OVLBI-ES MEMO NO. 43

SOFTWARE STANDARDS FOR THE GREEN BANK OVLBI EARTH STATION PROJECT

L. D'Addario
93/07/26

This memo summarizes the standards for file organization, file maintenance, and coding adopted in a meeting on June 25 attended by G. Langston, D. Varney and myself.

1. Directory Structure For Common Files

The initial directory structure under ~ovlbi on the Green Bank network was described in OVLBI-ES Memo No. 33 on 31 August 1992. Various changes have been made since then. G. Langston will prepare a revised explanation of the structure, including a description of the files to be kept in each subdirectory. This will be kept up to date and a copy will be maintained in ~ovlbi/doc. (Note that ~ovlbi/doc is also available to users outside Green Bank via anonymous ftp under /ovlbi/doc.)

2. Code File Maintenance Procedures

All source files are maintained under SCCS (the Unix Source Code Control System utility) in various directories. The selection of the correct files for building any particular program is handled by the Unix "make" utility via "makefiles." The design of the makefiles is particularly critical; they are also maintained under SCCS.

There are two major subsets of programs: those designed to run in the real time system under VxWorks, and those designed to run off line under Unix.

When a change is made to any piece of code, a rigorous procedure must be followed for installing the change. We will automate this procedure as much as possible using make files and scripts. For the real time system, the necessary steps are somewhat complex and are described in Appendix A. Some currently available maintenance scripts are described in Appendix B. For both real time and off line code, a note describing each change is to be appended to a chronological log file ~ovlbi/target/changes.log. This file will be periodically (about monthly) moved to ~ovlbi/doc and renamed to describe the dates covered.

For the real-time system, the current source code files and make files are kept in ~ovlbi/code. These are read-only copies extraced from the fundamental archive copies that are kept in several SCCS directories (see explanations in ~ovlbi/doc). Also kept here are C-shell scripts designed to assist in file maintenance.

For the off line code, current source files, make files, and maintenance scripts are in ~ovlbi/offline. As above, these are read-only copies with the archive copies in ~ovlbi/offline/SCCS. Unix executables are in ~ovlbi/bin, which may also include executables needed for the OVLBI project but developed elsewhere and for which we do not maintain the source code.

For all source code that we use but do not maintain (including Category A [unmodified] VLBA code, the NAIF package, and unmodified VxWorks files supplied by Wind River), the archive copies are maintained elsewhere but secondary copies may be kept in other directories under ~ovlbi.

3. Naming Conventions

In general, the naming conventions described here are to be applied to all newly created entities after this date. Already existing ones may be revised to conform as time permits, but this will not be immediately required.

A. File Names

C source code will use file names that begin with a lower-case letter and are in lower case except for the first letter of words in multi-word names and special ending letters, described below. The underscore character _ is to be avoided. The standard suffix ".c" will be used. Example: twtControl.c

C include files (header files) will use all-upper case names with suffix ".H". Words in multi-word names may be separated by underscores if needed for clarity. This distinguishes our application code header files from VxWorks and Unix header files. Note that the VLBA real time code also follows this convention.

Fortran source files and include files will follow the same conventions except that the suffixes will be ".f" and ".I".

Category A VLBA code files (i.e., unmodified code used by us) will be kept under the identical names used in the VLBA project, whether or not they conform to our naming conventions.

Category B VLBA code (i.e., code modified for our use) will have the letter "O" (upper case) added to the VLBA's root name; e.g., "menu.c" becomes "menuO.c". For file names that are already all in upper case, we will add "_O" instead; e.g., "STATION.H" becomes "STATION_O.H". Exceptions to the latter rule will be allowed if the modified include file is required by unmodified VLBA code (category A), in which case the file name may be left unchanged.

OVLBI-ES specific modules in certain categories may be identified by an upper-case letter at the end of the root name. In particular, those C modules that implement an operator "screen" for the real time system will be identified by end letter "S"; e.g., "acu45S.c". (Note that the VLBA code does not use this convention.) File names ending in "*menu.c" are reserved for menu-generating functions of operator screens (following the VLBA convention).

B. Functions, Subroutines, and Other Identifiers

Long names that are descriptive of the item involved are encouraged to the extent permitted by the language. Standards for upper and lower case do not apply in languages where case is not significant, e.g. Fortran.

Names are to be in lower case except for the first letters of words in multi-word names. The first character of a name must be a lower case letter. The underscore character is to be avoided.

The following are encouraged but not required: the names of pointer variables should start with "p"; the names of VxWorks tasks should end in "t"; the names of object libraries should end in "Lib.o".

C. Macros

The names of macros (i.e., the subjects of #define directives) will be in all upper case. Within C modules, this distinuishes them from function calls and other language elements.

4. Documentation

We adopt in its entirety the documentation standards of the VLBA control system, described in Section I of VLBA Memo No. 468 (by B. G. Clark, 1985 July), subject to the following notes and additions.

The information on version number and revision date will be inserted in the module's documentation automatically by the SCCS utility. Files containing a blank documentation section for C programs, Fortran programs, and script/make files will be maintained for editing into new modules.

The testing section of the imbedded documentation may contain references to data or script files used to carry out the tests. Such files will be maintained by the programmer who did the testing in his own directories.

The documentation not included in the code modules themselves will be as follows: (i) Design Documents describing the overall design of a software subsystem, usually written before implementation, and revised as necessary so that they remain current. (ii) Program Manuals, describing the actual implementation of subsystems that span several modules or that are too complex to explain within the module. These are intended for use by programmers. (iii) User Manuals, for use by operators and other staff, covering those modules for which they are appropriate. All of the above are kept in ~ovlbi/doc and maintained under SCCS. In addition, during development there may from time to time exist preliminary and partial versions of any of these documents; they should be placed in ~ovlbi/notes for reference by the development staff.

