# The AIPS Unix Porting Reference Manual

*Version 15JUL94*

*Patrick P. Murphy*
*National Radio Astronomy Observatory*
*520 Edgemont Road*
*Charlottesville, VA 22903–2475, USA*
*(804) 296-0372*

`pmurphy@nrao.edu`

*Requests for help etc. to* `aipsmail@nrao.edu`

*World Wide Web accessible at*
`http://info.cv.nrao.edu/aips/`

Ignore this page

# TABLE OF CONTENTS

# 1. INTRODUCTION

This document is intended as a reference for *AIPS* managers installing a new version of *AIPS* on a Unix* machine. It describes the 4 phases of *AIPS* installation and much of what goes on behind the scenes. There is a separate document (The *AIPS Unix Installation Summary*) that covers the basics of installing *AIPS* on the more common systems (Suns, IBM RS/6000, Convex, HP, DecStation, Alpha, SGI, Linux); if you are installing *AIPS* on one of these systems, please use that document as your primary guide. You should use this "porting reference" when installing *AIPS* on new, untried systems, and/or if you want a more detailed explanation of the various steps of installation for all systems, new or otherwise.

*AIPS* has had a long association with the proprietary VMS system from Digital Equipment Corporation, and the installation process described here has many of its roots in the AIPS VMS install process. Hence there is no "*make aips*" command or overall "Makefile". Furthermore, because there are many variants of unix, it is not always possible to have an installation procedure that will run on all of them with no problems whatsoever; the shell scripts that comprise the bulk of the Unix installation process for AIPS come close, or so the author hopes.

With the `15JUL94` release, there are once again two types of installation available: a "source-only" tape, which will require re-compilation and rebuilding of the programs, and a "binary" tape which has pre-built programs as well as the source code. The source-only tape is available (as usual) via ftp on the internet, but due to their large size, the binary tapes are only available as actual tapes (DAT and Exabyte). Also, binary tapes are only available for a limited set of systems (see below).

Regardless of whether you are performing a new port, a source-only installation on a well-known system or a binary installation, the initial procedures are essentially the same. For an overview of the installation process, the reader is again strongly recommended to read the *AIPS Unix Installation Summary*. The following sections also give a brief tour of the work involved.

## 1.1 Overview of the Installation Process

There are four basic steps to installing *AIPS* on a Unix system. Each step has an associated shell script (Unix equivalent to a VMS command procedure or a DOS batch file), and these have names like `INSTEP1`, `INSTEP2`, *etc.*. In reading these following sections, don't worry too much about the details; just try to get an overall view of what needs to be done.

`INSTEP1` is common to both source-only and binary installations, whereas the remaining INSTEPs (2, 3, and 4) are only necessary for source-only installations. However, there is some advice in the sections on some of these subsequent steps that is relevant to binary installations; where this is the case, it will be indicated at the beginning of the section or sections.

### *1.1.1 INSTEP1: Setup and Unpacking*

The first steps of the installation/porting process involve optionally creating an `AIPS` account, loading the installation tape or file to disk and running an initial shell script. This is the same `INSTEP1` script as would be used for a system that *AIPS* already runs on, *e.g.*, Sun4. The script will ask you several questions and will attempt to set up the directory structure and configuration files needed for an *AIPS* installation.

Unlike the previous versions of this script, the `15JUL94` version can actually offer to start the editor for you on the relevant files that need modified and/or customized. It also tries to detect as much about your system as it needs to know, even checking printcap files for printer names if necessary.

* UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

### *1.1.2 INSTEP2: Massive Compilation of Libraries*

The `INSTEP2` shell script will recompile all the necessary subroutines into the right libraries. It is not necessary to run it for binary installations. This script, and the `INSTEP3` and `INSTEP4` scripts will have been copied to an architecture-specific area; this allows different architectures to be installed at the same time. There is unfortunately no *Makefile* for this or the following steps, although the scripts comes fairly close to doing most of the same things.

For new ports, it will be necessary to create some new directories, define names for them in `AREAS.DAT`, edit the `LIBR.DAT` file, modify the compiler and linker settings, and more, before starting `INSTEP2`. For "known" systems, most of this is unnecessary (though running `INSTEP2` is necessary). The script can take a *long* time on some systems; there is no formula for exactly how long as it depends on compiler efficiency and processor speed. On fairly modern, fast RISC machines, it may only take a couple of hours. There are about 1400 routines in all, depending on the target system. One nice feature about `INSTEP2` (and the following steps too) is that they checkpoint themselves. This means that if the script stops for any reasons (*e.g.*, you run out of disk space, the computer crashes, someone accidentally kills the process, or a routine fails to compile), you can continue where you left off without having to redo the previous good compilations.

If you are porting *AIPS* to a new system, there are some machine-specific or "Z" routines that require local development. Unless the new system is very different from most Unix systems, there will probably only be a handful of these. The generic versions should suffice well into the installation process, or you may find acceptable versions in one of the other systems or "architectures".

The number-crunching heart of *AIPS* is the set of "Q" routines. Originally intended for Array Processors, the "pseudo array-processor" or PSAP routines is general enough that it does not need modified for most new ports. You may want to look at creating optimized versions of some of these to increase performance, however.

The X11-based TV server and its set of "Y" routines should be adequate for most new ports. In the unlikely event that you actually have a dedicated TV device similar to the IIS or DeAnza systems, there is a stubbed versions of the Y-routines that will do until you are ready to address the problem of local development for unsupported TV devices. *For systems that AIPS is already ported to, there should be no need to develop local Z, Q, or Y routines.* However, consult the appendices of this document to check for notes on your system hardware and operating system versions.

We ask that you please *DO NOT* simply edit the distributed versions of text files if you find they require require modification. If you ask NRAO to provide support for problems that develop later, it will be much harder for us to figure out what is going on if you have done this. Instead, please either use appropriately named architecture-specific directories if you are doing a new port or significant development; or preserve the file in its original directory, *e.g.*, as `<filename>.NRAO`. Your cooperation in doing this will be much appreciated!

As about 400 routines precede the Z-routines, the first part of `INSTEP2` is a good time to look at the few Z-routines that require local development *for new ports only* (see the appendix on this topic). If your system is fast at compiling, you might want to address the Z routine changes before starting `INSTEP2`. On processors where "little endian" byte ordering is used, the value of `BYTFLP` as set in the routine `ZDCHI2` must be correct (see the Dec, Alpha, or Linux versions; most other systems are big endian). Also, the values for `SPFRMT`, `DPFRMT`, `TTYCAR` as well as other system constants and parameters defined in `ZDCHI2` should be checked and modified as required.

### *1.1.3 INSTEP3: Building Essential Tasks*

The third step of the installation process involves compiling and linking a small but highly representative subset of *AIPS* application and maintenance tasks. The programs involved are:

- The programs essential to create/maintain *AIPS* system files
- The *AIPS* task itself, and
- About a dozen of the most used tasks.

The programs, and "canned" versions of the *AIPS* system files are provided on binary tapes, but they both must be created for source-only installations. Additionally, even for binary tapes, the parameters in the "canned" system files may not suit the needs of the site or system on which you are installing *AIPS*; if so, it will be necessary to re-generate the system files. `INSTEP1` will offer to do this for you.

The system files include a system parameter file, an accounting file, batch queue and batch work files (the batch feature is optional), a password file, image catalog files, a task communications file, a "gripes" file, and *POPS*† memory files. These are created via the program `FILAIP` which prompts for its inputs. After creation, the memory files must be initialized by the `POPSGN` program. Finally, the program `SETPAR` should be run to set the system parameters not set by `FILAIP`; usually this just means setting the system name.

For those systems with *AIPS* TV displays other than X11 or Sunview screen servers, the program `SETTVP` should also be run to change the default TV parameters. It is not necessary to run it otherwise, unless you changed the parameters in the header files of the screen server(s).

After these programs have been run, you are ready to run a basic stripped-down *AIPS* system (or a full-blown one if installing from a binary tape). Other details, such as network services, need to be addressed before the network and graphical features of *AIPS* can be activated.

For networked installations, *i.e.*, where more than one host of possibly different architectures will share the *AIPS* directory tree, you will need to perform additional setup tasks. Most, but not all, of this has been encapsulated in the shell script `SYSETUP` and a `SETSP` program (the latter is compiled in `INSTEP4`).

Most of the task binaries generated by `INSTEP3` are the same ones used in what is known as the "Dirty Dozen Test" or DDT benchmark. This test is used to regenerate images and UV datasets which are then compared against a master set in order to certify an installation, and provide timing information. The master set is available on request from the AIPS group, or via anonymous ftp. For further details, see *AIPS* memo 85. There are additional certification tests used internally (`VLAC`, `VLAL`); contact the *AIPS* group for details on these.

### *1.1.4 INSTEP4: Building Everything Else*

The fourth step of the installation process is much like `INSTEP2` and `INSTEP3`, and it will attempt to compile and link all the tasks in the *AIPS* system. As this includes 316 separate programs, this can easily take as long as, or longer than `INSTEP2`.

## 1.2 "Logicals" or Environment Variables

Throughout this document, *AIPS* directories will sometimes be referred to as *logicals*; this term may be a bit confusing. It originates from the earlier VMS port of AIPS where extensive use was made of "logical names". These are implemented as environment variables under Unix. In addition to pointing at source code directories, they may also point at data areas, tape drives, TV display and graphics devices.

The notation used to refer to these environment variables here is the same that you would use to actually expand them on the Unix command line. This means that, for a variable called ARCH, if you type:

```
$ echo $ARCH
```

the computer will "echo" or print the value contained in that variable. In the remainder of this document, such variables (or rather their contents) will be referred to by, *e.g.*, `$ARCH` in typewriter font with the dollar sign preceding the name.

A few of the most common "logicals" or environment variables, used very frequently in this document and in *AIPS*, are:

† People Oriented Parsing System, *i.e.*, the *AIPS* command language

- **$ARCH**, the architecture of the computer (a combination of the Operating System and Hardware)
- **$HOST**, the host or computer name
- **$SITE**, the site name (may cover multiple hosts)
- **$SYSUNIX**, a directory where many Unix shell scripts, setup files and programs are kept
- **$SYSLOCAL**, a site- and architecture-specific directory where local setup information and *AIPS* utility programs are kept
- **$AIPS_ROOT**, the top-level directory of your *AIPS* installation.

Some of the essential *AIPS* environment variables will be defined for you by the **LOGIN.CSH** or **LOGIN.SH** scripts; you will be instructed to insert a call to one of these in your login file later in the installation process. However, the bulk of the variables are defined after one of these files has been called, usually by typing **$CDTST** at the Unix command line. This will define a variable for most of the directories in the *AIPS* directory tree; these are required by the compilation and linking shell scripts, but not for routine running of *AIPS*itself.

While almost all Unix systems now have the `printenv` command, some may not have an environment big enough to fit all the *AIPS* programming environment variables. This is especially true if your *AIPS* root directory is a long pathname. If you get messages about "environment overflow" or find that some variables are just not defined when they plainly should be, it may be necessary to remove the symbolic link or shell script **PRINTENV** from your **$SYSLOCAL** area, and build the **PRINTENV.C** program (found in **$SYSUNIX**) there instead.

## 1.3 Directory Structure

The original directory structure of AIPS is described in considerable detail in the AIPS programmers manual *Going AIPS*, Appendix A. A version of this in LaTEX format may be found in **$DOCTXT/APPENDIXA.TEX***. For the **15APR92** and subsequent versions of *AIPS*, some of this structure has been revised to allow for multiple hosts and architectures sharing the same directory tree.

At NRAO, multiple (dated) versions of *AIPS* need to exist side by side, so we have adopted a scheme where the date of a given version (*e.g.*, **15JUL94**) is encoded in the directory under which that version is kept. This version scheme is described in the section "AIPS Versions" later in this document. Thus, in the top-level directory where your *AIPS* installation will live, there will be the following sub-directories:

| | |
|---|---|
| 15JUL94 | Source and version-specific files (most of *AIPS*) |
| BIN | Contains some files used during a binary installation |
| DA00 | Holds areas for host-specific system files, also config files |
| FITS | Default location for import/export of FITS disk files |
| RUN | User-created run files are stored here (optional) |
| TEXT | Ionospheric data, color schemes (OFM's), and documentation (PUBL) |

In addition to 15JUL94, there may be similar directories for older versions of *AIPS*. Under each version-date directory are (at least) the following:

| | |
|---|---|
| AIPS | AIPS main programs and subroutines |
| APL | Applications programs and subroutines |
| DOC | Documentation area (see also AIPSPUBL above) |
| HELP | Help/Explain/Input files for tasks, verbs, *etc.* |
| HIST | Code Management history files |
| INC | Include files |
| Q | Numeric-intensive ("Pseudo-Array-Processor") tasks and routines |
| QY | Tasks, routines that depend on both Pseudo-AP and TV routines |
| RUN | Area for standard procedure or "run" files (DDT for example) |
| SYSTEM | System and Machine specific files |
| Y | TV-specific tasks and routines |

* printed copies are available from the AIPS Group in Charlottesville

Finally, there are (since `15APR92`) the following architecture-specific areas under each version-date directory:

| | |
|---|---|
| ALPHA | Digital Alpha AXP systems running OSF/1 |
| ALLN | Alliant Systems |
| CRI | Cray Research, Incorporated (UniCos) |
| CVEX | Convex Systems |
| DEC | Digital Equipment Corp; Ultrix on DecStations/DecSystems |
| HP | Hewlett-Packard 9000 series 700 "snake" systems |
| IBM | RS–6000 series systems running AIX |
| IBM3090 | Mainframes running AIX† |
| LINUX | Intel PC (386, 486, pentium) running Linux (*NOT* DOS!) |
| SGI | Silicon Graphics running Irix version 5 |
| SOL | Sun-4 series (Sparc) running SunOS 5 (Solaris 2) |
| SUN4 | Sun-4 series (Sparc) running SunOS 4 (Solaris 1) |
| SUN3 | Sun-3 series (Motorola 68020/68030) systems |

Of all these, only ALPHA, DEC, IBM, LINUX, SOL, and SUN4 are actively supported in-house by NRAO (*i.e.*, we have these systems and they all run *AIPS* for us). CVEX support is not what it used to be due to our lack of a local system (since March 1994 when `yucca` was de-commissioned). HP and SGI are supported via guest account at a couple of outside site (you know who you are; we are grateful!) and they generally have better support than other non-in-house systems. Under each of these directories there should be the following architecture-specific (but host-independent) areas:

| | |
|---|---|
| ERRORS | Has host-specific areas; not much used anymore |
| INSTALL | for performing the installation; `INSTEP2,3,4`, logs, *etc.* |
| LIBR | Object (random archive) Libraries (in subdirectories) |
| LIBRDBG | Object libraries compiled in debug mode (optional) |
| LOAD | Binary (executable) files |
| MEMORY | *POPS* memory file (first, *POPS*-number-independent, part) |
| PREP | for staging of temporary and listing files in compilation |
| SYSTEM | Arch-specific shell scripts, *etc.*; `SYSTEM/$SITE` is your `$SYSLOCAL` area. |
| TEMPLATE | Only for network installation; template system DA00 files. |

## 1.4 Software Overhaul and Data Compatibility prior to 15APR89

The information in this section is now at least 5 years old, but for the sake of historic completeness, is included here.

