System management for aips++

Part 1: organization and distribution

Mark Calabretta
aips++ programmer group
1992/Jan/30, revised 1992/Mar/14

1 Introduction

This discussion paper deals with the organization of **aips++** and its distribution to end-users. It is a crystallization of material presented in the "aips2-tools" email exploder between July and December 1991.

Organization encompasses that of the source code and the **aips++** "system", in particular the structure of the **aips++** directories on disk, and use of a source code management system.

Distribution of **aips++** extends on the one hand to members of the **aips++** consortium, whose copies need to be kept closely in synchrony, and on the other to end-users, who will usually only require major releases. It covers only the physical transport of **aips++** code, and certain installation requirements. System generation is addressed in part 2 of this document.

To date, the discussion on **aips++** system management has been set mainly in the context of the unix operating system. In practice compliance to POSIX.2 (Shell and utilities) is all that will be required. This standard is in an advanced state but has not yet been finalized.

2 Directory hierarchy

In designing the **aips++** directory structure, the discussion in the aips2-tools exploder seemed to reach a consensus on the following guiding principles

- Only one copy of the source code should need to be present at any one site (in this context "site" effectively means the local area network, or LAN).

  The current practice of installing $\mathcal{AIPS}$ and similar packages on one machine of each architecture in each LAN (or even on several machines of the same architecture!), can be circumvented by careful design of the directory tree. This will have advantages in maintainability, as well as more efficient use of disk space.

  NFS (or similar) provides the mechanism for realizing this requirement.

- Multiple machine architectures must be supported transparently.

  This effectively means that multiple architecture-specific system areas (binaries, libraries, etc.) must be logically divorced from the **aips++** code areas and from each other.

  Divorcing the code and system areas will also offer end-users the option of deleting the source code once they have installed the system. In a similar vein, the BIMA user specifications require **aips++** to be "configurable" in the sense that synthesis, single-dish, and VLBI code be separable, with the option of not installing one or other of them.

  A side effect of this requirement is that architecture-specific work areas will be needed to hold the intermediate products of compilation.

- The directory structure should allow logically distinct parts of **aips++** to reside on file servers as is most appropriate.

  This may be realized most conveniently by having the **aips++** code and architecture-specific system areas stem from a single directory node, thereby allowing one machine to act as an NFS server for the code and/or system areas which clients mount on a single directory node. A machine of a particular architecture may be selected to act as NFS server of the **aips++** system for other machines of similar architecture.

  Likewise, the entire **aips++** tree should stem from a single directory node and be relocatable.

  In conjunction with the first two requirements, this requirement implies that the **aips++** system be installable entirely from information contained in the code subdirectory tree.

  The code subdirectory tree would be mounted read-only from the server.

- Third-party software should be logically separate from the rest of the **aips++** code.

  It should reside in its own subdirectory or subtree.

  Libraries and utilities such as *PGPLOT*, and GNU *make* which will be a required part of **aips++** will be made available optionally as part of the **aips++** distribution (see below). Where recipients already have the correct version of this software installed and don't want a duplicate copy, they may opt not to accept the copy distributed with **aips++** and rely on their own.

- Distribution of **aips++** within multiple sites belonging to a single institution should be facilitated.

  Many institutions have several sites where **aips++** will be maintained in parallel. For $\mathcal{AIPS}$ Charlottesville-Socorro were such a pair for NRAO, and similarly Epping-Parkes-Narrabri for ATNF. The **aips++** directory tree can be structured to facilitate support for multiple sites. In particular, site-specific system areas can hold site-specific databases, for example data aging limits or data disk bookings. In a well designed system, upgrades may consist simply of copying the whole directory tree (or deltas) from one (master) site to another (slave) site.

- **aips++** must support multiple hosts within a site.

  Networks with dozens of workstations, each possibly with their own peripherals, are now the norm. Host-specific databases may be required to manage these peripherals, for example, and these should be supported in host-specific subdirectories.

- A release and development version of the **aips++** executables should be kept at consortium sites.

  $\mathcal{AIPS}$ traditionally maintained three versions (`OLD`, `NEW`, and `TST`). This was the minimum required to ensure that the distributed version (`OLD`) was sufficiently mature, bug-fixes being applied to the `NEW` and `TST` versions, and code development restricted solely to the `TST` version.

