigned long   whole;
};
typedef union NumVersionVariant NumVersionVariant;
```

Mention of third-party sites and third-party products is for informational purposes only, and constitutes neither an endorsement nor a recommendation. Apple assumes no responsibility with regard to the selection, performance, or use of these vendors or products.

Back to top

References

Inside Macintosh: Providing Version Resources

Back to top

Change History

01-April-1988	Originally written.
1-Oct-1990	Revised as *Technote OV 12 -- Version Territory* to reflect the changes in MPW C 3.1
1-June-1998	Updated to clarify the use of the `NumVersion` structure.

Back to top

Downloadables

	Acrobat version of this Note (K).	Download
	Acrobat version of Inside Macintosh: Macintosh Toolbox Essentials.	Download
	Universal Interfaces 3.1	Download

Back to top