5. Coding Standards

Adherence to ANSI standards is encouraged. C programs should conform to ANSI C as much as possible, and "make" files will be set up to cause compilers to produce warning messages for non-conforming code. In particular, forward declarations and external declarations of C functions should always be included and should use the ANSI prototype form so as to allow type checking by the compiler. Fortran programs should be written in compliance with the ANSI standard as much as possible, but ANSI extensions such as calling C from Fortran and using INCLUDE files are allowed. (Strict adherence to ANSI Fortran may not be desirable because it limits the length of variable names; this question is currently not resolved.)

Variables should not be made visible outside the module(s) in which they are needed. Thus, variables local to a function should be declared within the function; and variables local to a C module (file) should be declared "static". (Note that VxWorks provides macros LOCAL and GLOBAL for use in type declarations in order to clarify the progammer's intent; the use of these is encouraged where appropriate.)

The precision of numeric variables should always be well defined, and therefore declarations whose results are compiler-dependent or machine-dependent should be avoided. For example, in C, integers should be declared "short" or "long" and not "int" unless the programmer is very sure that the precision is unimportant.

Code should be placed in a header ("include") file only if it is needed by more than one module, and then only if it is longer than a few lines or unusually complex. Otherwise, declarations and definitions should be placed in the module where they are used. The header file's documentation should include a list of modules using it or (in case a large number of modules will need it) a description of the set of modules using it.

Indentation should be used to clarify the structure of the code, but no specific style is required as long as it is consistent within a module. However, a programmer who is modifying someone else's module is required to follow the original style. Long blocks (more than about 20 lines) should include matching comments at the start and end to further clarify their range.

6. Additional Standards and References

We acknowledge B. G. Clark (VLBA Memo No. 468) and Wind River Systems (VxWorks Programmer's Manual, Appendix C) for many of the ideas contained here. These documents contain additional standards to which we recommend, but do not require, adherence.

APPENDIX A: UPDATING PROCEDURE FOR REAL-TIME CODE

[Directory references in these notes are relative to /s3/ovlbi .]

A. To install a new module, it's necessary to do all these steps, in order:

1. Be sure that all the required source code is in either ./vlba-all, ./SCCS, or ./SCCS-vlba.
2. If a new VLBA module (or one that the GBES is not currently using) is being added, then edit MakeCopyVLBA to cause a copy of that module to be retrieved. Note that MakeCopyVLBA is under sccs, so editing requires checking it out and back in. Extract a current version of MakeCopyVLBA into ./code.
3. Edit MakeCode to cause the new module to be compiled and linked into an appropriate library. Since MakeCode is under sccs, editing requires that it be checked out and back in.
4. In ./code, execute the script ./code/CopyAll to cause current versions of all source code modules and makefiles to be retrieved.
5. In ./code, run % make -f MakeCode . The resulting libraries are automatically copied to ./target.
6. In ./target, edit gbes.vws to cause any new libraries to be loaded, or to make any other changes needed to support the new module. (This step is not needed if the new module was added to an existing library.)
7. In ./target, edit changes.log by adding an entry at the end that explains what you did.

B. To install a revised version of a previously-installed module, do steps 1, 4, 5, and 7 of part A.

C. To delete a module:

1. Delete (using % rm <name>) the source code from ./code.
2. Edit MakeCode to remove all references to the module.
3. If it was a VLBA module, also edit MakeCopyVLBA to remove any mention of it.
4. In ./code, run % make -f MakeCode.
5. In ./target/gbes.vws, remove references to the module, if any.
6. Edit ./target/changes.log by adding an entry to explain what you did.

Note: All of our code-control modules, including scripts and makefiles, contain a HISTORY section within comments near the beginning. Whenever one of these modules is edited, it is essential to add an entry to the history to explain what changes were made.

APPENDIX B: SOME CODE CONTROL SCRIPTS

```
# File CopyAll, version 1.2, released  93/07/15 at 13:58:21
#   retrieved by SCCS 93/07/15 at 13:58:23
#%% Script to copy all GBES source code from archives to current directory.
#:: CopyAll
#
HISTORY:
# gil 930715  Changed order so that OVLBI files overwrite VLBA files.
# lrd 930515  Changed to use make rather than cp to get VLBA code.
#             Put under sccs.
# lrd 930323  initial version.
#
#DESCRIPTION:
# csh script to extract all relevant files from the archive
# directories:
# (a)   /s3/ovlbi/vlba-all      VLBA code that we are using unchanged
# (b)   /s3/ovlbi/SCCS-vlba     VLBA code that we have modified
# (c)   /s3/ovlbi/SCCS          our code, specific to the GB OVLBI ES
make -f MakeCopyVLBA
sccs get /s3/ovlbi/SCCS-vlba
sccs get /s3/ovlbi/SCCS
```

The following two scripts, "checkout" and "putback", are intended to automate the process of modifying and re-installing source code that is controlled under SCCS and kept in any of our several SCCS directories. Detailed explanations of their use are given in ~ovlbi/notes/README.ovlbi.bin

```
## FILE  "~ovlbi/bin/checkout"
## usage:  checkout <filename>
## Script to determine which ovlbi directory contains a requested file.
## If the file is found and if it is not already checked out for editing,
## then an editable copy is checked out and placed in the current directory.
...

## FILE "~ovlbi/bin/putback"
## usage:  putback <filename> [<directory>]
# Putbck puts ovlbi code into the proper directory after editing.
# last edited by glen langston on 93 jul 16
...
```