The *AIPS* application software was "overhauled" for the `15OCT89` release. All Fortran language software was changed to use a pre-processor and to appear to the compiler as ANSI-standard Fortran 77. This should make little difference to the installation of *AIPS*. However, no user data files from previous releases may be used by the `15OCT89` and later releases. All data must be read into `15OCT89` and later releases from FITS disk or tape or from other supported tape formats. Also, all programs written by users at non-NRAO sites prior to this release need to be overhauled. Software to assist in this is provided; see the file `$DOCTXT/F77CONV.TXT` for more details.

## 1.5 AIPS Versions and Multiple Architectures

### *1.5.1 The Canonical Three Versions: OLD, NEW, TST*

Up to recently, the *AIPS* system has been evolving continuously at NRAO, and three separate versions of *AIPS* were separately maintained (`OLD`, `NEW` and `TST`). Periodically, a new `TST` version was created, the previous `TST` version redefined as `NEW`, and the previous `NEW` version redefined as `OLD`. This `OLD` version was then distributed to non-NRAO sites as required, after testing of the installation process and verification of correctness via the DDT tests†. This process, known internally as the *quarterly update*, was (surprise) usually done every three months.

† "Dirty Dozen Tests", see *AIPS* memos 73 and 85

The *AIPS* systems are referred to by the date of their intended release, *e.g.*, `15JUL94`. NRAO staff are the main users of `TST` and often find bugs and problems first. Visitors to NRAO are recommended to use `NEW`, which was changed only to fix serious bugs. In this way, the code ultimately shipped to outside users has been rather thoroughly exercised, at least on NRAO systems.

However, with the development of the *AIPS*++ system proceeding (there has already been one library release), the `15APR91` release of "Classic" *AIPS* was the last one issued on a quarterly basis. NRAO is issuing `15JUL94` and will continue to issue versions most likely on a bi-annual basis, the next one planned being `15JAN95`. The decision to make further releases or to change this bi-annual release policy will be made as needed. Contact the *AIPS* group at `aipsmail@nrao.edu` for more details. In addition, the released version of "Classic" *AIPS* now corresponds to the `NEW` version in-house.

Because of the large amount of disk space needed for each version (see requirements below) most sites maintain only one release (*i.e.*, `OLD=NEW=TST`). However, some sites have found it very useful to bring up a new version of AIPS defined that coexists with an older, established version. After the new version checks out, the old version can be moth-balled on tape. However, versions of *AIPS* starting at `15APR92` are incompatible with earlier versions and it is difficult if not impossible to make a pre-`15APR92` version coexist with a later version (it can be done but requires work/expertise). Too many system-level directories are in different places. Also, as the `HOSTS.LIST` format was changed in `15JUL93`, it may be difficult to have it and older versions `15OCT92` or `15APR92` coexist with it. There are a few steps necessary to get such systems to co-operate; the details are spelled out in the `15JUL94` release notes later in this document.

### *1.5.2 Multiple Architectures*

As is alluded to elsewhere in this document, it is possible to support more than a single type of computer/OS from the same *AIPS* directory tree. This of course requires that you use the Network File System (NFS) or its equivalent, to make the same directory accessible from multiple systems. It is critical that the top-level "aips root" directory appear to have identical names on all systems on which you make it available, otherwise you will defeat the network scheme that *AIPS* uses.

Once you have installed *AIPS* on one architecture, here are the steps you must go through to add a second.

- If you have two binary tapes, unpack the second one in the same place as the first. You may get complaints about it not overwriting existing files, but as these refer to the source files and these are identical on all tapes, there is not a problem. Ignore such messages.
- If you have a binary tape but want to extend your system to a second architecture, you do not need an extra tape or tar file. All the source needed for compiling *AIPS* for an additional architecture is already on your tape.
- Log in to the "second architecture" system using the same account name you used while installing *AIPS* on the first. If necessary, create this account on the second system.
- Start `INSTEP1`. As it will find the `INSTEP1.STARTED` file from your first installation, it will pick up such things as the site name and aips root area. While it will ask some questions that are strictly speaking not necessary, it should detect most of the things set up by the first installation.
- If you are performing a source-only installation, proceed as you did for the first architecture with preparations for running `INSTEP2`, `INSTEP3` and `INSTEP4`. If the architecture you are building on is a well-known one to *AIPS*, you can probably skip `INSTEP3` as it is a subset of `INSTEP4`.

If your two (or more) architectures are compatible at the binary data level (identical byte order and floating point format), you need *not* re-run the `FILAIP` and `POPSGN` programs on the second architecture *if* you already have done so on the first. If you simply copy the files in the template area for the first system to a template area for the second (or make the template directory for the second system a symbolic link to the first), then the `SYSETUP` script will finish the configuring of the system files for you. You will still need to run `SETPAR` to set the system name.

### *1.5.3 Checklist*

The list of items below is intended as a quick summary of what is required to complete an installation. For most installations, the majority of the work will be done by INSTEP1; where manual intervention is needed, this is indicated by items *entirely in italics*. Also, items not relevant to binary installations are indicated by a **o** instead of a •.

- *Create an* aips *account if wanted, and read the tape (or tar file) to disk*
- If you want to have INSTEP1 automatically start your favourite editor on files that need to be edited by hand, make sure you have defined the EDITOR environment variable (*e.g.*, set it to emacs or vi or whatever).
- If necessary, convert critical shell scripts, including INSTEP1. This is only necessary on some systems where the Bourne shell lacks certain features (*e.g.*, shell functions). Of the known systems, only Convex, Ultrix and to a lesser extent OSF/1 have had to use alternate shells for some scripts. Generally the Korn shell is a good replacement. All files in $INSUNIX, and all *.SH files in the architecture-specific system area are converted initially by INSTEP1. Additional files copied to $SYSLOCAL also get converted later. The conversion simply involves editing the first line of each script.
- Define $HOST (hostname), $ARCH (architecture), and $SITE (site name). See item on HOSTS.LIST below. If an old INSTEP1.STARTED file from a previous installation is found, $SITE and $AIPS-_ROOT may be extracted from it. INSTEP1 will not automatically detect an old version in any other way.
- If Architecture is new, create 15JUL94/$ARCH directory and make LIBR, LOAD, MEMORY, PREP, SYSTEM and optionally TEMPLATE subdirectories. Also make $SYSLOCAL which is 15JUL94/$ARCH/-SYSTEM/$SITE.
- If 15JUL94/$ARCH/SYSTEM already existed, copy all files from there to $SYSLOCAL. Copy subset of files in $SYSUNIX to the new $SYSLOCAL also. Convert some of these for Convex OS and Ultrix shells.
- Copy some $SYSUNIX files to the $AIPS_ROOT area, being careful to save older versions already there. If an older 15JUL93 or 15JAN94 version was found, skip some unchanged files. Run AIPSROOT.DEFINE.
- *Insert a* source LOGIN.CSH *in the* .login *file (or a* . ./LOGIN.SH *in the* .profile*).*
- Make a symbolic link in $SYSLOCAL to the START_AIPS shell script. For Convex/Ultrix, copy the file instead, and convert its shell.
- Define the *AIPS* versions in AIPSPATH.CSH and AIPSPATH.SH (both of these in $AIPS_ROOT).
- Create $NET0 ($AIPS_ROOT/DA00), copy DADEVS.LIST and NETSP there, define this host's disks in these files, and make a directory here called $HOST. Also create an area for FITS files (defaults to $AIPS_ROOT/FITS), or make it a symlink to somewhere else.
- If necessary, copy PRDEVS.LIST from SYSUNIX to $NET0; add printer definitions from an old PRDEVS.SH file or from /etc/printcap if possible.
- Copy the TPDEVS.LIST file to $NET0 and make entries in it for local tape drives.
- Create the binaries NEWEST, PRINTENV (or make a symlink to printenv) and PP.EXE in $SYSLOCAL. Also create F2PS and F2TEXT. (Binary installations: copy these from the BIN/$ARCH area).
- Make an entry in $AIPS_ROOT/HOSTS.LIST for this host, and any of like architecture that will run *AIPS*.
- *For new ports, make sure all necessary new source directories have been created (one for Z routines to parallel the* $APLSUN*,* $APLSGI*, etc., areas; possibly for Q and – if you can't use X11 – Y routines too).*
- *Check that the above local directories are defined as environment variables in both* CDVER.CSH *and* .SH *in* $SYSLOCAL.
- *New ports only: Check* LIBR.DAT *and* INCS.SH *in* $SYSLOCAL *and make sure you have included any new Q, Y, or Z areas local to your site, and the search path for included source text.*

- *Edit the* *OPT.SH *files in* $SYSLOCAL *to reflect your choice of debug, optimization, etc., for compiling and linking. Also edit* $SYSUNIX/FDEFAULT.SH *to set fortran options if you changed them in* INSTEP1*, and check* $SYSUNIX/OPTIMIZE.LIS.
- Choose whether or not to use shared libraries on Sun/HP, and a separate set of debug libraries on all systems.
- Make XAS, the X11 (X-windows) TV server if desired.
- Create $ARCH/INSTALL and create symlinks to $INSUNIX/INSTEP[234] there. Move there and get ready to start INSTEP2.
- *Check the* **Vendor Specific Notes** *to see if there are any special action items for your system.*

## 1.6 Disk Space Requirements

A full-blown *AIPS* system with no files removed for disk space saving will take anywhere from 200 to 300 Megabytes, depending on several factors. Most of this is tied up in the binaries (executable files, the *.EXE files in the $LOAD area). The source code as distributed on tapes takes up considerably less, and in fact the compressed tar file for 15JUL94 when obtained via internet ftp is only slightly more than 24 Megabytes, or 17 Megabytes for the version compressed with the GNU gzip utility. The uncompressed source code is about 75 megabytes, including the TEXT area for documentation, *etc.*The *AIPS Unix Installation Summary* has a list, in section 5.1, of the sizes of files on the binary tapes as noted in the preparations for the release.

### *1.6.1 Text Files*

The disk space required for all of the text files that come on the installation tape is roughly 50 Mbytes. However, quite a bit of this is not required on disk for a Unix *AIPS* installation. The specific areas that can be removed after the installation is complete are:

- TEXT/PUBL, Documentation, including PostScript files
- 15JUL94/DOC, More documentation (over 4 Megabytes)
- 15JUL94/HIST, *AIPS* history, code management, *etc.*
- 15JUL94/SYSTEM/VMS, VAX/VMS system code
- 15JUL94/APL/DEV/VMS, VAX/VMS Z-routines
- 15JUL94/Q/DEV/FPS, FPS Array Processor Hardware code
- 15JUL94/Y/DEV/???, Any TV's you don't have, *e.g.*, DEA for DeAnza hardware
  (The TV servers use SS)

If you are really tight on disk space, you can always back up the source code to tape and remove all fortran, c, and include files:

```
% find 15JUL94 -name '*.FOR' -exec rm -f {} \;
```

and similarly for the files *.C, *.INC, and any *.o or *.f or *.c files in $PREP left over from your installation. The VMS areas can be safely removed if you are not going to use them (all-unix sites won't).

In addition, you can remove any 15JUL94/???/SYSTEM areas that do not match any of your systems and also any non-native Z routine areas under 15JUL94/APL/DEV/. Do *not* attempt to remove the HELP and RUN areas; they are necessary for the correct functioning of AIPS.

### 1.6.2 Binary Files

The disk space required for binary files (object libraries, program object modules, executable modules and system files) will be much greater than for the code. The bulk of most *AIPS* systems is taken up by the contents of the `$LOAD` area, i.e. the executable or binary program files. It is possible to use the unix utility `strip` to remove the symbolic debugging information from the files `$LOAD/*.EXE`; how much space this saves will be system dependent. It will probably be significant, however.

The space used by the system files will vary depending on the size of your installation. With the binary distributions, the supplied system files are designed for use with a moderate size setup with up to 10 *AIPS* hosts, 4 interactive and 2 batch users; varying these numbers will in turn change the sizes of the files. For the 15JUL94 release, the size of the system files on the binary tapes is about 8 megabytes.

Space used by the program binaries will depend on architecture, and many other details. Factors that can influence the sizes of these files include:

- Debug options options used when *AIPS* is built, *e.g.*, whether the `-g` option to include symbolic debugging information is used;
- Whether or not you "strip" the `*.EXE` files; this removes any debugging information (binary tapes are shipped with stripped `.EXE` files);
- Type of system; RISC systems (Reduced Instruction Set Computers; *e.g.*, Sparc, Alpha, HP, SGI) produce larger binaries than CISC (Complex Instruction Set, *e.g.*, Intel, VAX)
- How much use is made of shared libraries. The IBM AIX operating system, for example, makes extensive and efficient use of system shared libraries. Also important is whether or not *AIPS* shared libraries are used (their use is a trade-off; see below).

There are some tasks (*AIPS*ese for executables or binaries) that you will probably never use or need. Make sure you have a backup before you delete anything!

On Sun and HP systems, *AIPS* allows for the use of *AIPS*-specific shared libraries. This is achieved through special versions of `$LIBR` and `$LINK` in the `15JUL94/$ARCH/SYSTEM` area (which will be copied to your `$SYSLOCAL` by the installation procedure). The file `USESHARED` must exist in `$SYSLOCAL` in order to turn this option on; if it does not exist, the procedures will not create *AIPS* shared libraries. The first stage of the installation, INSTEP1, will query you on whether you want this feature or not, if you are performing a source-only installation. Another file, `NOSHARE.LIS`, must also exist. It specifies any programs that should *not* be built with shared libraries. There are many files in the `SUN4` version of this file, many of these probably need not even be there. There are no Solaris or HP versions of the file. All binary tapes are shipped from NRAO with static *AIPS* libraries.

