2.0 Version Strings

By Carolyn Scheppner

Unlike the 1.3 *version* command, the 2.0 *version* command has the ability to search Amiga files for a version string. If you try the 2.0 *version* command on any of the 2.0 Workbench commands, you will find that almost all Workbench commands now contain these special version strings. For example, running *version* on the current version of *SYS:Utilities/More* will output *More 37.2*.

This embedded version string provides a simple way for a user to determine the specific version of a command. This is extremely useful for bug reports and phone support. You may enter these strings in your code yourself and update them by hand when required, or you may automate updates by using the *bumprev* tool (provided on a variety of DevCon disk sets and also in the Preliminary Software Toolkit II).

Hand-Coded Version Strings

The hand-coded method can be used in text files and is often quite suitable for simple programs with a single code module. If you code the version strings by hand, they should be formatted like the examples below. The example hand-coded strings are for a program named *myapp*, version 37.1, date 20-Mar-91 (20.3.91):

In C:

```
UBYTE versiontag[] = "\0$VER: appname 37.1 (20.3.91)";
```

In assembler:

```
versiontag     dc.b 0,'$VER: myapp 37.1 (20.3.91)',0
```

In a text file:

```
$VER: myapp.doc 37.1 (20.3.91)
```

Note that the `NULL` ("\0" or 0,) at the beginning of the versiontag string is not necessary but can be useful if you choose to #define the string and wish to give a version number to a C program with no data segment. With the initial `NULL`, you can concatenate a #defined versiontag string onto an arbitrary immediate string used in your code to get the versiontag into your code segment.

Automating Version Numbering with *Bumprev*

The *bumprev* tool and the include files it creates are what we use internally to give version numbers to system ROM modules, disk-based devices and libraries, and 2.0 Workbench and Extras commands. *Bumprev* creates or updates three files -- a *name_rev.rev* file which contains the current revision number, and the C and assembler include files called *name_rev.h* and *name_rev.i*. These include files contain #defines (.h) or macros (.i) to define the name, version, revision, and date of your program in a variety of string and numeric formats.

By using the appropriate include file in one or more of your code modules, you can use these #defines (or macros) in place of hardcoded version and revision information. This way, whenever you ‘‘bumprev’’ your revision files and recompile (or reassemble) your program, all version, revision, and date references in your program will be automatically updated. You can even include a *bumprev* call in your makefile for automatic revision bumping on every make (although this can update the version number more often than is really necessary).

The usage of bumprev is: `bumprev <version> <name_rev>`
For example: `bumprev 37 myapp_rev`

The first time you use the above example *bumprev* call, it creates a *myapp_rev.rev* file containing ‘‘1’’, and *myapp_rev.h* and *.i* files containing a variety of version and revision #defines (or macros) for version 37.1. The next time you use the same *bumprev* command it updates the files so that all #defines (or macros) are for version 37.2.

Bumprev does have some caveats. If you accidently type ‘‘bumprev 37 *myapp*’’ (instead of *myapp_rev*), *bumprev* will gladly overwrite any *myapp.h* or *myapp.i* file you happen to have rather than complain or automatically insert *_rev* into the output file names. Also, to make a major version switch (for example from 36 to 37), you must first delete the *myapp_rev.rev* file to make *bumprev* start the revisions over again at 1. Note that the 2.0 convention is for a major version number of 37 (to match the OS major version).

Here are example *_rev.h* and *_rev.i* files as generated by *bumprev*, and fragments of C and assembler code which include and reference these files.

Example *myapp_rev.h* generated by *bumprev*:

```
#define      VERSION       37
#define      REVISION      1
#define      DATE          "20.3.91"
#define      VERS          "myapp 37.1"
#define      VSTRING       "myapp 37.1 (20.3.91)\n\r"
#define      VERSTAG       "\0$VER: myapp 37.1 (20.3.91)"
```

Code example which includes *myapp_rev.h*:

```
/* myapp.c */
#include <exec/types.h>
#include <dos/dos.h>

/* stdlib.h and stdio.h contain prototypes for exit and printf.
 * Amiga.lib IO users could instead use <clib/alib_protos.h>
 *  and  <clib/alib_stdio_protos.h>
 */
#include <stdlib.h>
#include <stdio.h>

#include "myapp_rev.h"

/* NOTE: we reference VERSTAG version string for C:VERSION to find */
UBYTE versiontag[] = VERSTAG;

/* NOTE: we concatenate program name and version (VERS) with our copyright */
UBYTE Copyright[] = VERS " Copyright (c) 1991 CATS, Inc.  All Rights Reserved";

void main(int argc,char **argv)
    {
    /* Print our Copyright string.
     * Copyright string includes our myapp _rev.h version and date
     */
    printf("%s\n",Copyright);
    exit(RETURN_OK);
    }
```

Example *mylib_rev.i* generated by *bumprev*:

```
VERSION     EQU  37
REVISION    EQU   1
DATE       MACRO
    dc.b    '20.3.91'
    ENDM
VERS       MACRO
    dc.b    'mylib 37.1'
    ENDM
VSTRING    MACRO
    dc.b    'mylib 37.1 (20.3.91)',13,10,0
    ENDM
VERSTAG    MACRO
    dc.b    0,'$VER: mylib 37.1 (20.3.91)',0
    ENDM
```

Code example which includes *mylib_rev.i*:

```
* This is an example of an initial library code module
* Mylib_rev.i is generated with bumprev

      nolist
        include "exec/types.i"
        include "exec/initializers.i"
        include "exec/libraries.i"
        include "exec/resident.i"

        include "mylib.i"
        include "mylib_rev.i"       ; Bumprev revision include file
      list

            ; external
        xref    InitLib             ; init function
        xref    FuncTable           ; function table
        xref    EndSkip             ; End of code segment


; code at start of file in case anyone tries to execute the library as a program

        entry   FalseStart
FalseStart
        moveq   #-1,d0
        rts


ResidentNode
        dc.w    RTC_MATCHWORD       ; RT_MATCHWORD
        dc.l    ResidentNode        ; RT_MATCHTAG
        dc.l    EndSkip             ; RT_ENDSKIP
        dc.b    RTF_AUTOINIT        ; RT_FLAGS
        dc.b    VERSION             ; RT_VERSION        ;From mylib_rev.i
        dc.b    NT_LIBRARY          ; RT_TYPE
        dc.b    0                   ; RT_PRI
        dc.l    LibName             ; RT_NAME
        dc.l    IDString            ; RT_IDString       ;Contains VSTRING
        dc.l    InitTable           ; RT_SIZE           ;  from mylib_rev.i


LibName:        DC.B    'mylib.library',0
IDString:       VSTRING                                 ;From mylib_rev.i
        CNOP    0,2

InitTable
        dc.l    XMyLibBase_Size
        dc.l    FuncTable
        dc.l    DataTable
        dc.l    InitLib

DataTable
            ; standard library stuff
        INITBYTE    LN_TYPE,NT_LIBRARY
        INITLONG    LN_NAME,LibName
        INITBYTE    LIB_FLAGS,LIBF_SUMUSED!LIBF_CHANGED
        INITWORD    LIB_VERSION,VERSION                 ;From mylib_rev.i
        INITWORD    LIB_REVISION,REVISION               ;From mylib_rev.i
        INITLONG    LIB_IDSTRING,IDString               ;Contains VSTRING
                                                        ;  from mylib_rev.i
            ; library specific stuff

            ; end of init list
        dc.l        0

        end
```

❖