  However, this also meant that the $\mathcal{AIPS}$ programmers were always at least six months ahead of the rest of the $\mathcal{AIPS}$ community, and important bug fixes took at least three months to be distributed. On the other hand, bug fixes submitted by end users referred to a system long since removed from the computers at Charlottesville.

  aips++ should aim for a more direct association between programmers and end-users by keeping only two versions, released and development. The slightly increased risk of distributing bugs can now be offset by distributing fixes via worldwide networks. This mechanism was not available for $\mathcal{AIPS}$ until well into the project. The actual mechanism of distributing code and bug-fixes is discussed below.

A possible directory hierarchy designed to satisfy the above requirements may resemble the following scheme:

```
                   +--- PGPLOT-4.9.tar.Z
                   +--- SLALIB.tar.Z
                   +--- ATELIB.tar.Z
   +--- import ----+--- cool.tar.Z
   |               +--- make-3.62.tar.Z
   |               +--- cvs-1.2.tar.Z
   |               +--- rcs-5.5.tar.Z
   |               +---   :
   |
   |               +--- install ---+---
   |               |
   |               +--- kernel ----+--- include
   |               |               +-- implement
   |               |               +-----  Z
   |               |               +-----  Q
   |               |               +-----  :
   |               |               +--- scripts
   |               |               +---- data
   |               |               +----- doc
   |               |
   |               |               +--- include
   |               |               +-- implement
   |               +-- synthesis --+--- scripts
   |               |               +---- data
   +---- code -----+               +---- doc
   |               |
   |               +---- vlbi -----+...
   |               +---- dish -----+...
   |               +---- atnf -----+...
   |               +---- bima -----+...
   |               +---- drao -----+...
   |               +---- nfra -----+...
   |               +---- nral -----+...
   |               +---- nrao -----+...
   |               +---- tifr -----+...
   |               +--- contrib ---+...
   |
   +---- data -----+...
   |
   |
   |                               +----- lib
   |                               +----- libdbg
   |                               |
   |                               +----- bin
   |                               +----- bindbg
   |                               |
   |                               +----- tmp
   |                               +----- tmpdbg
   |                               |
   |                               +----- data
   |                               |
---+               +---- apex -----+
   |               |               +----- doc -----+--- ascii
   |               |               |               +--- info
   |               |               |               +--- manl
   |               |               |               +--- catl
   |               |               |               +---   :
   |               |               |
   |               |               |               +--- msgs
   +--- (arch1) ---+               +--- (site1) ---+--- (host1)
   |               |               |               +--- (host2)
   |               |               |               +---   :
   |               |               |
   |               |               +--- (site2) ---+...
   |               |               +---    :
   |               |
   |               +---- base -----+...
   |
   |
   +--- (arch2) ---+...
   +---   :
```

Example names
arch: `sun4, sun3, ibm, cvex, vax, pc,` etc.
site: `epping, narrabri, parkes, mopra` (for the ATNF installation), etc.
host: `baboon, lemur, nrao1` (at Charlottesville), etc.

Notes:

- `~aips++/import` holds public domain subroutine libraries and stand-alone utilities
- `~aips++/code/install` holds installation scripts including template system scripts
- `~aips++/code/kernel` holds core routines written by the **aips++** group
- `~aips++/code/synthesis` holds **aips++** tasks specific to synthesis arrays
- `~aips++/code/vlbi` holds **aips++** tasks specific to VLBI processing, in addition to tasks in `synthesis`
- `~aips++/code/dish` holds **aips++** tasks for single dish processing
- `~aips++/code/`{`atnf,bima,drao,nfra,nral,nrao,tifr`} hold instrument-specific packages
- `~aips++/$ARCH/base` is the release version of **aips++**
- `~aips++/$ARCH/apex` is the development version of **aips++**

The `.../install` directory will contain subdirectories - possibly architecture-specific - which will contain all the source code and shell scripts necessary to build the system procedure scripts and place them in the `.../$ARCH/bin` directories.

Division of the `.../kernel` subdirectory has not been considered in detail. However, the `Q`- and `Z`-routine directories at least will need to be further subdivided.

The division of **aips++** tasks into `synthesis`, `vlbi`, and `dish` is motivated by the BIMA user requirement that the installation be configurable for these purposes. It is not clear whether any of these directories will actually need `implement` and `include` subdirectories.

The `apex` directories would only contain files which differ from those in the `base` directories. Search paths could operate for the executables, libraries, documentation, etc., so that someone wanting to run the development version of **aips++** would set their search path with

```
~aips++/$ARCH/apex/bin:~aips++/$ARCH/base/bin:...
```

Search paths for documentation, and other areas may have to be implemented within **aips++** itself.