For shared libraries, the resultant size of the binaries is on the order of 100 Megabytes for `SUN4`. If the system is built without using shared libraries and in debug mode, the size will be closer to 300 Megabytes. As indicated above, stripping the `*.EXE` files, and/or building without debug flags, will reduce the size somewhat. There is a trade-off, however; if you use shared libraries, you will most likely need to *increase the swap space on your systems.* At NRAO, we typically have between 70 and 105 megabytes of swap. In a site with more than two or three *AIPS* hosts, the savings from smaller binaries in `$LOAD` are more than offset by the amount of disk space that has to be reserved for swap on each system.

There have been some problems with shared libraries on newer OS versions of HP–UX. Caution is advised for HP installers; if disk space is not a major issue, you may want to stick with non-shared libraries.

## 1.7 System Directories

The directory 15JUL94/SYSTEM/UNIX contain shell scripts and configuration files used by *AIPS*(the older VMS command files are in the 15JUL94/SYSTEM/VMS area). In addition, there are directories under 15JUL94/$ARCH/SYSTEM for different flavors (architectures) of Unix. A list of these may be found earlier in this document under the *Directory Structure* section.

The installation procedure will create a subdirectory for site specific procedures and definitions. This will be 15JUL94/$ARCH/SYSTEM/$SITE and is usually referenced by its environment variable name, $SYSLOCAL. There is also a directory called 15JUL94/SYSTEM/UNIX/INSTALL where the Unix version of the *AIPS* install scripts are stored. As part of INSTEP1, a site- and architecture-specific area 15JUL94/$ARCH/INSTALL is created, and is populated with symbolic links pointing at the master versions of INSTEP2, INSTEP3, and INSTEP4. The compiling and linking for each architecture will then be done from the architecture-specific area, enabling simultaneous installations to be performed in a single AIPS_ROOT directory tree. Procedures for performing nightly updates (*i.e.*, the so called "midnight" jobs) from the master *AIPS* site in Charlottesville are stored in 15JUL94/SYSTEM/UNIX/UPDATE and its subdirectories; you will most likely not need these.

There are in addition two directories under the CVEX area of Unix System code. These reflect the different system requirements at NRAO's two former Convex sites, and are provided as examples of working Convex installations. However, all of NRAO's Convex systems have now been de-commissioned and there may be problems with newer versions of Convex OS.

## 1.8 Device and Machine Specific Code

The *AIPS* conventions for code that is of necessity specific to a given system or device is that the routine name begin with one of the letters Q, Y or Z. The Q routines were originally for array processor specific code, such as for Floating Point System's AP120 model; these routines have evolved into pseudo-AP routines on general systems, and into highly vectorizable machine specific code on computers that support vector operations, such as Convex. These routines are generally good candidates for high levels of optimization.

The Y routines indicate dependence on a specific TV or image display device. This includes not only older devices such as the International Imaging Systems (IIS) Model 70, 75 and IVAS, but also the *AIPS* TV servers. Since the 15APR92 release, all TV servers use the same set of Y routines ($YSS). NRAO has standardized on the XAS TV server which now has all the functionality of the Sun-specific XVSS. Considerable effort has been expended to make XAS as fast as possible. XAS should run on most, if not all, X-window displays (for X11 Releases 4 and 5 and variants thereof, including OpenWindows).

Finally, the Z routines contain code that is dependent on the specifics of different operating systems and different hardware.

These routines will be described in some detail in the sections that follow.

### *1.8.1 Z-routines*

The directory 15JUL94/APL/DEV and its subdirectories contain operating system, architecture and device dependent source code. Generic forms that are the same for all systems are kept in 15JUL94/APL/DEV itself, and as you descend the directory structure, the more specific the code becomes: Unix versus VMS, Bell (probably better if it were called SYSV) versus Berkeley Unix, Dec versus Sun, and so on. At the top level, some routines don't do more than call other routines in the same area or lower levels, or some combination thereof, and are all Fortran (with one exception). In the Unix sub-tree, most of the routines are written in C.

### *1.8.2 Y-routines*

Subroutines that deal only with the specifics of *AIPS* TV display, or Y-routines, are found in area `15JUL94/Y/DEV` and its subdirectories. The range of "TV" types covered ranges from the TV servers that use windowing systems such as X11, to the older dedicated image display devices from several different vendors. There is also an area containing stubbed versions (`15JUL94/Y/DEV/STUB`) for installations with no TV or windowing system. The most generic forms are kept in `15JUL94/Y/DEV` with more specific code in the subdirectories. Current TV devices include:

- `SS`: AIPS TV Servers, XAS, XVSS, or SSS; XAS over the network
- `IIS`: International Imaging Systems, Models 70 and 75 (in separate subdirectories)
- `IVAS`: The IIS IVAS device
- `DEA`: DeAnza (Gould) TV displays
- `STUB`: Stubbed routines that do nothing
- `VTV`: Virtual TV, "TV by wire" over the network (obsolete)

The `VTV` area is only used if you use `TVMON`; this is not necessary if you follow the standard network scenario used by *AIPS* for multiple TV servers. The IIS Model 70 actually formed the basis for the model behind the TV servers and their capabilities.

### *1.8.3 Q-routines*

The most computationally intensive portion of the *AIPS* source code has been isolated into what we call Q-routines and resides in the subdirectories of `15JUL94/Q/DEV`. Originally, this was intended to support code specific to array processors, such as those from FPS (Floating Point Systems), but with the widespread availability of fast, inexpensive RISC and other fast processors, the attractiveness of such add-on devices has diminished considerably.

In the early days of the "VAX" era, a pseudo array processor library (`15JUL94/Q/DEV/PSAP`) was developed for installations with no array processor (*i.e.*, most). These routines emulate an array processor in memory. Since then, they have become the standard for most Unix installations. Under the `PSAP` area are directories for systems that have vector capability such as Alliant and Convex. There are also other directories here for machine-specific code, usually modified for reasons of precision or similar considerations.

As previously stated, there are a large number of directories here that can be safely removed. Strictly speaking, you need only keep the subdirectories relevant to your installation. However, if you are porting *AIPS* to a new operating system or architecture, or dealing with a new windowing system or an unusual computing architecture (*e.g.*, multiprocessing or parallel), you may find the code in some of the existing directories useful as models.

## 1.9 Programming and Execution Tools

The directory `$SYSUNIX` contains many procedures and some programs used in Unix *AIPS* programming and operation. Most of these are written to be as generic as possible. However, some may require modification for a new port. Most of these require that the *AIPS* programming environment variables be defined prior to their use. This is normally done via the `$CDOLD`, `$CDNEW` or `$CDTST` commands, whichever is appropriate; these are described later in this document and redefine on the fly the many environment variables that point to the different directories (and that depend on the version date). For most installations, this will only be done once, typically in the `.login` or `.profile` file.

There are several other architecture-specific system directories `15JUL94/$ARCH/SYSTEM` as already mentioned. These may contain architecture-specific versions of the procedures and/or programs. `INSTEP1` will copy the most appropriate of these to a new `$SYSLOCAL` area in a rational way, using `15JUL94/$ARCH/SYSTEM/$SITE` as the directory for `$SYSLOCAL`. If none is appropriate (*i.e.*, you are doing a new port), choose the closest or if necessary the generic `$SYSUNIX` files. The few C programs and the preprocessor (in the `$SYSUNIX` area, currently `NEWEST.C`, `PRINTENV.C`, `PP.FOR`, `F2PS.C`, and

F2TEXT.C) will need to be compiled and linked as part of the installation. Again, this will be attempted by INSTEP1 but you may have to do it by hand if it fails. If there is a printenv command in your path during INSTEP1, then a PRINTENV symbolic link is simply made to it in $SYSLOCAL. Again, it may be necessary to build PRINTENV by hand if the system printenv cannot handle a large number of rather long environment variables.

Detailed descriptions of each procedure/program and its usage can be found as comments in the text. Also, most are designed to give you a synopsis of their usage if invoked incorrectly (*e.g.*, no arguments). A brief description of each is given here. First the procedures or shell scripts that will be copied to the $AIPS_ROOT area:

| | |
|---|---|
| AIPS.BOOT | (optional) commands to run at boot time: starting TPMON daemons; should be run under the aips account, not root (or at least make TPMON set-uid to aips). |
| AIPSASSN.SH | and .CSH: Local overrides, reserved-tv/message-terminal definitions. |
| AIPSPATH.SH | and .CSH: Path setting, version names. |
| AIPSROOT.DEFINE | Sets AIPS_ROOT in other scripts via sed (used by INSTEP1). |
| HOSTS.SH | and .CSH: Define HOST, SITE, ARCH (uses $AIPS_ROOT/HOSTS.LIST). |
| INSTEP1 | Will be a copy of the $INSUNIX/INSTEP1 procedure. |
| INSTEP1.STARTED | holds definitions of SITE and AIPS_ROOT from one run of INSTEP1 to the next. Also stores the *AIPS* version, e.g. 15JUL94. Keep it around for a possible 15JAN95 installation. |
| LOGIN.SH | and .CSH: *AIPS* setup to be called on login (calls HOSTS.*, AIPSPATH.* and AIPSASSN.* in that order). Insert a call to the relevant one from the .login or .profile of any account that will run *AIPS*. |
| PRDEVS.SH | Symbolic link to the generic $SYSUNIX version; inserted here for backwards compatibility with older versions of *AIPS*. See below. |
| START_AIPS | Starts up the *AIPS* program (via AIPSEXEC); called via aips or AIPS symlinks in $SYSLOCAL. |
| START_TPSERVERS | Starts up TPMON daemons for remote tapes (may be called via rsh). If called with -d as argument, gives verbose messages. |
| START_TVSERVERS | Starts up XAS and the other window servers MSGSRV and TEKSRV (may be called via remote shell command). |
| TVALT | Figure out your windowing system, if any. |

TVDEVS.SH Define names of hardware TV's such as IIS, alternate load areas if any (if none of either, needs no change and can be a symlink to $SYSUNIX version).

The following shell scripts are always copied to $SYSLOCAL:

| | |
|---|---|
| CDVER.SH | and .CSH: Change *AIPS* version and programming environment variables (called by $CDTST, *etc.*) |
| CCOPTS.SH | Define name and options for C compiler. |
| ASOPTS.SH | Define name and options for Assembler (if needed, *e.g.*, on a convex). |
| LDOPTS.SH | Define how to link-edit, usually with Fortran command. |
| INCS.SH | Defines search path for INCLUDE files for preprocessor. |

And the following scripts are available in the generic $SYSUNIX area:

| | |
|---|---|
| AIPSCC | Runs local C compiler (uses $SYSLOCAL/CCOPTS.SH). |
| AIPSEXEC | The script that actually starts *AIPS* (called by START_AIPS). |
| ALLRSH | Do a command on all *AIPS* hosts via rsh or remsh. |
| AREAS | (Re)Makes AREAS.SH and .CSH from $SYSAIPS/AREAS.DAT. |
| AS | Runs local Assembler (uses $SYSLOCAL/ASOPTS.SH). |
| BATER | Starts up program BATER (*AIPS* in batch mode). |

| | |
|---|---|
| `COMLNK` | Runs Compiler and Linker with appropriate object libraries (uses `SEARCH`, `PP`, `FC` and `LINK`). |
| `COMRPL` | Runs Compiler only, not for main programs (uses `SEARCH`, `PP`, `AS` or `CC` or `FC`). |
| `CREADIR` | Creates empty *AIPS* directory tree based on `$SYSAIPS/AREAS.DAT`. Obsolete. |
| `DADEVS.SH` | Defines data disks from files `NETSP` and `DADEVS.LIST` in `$NET0` (or `DADEVS.LIST` in `$DA00`, or `.dadevs` in login area). |
| `DASETUP` | Sets up data disks with aipsmgr/aipsuser ownership and permissions. Not used at NRAO so may need some work. |
| `DATOUCH` | List data directories, causing them to be mounted (if the automounter is used). |
| `FC` | Runs local Fortran compiler (uses `FDEFAULT.SH` and `FCLEVEL.SH` in `$SYSUNIX`). |
| `FCLEVEL.SH` | Reads file `OPTIMIZE.LIS` from `$SYSLOCAL` or `$SYSUNIX` and determines debug, optimize level for the program or subroutine being compiled. Called by `FC`. |
| `FDEFAULT.SH` | Defines default compiler names, switches, and optimization levels for fortran compilations (**Needs filled in for new ports**) |
| `GREP` | Finds files containing strings in subdirectories. |
| `LGREP` | Like GREP except this one is case sensitive. |
| `LIBR` | Builds/updates object libraries in `$LIBR/*/` (uses `$SYSLOCAL/LIBR.DAT`). |
| `LIBS` | Generates library list needed for programs in a given source code area; uses `$SYSLOCAL/LIBR.DAT` (versions also in Sun, Sol and HP local areas). |
| `LINK` | Runs local Linker (loader); uses `$SYSLOCAL/LIBR.DAT` and `LIBR` (versions also in Sun, Sol, IBM, HP local areas). |
| `MAKEAT` | generates file containing list of filenames (uses `$SYSLOCAL/LIBR.DAT`). |
| `PP` | Shell wrapper to preprocess source code (for fortran, calls `$SYSLOCAL/PP.EXE`). Local version for `IBM` and `CRI`. |
| `PROG` | "What directory is program FOOBAR in?" |
| `PWD` | Show current working directory without automounter (*e.g.*, `/tmp_mnt`) directory; may fail for some configurations. Uses `$PWD` if it can. |
| `RUN` | Run programs other than `AIPS` or `BATER`. Calls `DADEVS.SH`. |
| `SEARCH` | find appropriate version of a module (calls `NEWEST`, uses files `$SYSLOCAL/SEARCH*.DAT`). |
| `SHOPH` | Used for checking the shopping lists used by the new `ABOUT` adverb. |
| `SSSERVERS` | Start up Sunview version of *AIPS* TV server. (may not work; untested for years). |
| `STARTPMON` | Start up `TPMON` daemon(s). Called by `START_TPSERVERS`. |
| `SYSETUP` | Initial or incremental setup of host system areas. Handy when adding a new *AIPS* host. |
| `TPDEVS.SH` | Defines tape names, uses table `$NET0/TPDEVS.LIST` |
| `WHICH` | *AIPS* wrapper for "find" command. |
| `WHOUSES` | Shows which binaries (`*.EXE`) use a given routine. Takes a *long* time. |
| `XASERVERS` | Start up the XAS X11 *AIPS* TV server. Called by `START_TVSERVERS`. |
| `ZDICC2` | NRAO-specific Dicomed film recorder spooler. |
| `ZLASCL` | Sends graphics output to appropriate printer. |
| `ZLPCL2` | Sends text output to appropriate printer. Local versions for `SOL` and `CRI`. |

Finally, there are five programs in `$SYSUNIX`:

| | |
|---|---|
| `F2PS.C` | (New for `15JUL94`) Convert Fortran output to PostScript (tm) in a few different layouts; |
| `F2TEXT.C` | (New for `15JUL94`) Convert Fortran output to plain text (*i.e.*, filter the carriage control in col. 1); |

| | |
|---|---|
| `NEWEST.C` | Find newest file in argument list; |
| `PP.FOR` | The Fortran Preprocessor for *AIPS*; and |
| `PRINTENV.C` | copy of Berkeley `printenv` for systems lacking it or systems where it can't handle the 200+ names used by *AIPS*. |

At least two of these programs — `NEWEST` and `PP` — have to be built prior to using any of the other tools, *i.e.*, before most of the installation can proceed. The `INSTEP1` procedure will attempt to build binaries in the `$SYSLOCAL` area. The rest of the tools will assume (a) that `$SYSLOCAL` is in your path (the `LOGIN.*` scripts put it there) and (b) that the programs above are built and placed in that area.

