VLB ARRAY MEMO No. 468

SOFTWARE RELATED STANDARDS FOR THE VLBA PROJECT
(Standards as implemented for Monitor/Control Software - 1985 July)

B. G. Clark

This document covers standards for documentation, coding, and updating procedures for the VLBA real-time systems, including those run in the station computer, the array control computer, the correlator control and archive writer computers, and any other computers in the system whose entire software effort exceeds a few pages of code. It includes firmware as well as software, and command files as well as code. It does not cover any software development efforts downstream of the archive writer (which have their own standards) with the possible exception of programs that run on archive writer files to assist in the diagnosis or control of the correlator.

Section I. Documentation.

Module specific documentation will be found in the module to which it applies, set off by delimiters as discussed below. In addition, there may be program system documentation, which will be a separate file with a filetype of .TXT.

A. Program module descriptions. These will consist of the following sections.

1. A statement of purpose and overview, hereafter referred to as a synopsis, delimited as described below.
2. References as applicable, especially references to datastructure documentation.
3. Functional description, including
   a. description of function and use of all arguments to a subroutine
   b. description of all run-time switches or options
   c. algorithms used
   d. timing or memory use if important
   e. external subroutines called
4. Statement of testing performed before initial submission and at each major revision.
5. Audit trail, giving revision number, programmer, date, and a brief (less than 80 character) description of the change, delimited as described below.
6. Keywords and index lines for making machine constructed indices to software, delimited as described below.
7. A statement of the program's language and required environment.
8. A version number, of the form x.y (x is major revision, y is minor revision, for instance version 1.0 for an original submission; version 1.10 follows version 1.9 if needed). This version number shall be accessible to the program via the macro facility of the language.

B. Datastructure descriptions. These will consist of the following sections.

1. A statement of purpose and overview.
2. A description of each item. This may optionally consist of comments intermingled with the actual declarations of the structure. In this case, the documentation section is the whole module.
3. A list of known programs using the structure.

4. Audit trail, language, environment, index line, and keywords as above.

C. Program system descriptions. These will consist of descriptions of how various modules can fit together to make a subroutine package, or how several programs work together to meet the needs of their users.

D. Documentation delimiters.

There are three levels of documentation: The documentation area, which for complicated main programs, might run many pages; the synopsis, which would contain sufficient information for a programer to use a subroutine or decide if he wants to steal the code of a main routine, and which should have a goal of perhaps ten or fifteen lines; and the index, which would have a one-line entry, filed under various keywords.

Documentation and synopsis material is delimited by start and stop indicators. Synopsis material is automatically included in the documentation file.

In addition to the above, there will be a language indicator, a version number associated with the program, indicated by an appropriate delimiter, and an audit trail, indicated by start and stop delimiters.

The beginning of a file serves as a beginning of synopsis delimiter. All delimiters start in the first character of a record except as indicated below. ALL delimiters are reserved, so that the documentation processor need not know what language the module is written in. For convenience of the various languages, though, serveral delimiters with the same meaning are allowed.

	C	FOR	BAL	PAS
Start of documentation:	/*+	C+	*+	▮+
End of documentation:	-*/	C--	*--	--▮
Start of synopsis:	/*++	C++	*++	▮++
End of synopsis:	---	C-	*-	---
Note: end-of-documentation also ends synopsis.				
Start of audit trail:	++$	C+$	*+$	++$
End of audit trail:	--$	C-$	*-$	--$
Index line indicator:	%%	C%	*%	%%
Note: only one per file permitted; must be 80 characters or less.				
Keyword indicator	::	C:	*:	::
Keywords separated by blanks or commas, only first 8 characters of each used. Filename is automatically a keyword.				

Language indicator: LANGUAGE: C
CLANGUAGE: FORTRAN
*LANGUAGE: ASSEMBLER
LANGUAGE: PASCAL

Note: only one per file permitted.

Version indicator #define version
<tab or blanks> PARAMETER VERSION
<tab, blanks, .> IDENT (or IDNT)
CONST VERSION

Note: only one per file permitted.

Section II. Coding standards.

It is recognized that no one language will be suitable for all applications; however, unless there are strong considerations otherwise, modules should be coded in C. In any event, the language should be one of the following: Macroassembler, Fortran, Pascal, or C. Macroassembler should not be used in VAXs or 68000s except in subroutines.

A. General considerations.

As much as possible the terms "Program module" and "Program source file" should be made interchangeable. That is, only one externally visible program module should be included in a file, as stated below.

The module should begin with the documentation area as described above. The programname declaration and the declaration of the arguments of a subprogram should be included within the documentation area.

Unless there are good and specific reasons to the contrary, all integers should be 32 bit (that is, the C int declaration and the Fortran Integer*2 declaration are to be avoided, in favor of the long and Integer*4 declarations.