The `libdbg`, `bindbg`, and `tmpdbg` areas are provided for debug versions of the libraries, executables, and temporaries and will be discussed in part 2 of this document.

The `doc` area would contain documentation compiled from the code `doc` areas. Since the form of the compiled text may be different for terminals, workstations, etc., it might be necessary to have subdirectories holding documentation tailored for each of these cases.

Directories should contain README files to explain their purpose. It should also be noted that POSIX.1 effectively limits file names to 14 characters, and pathnames to 255.

3 Code management

Participants in the aips2-tools exploder generally agreed that universal read access should be allowed to all parts of **aips++** , and that write access should be arbitrated by some form of code management system. Although there was some disagreement as to what this should consist of, the majority view seemed to favour the Concurrent Versions System (*CVS*), a successor of the unix Revision Control System (*RCS*) on which it is built. At this time, however, purchase of the *TeamNet* software management system is being considered, for which consent will be required from consortium members. The remainder of this section and aspects of the next (Distribution) discusses the use of *CVS* as a fall-back option in case *TeamNet* is not adopted.

CVS has the advantage that it is freely available in the public domain, although it is unclear if it has yet been implemented successfully on anything other than SUN systems. *CVS* is optimized for tracking new releases of source code while maintaining local modifications to earlier releases. This should help to encourage the **aips++** user community to implement their own bug-fixes and enhancements, and develop completely new applications independently of the **aips++** consortium. This is in line with a sentiment expressed in the tools exploder that code development at remote sites should be supported.

CVS encourages concurrent development of code. Code cannot be locked for exclusive access, and in the situation where more than one programmer simultaneously modifies a file, it is left to the programmer who checks the code in last to reconcile any conflicting changes. This is at variance with our current experience with code management systems, and is currently contentious. In my opinion, *CVS* aggravates the problem by encouraging entire subdirectory trees to be checked out in one go, and by not providing a mechanism to determine who else has checked out a copy of a module. Checkout on a file-by-file basis is possible by making each of them a separate module, but at the expense of manually maintaining a database with an entry for each individual file. On the other hand, it should be possible to automate this. However, *CVS* modules must have unique names. Since **aips++** will have a code search-path mechanism for the Q- and Z-routines, a distinction would need to be made between such files if they are implemented as separate modules.

As an alternative to a completely concurrent system we might consider allowing exclusive use of an individual file for a limited period of time (say one week) and thereafter allow concurrent checkout. This would require hacking of the *CVS* source code itself.

Once the members of the development team separate and return to their home institutions, they will need network access to Charlottesville in order to check out code, retrieve it, and later replace it. At the moment, ATNF, BIMA, NRAL, NRAO, and DIT (the institution providing *CIC*) are well connected to the internet. DRAO, and NFRA plan connections to the internet within the next few months. Failing that, each of these sites currently has modem access to a site which is already on the internet. In the medium term, TIFR will only have access via modem to an internet site. However, this should be adequate for logging into Charlottesville, checking out code, and mailing it to themselves. Code distribution for TIFR will however need to be a little more sophisticated and is described below.

4 Distribution

The master copy of the **aips++** source code will be kept on one machine at Charlottesville (baboon) for redistribution to all other consortium sites. In order to preserve the notion that consortium sites "own" their own parts of **aips++** , they may be given exclusive rights to check code in and out of these areas. It will be their responsibility to update the master copy in Charlottesville with bug-fixes and enhancements, and thereby make it available to the other consortium sites.

At the June 1991 **aips++** meeting it was decided that consortium sites would play a role in redistributing **aips++** to their geographical region in order to share the load. ATNF, for example, would cater for the Australian astronomical community. Since distribution will mainly be via anonymous *ftp*, this scheme would be observed as a gentleman's agreement by **aips++** recipients. For example, there would be nothing to stop an Australian site from acquiring **aips++** directly from Charlottesville, they will simply be asked to get it from Epping. End users should register with their local consortium site for the purpose of receiving bug-reports, newsletters and the like, and also for statistical and political purposes. The user database should include the hardware configuration and operating system releases so that users with like systems may be put in contact with each other (by agreement). A master list of all **aips++** sites should be maintained at Charlottesville.