# 2. Step by Step Installation Instructions

## 2.1 The AIPS Account

In an installation where many people will be using *AIPS*, it may be a good idea to have a dedicated account to run *AIPS* from. A good name for the account is `aips` but this is not essential. On the other hand, for a single user installation, creating such an account is not necessary. The default login shell may be the Bourne shell, C shell, Korn shell, or the GNU BASH shell, although the C shell is most commonly used. There have been some reports of problems with `tcsh` during the installation process.

At NRAO, most *AIPS* usage is conducted from this account, but AIPS may also be executed or programmed from private accounts *provided the AIPS system directories are set up with the proper group and/or world permissions.* If you want everyone at your site to be able to run *AIPS* from their own account, you should consider setting up a new group (*e.g.*, `aipsuser`) and adding people who are going to run *AIPS* to it; it need not be anyone's primary group. You may already have an existing group that can be used for this purpose. As the only alternative to an *AIPS* group is to set the protection on memory file, system, and data areas to world read/write (*this is a security risk and NRAO does NOT recommend it*), you are **strongly** encouraged to consider the group approach.

If you have a large *AIPS* installation, you may want to mimic NRAO's approach and have an `aipsmgr` account for the *AIPS* manager, in addition to the `aips` account. In that case, the `aipsmgr` account should be the owner of all files and directories associated with *AIPS*. Also, `aipsmgr`, the `aips` account (if any), and any other accounts that will run *AIPS* should belong to the `aipsuser` group mentioned in the previous paragraph. Then the user data areas and *some* system directories (`$NET0/*`, `$DATA_ROOT/*`, `$TSTMEM/*`) should be set:

- to be owned by group `aipsuser`, and
- `chmod g+ws`, *i.e.*, with group write enabled and the "setgid" bit on.

The "setgid" bit causes files created in these areas to have the same group ownership and permissions as the directories. However, note that for this "setgid" scheme to work, any disks that are NFS-mounted should be mounted by the client with the `grpid` option, if available. At NRAO, we mount *AIPS* data disks with options `rw,grpid,hard,intr`. `INSTEP1` attempts to apply this option where appropriate.

Do *not* set the "t" bit on these directories; it will cause many problems. If this bit is set on `/tmp`, it can potentially cause problems for *AIPS* as well.

The `$SYSUNIX/DASETUP` file has the details on how NRAO sets up *AIPS*-related accounts and ownership and protection on the important directories. It cannot be used if you use an automounter `DATA` map — the `root` account will probably have to set up the areas and the automounter maps — but is useful for setting up local disks. This script has *not been used* for some time at NRAO.

The *AIPS* startup script by default sets the protection mask to `umask 002` which allows the owner, and members of the same group, read and write access to the files and directories. This is set in `$SYSUNIX/AIPSEXEC`. The `ZCREA2.C` routine also sets a protection mask `PMASK` but this is wide open (`0666`). Change these as needed but try to avoid inconsistencies.

The source and binary files for an *AIPS* system may need to reside in one partition. The `mv` command is used by the installation and programming tools, and some older systems do not support moving files across partitions. Most modern systems tolerate this, however. In a network environment where more than one architecture is supported in the same *AIPS* directory tree, it may in fact be desirable to put the architecture-specific areas on different machines. As an example, suppose you have a network of Suns, some using SUN4 and one using SOL. You would then want the `$AIPS_ROOT/15JUL94/SOL` directory to be on the Solaris machine itself; this can be easily accomplished via a symbolic link:

```
cd $AIPS_VERSION
mv SOL SOL.OLD
ln -s SOL /home/foo/aips/SOL
```

Then you create the `LIBR`, `LOAD`, `SYSTEM`, `INSTALL`, `PREP`, `TEMPLATE` and `SYSTEM/$SITE` areas in the new directory on the Solaris host and move stuff into them from the old one.

Another advantage of this scheme is that it reduces the disk space requirements on the partition where the aips directory tree lives.

## 2.2 Loading the Installation Tape

In the distributed, networking world that most *AIPS* users now live in, there are many ways of loading an *AIPS* distribution onto disk. These include:

- Using the Internet and `ftp`
- Reading from a Local Tape Drive
- Reading from a Remote Tape Drive

Each of these possibilities requires a slightly different recipe in order to unpack the archive and get the sources in their correct directories. In all cases, however, you should be using the `aipsmgr` account if you have one, or failing that the `AIPS` account. If you choose not to create either of these accounts, use the account — perhaps your own — that will be the primary user of *AIPS*. For whichever account you use, make sure you are in its login area, or wherever the root of the `AIPS` directory tree will be.

The `tar` operation (see below) will create several files in this area: a directory containing the *AIPS* code (`15JUL94`), a second directory for some ancillary files (`TEXT`), a third directory used in the binary installations (`BIN`), and a symbolic link to the `INSTEP1` procedure. On binary tapes, there will also be a directory `DA00` which contains pre-built versions of the *AIPS* system files. The actual program binaries are in their final resting places on the binary tapes; there is no need to move them around after reading the tape.

The `15JUL94` directory contains a hierarchy with subdirectories `AIPS`, `APL`, `DOC`, `HELP`, `HIST`, `INC`, `Q`, `QY`, `RUN`, `SYSTEM` and `Y`. It also contains architecture-specific directories for many systems. You may want to remove any old `TEXT` area before unpacking the tape (or tar file), unless you modified files in it (unlikely) in which case you should move it prior to loading the tape or tar file. If you leave an old `TEXT` area, the reading from tape will fail to load newer versions of any changed files in that area.

### *2.2.1 Internet and ftp*

NRAO now encourages sites to obtain the source-only version of *AIPS* via `ftp` over the internet. If interested in this means of getting *AIPS*, contact the *AIPS* group at Charlottesville for details (send e-mail to `aipsmail@nrao.edu`). Unfortunately we cannot provide *AIPS* under "anonymous" ftp; therefore a special `getaips` account and (frequently changed) password is used; you will be provided with the latter when you make arrangements to obtain *AIPS* in this way.

If you used `ftp` to get the *AIPS* system, you will have ended up with either one large compressed `tar` file (`15JUL94.tar.Z`), or many smaller ones (`15JUL94.tar.Z.01`, `.02`, *etc.*). If you have many small files, just `cat` them together into one:

```
% cat 15JUL94.tar.Z.* >15JUL94.tar.Z
```

If you do not have very much disk space, you may want to combine them in the following manner (Bourne/Korn/bash shell):

```
$ mv 15JUL94.tar.Z.01 15JUL94.tar.Z
$ for i in 15JUL94.tar.Z.* ; do
>  cat $i >>15JUL94.tar.Z
>  rm $i
> done
```

This will go through each piece, appending it to the first one and removing the appended piece right away. Once you have the large file, uncompress it and extract the files from it as you would from a tape:

```
% uncompress 15JUL94.tar.Z
```

or, if you got the gzipped version, use:

```
% gunzip 15JUL94.tar.gz
```

Then:

```
% tar xvf 15JUL94.tar
```

or:

```
% zcat 15JUL94.tar.Z | tar xvf -
```

The second form can be used if your system has `zcat` (most do; some versions even work with gzipped files). Once you are confident that the archive unpacked correctly, you can remove the `.tar*` files from disk. If you wish to be careful, you can save them on tape first.

### 2.2.2 Local Tape Drive

Insert the distribution tape into the drive and use `tar` to unpack it, *e.g.*:

```
% tar xvf /dev/nrst0
```

This would do the trick on most Sun systems with the tape drive on a SCSI bus; use `/dev/nrmt0` for VMEbus based tape drives.

The tape drive names will be different on other systems:

- For SunOS 5.x, HP and SGI systems, use `/dev/rmt/<n>ln` where `<n>` is as above.
- On the IBM RS/6000 series, use `/dev/rmt<n>` where `<n>` is a number (0, 1, 2...). Use `smitty tapes` on IBM's to see which tape drive is which.
- On DecStations and Alpha systems, use `/dev/rmt<n>h` or `/dev/rmt<n>l` (for high or low density).
- On a Convex, use `tpmount` to select a tape drive (then `tpqueue` and/or `tpq` to see which one it is if you have many), and use the device name reported by the command.

If in doubt about tape names, check your system documentation, starting with the manual pages for `mt` if any, or ask your system administrator. "*There is no such thing as a standard Unix tape drive*" — the author.

The `v` flag in the argument to `tar` is not necessary; it simply causes each filename read from the archive to be printed out as it is unpacked.

### 2.2.3 Remote Tape Drive

This situation is a little trickier to do. Some implementations of `tar` will support reading remote tape drives (*e.g.*, GnuTar) but most will not. However, use of a few standard Unix utilities can do the trick:

```
% rsh tapehost 'dd if=/dev/rmt0 bs=20b' | tar xvBfb - 20
```

(On HP systems, use `remsh` instead of `rsh`) You use the appropriate network name of the remote system (the one with the tape drive) in place of `tapehost`, and the correct tape device name as argument to `if=`, of course. This command should be given on the destination machine. You can safely omit the `v` option if you do not want the verbose listing of all files as they are unpacked.

Before you try this, check your local system documentation to make sure the syntax of the `dd` and `tar` commands is as expected. In the example above, the `bs=20b` forces blocking in 20-block chunks (*i.e.*, 10k bytes), and the five letters after `tar` force extraction, verbose, force multiple reads per block if needed, specify input filename, and specify blocking factor in that order. The minus following that tells it to use the standard input (*i.e.*, output of the `rsh` command via the pipe) and the 20 is the argument for `b`, the blocking factor.

You should also make sure that the `rsh` command will work; check your documentation (manual pages, whatever) on `rhosts` and/or check your `.rhosts` file in your login area or the `/etc/hosts.equiv` file.

## 2.3 Installation Step 1

This step involves some editing, moving, and general setup. The `INSTEP1` shell script that will perform most, but not all, of the setup for you. It will prompt you with a set of questions. If you are porting *AIPS* to a new system, it will still attempt to do most of the setup, although more work will be required on your part after the procedure finishes.

The rest of this section will describe the individual questions the `INSTEP1` procedure asks, what they mean, and what they actually do "behind the scenes". The intent here is to give the reader a fuller understanding of what is necessary for an *AIPS* port or installation than the shorter *AIPS Unix Installation Summary*.

To get started, you should first make sure you are in the "aips root" area. This should be where you unpacked the distribution, and you should see the files `15JUL94`, `TEXT`, `BIN`, and `INSTEP1` (also `DA00` for binary installations). The directory you are in will be defined by environment variable `$AIPS_ROOT`.

If for some reason the `INSTEP1` file is *not* there, just type:

```
$ ln -s 15JUL94/SYSTEM/UNIX/INSTALL/INSTEP1 INSTEP1
```

It may be necessary on systems that do not support symbolic links to actually copy the file.

If you want `INSTEP1` to offer you the choice of editing the various configuration files during the course of the installation, you should make sure to have the `EDITOR` environment variable defined before starting.

Once you have reviewed the rest of this subsection, you then start the procedure (just type its name, *i.e.*, `INSTEP1`). The shell script itself is fairly well commented and is divided into about 17 sections, each one numbered in the comments and (roughly) corresponding to one of the following sections.

### *2.3.1 Preliminaries: Convex, Ultrix Bourne Shells*

During the recent enhancements of the various shell scripts, certain features, notably the use of shell functions, have been added that are not available in some of the more primitive versions of the Bourne shell. The systems known to be affected are Convex (OS9) and Ultrix (4.3) systems, and to a lesser extent OSF/1 (Dec Alpha)..

The `INSTEP1` procedure should be able to detect the "inferior" shells on the above-mentioned systems, and it will attempt to convert certain critical shell scripts to an alternative shell. For Ultrix, this is `/bin/sh5` (the System V shell), and on Convex and Alpha systems `/bin/ksh` (the Korn shell) is used. INSTEP1 writes a shell script called `sh-cvt` in the aips root area which adds a line at the beginning of each shell script it operates on, e.g. `#!/bin/sh5`.

The following files will be converted. As the first of these is `INSTEP1` itself, the process may get stuck after some or all of the converting. This has occurred in the past (on Convex OS) but all scripts apparently got converted anyway and a restart succeeded.

1) The installation scripts `INSTEP1` through `INSTEP4`;
2) All `*.SH` files in the `CVEX/SYSTEM` or `DEC/SYSTEM` areas

Later on, the `sh-cvt` script will be used on additional files.

On OSF/1 systems (Dec Alpha), the Bourne shell is better but still not quite good enough for `INSTEP1` so it modifies itself to use the Korn shell, asks to be restarted and stops. Just do the obvious: type `INSTEP1` again.

### *2.3.2 Protection Mask*

The procedure will now ask if the current `umask` or protection mask is acceptable:

```
INSTEP1:  The current umask (protection mask) is 0022
INSTEP1:  (check the man page on umask if you don't know what it is)
INSTEP1:  is this umask acceptable?  (NO)
```

You should answer with either `Y` or `N`*, though a simple `<return>` will give the default response indicated (if a default response is indicated; some questions require a specific answer). In general, restricting access to the files that will be created is a good idea; the mask indicated above will deny anyone except the owner of the files write access. Alternately, the `umask` can be set to 0002 which allows members of the same group write access. Note that *AIPS* requires write access to some files (*e.g.*, memory, system files in `$DA00`, and of course data areas). The setting of 0000 will allow any user read and write access but this is not recommended for security reasons. (The Bourne shell `umask` command gives 4 digits unlike the c-shell equivalent, and some shells only show two figures but they are the relevant two).