All error messages which may be seen by the user should include the name of the module in which they are generated.

2. Layout.

a. Except in macroassembler modules, the block structure of the program shall be indicated by suitable indentation. The programmer who modifies a module should seek to conform with the convensions of the originator rather than employing his personal style.

b. Important comments (those applying to more than four or five lines of code) should occupy a line by themselves, with a blank line preceding and following.

3. Datastructures. Datastructures used in more than one module, or datastructures likely to be generally useful, or complex datastructures used extensively within a module should not be declared in the module, but included by an include statement from a separate file.

4. Modularization. Program modules should be included in separate source files. Separately callable subprograms should be considered separate modules. Separate subprograms should be included in a module only if it is inconceivable that they would be called from outside the main routine of the module.

5. Subroutine packages. Subroutines that are part of a package

should be declared by the inclusion of a declaration file rather than separate declaration within the module.

B. Declaration modules.

Declaration modules are used to define macros, subroutine parameters, and datastructures. Declaration modules should not be nested (that is, a declaration module should not include an include or #include statement).

1. Macro modules. Macro names should bear a visible relationship to the module name.

2. Subroutine declaration modules. These modules should be used to declare subroutine packages only. Modules declaring a single subroutine lead to a proliferation of modules that is more confusing than inline declarations. The existence of a subroutine package declaration module implies the existence of a subroutine package documentation file.

3. Datastructure declaration modules.

a.) For macroassembler consist of equates and macros which generate only equates.

b.) For Fortran, consist of variable declarations and either a named common declaration following, or an equivalenced array following. All variable names consist of the first two letters of the name of the common or array, and a four character identifier. That is, the named common /SOURCE/ might include real variables SORA and SODEC, and the character variable SONAME.

D. Fortran.

No extensions beyond ANSI standard Fortran 77 should be used except IMPLICIT NONE, INCLUDE, and inline comments. Arithmetic IFs and assigned GOTOs are considered obsolete and should not be used. EQUIVALENCE should be used only in datastructure declaration modules as described above.

E. Pascal.

No extensions to standard Pascal should be used except as explicitly stated in the module header (the most useful being the ability to set a pointer to a declared structure, the ability to do arithmetic on pointers, character string extensions, and direct access I/O). Record structures should not be declared within separately compilable modules but by #INCLUDEs of structure declaration files. Structures should be passed between modules (both for input and for output) by including pointers as parameters, rather than by including the structure itself as a parameter.

F. C

Structures passed to subprograms should not be declared internally within the subprogram, but should be #included.

G. Macroassembler

Macroassembler internal subroutines should, unless strong reasons exist to the contrary, use the Fortran, Pascal, or C linkage conventions as appropriate.

III. Update procedures.

A. Module residence conventions.

The fundamental copy of all source code will be in VAX3::[VLBSOFT.CODE]. Secondary copies will be maintained as needed on CVAX, PHOBOS, and the VME-10. Source code will be entered into the secondary copy areas only by means of copies from the primary area. It is not permitted to do simultaneous or independent copies to the secondary areas.

The fundamental copy of all documentation will be in VAX3::[VLBSOFT.DOC].

Relocatable object code will be generated in [VLBSOFT.CODE], and the relocatable object libraries will be found here also. There will be one relocatable library for each language, rather than separating libraries by function. There are two exceptions: libraries purchased as a unit from an outside vendor will be maintained as a separate library, and routines which are hardware specific, and for which more than one type of hardware is likely to be used (for instance, terminal formating routines) can be kept in separate libraries.

The area [VLBSOFT.OLD] will contain copies of the relocatable libraries prior to the last periodic update. In addition, for each program submitted for update, the previous version of source, object, and, if appropriate, load module, will be copied into [VLBSOFT.OLD] from [VLBSOFT.CODE].

The area [VLBSOFT.NEW] will contain source, object, object libraries, and load modules from programs submitted for update, but not yet incorporated by a periodic update.

On the Versados systems, a subsidiary copy of the source code and individual relocatables will be maintained in area 9000. Object libraries and load modules will be maintained in area 0.