The anonymous *ftp* directories for **aips++** will contain compressed *tar* files of the complete plain text copy of the latest release of **aips++** . Separate tar files would be supplied for each of the `aips++/code` subdirectories to allow end-users to fetch only those parts of **aips++** that they were interested in. These would be used to build the `base` version of the **aips++** system. Sites which are not on the internet will require distribution of **aips++** by tape. In addition to the full release, there should also be a separate area for patch files for significant bug-fixes effected since the last full release. These should contain bug-fixes only and be independent of the latest revision of **aips++** which will probably contain new bugs. *AIPSSERV*, the *AIPS* email server, could be used to distribute these bug fixes to sites not on the internet. A list of the email addresses of **aips++** recipients should be maintained at each consortium site, and when a new bug patch is received email should be sent to all sites automatically.

In order to protect ourselves from end-users who are too tardy in updating their aips++ installation we should declare from the start that we will not support releases of aips++ which are too old. Two years seems like a fair and reasonable time.

The policy adopted in redistributing third-party software is that it be made available with the **aips++** release, but that end-users not be required to duplicate what they already have. This can be achieved by having a separate anonymous *ftp* area containing individual compressed tar files for each package. Recipients can chose what they want, or *mget* the whole lot.

The subject of distributing binaries was raised in the tools exploder discussions. It would be fairly straightforward to distribute binaries for all architectures and operating system releases for which **aips++** can be built at consortium sites. However, it is inevitable that many sites will not be able to take advantage of this because of incompatibilities in the revision of the operating system, compiler, etc.

Consortium sites will maintain their copy of the **aips++** source code in *CVS* repositories. A compressed tar file of the `base` version of the *CVS* repository would be kept in an anonymous *ftp* directory separate from the plain text version. Two schemes for distributing updates to the *CVS* repository could be adopted. Both are based on distributing complete copies of individual *RCS* files from within the *CVS* repositories.

- Have daily incremental updates which only include files changed since the last daily update. In order to rebuild the most recent version of **aips++** from the `base` version each of these incremental updates would have to be applied in succession. The onus of version tracking would be on the recipients.

- Have updates which always refer back to the last `base` release so that only a single update file is required to bring the `base` release up-to-date. This will obviate the need for detailed accounting of release versions at the recipients end, and prevent copies of the source code at consortium sites from getting out of step. However, because this scheme causes all changes made to the base release to be propagated each night, the size of the update file will be larger, and *make* will have to do more work than in daily update scheme.

In fact, these two schemes are not mutually exclusive, both could be catered for at the Charlottesville end. For example, it might be reasonable for consortium sites to apply daily updates during the week, and do a full update each weekend to guarantee the integrity of the system on that time scale. Either way, a separate procedure would also have to be delivered by Charlottesville to delete files from the *CVS* repositories at the receiving sites to account for files which had been removed from the master copy.

The ~ftp/aips++ directory hierarchy might resemble the following

```
                   +---- README
                   |
                   +--- import ----+--- make-3.62.tar.Z
                   |               +--- PGPLOT-4.9.tar.Z
                   |               +--- libg++-1.39.0.tar.Z
                   |               +---  :
                   |
                   |               +--- install ---+--- README
                   |               |               +--- VERSION
                   |               |               +--- V3.1.tar.Z
                   |               |
                   |               +--- kernel ----+--- README
                   |               |               +--- VERSION
                   |               |               +--- V3.1.tar.Z
                   +---- code -----+
                   |               +-- synthesis --+--- ...
                   |               +---- vlbi -----+--- ...
                   |               +---- dish -----+--- ...
                   |               +---- atnf -----+--- ...
                   |               +---- bima -----+--- ...
                   |               +---- drao -----+--- ...
                   |               +---- nfra -----+--- ...
                   |               +---- nral -----+--- ...
                   |               +---- nrao -----+--- ...
                   |               +---- tifr -----+--- ...
                   |               +--- contrib ---+--- ...
---+---- aips++ ---+
                   |
                   +----- sun4 ----+--- README
                   |               +--- VERSION
                   |               +--- OS4.0.3c.tar.Z
                   |               +--- OS4.1.1.tar.Z
                   |
                   +----- cvex ----+--- README
                   |               +--- VERSION
                   |               +--- OS10.1.tar.Z
                   |       :
                   |       :
                   |
                   +-- bug_fixes --+--- README
                   |               +--- VERSION
                   |               +--- V3.1.5.Z
                   |
                   +-- consortium -+--- README
                                   +--- VERSION
                                   +--- CVS3.1.tar.Z
                                   +--- CVS3.2.tar.Z
                                   +--- CVS3.3.tar.Z
                                   +---  :
                                   +--- CVS3.update.tar.Z
```

Both versions of the update mechanism are represented in the ~ftp/aips++/consortium subdirectory, where CVS3.update.tar.Z represents the update referred back to the base release, CVS3.1.tar.Z.