It is better to have all *AIPS* users be a member of a group, *e.g.*, `aipsuser` as outlined earlier in this document. The Installation script will attempt to set the "set-gid" bit on certain directories and set the protection on them so that all members of the primary group of the `aips` account (or whatever account is doing the installation) will have privilege to write to files created in them.

### *2.3.3 Existing Installation*

The way in which `INSTEP1` finds a previous installation is via the presence of the file `INSTEP1.STARTED`. If this file exists, it is scanned at this point and the version defined therein is extracted. Then it is moved to `INSTEP1.STARTED.OLD` and you are given the chance to have the previously defined `SITE` and `AIPS_ROOT` extracted from this older file. Regardless of whether you choose to get these older values, a new `INSTEP1.STARTED` is created with the current version date `15JUL94`.

If the older version is `15JUL93` or `15JAN94`, certain files in `AIPS_ROOT` will be preserved as they have not changed in the `15JUL94` release.

### *2.3.4 Host Name*

The installation script will attempt to find the Unix `hostname` command in your default path. If not found, it checks if `uname` is available. For Convex systems, it makes a symbolic link in the current area to the dummy `uname` command in `$SYSCVEX`. If neither of these is found, it will query you for the hostname. In this case you should enter the simple hostname with no domain, *e.g.*, `orangutan` instead of `orangutan.cv.nrao.edu`. It converts the string to uppercase and (where `hostname` or `uname` is used) strips off any domain part of the hostname.

The name is kept in the environment variable `HOST`. It will be defined by the file `LOGIN.CSH` or `LOGIN.SH` after the procedure finishes.

If your site uses fully qualified host names, e.g. the `uname -n` command (or `hostname` on systems where `uname` is not available such as Convex) prints out `orangutan.cv.nrao.edu` instead of just `orangutan`, you need to make sure there is an alias for your system in the `/etc/hosts` table, *e.g.*:

```
192.33.115.11     orangutan.cv.nrao.edu orangutan
```

Here, the second mention of "orangutan" is the alias. Some networks have the `/etc/hosts` file distributed via NIS (yellow pages) and others use the Domain Name Service (DNS). Contact your system administrator and find out where this information can be set and have the alias inserted if it's not there already. One way to find out is to try, *e.g.*, `telnet orangutan` (replace with your hostname, minus any domain) and see if it works. If it does, then you don't need to worry. If not, have it fixed, otherwise the *AIPS* TV, remote tapes, and graphics and message servers will not work. (There is a call in the Z routines to `gethostbyname()` and `AIPS` will feed the unqualified host name into it).

In addition, `uname` on systems such as SunOS 4.1 will return a maximum of 9 characters. SunOS 5 does not have this restriction. Because of this, NRAO has found it necessary to create additional

* For all yes/no questions, the script will convert whatever you type to uppercase and only look at the first character.

aliases for systems like `antarctica` and `pantagruel` (dropping the tenth and any subsequent characters to create aliases, e.g. `antarctic`).

### *2.3.5 Site Name*

The site name is used to set up certain site-wide files and directories; when prompted, you should enter a string that can be a valid filename. At NRAO we use `NRAOCV`, `NRAOAOC`, `NRAOVLA`, and `NRAOGB`. The string you enter will be converted to uppercase. The site name is kept in environment variable `SITE`. This is stored in the file `INSTEP1.STARTED` so you do not have to enter it again if you interrupt the installation procedure and start over. If an old `INSTEP1.STARTED` is found, the site name will — at your approval — be picked up from it by `INSTEP1`.

It is possible for multiple sites to share the same `AIPS_ROOT` area. This is quite useful if you want to support machines with different byte order (big, little endian) using just one installation. For example, NRAO has two "sites" within the Charlottesville installation, `NRAOCV` for Sun and IBM RS/6000 systems, and `VCOARN` (spell it backwards) for Dec Alpha, Dec Ultrix, and Linux systems. As the hosts and tv information obtained by the various scripts from`HOSTS.LIST` is restricted to a single site name, it is possible to mix/match different systems in this manner.

### *2.3.6 Architecture*

There is no general way of telling what type of computer a unix shell script is running on, but `INSTEP1` looks for a few telltale hints. The aim here is to get an environment variable `ARCH` defined, and create a directory of the same name, *i.e.*, `15JUL94/$ARCH`, to hold subdirectories for binaries, libraries, and the like. The current known architectures are:

| | |
|---|---|
| `SUN3` | Sun Micro-systems (Motorola 680x0), SunOS 4.x (Solaris 1.x) |
| `SUN4` | Sun Micro-systems (Sparc), SunOS 4 (Solaris 1.x) |
| `SOL` | Sun Micro-systems (Sparc), SunOS 5 (Solaris 2.x) |
| `CVEX` | Convex, Convex OS |
| `DEC` | Digital Equipment Corporation DecStation/system, Ultrix 4.3/DEC Fortran |
| `ALPHA` | Digital Alpha AXP, OSF/1 1.2, 1.3, and 2.0? |
| `LINUX` | Intel 386/486/pentium PC's with Linux (not DOS) 0.99 through 1.1.47 (at least) |
| `IBM` | IBM RS–6000 series, AIX 3.1 and 3.2 |
| `HP` | Hewlett-Packard 9000/700 systems, HP-UX 8 or 9 |
| `SGI` | Silicon Graphics (SG–Irix version 5) |
| `IBM3090` | IBM Mainframes, AIX (not fully supported) |
| `ALLN` | Alliant (Concentrix) |
| `CRI` | Cray Research, Inc. (UniCos) |

The `INSTEP1` script can usually discriminate among all of these, except for the last three. The way it does so is somewhat convoluted, but goes something like this. First it sees if `/usr/bin/arch` exists. If so, it sets `ARCH` to the output (uppercased) of this command. This covers Sun3 and Sun4. On Solaris the directory is `/usr/ucb/arch` so it looks there next. If neither of these are found, it checks for a directory `/usr/convex` which is usually only to be found on (surprise) Convex systems. If this in turn is not found, it looks for the `uname` command, first in `/bin` and then in `/usr/bin`. If either `uname` or `arch` is found after all this searching, they are used to get the architecture.

After all this, the `ARCH` environment variable is examined and possibly changed. `AIX` is changed to `IBM`, `ULTRIX` to `DEC`, `OSF1` gets converted to `ALPHA`, `HP-UX` to `HP`, and `I386` or `I486` to `LINUX`. In addition, for `SUN4`, the OS revision (4.something or 5.something) determines whether it is converted to `SOL`. Also, for Solaris, `/usr/ccs/bin` is added to the path if (a) it's not already in it, and (b) the directory exists. If you see a message, you might want to consider adding this directory permanently to your path in your login or profile.

The script then asks for confirmation on what it has found, or tells you that it is an "unexpected system" or perhaps a "system not familiar to this script". If the script just plain guessed wrong, you can correct it at this point (let us know so we can investigate and figure out where we went wrong too). If you are doing a new port of *AIPS*, enter `NO` at this point. Then the script will ask for the name of the architecture, and if it's new, you will be asked for confirmation.

The following is done for new ports and also for known systems, and the directory creation and setgid steps are skipped if the directories already exist:

- Create `15JUL94/$ARCH` directory, makes it setgid;
- Create subdirectories `ERRORS`, `ERRORS/$HOST`, `LIBR` (for object libraries), `LOAD` (for binaries), `MEMORY` (for the *AIPS* memory file), `PREP`, `SYSTEM` (system-specific shell scripts), and `TEMPLATE` to hold templates of system setup files;
- Make all these subdirectories "setgid".

### *2.3.7 Create SYSLOCAL Area*

The `$SYSLOCAL` directory is architecture-specific, and is found under `15JUL94/$ARCH/`. `INSTEP1` will create it using your site name, *i.e.* `15JUL94/$ARCH/SYSTEM/$SITE/`. Thus it is the same for all hosts of the same type in a given installation.

Once the directory exists, any files in the `15JUL94/$ARCH/SYSTEM` area are copied to it. Then a set of standard files are copied to this new `$SYSLOCAL` area from the `$SYSUNIX` area `15JUL94/SYSTEM/UNIX`, only if they don't already exist in the `$SYSLOCAL` area. In this way, all the necessary shell scripts and setup files are copied, but any architecture-specific versions are used in place of the generic ones.

The files conditionally copied from `SYSUNIX` are: `CDVER.CSH` and `.SH`, `CCOPTS.SH`, `LDOPTS.SH`, `ASOPTS.SH`, `INCS.SH`, and `LIBR.DAT`. Some of these will need to be edited after `INSTEP1` completes, at least for new ports of *AIPS*.

For Convex and DEC–Ultrix installations, some additional files are converted to the Korn or System V shells at this point (see above). These are: any `*.SH` files, and copies of `AIPSEXEC`, `LIBR`, and `LINK` from `$SYSUNIX`.

If you had modified any `$SYSLOCAL` files in some older versions of *AIPS* (either `15OCT92` or `15APR92`), you should now look at differences between the distributed version and your customized version. Also, if you had added any purely local files, you might want to copy them to the new `$SYSLOCAL` area from the old one.

### *2.3.8 Define the "aips root" Area*

As already mentioned, this area should ideally be the login area of the `aips` account. It should also be the directory you were in when you started `INSTEP1` and where the `15JUL94` directory is located. If the environment variable `AIPS_ROOT` is not defined, you will be given the name of the current directory (with `/tmp_mnt` removed from it in case you are using the automounter) and asked to confirm if this is the `AIPS_ROOT` area. If not, you will be asked to enter the directory name for it. If what you tell it is not a directory, you will be given the option of creating one.

If the `AIPS_ROOT` variable is already defined, it just asks you if it is correct. Again, you are given the option of re-entering it (but not creating it) if it's wrong.

Finally, the script will stop dead in its tracks if there is a period (dot, ".") in the `$AIPS_ROOT` pathname. Such a name, *e.g.*, `/u/j.doe/aips`, will cause serious problems to `INSTEP2`, and it is far simpler to either rename the directory or pick one without a dot in its name.

### *2.3.9 Printer Setup*

The manner in which printers are configured has changed in 15JUL94. In older versions, the installer had to insert actual shell variable assignments in the PRDEVS.SH file which got copied to $AIPS_ROOT. Now, the definitions are placed in $NET0/PRDEVS.LIST, which is a file similar in appearance and concept to the other list files (HOSTS.LIST, TPDEVS.LIST, DADEVS.LIST).

There is a sample PRDEVS.LIST file in $SYSUNIX which contains some definitions that are used in the Charlottesville *AIPS* installation. The actual in-use version is somewhat more interesting and is shown here:

```
#-----------------------------------------------------------------------
# PRDEVS.LIST -- List of AIPS Printers for your site
#-----------------------------------------------------------------------
# Place a one-line entry for each printer that you wish to make
# available to AIPS users.  The first column should have the printer
# name as known to /etc/printcap or equivalent.  The second is the AIPS
# printer type, one of PS, PS-CMYK, QMS, TEXT, or PREVIEW for PostScript
# (tm), Color PostScript, QUIC (QMS or Talaris), plain text, or screen
# previewers respectively.
#
# Column three specifies a comma-separated (NO SPACES!) list of options
# for each printer.  Right now there are four possible values: NONE,
# DUP to indicate a duplex printer, DEF to indicate a default printer,
# and BIG[=VALUE] to indicate that this is _the_ "big" printer and any
# print job with more lines than VALUE (default 1000) will be diverted
# there regardless of what printer the user chose.
#
# This version for NRAO-Charlottesville.
#-----------------------------------------------------------------------
# printer  type     AIPS options   Description

ps3dup     PS       BIG=1000,DUP,DEF Library FAST PostScript printer, Duplex
ps3        PS       None             Library Fast PostScript printer, normal
ps1        PS       NONE             Postscript printer downstairs
ps2        PS       NONE             Postscript printer in room 215
pscolor    PS-CMYK  NONE             Color Postscript printer, AIPS Cage
35mm2k     PS-CMYK  NONE             Postscript 35mm slides
ghostview  PREVIEW  NONE             GNU Ghostview X-windows previewer
pageview   PREVIEW  NONE             Sun OpenWindows previewer
lpgb       REMOTE   NONE             QMS1725 Postscript printer in GREEN BANK
$PRINTER   PS       NONE             Printer ${PRINTER-not.available}, type PS
```

There are four main fields in the file; the first three must be delimited by a space or spaces, while the fourth (the description) may contain spaces. The third field (options) may contain multiple items but these must be separated by commas only. The comments in the file describe the format and allowed values fairly well. There are two interesting features shown above that may be useful to other sites. The first is the "remote" printer lpgb, and the second is the last printer $PRINTER.

For any printer you define as "remote", the file to be printed will be passed to a shell script of the same name as the printer. In the above example, there is a shell script lpgb that takes one argument (the filename) and, via a rsh command, copies it to a remote printer. In cases where you want to give access to remote systems without having your system administrator configure extra remote printers for the local system, this can be quite useful. However, the scheme is only as good as the shell script you supply.

If you define **$PRINTER** as one of the printers exactly as indicated in the last line of the example above, this definition will only be useful if the user has defined this variable in their environment. If the user has set the variable to **myprint** for example, they will see the line:

```
10.  Printer myprint, assume PostScript
```

whereas a different user who has no printer defined will instead see:

```
10.  Printer not.available, assume PostScript
```

This is a useful way of configuring personal printers for individuals without giving everyone easy access to them.

The **INSTEP1** script will attempt to take the information from an old **PRDEVS.SH** file, if one is found, and convert it to the newer format automatically. If you have the **EDITOR** environment variable defined, you will be given the option of editing the resulting file at that point.

If an older **PRDEVS.SH** file is not found, the script presents you with a summary of the types of printers and the options. Then it checks if there is a file **/etc/printcap**, and if so it will attempt to parse its contents and extract the names of all printers defined therein. The **printcap** file is not universally used, but it can be found on most Berkeley-based systems such as Suns running SunOS 4. For each printer found, you will be asked if you want to include it, its type, any options, and a description. This information is added rather crudely to the **$NET0/PRDEVS.LIST** file.

If the **printcap** file cannot be found, you will be asked how many printers you want to define, and will be asked for the printer name, plus the same information as outlined in the previous paragraph.