B. Module naming conventions

Unfortunately, because the Versados only supports two letter file types, different conventions have to be used on that system from that of the VAX. The VAX convention for file types is as follows:

C source code	C
Fortran source code	FOR
Pascal source code	PAS
Macroassembler source code, macros, equates	MAR
Pascal structure or data definitions	PDC
Fortran named common declarations	FCM
C structure or data definitions	H
C macros	H
Module documentation	DOC (A file.DOC file will be extracted from the p program file)

Package documentation	TXT
VAX command files	COM
Versados command files	CF

The following file types will be used on Versados systems:

C source code	C
Fortran, Pascal, Macroassembler source code	SA
Macroassembler macros or equates	SM
Pascal structure or data definitions	PS
Fortran named common declarations	FC
C structure or data definitions	H
C macros	H
Command files	CF

C. Command files.

Command files are subject to the same update procedures as source code files, and should be submitted along with the source files when a program is originated. Command files should have the same name as the program module. The single command file associated with the module should, if the module is a subroutine, produce relocatable object. If it is a main program, it should compile the main program and linkedit with appropriate libraries, producing both relocatable object and a load module. Command files need not be submitted if no parameters or switches need be supplied to the compiler or linker, irrespective of wwhat directory or process the module is being compiled from

All specification of the directories from which INCLUDE files or object libraries are to be found is to be specified in command files, NOT in the program text. Command files submitted for system update should not contain specific references outside the area in which the program is being compiled and linked (except, on the Versados systems, explicit references to area 0 may be included). The programmer working on an existing program will make up his own command file with pointers to the appropriate libraries, etc, which he does NOT then submit with the program.

Sysgen command files for all dedicated VLBA computers are also subject to the requirement of format submission, as are any special routines written for incorporation in the systems.

D. Procedure for submitting for update.

For an existing program, one should first type the file VAX3::[VLBSOFT.NEW]CHECKOUT.TXT, which will list all modules currently being worked on and modules submitted for update. One then finds the appropriate copy of the module and copies it into his own area, and sends by VAXMAIL to VAX3::VLBSOFT a statement of which module he is working on.

When he has completed work on the relevant source and documentation (at least including updating the audit trail in the documentation area), he finds a copy of the update submission form in VAX3::[VLBSOFT]UPDATEFRM.TXT, fills it out, and sends it by VAXMAIL to VAX3::VLBSOFT. When the spirit moves him, the maintainer of VAX3::[VLBSOFT.NEW] will copy the relevant files into that area and transmit an acknowledgement by VAXMAIL to the sender. He may then

compile the module and update libraries and load modules in that area. He will at that time produce a command file which will, when executed in [VLBSOFT.CODE], produce the new, updated [VLBSOFT.CODE]. The commands to relink the main programs for included subroutines will be generated from the documentation for those main programs. Programmers MUST include accurate statements about included subroutines in this documentation.

E. Procedure for executing periodic update.

Updates will occur at fixed intervals; however, the spacing of intervals may change from time to time. Adequate notice to those involved will be provided.

The update will involve the following steps:
1). Initial operations in VAX3:
a). The relocatable libraries and all source, object, load, and command files expected to change in the course of the update will be copied from [VLBSOFT.CODE] to [VLBSOFT.OLD].
b). All ASCII files will be copied from [VLBSOFT.NEW] to magnetic tape for permanent record.
c). The new ASCII files will be copied from [VLBSOFT.NEW] to [VLBSOFT.CODE].
d.) [VLBSOFT.CODE] will be updated by
   i.) Compilation of subroutines and updating of libraries, and
   ii.) Recompilation and relinking of main programs.
e.) [VLBSOFT.DOC] will be updated by copying the documentation files into it.
f.) The contents of [VLBSOFT.NEW] will be deleted, and the other [VLBSOFT] areas purged.
g.) [VLBSOFT.*] will be backed up on magnetic tape.
h.) The maintainers of the subsidiary code areas will be informed of update completion and the names of the command files used in the update.
2.) Maintainers of subsidiary code areas in other VAXs will perform updates as appropriate, using the command files supplied.
3.) Maintainers of Versados areas will perform updates as best they can, using the VAX command files as reference.

F. Strictures against use of software outside the herein described system.

In many applications in real-time systems it is more important to know just what is running than it is that it is right. Therefore, only that code that has been through the update procedure will be used for production purposes. If the code absolutely will not work, the use of ad hoc fixes should not be used, but one of the following: 1) If an error in the last update caused a main program to cease working, the update for that program may be rescinded by copying the appropriate files from [VLBSOFT.OLD] into [VLBSOFT.CODE] (and equivalent copies on Versados systems). 2) If an error in the last update caused an important subroutine to quit working, the update should be rescinded in toto, by restoring [VLBSOFT.CODE] from the backup tapes. 3) If a hardware change or some such causes things to stop working, a full update should be done on an emergency basis.

No module will be accepted initially without appropriate documentation. No module will be accepted for update without incrementation of the version number and entry in the audit trail.

Documentation is fundamental. Programmers are enjoined not even to admit the existence of a module before the documentation is in existence.

IV. Common subprograms.

An index to existing subroutines will be maintained in VAX3::[VLBSOFT.DOC]PROGINDEX.TXT. Programmers are strongly urged to use programs from this list instead of doing their own. Similarly, they are urged to submit programs in a form to be maximally useful to others. (This often takes the form of submitting the routine in several useful pieces, with a master routine that calls them in succession).