The script which deletes files removed from the *CVS* repositories in Charlottesville would be contained within the update itself.

In general, the compressed tar files will probably have to be split into reasonable sized files. The `VERSION` files will simply contain a version number for comparison with the recipients installed version. The procedure which automatically fetches the update file at consortium sites, or the bug-fix file at end-user sites, will check this version number against their currently installed version and halt if it hasn't been incremented. The `base` release number will always be of the form "n.1", and the bug-fixes which branch from the `base` release, will consequently have a third version number field, "n.1.m". In general there will be one new incremental release per day in the life of a `base` release, so the second version number field will increase to quite high numbers.

The mechanics of updates to consortium sites will require a cron job which runs at Charlottesville each evening to produce an update and place it in the `~ftp/aips++/consortium` directory. Some time later at the recipients end, a cron job will run to fetch the update, apply it, and then *make* **aips++** . Because of the international nature of the project, the effect of time zones must be taken into account. The following timetable has been constructed on the assumption that the update is placed in the anonymous *ftp* area in Charlottesville at 1900 local time. Note that each consortium member may have several sites at which **aips++** is under active development. The time difference quoted is with reference to Charlottesville, without allowance for daylight saving.

Site	time difference (hr)	local time	start time	wait time (hr)
ATNF				
Epping	+15	1100	1900	8
(Culgoora)	+15	1100	1900	8
(Parkes)	+15	1100	1900	8
BIMA				
Illinois	-1	1800	1900	1
Maryland	0	1900	2000	1
(Berkley)	-3	1600	1900	3
DRAO				
Penticton	-3	1600	1900	3
NFRA				
Westerbork	+6	0100	0200	1
Groningen	+6	0100	0200	1
Leiden	+6	0100	0200	1
NRAL				
Jodrell Bank	+5	0000	0100	1
NRAO				
Charlottesville	0	1900	1900	-
Greenbank	0	1900	2000	1
Socorro	-2	1700	1900	2
TIFR				
Poona	+10.5	0530	1900	13.5
Bombay	+10.5	0530	1900	13.5

As shown by the timetable, most of the western hemisphere sites should be able to collect the **aips++** update and begin a *make* within a few hours of it being made available in Charlottesville, even allowing for a minimum wait-time of one hour. In the worst case, NFRA will have only half a night to re*make* **aips++** , but this should be sufficient. However, assuming that the update should not begin before 1900 local time, the Australian and particularly the Indian sites will be somewhat disadvantaged by long wait times. Also, since TIFR is not on the internet, the update files will have to be distributed to them by email using *AIPSSERV*.

The converse of source code distribution is that of receiving feedback from end-users. Electronic mail addresses should be set up to deal with this, and in particular, separate addresses should be provided for bug-fixes, bug-reports, and requests. The first two categories will have a high priority. End-users should mail bug reports to their local consortium member, not necessarily Charlottesville. However, consortium members should forward all email reports to Charlottesville for cataloguing in a central repository. In some cases, consortium members may need to pass on certain bug reports to other consortium members who have responsibility for a particular piece of code.

5 Installation

One of the main points to emerge from the tools exploder discussion and also the user specifications was that installation of **aips++** must not require root privilege. This means that we cannot make use of certain unix system features which would greatly alleviate some problems, particularly in the area of initiating **aips++** and network services.

Many aspects of installation are covered in part 2 of this document. One of the user requirements was that installation be easy. The intention is that installation of **aips++** for supported architectures will at most consist of entering site and host information into editable ASCII site and host databases followed by a single *make*.

Installation of subsequent `base` releases should be even easier in one sense, since the databases will already be present. For sites undertaking their own code development, *CVS* provides a mechanism for merging their modifications with the new release code.

Specific user or group ids will not be required for **aips++** . However, we will recommend a set of **aips++** user and group names and ids for those sites who wish to create a separate account for the **aips++** package.

There is a potential conflict in requirements for access to **aips++** , in that some sites will want to allow free access to all users, while others may want to restrict access to members of a particular group, particularly with regard to usage of data directories. This will affect the group ownership and permissions of the **aips++** executables and data directories, and a procedure may have to be provided to reset these accordingly.