### *2.3.10 Copy system files to the aips root*

There are currently 16 files that will be copied to the **AIPS_ROOT** area from **$SYSUNIX**. If you have a **15JUL93** or **15JAN94** installation and the **INSTEP1** script detects this, only 10 of these will be copied and the remaining 6 will remain untouched. Be warned, however: the script says:

```
INSTEP1:  Will NOT clobber your AIPSASSN.*, AIPSROOT.DEFINE, HOSTS.*,
INSTEP1:  or LOGIN.* files (15JAN94 INSTEP1 clobbered HOSTS.LIST, Sorry
INSTEP1:  The others in $AIPS_ROOT will have to be updated.
```

("clobber" here means overwrite) Most of these are Bourne shell scripts; three are C shell scripts for users with the C-shell as their login shell, and one is just a configuration file. The script will copy them all to the **$AIPS_ROOT** area and set the execute bit on for the copied version and off for the **SYSUNIX** version.

If any of these files already exists in the **$AIPS_ROOT** area, you will be asked for each one, *e.g.*:

```
Warning!  Old AIPSPATH.SH found in AIPS_ROOT
Some older versions of AIPS_ROOT files may be
incompatible with this version of AIPS, especially
if they predate the 15APR92 release

Do you want to MOVE the old AIPSPATH.SH file to AIPSPATH.SH.OLD? (NO)
```

The files copied are:

| | | |
|---|---|---|
| AIPSASSN.CSH | HOSTS.SH | START_TVSERVERS |
| AIPSASSN.SH | HOSTS.LIST | TVALT |
| AIPSPATH.CSH | LOGIN.CSH | TVDEVS.SH |
| AIPSPATH.SH | LOGIN.SH | AIPS.BOOT |
| AIPSROOT.DEFINE | START_AIPS | |
| HOSTS.CSH | START_TPSERVERS | |

The format of **HOSTS.LIST** was changed in **15JUL93**, so if you have an earlier version, you need to make sure it uses the newer **HOSTS.CSH**, **HOSTS.SH**, and **TVDEVS.SH** scripts. In order to maintain compatibility between a **15OCT92** release and a newer one, you should do the following:

1. Move the two directories and make symlinks to `15JUL94`:

```
cd $AIPS_ROOT/15OCT92/$ARCH/SYSTEM
mv $SITE $SITE.OLD
ln -s $AIPS_ROOT/15JUL94/$ARCH/SYSTEM/$SITE $SITE
cd $AIPS_ROOT/15OCT92/SYSTEM
mv UNIX UNIX.OLD
ln -s $AIPS_ROOT/15JUL94/SYSTEM/UNIX UNIX
```

2. Rename the `TEKSERVER` binary to `TEKSRV`:

```
cd $AIPS_ROOT/15OCT92/$ARCH/LOAD
mv TEKSERVER.EXE TEKSRV.EXE
```

### 2.3.11 Run AIPSROOT.DEFINE

This shell script will set the value of the `AIPS_ROOT` environment variable automatically in all of the shell scripts that need it. If you do not specify the directory name for `AIPS_ROOT` as an argument when you run the script, it prompts for the name. `INSTEP1` will provide the name you already entered.

### 2.3.12 Create the AIPS command

In versions of *AIPS* prior to `15JAN94` you were advised to create a file in `/usr/local/bin`. That method has now been superseded. Now the `INSTEP1` script will automatically make a symbolic link in the `$SYSLOCAL` area called "`aips`" that points at the version of `START_AIPS` in `$AIPS_ROOT`. It also makes an additional one called "`AIPS`" for those users who like to work in upper case. Thus, for users to avail of this scheme, they will have to "source" or "dot" the relevant `LOGIN.*` file in their `.login` or `.profile`.

For DEC/Ultrix and Convex systems, the above symbolic links are removed, and the `$AIPS_-ROOT/START_AIPS` script is copied to `$SYSLOCAL/AIPS`, and converted to the relevant shell. A symbolic link `aips` is then made to it. This allows non-Ultrix and non-Convex architectures to co-exist with these systems.

### 2.3.13 Set AIPS Version Names

The system as shipped can accommodate up to three simultaneous versions of *AIPS*. These are usually referred to as `OLD`, `NEW`, and `TST` and reflect NRAO's internal setup. There will be environment variables declared in the `AIPSPATH.*` files that match these names against actual versions of *AIPS* (*e.g.*, `15JUL94`). The `INSTEP1` script first scans the `AIPSPATH.SH` file to see what the existing definitions are, prints these out and asks if these are what the installer wants. If you enter `NO`, you will then be asked what they should be.

For most sites, all three versions should be the same; this is the default as shipped. However, be warned that versions of *AIPS* starting with `15APR92` are **incompatible** with earlier versions (*i.e.*, `15APR91` or earlier). You should not attempt to mix them; memory and other files will be in the wrong places and the `$CDTST`, *etc.*, version-changing commands will not work if you do mix them. There are other incompatibilities, though these are more to do with the data and techniques as opposed to system-type differences. Check the recent *AIPS*letters for details. Also see the above note for compatibility between `15OCT92` and later releases.

Once you have entered the revised version names, the script puts these names back into both `AIPSPATH.SH` and `AIPSPATH.CSH`.

### 2.3.14 Make DA00 (system file) Area

Prior to 15APR92, the *AIPS* data areas had names like DA00, DA01, and so on. The first one was reserved for system files, and the other ones were used almost exclusively for data files. Having the system file area in the same scheme as the data areas often led to some confusion.

The system file area is of necessity host-specific, so the environment variable $DA00 now points to a directory $AIPS_ROOT/DA00/$HOST. Each host in your *AIPS* network will have its own private area like this. The INSTEP1 script will also allow group-write on the $AIPS_ROOT/DA00 and $DA00 areas as well as setting the "setgid" bit (this forces the directory's group-ownership on files created in the directories).

The network scheme relies on the $DA00 subdirectories for all hosts being in the same directory, as various files will be hard linked across them. See the description of SYSETUP later in this document. Also see the note in the next section on NFS mounts and the grpid option.

### 2.3.15 Set up Disks (directories) for this host

Since the 15APR92 release of *AIPS*, the data areas are by default organized so that they all reside under a single directory called the DATA_ROOT. On NRAO systems this is typically /DATA. The convention for disks is that they take the (uppercase) name of the host followed by an underscore and a number. So on a host rhesus, for example, the first two disks would be /DATA/RHESUS_1 and /DATA/RHESUS_2. This will enable the *AIPS* user to select disks from different host machines at startup. The criterion for the da=FOOBAR command line option to the aips command to select disks is simply that the string FOOBAR be somewhere in the name.

For most installations, it will be easiest to make the data root area something like $AIPS_ROOT/-DATA/ and set up the data directories or "disks" as symbolic links pointing to existing *AIPS* data areas. The INSTEP1 script now supports this and assumes you will want to take this approach.

If your *AIPS* network is going to be in any way large (more than about 3 machines), then you will probably save yourself and your systems administrator some work by using the automount facility available on most unix systems. At NRAO, we have gone further than just using basic automounter technology, and we have set up an automounter "DATA" indirect map that greatly simplifies accessing a large number of differently named AIPS data areas. Here is a portion of it:

```
GORILLA_1 -rw,grpid,hard,intr gorilla:/gorilla/aips/GORILLA_1
GORILLA_2 -rw,grpid,hard,intr gorilla:/gorilla/aips/GORILLA_2
GORILLA_3 -rw,grpid,hard,intr gorilla:/gorilla_2/GORILLA_3
```

and the DATA entry in the auto.master map is:

```
/DATA auto.DATA -rw,grpid,hard,intr
```

This shows that three disks from host gorilla are accessible from any machine using these automounter files (which we distribute via NIS/YP) as /DATA/GORILLA_1, and so on. Two of these are on the same physical partition and are separated purely for the owner's logistical use (different projects, *etc.*). Note the grpid option above; it is essential in order to make the "setgid" bit work correctly. Unfortunately not all NFS servers support the grpid option.

Systems that NRAO has used this automounter scheme successfully with include: Sun3, Sun4, Solaris, IBM RS/6000 (AIX 3.2; it did *not* work for version 3.1) Convex, DecStation (Mips/Ultrix 4.0, 4.3, and Alpha/OSF–1). With a little effort, our Linux guru was also able to get it to work. It is possible to use a hybrid scheme with most systems using the DATA automounter map and some using explicit NFS mounts at boot time, but it is preferable to have most systems using the automounter.

The INSTEP1 procedure will ask for the name of the DATA_ROOT area. It will only ask for this the first time it is run. The definition you enter will be inserted in AIPSASSN.CSH and AIPSASSN.SH. If the directory DATA_ROOT does not exist, it offers to create it for you. *Do not let it create this directory if you plan on using the automounter*, as root privileges will be required for that.

Certain tasks (such as the TPMON remote tape dæmons) require the FITS area to exist. This is the default staging area for import and export of FITS-format disk files to and from AIPS. If there already exists a directory or symbolic link $AIPS_ROOT/FITS, or if environment variable $FITS is defined, then nothing is done. Otherwise you are given the option to create the directory in the standard area, or make it a symbolic link to another area. Some sites may want to have a large scratch area on another file-system for this purpose.

If required, `INSTEP1` copies the `DADEVS.LIST` and `NETSP` files from `$SYSUNIX` to the `$NET0`) area (actually `$AIPS_ROOT/DA00`), and checks the latter file to see if this host already has disks defined. If not, it asks how many disks are to be added for this host, and asks you to confirm the name of each one. You can elect to change any or all of the default names at this stage.

For each "disk", it first asks if you want to make it a symbolic link to another directory. If you do, you will be prompted for the name of the directory the symlink should point at. This can be quite useful if you have existing *AIPS* user data areas that you want to make accessible. Do *not* make a reference to an old `DA00` area from `15APR91` or earlier; that is a system file area and has no user data.

If you elect not to make a symbolic link, it offers to create the directory if necessary, and will then automatically create the (empty but *absolutely necessary*) `SPACE` file in each one.

After all "disks" have been entered, you are encouraged by the `INSTEP1` script to inspect the two files mentioned in the last paragraph. If your `$EDITOR` environment variable is defined, the script will ask if you wish to edit them now.

The `NETSP` file overrides some system parameters previously set via the `RUN SETPAR` program, *i.e.*, the `$DA00/SPC00000;1` file. The `TIMDEST` limits and the reservation system — where certain *AIPS* disks can be restricted to certain users or set only for scratch use — are controlled from the single `NETSP` file for your entire network of *AIPS* hosts. The `DADEVS.LIST` file is for indicating which disks are required and which are optional, as well as for user disk selection at startup (on a per-host basis, *i.e.*, select a set of hosts and get all disks from those hosts only).

### *2.3.16 Fill in the Tape Drive Names*

In older versions of *AIPS*, the tape drive assignments were made in a rather complicated manner by editing a pair of shell scripts `AIPSASSN.CSH` and `AIPSASSN.SH`. Since the `15JUL93` release, this method has been replaced with a single, site-wide file `TPDEVS.LIST` that holds definitions of *AIPS* tape drives for all hosts in your network. As with the other list files, it is plain text and automatically set up for you by `INSTEP1`. This file will live alongside `DADEVS.LIST` and `PRDEVS.LIST` in the `$NET0` area. If it exists, and the current host has a tape drive or drives defined in it, the file will not be updated or modified. If it exists and there are no entries in it for the current host, then you are given the option of adding one or more.

There are three columns of information in this file. The first has the hostname where the tape drive resides, in uppercase (no domain name, just the simple host name). The second contains the actual device name, *e.g.*: `/dev/nrst0`. The third (and subsequent columns) contains a comment which is printed out at startup time. Suppose on host `FROBB` (a Sparc at SunOS 4.1.2) you had a single Exabyte tape drive on the *SCSI* bus. The line in `TPDEVS.LIST` would look like:

```
FROBB      /dev/nrst0        Exabyte 8500
```

Then when you start up AIPS on frobb, you will see (after the disk assignments message) another message:

```
Tape assignments:
   Tape 1 is Exabyte 8500 on FROBB
   Tape 2 is REMOTE
   Tape 3 is REMOTE
```

The `INSTEP1` procedure will prompt you for the tape names on your local host only, and ask for a (human-readable) description of each. It is important that you read the machine-specific instructions that `INSTEP1` prints out, as most systems will overload the device name with additional functionality.

As an example, consider the attribute of *no-rewind on close*. This is what forces the tape drivers *not* to rewind the tape anytime the device is referenced in a `close()` statement, and it is essential that *AIPS* use the correct device name to ensure this (rewinding on close) does not happen. On SunOS version 4, this is ensured by prepending the name with the letter `n`, *e.g.*for the first tape on a SCSI interface, the device name will be `/dev/nrst0` for the no-rewind device and this is the exact name you should use. However, on IBM RS–6000/AIX, the corresponding device name would be /dev/rmt0.1, **but** you should use /dev/rmt0 when entering the information in INSTEP1. The reason is that *AIPS* knows enough to add on the "point-something" for AIX device names.

The ZMOUN2.C routine in the architecture-specific Z routine area will modify the actual device name used where multiple densities are supported. This usually means 9-track tape drives or Exabyte 8500 devices. If you have such a device, you *must* enter it so that the density modification code in the Z routine produces the correct name for different densities.

Here is a brief summary of the form of device names for known systems, and how the ZMOUN2 routine modifies the name:

- SUN4 /dev/nrst<x> for SCSI devices, where you replace <x> with $0, 1, 2, \ldots, 7$ for the tape drive number. *AIPS* will add 8 or 16 to the number if higher density is indicated by the DENSITY adverb; it is imperative that you enter the base number, otherwise the density algorithm will get confused (though not half as confused as you). For Solaris 2 (SunOS 5), see SOL below.
- SUN3 See SUN4 above.
- SOL /dev/rmt/<x><d>n, where <x> is $0, 1, 2, \ldots$, and <d> is the density (l, m, h for low, medium, or high). *AIPS* will modify <d> on the fly depending on the density desired, and the density letter *must* immediately follow the tape number <x>.
- IBM /dev/rmt<x> where <x> is $0, 1, 2, \ldots$ for the tape drive number. Do *not* use a full name, *e.g.*, /dev/rmt0.1 as *AIPS* will add the correct qualifier. The density selection *only* works for the standard rmt driver (*e.g.*, third party drivers like wb are not supported).
- CVEX /dev/rmtxx *literally*. *AIPS* will replace the xx with the appropriate numbers. You must preserve the order, i.e. system tape drive zero must be *AIPS* drive 1, system drive one is *AIPS* drive two, *etc.*.
- DEC /dev/nrmt<x><d> where <x> is $0, 1, 2, \ldots$ for the tape drive number, and <d> is the density: l for low, m for medium, h for high. Be sure to set <d> to l initially.
- ALPHA Identical to DEC.
- HP /dev/rmt/<x><d>n where <x> is $0, 1, 2, \ldots$, and <d> is the density (l, m, h as for DEC). The n is for no-rewind. *AIPS* will modify the <d> according to the density desired on the fly. As with SOL, the density indicator must immediately follow the tape number.
- LINUX /dev/nrst<x>, similar to Sun, for SCSI tape drives, or /dev/nrmt<x> for non-SCSI drives. There is some local tape drive support in 15JUL94 on Linux; however, most PC tapes are quarter-inch cartridges which cannot backspace over files/records and are thus unusable from within *AIPS*.
- SGI Identical to HP.

The relevant code containing the logic for handling tape drives is found in ZMOUN2.C and ZTAP2.C in the Z-routine areas for the different architectures. If you modify this code for your system, please let us have a copy of the modified (working!) code along with a thorough description of your system (hardware, operating system, any relevant patches), tape controllers, tape drivers and the tape drive itself. Also remember that the rules for tape names and behavior (*e.g.*, reporting status of drive, error codes) are probably the least standardized thing in the Unix environment, and what works for your drive may not work for someone else's. There isn't even anything useful in the Posix standards about mag tapes (please contact the author if you know this is not so!).

### *2.3.17 Make Preprocessor and other utility programs*

Before any programs, including the preprocessor, can be defined, the procedure needs to know what the native Fortran and C compilers are on the current host. This information is requested by `INSTEP1` even if you are doing a binary installation. Its first guess is obtained from the `$SYSLOCAL` file `CCOPTS.SH` and the output of the `$SYSUNIX` file `FDEFAULT.SH`, which come pre-configured for several well-known *AIPS* architectures. You will be asked to confirm if this guess is correct or not. For Sun3 and Sun4 systems, if the C compiler is not set to `acc`, the script strongly urges you to change it to this (ANSI compliant C compiler), as it is what NRAO now uses in-house. This is not the same as the (Kernighan and Ritchie style) default C compiler available in SunOS 4.1.

For new ports the first guesses at the compiler names may well be wrong. If you enter `NO`, it will ask you for the names of both compilers. You can either enter the command name or the fully qualified path name, *e.g.*, `f77` or `/usr/lang/f77` on suns. The latter form is necessary if the compiler directory (in this case, `/usr/lang`) is not in your default path. Note on Solaris–2, the compiler location is probably in `/opt/SUNWspro/bin`.

If you change what the default Fortran or C compiler is, the script attempts to update this information in the relevant files. For Fortran, the only one it can update is `$SYSLOCAL/LDOPTS.SH`, but you will need to modify `$SYSUNIX/FDEFAULT.SH` by hand (if `$EDITOR` is defined, it offers to start the edit for you now). For C, the file `$SYSLOCAL/CCOPTS.SH` is automatically modified.

If you do not specify full pathnames for the compilers and they are not in your default path, the `INSTEP1` procedure will stop immediately. You should either restart and specify the full pathnames, or modify your `PATH` to add the relevant area in your `.login` or `.profile`, log in again, and restart.

There are currently five utility programs that need to be compiled for the source-only installations, before any of the *AIPS* tools, including the compile and link scripts, can work. These are:

1) `PRINTENV` to print out all environment variables;
2) `NEWEST` to determine which of two files is newer;
3) `PP.EXE`, the Fortran pre-processor;
4) `F2PS`, the Fortran-to-PostScript filter; and
5) `F2TEXT`, the Fortran-to-text filter.

For binary installations, pre-built versions of these are copied from the `$AIPS_ROOT/BIN/$ARCH` area, and no re-compilation should be necessary. For source-only installations, the `INSTEP1` script will attempt to build the programs for you.

The first of these is only needed on systems that do not have `printenv`, or where the system-supplied `printenv` cannot handle the large environment needed to rebuild *AIPS*. If the script finds `printenv` in the path anywhere, it will simply make a symbolic link `PRINTENV` in the `$SYSLOCAL` area to it. Otherwise, it will make a symbolic link to `$SYSUNIX/PRINTENV.C` in `$SYSLOCAL/printenv.c` and compile it. The only qualifier on the C compile command used is `-o` to specify the output file. You may need to do this by hand if your `printenv` command cannot handle more than about 200 long (50–60 character) environment variables. Most systems can easily handle this load.

The second file is `$SYSUNIX/NEWEST.C` which again is copied (for all architectures this time) to `$SYSLOCAL` as `newest.c` and compiled with the `-o` qualifier to make `NEWEST`.

A symbolic link is made to the preprocessor source `$SYSUNIX/PP.FOR` via `$SYSLOCAL/pp.f` and a Z-routine to translate environment variables `15JUL94/APL/DEV/UNIX/ZTRLOG.C` is also symlink-ed as `$SYSLOCAL/ztrlog.c`. The latter is compiled with the `-c` qualifier and a `-I` reference to the *AIPS* include area which should create `ztrlog.o`. On IBM RS/6000 and HP systems, the trailing underscore in the declaration of `ztrlog.c` is removed before the compilation. On Cray systems, this is also done and the reference is converted to uppercase.

Then the preprocessor itself is compiled and linked in one step, including the `ztrlog.o` object file. For IBM's, the `-lc` option is added to force the linker to use the C language version of `getenv()` instead of the fortran version. For Linux, a three-step process is used: (1) run `f2c` on the Fortran source, (2) change the references to `CALL EXIT` in the resultant C code, and (3) compile the edited C code with `gcc`. Also, on HP systems, the `CALL EXIT` is replaced with `STOP`.

Finally, the F2PS and F2TEXT filters (new to the 15JUL94 release), are compiled. These are relatively simple programs that take the Fortran-formatted output of AIPS and other tasks, and format it for either PostScript or plain text printers. F2PS will use the value of "number of lines per printed page", item 34 in the SETPAR menu, to decide what format and orientation is used. The table below summarizes the command line arguments and their effects on F2PS:

| Argument | Lines | Argument | Lines | Orientation | Font size |
|---|---|---|---|---|---|
| -S | 97 | -S/-A4 | 95 | Portrait | 6.8 |
| -R | 61 | -R/-A4 | 58 | Landscape | 8.0 |
| -M | 54 | -M/-A4 | 52 | Landscape | 9.0 |
| -L | 52 | -L/-A4 | 47 | Landscape | 9.25 |

The arguments are passed to the program via the ZLPCL2 shell script in $SYSUNIX. As shipped, this script causes the program to assume "A" size paper (8.5 × 11 inches); if you wish to use A4 size instead, edit the shell script and add the -A4 argument in the appropriate place.

Both F2PS and F2TEXT will accept Fortran formatted output, *i.e.*, column 1 of the files they process is assumed to be the special formatting column as specified in the Fortran 77 standard. The programs will interpret this column, and act accordingly (new page for form feed, skip 1 or 2 lines as appropriate). F2TEXT will take an extra optional argument, -e, if the last page of a print job is to end with a form feed; this may not be necessary with most printers, and the default is no ending form feed. Again, change this as needed in ZLPCL2.

If any of the above task builds fail, you will have to investigate the problem, correct it if possible and compile the program by hand.

### *2.3.18 Shared and Debug Libraries*

You are given the option to choose whether you want to build "shared" or dynamic *AIPS* libraries on some systems. In addition, on all systems you are also asked whether you want to build a second set of libraries that are compiled completely in debug mode. The shared option is only available on Sun3, Sun4, Solaris, and HP systems, and they are not totally debugged on HP.

The binary tapes are shipped with non-debug, non-shared *AIPS* libraries, and the rest of this section is not relevant for such installations.

The advantage of using shared libraries is that it reduces the size of the binaries (executables, files in the LOAD area) considerably. Depending on your system, the total size of the binaries may decrease by about 50 megabytes. However, there is a trade-off with the amount of swap space you will need. At NRAO we have found it necessary to use anywhere from 70 to 110 megabytes on Suns for swap space (more on larger systems like the Sparc10). Obviously, if you have to increase the amount of swap on many systems, you will end up using more disk space than you will save by reducing the size of the binary files. Conversely, on a single system or an *AIPS* network of fewer than about three systems, the savings may be worth it.

On HP–UX 9.01 it has been noted that using shared libraries seems to cause fatal errors in critical parts of *AIPS*. However, on an older version of the OS (8), the initial port was able to make use of shared libraries with no apparent side effects. Caution is recommended and you may want to stick with the non-shared libraries if you have a HP system. As the NRAO *AIPS* group still does not have access to an in-house HP–UX system, our progress in debugging this problem has been slow.

Even if you ask for the shared libraries, the regular non-shared versions have to be built anyway. Thus, if you chose share-able libraries, all it really takes to convert a given task to use the static libraries is a relink. The mechanism for deciding whether or not to link against shared libraries is the presence or absence of the file $SYSLOCAL/USESHARED (the contents or size don't matter). Also, there is a list $SYSLOCAL/NOSHARE.LIS containing simple (basenamed) filenames of programs that must *not* be linked against the shared libraries. This file exists for SUN4 installations (most of the entries in it probably don't need to be there) but not for other architectures. You can always add to this list if you find a task that does not behave well with shared libraries.

Previously, linking against shared libraries had sometimes produced warnings about unresolved references. On Solaris 2.x, this is a fatal error and the resulting task simply will not run. The *AIPS* group has tried to remove all such occurrences of these references, so you should not see this message anymore. If you do, it means something is not right.

Finally, the key on whether or not to generate a debug set of libraries is determined by the presence or absence of the file `$SYSLOCAL/DOTWOLIB`, which was called `DOTWOBIN` in older versions. This option will not only generate a second set of libraries under the `LIBRDBG` directory parallel to the `LIBR` area, but it will also leave the pre-processed source code around in the `$PREP` area too, as it is necessary for debugging purposes to have access to the source code.

You most likely will not need the debug libraries. This feature is useful at NRAO, and probably at other sites where active *AIPS* code development or modification or debugging is occurring.

### *2.3.19 Create HOSTS.LIST and/or add this host to it*

This file is the hub of the "network" scheme for *AIPS*. It lists, for a given site, all the hosts that `aips` and/or the *AIPS* TV server will run on, their architectures, and a brief half-line summary of what the machine is. Versions of *AIPS* `15JUL93` or earlier had a different format for this file, but the newer format *should* be backwards compatible, as long as your older versions of *AIPS* use the newer shell scripts in the `$AIPS_ROOT` area. In other words, using the new format will be possible in most cases for `15OCT92` and `15APR92` versions if you want to use them as well as `15JUL93`, `15JAN94`, or `15JUL94`.

As supplied, the file looks like this:

```
AIPS hosts.  Spacing/position is significant for columns 1 and 2.  The
first item must be in column 1 and there must be exactly two spaces
between it and column 2.

Fields: 1 = "+"  host runs AIPS and can/will run a TV server (e.g. XAS),
            "-"  host is a TV server only (does not run AIPS), and
            "="  host is an X11 display terminal (does not run AIPS).
        2 = hostname, uppercase (NOT qualified, i.e. FOO, not FOO.EDU)
        3 = architecture (CVEX, IBM, SUN4, SUN3, HP, ALLN, CRI, IBM3090,
                          DEC, SGI, ...)
              Note: byte order and f.p. format MUST be identical on all.
        4 = Site.  Use same string for all your entries at one site.
        5 = Default server for X terminal (ignored for AIPS hosts, TV
              servers)
        6 = description (for people and your own benefit).

SAMPLE SETUP FOLLOWS (you can delete any lines starting with "*", and
        use a "+" or "-" in column 1 instead)

1  2          3      4        5           6
*  GOLDY      CVEX   MYSITE   NONE        Convex 3800, Computer center
*  ZIPPY      IBM    MYSITE   NONE        IBM RS/6000-560, Astro lab
*  HAPPY      SUN4   MYSITE   NONE        Sun 4/75 color, gx accellator
*  OLDIE      SUN4   MYSITE   NONE        Sun ELC mono, TV server
*  OTHER      DEC    MYSITE   OLDIE       DecStn 3100 color, TV display
```

These five machine entries are of course fictitious and are for illustration. The asterisks in column 1 are present for the `INSTEP1` procedure to detect and remove.

The comments for fields 3 and 4 are old. You can in fact mix different architectures in the same `$AIPS_ROOT` area, but you have to assign different site names. As already mentioned, at NRAO/Charlottesville the `HOSTS.LIST` file has two sites: `NRAOCV` for the Suns and IBM RS/6000 systems, and `VCOARN` (NRAOCV spelt backwards) for systems with reversed byte order: Dec/Ultrix, Dec/Alpha, and Intel/Linux.

When `INSTEP1` is run for the first time, it checks the `HOSTS.LIST` file to see if the first machine above ("goldy") is still in place. If so, it removes all entries that start with an asterisk. Then for each host it is run on, it asks for a short description. Then it inserts a line representing that host in the file.

You will then be asked if you want to add entries here for other hosts of the same architecture that you want to run *AIPS* on. The script will continue to ask you for hosts and descriptions, inserting them in the file, until you enter a blank line for a host. While you do have to re-run `INSTEP1` on a sample machine of each architecture in your *AIPS* network, you do *not* need to do so for the additional hosts of the same architecture.

After you have entered all *AIPS* hosts for the specific architecture, the `INSTEP1` script will allow you to enter both TV servers and TV display machines for *AIPS* in the same manner as you entered the regular hosts. Here a "TV server" is a computer that will run the `XAS` image display for *AIPS*, but with the intent of displaying it elsewhere. Conversely, a "TV display" host is one that receives this display. There is a limit of one instance of a TV server (`XAS`) per *AIPS* host or TV host. Conversely, given enough hosts, there is no such limit on the number of TV servers you can display on a single X11 screen; you just need to have each server come from a different host.

Using the `tv=` command line option of the `AIPS` command, you can in fact redirect (via X11) the display anywhere, *i.e.*, you are not restricted by the display hosts you put in this file. However, it is useful to be able to specify, for a given TV display host, which other host is its *default* TV server. Then when the startup scripts determine (from `who am i` or variants thereof) what system you are actually sitting in front of, they can automatically have `XAS` start up on the correct server host.

You can also choose, using the `tv=` command line option, a separate TV host other than the system on which you are running `AIPS`. In other words, the `AIPS` host, TV host, and TV display can be three separate systems. However, the TV host and `AIPS` host have to be regular *AIPS* systems listed in `HOSTS.LIST`.

### *2.3.20 Make the X11 TV Server (XAS)*

You can choose to have `INSTEP1` attempt to make the default *AIPS* TV server — XAS — for you. This TV server should work on all X11R4 and X11R5 based windowing systems (including OpenWindows versions 2 and 3, Dec-Windows, Motif, fvwm and twm/tvtwm).

If you answer `YES`, the script will unpack the archive in the directory `15JUL94/Y/SERVERS/XAS`. For Linux, the unpacking is performed via the `UNSHR` program provided in the `YSERV` area, as the bash shell gets very confused with the characters in the `Makefile` in the `XAS.SHR` archive. Then it will look for the X11 include and library areas. It attempts to do so in a consistent way, by first checking for the file `lib/libX11.a` in a given area, and if it finds that, double-checking to make sure `include/X11/Xlib.h` is in the same area. Also it uses some predefined values for some systems. Here is a summary of the "logic" behind this:

- For Linux, use `/usr/X386/lib` and `/usr/include/X11`.
- If `$OPENWINHOME` is defined, use its include and lib areas.
- For Alpha systems, use `/usr/include/X11` and `/usr/include/lib`.
- If `/usr/openwin` exists, treat it as for `$OPENWINHOME` above.
- Check `/opt/local/lib` and `/opt/local/include`
- Check `/usr/local/lib` and `/usr/local/include`
- Check `/usr/lib` and `/usr/include`
- Check `/usr/lib/X11R5` and `/usr/include/X11R5` (valid for some HP systems)
- Check `/local/X11R5` (NRAO-specific area)

Obviously this set of areas could probably be more comprehensive, but complexity of the script is a concern too. If you think it guesses things incorrectly, you may just want to skip this section and do it yourself in the Makefile by hand later.

If the script cannot find the include and/or library areas for X11, it will prompt you for them. You should enter the *exact* directory locations of the include and library directories for your X11 installation.

XAS can take advantage of the "MIT shared memory" option available in some X11 servers. `INSTEP1` attempts to figure out if this is possible by running the standard X11 program `xdpyinfo` and searching for the string `MIT-SHM` in its output. It also searches for the file `X11/extensions/XShm.h` in the include area previously specified. If and only if: (a) it finds this and (b) the `xdpyinfo` check worked, it will assume that XAS should be made with the shared memory code enabled. You will be asked for confirmation before the "make" starts. If the wrong choice is made, and shared memory is enabled during the make, most likely the compile or link will fail. To change the option permanently in `XAS`, edit the `Makefile` in the `$YSERV/XAS/` directory to change `SHMOPT` to blank for no shared memory, or `-DUSE_SHM` otherwise, and then type `make` to remake it (Warning: `$LOAD` must be defined before you do this, and any `*.o` files should be removed first; there is no "clean" target in the Makefile).

If you have a shared-memory-enabled version of `XAS`, it will automatically detect if the server and display hosts are different; in this case it will not even attempt to use shared memory. However, if you want to inhibit its use, see the section below on X resources; the `AIPStv*useSharedMemory` resource can be turned on or off prior to starting the TV server.

On Suns, you will need to have the System V IPC options configured in your kernel. Check Sun's System and Network Administration Manual, chapter 9, for the details. In particular, you will want to have this in the configuration file:

```
options IPCSHMEM        #  System V IPC shared-memory facility
```

If it is not (or is commented out), then you have to rebuild the kernel — or have your sysadmin do it for you — to make use of the X11 shared memory extension.

Once the include and library areas are known to `INSTEP1`, it then modifies the `Makefile` to insert the revised definitions of `INCDIRS` and `LIBDIRS`. It also modifies `SHMOPT` if necessary. For Solaris, it adds `-lsocket` to the `STDLIBS` variable. Then it defines `LOAD` (the ultimate destination of the `XAS` binary) and calls `make`. This may or may not work. If not, don't panic. Just look at the error messages, try to see if it's looking at the correct library and include file area, and if all else fails, contact an X11 guru. If you can't find one, send email to `aipsmail@nrao.edu` and we'll do what we can to help. However, a failure in making XAS will *not* stop you from continuing with `INSTEP2` and the following steps.

### *2.3.21 Create Architecture-specific Install area*

In preparation for the next steps, the `INSTEP1` procedure makes an architecture-specific installation directory `15JUL94/$ARCH/INSTALL` and makes symbolic links to the files `INSTEP2`, `INSTEP3`, and `INSTEP3` in `$INSUNIX`. Obviously this is only done for source-only installations; there is normally no need to run `INSTEP2` or subsequent steps for binary installations.

You will be presented with a set of instructions for editing and checking various files at the end of `INSTEP1`. Details of what these mean are expanded on in the *AIPS Unix Installation Summary* and is not repeated here in the interests of conciseness. The important thing at this stage is to get `INSTEP2` started as soon as you can, as it can take a long time to finish. So you should check the compiler options, define the programming environment variables, and start `INSTEP2` as soon as you are ready.

## 2.4 Additional Setup for New AIPS Ports

This section describes some steps that may be needed in addition to the customization outlined at the end of the `INSTEP1` script. Most of it is only relevant if a new *AIPS* port is being done, but some sections may be of interest to those who want to — or have to — develop or enhance the system.

### *2.4.1 Local Source Code Directories*

Historically, there was often a need for each installation of *AIPS* to perform some local code development; Unix was not very widespread and there were many different architectures and systems to choose from. Since these "bleeding edge" days, there has been somewhat of a shakedown in the Unix software market, and standards bodies have rushed in trying to make things easier (except for mag-tapes; see author's complaint above!). Also, NRAO now has a moderately diverse selection of Unix based systems in-house that are accessible to the *AIPS* group.

If you are porting *AIPS* to a previously untried architecture, you will most likely have to create several new areas. The best way is to edit `$SYSAIPS/AREAS.DAT` and insert new area definitions. Typically, you will want one for Z routines somewhere under `APL/DEV`, and a system area analogous to `$SYSSUN`. This is the bare minimum, as you may also need a Q-routine area and an include area. Once you have modified this file, you should run `$SYSUNIX/AREAS` to regenerate `AREAS.CSH` and `AREAS.SH` in the same directory.

You will want to define a new architecture in the various shell scripts that need it: `START_AIPS`, `FDEFAULT.SH`, `TVALT`, and others. Just look to see where the `ARCH` variable is used/checked. Then you need to create a new `LIBR.DAT` file. The easiest way to do this is to take the closest architecture, e.g. for a hypothetical port to a Berkeley-like system, take the `SUN4` one and change things like `$APLSUN` to whatever you called the new architecture-specific area. For additional information about `LIBR.DAT`, see section 2.4.5.

The Z-routines are stored under `15JUL94/APL/DEV` (*i.e.*, `$APLGEN`) and this area has two main branches: `UNIX` and `VMS`. Under the former there are again two branches: `BERK` for Berkeley-like (BSD) systems and `BELL` for systems based on AT&T System V. There are a large number of routines (over 240 files) in the top-level `$APLGEN` area; many of these are wrappers for the machine-specific routines found in lower levels. This also applies to the other routines; the *AIPS* group has tried to generalize the lower level routines as much as possible, or even eliminate them.

Under the `APL/DEV/UNIX/BELL` area, there are separate directories for Cray (`CRI`), Hewlett-Packard 9000 series 700 (`HP`, Intel/PC (`LINUX`), Masscomp (`MASC`), Silicon Graphics (`SGI`), and SunOS 5/Solaris 2 (`SOL`). Under the `APL/DEV/UNIX/BERK` area, there are directories for Alliant (`ALLN`), Convex (`CVEX`), Dec Alpha/OSF–1 (`DEC`, IBM RS/6000 (`IBM`), Sun (`SUN`), and Vax (`VAX` for BSD Unix). There are subdirectories for `NRAO1` and `VLAC1` in the `CVEX` area (for specifying that we had IEEE hardware). However, the IBM 3090 area is a subdirectory of the IBM RS/6000 area. Similarly, in the Dec-Alpha (OSF/1) area, there is an `ULTRIX` subdirectory for the few routines that are different on Ultrix (4.3).

To maintain the hierarchical structure, you should position any new Z-routine directory such that it accurately reflects your system. for example, `15JUL94/APL/DEV/UNIX/BELL/SCO` might be the choice for a hypothetical port to the Santa Cruz Operation's version of Unix for PC's. Our System V UNIX system support has improved drastically with the advent of the SunOS 5 and HP ports. Note however that the IBM RS/6000 AIX port is firmly in the Berkeley area, despite its system V nature (it just has excellent Berkeley support and *AIPS* was ported to it before we rejuvenated our System V knowledge).

Y-routines are stored under `15JUL94/Y/DEV` and there is a directory for TV servers (`SS` which supports the screen servers XAS (for X11 including Sun's OpenWindows), SSS (Sun's obsolete Sunview only) and XVSS (Sun's OpenWindows only). The latter two are rarely used anymore. There are older areas here too, including one for Virtual TV (`VTV`, when the display is on a remote TCP/IP system; obsolete now). Other areas are for older image display hardware such as the IVAS and DeAnza. These areas have already been listed at the start of this document (Section 1.8.2).

In the exceedingly unlikely case that you have no TV (this now means that you have no access to X11, and don't have any older style hardware image display device like the $I^2S$), modify the `LIBR.DAT` to use `15JUL94/Y/DEV/STUB` for no TV support. In the even more unlikely case that you have an image display device that does not support X windows and for which there is no existing support in *AIPS*, you will need to create a new Y-routine directory under `Y/DEV` somewhere. You can safely postpone any Y-routine development by using the stubbed routines (see "Object Libraries and Source Code Search Paths", section 2.4.4).

If you do develop Y-routines, please make them *device specific only*, and try not to use operating system specific code. If you need to make system calls in a Y routine, do it through the Z routines or make up a new system-specific Z routine if you have to. Also, try to minimize the amount of device-specific code and make as much use of the generic Y routines as possible.

Finally, the Q-routines are stored under `15JUL94/Q/DEV`. The structure basically has two subtrees, one for Floating Point Systems (`FPS`) array processors, and the other for the so-called pseudo-array processor routines. The latter includes vectorized routines for some architectures. FPS models supported include:

- `16B/120B`, for 16 bit model 120B array processors

- `16B/5000`, for 16 bit model 5105, 5205, *etc.*
- `32B/190`, for 32 bit model 190

With the decline in use of Array Processors, the number of areas for the pseudo-AP (PSAP) routines has grown where they now outnumber the classical AP areas. NRAO has not had any active FPS array processors used for *AIPS* in quite some time. There are separate areas under the `15JUL94/Q/DEV/PSAP/` directory for:

- `ALLN` Alliant FX vector processors
- `CRI` Cray Research (UniCos)
- `CVEX` Convex vector processors
- `DEC` (empty)
- `HP` (empty)
- `IBM` RS/6000 (AIX) optimized PSAP routines; there is also a subdirectory for mainframe-specific routines (`IBM/3090/`).
- `SGI` (empty)
- `SOL` (empty)
- `SUN` Sun 3/4 optimized PSAP routines
- `VMS` VAX/VMS specific PSAP routines

For new ports, it's most likely you won't need one anyway as the generic pseudo-AP routines will do a good job on most systems. Otherwise, you should logically position your new directory in the above scheme.

### *2.4.2 Environment Variables for Areas ("logicals")*

Most of the directories in the *AIPS* directory tree have an environment variable associated with them. This practice comes from the VMS version where logical names were extensively used; these variables are often referred to as "logicals", "logical names" or just "areas". Most of these are defined in `AREAS.CSH` and/or `AREAS.SH` (former for C shell, latter for Bourne/Korn/BASH shell). Both of these files are automatically generated from the `$SYSAIPS/AREAS.DAT` file by the shell script `$SYSUNIX/AREAS`. Note that in the generic `$SYSUNIX` versions of the `CDVER.*` files, there is an `echo` statement which is designed to prompt you into putting the local definitions there. If you have already put them in `AREAS.DAT` and regenerated the shell scripts, there is no need for this and you can un-comment this from the `$SYSLOCAL` versions of the `CDVER` files.

Some UNIX systems may not be able to handle the size of an environment resulting from so many definitions, as mentioned earlier. The limiting factor is the value of `NCARGS` as defined in the file `/usr/include/sys/param.h` on most systems. This is the maximum number of characters that can appear in an argument list. Since the environment is passed as part of the argument list, a large environment can start to cause problems for "wild card" operations like `ls long_directory_path_name/*`. `NCARGS` varies somewhat but on most modern systems it is usually in excess of 24000. Convex-OS (as of release 9.0 at least) is an exception and is closer to 12000. On systems where it is smaller, you may find it necessary to eliminate some of the definitions. Alternately, you may be able to rebuild or reconfigure your system kernel with a sufficiently large value for `NCARGS` (or have your sysadmin do this). However, removing "logicals" for areas you may have deleted or don't need, and/or making sure the `$AIPS_ROOT` is a short name, are probably simpler solutions.

### 2.4.3 Login Procedures

Almost all of the *AIPS* procedures are written in Bourne shell syntax and have been exercised under logins where the default shell is either the Bourne shell, Korn shell, GNU Bash, or (most often) the C shell. Depending on your choice of shell, either `LOGIN.CSH` or `LOGIN.SH` should be incorporated into the login procedure for the AIPS account via:

`source LOGIN.CSH` (C shell syntax for `.login`)

or

`. $HOME/LOGIN.SH` (Bourne/Korn/Bash shell syntax for `.profile`)

There have been many problems with the use of `tcsh` as a login shell reported by several users; care should be exercised if you use this shell. Also, it is important to refer explicitly to `LOGIN.SH` to prevent the "dot" command from picking up the generic `$SYSUNIX` version of the file (and telling you that `AIPS_ROOT` is undefined!). Any other accounts that intend on using *AIPS* should also have one of these lines — or similar — in their login/profile file(s). However, please note that you may run into file locking problems and get multiple versions of `AIPS1`, `AIPS2`, *etc.*. with multiple accounts and/or multiple groups. This is a known problem on IBM RS/6000 AIX systems and there is no quick solution.

In addition, watch for the protection mask on the `/tmp` area; on some systems the "t" bit is set so as to prevent one user from deleting another user's file, even though the protection mask on the file itself permits such.

The values for `$SYSLOCAL` and `$SYSUNIX` are generated by `AIPSPATH.SH` and its `.CSH` sibling. The environment variable `PATH` is extended to include the current working directory ("dot" or "."), followed by `$SYSLOCAL` and `$SYSUNIX`. These are inserted at the beginning of the variable, and all the other elements in `PATH` that don't match these are then appended. *AIPS* definitions (environment variables) can be reset to whichever version you want through typing `$CDOLD`, `$CDNEW` and `$CDTST` if you have multiple versions. These strange creatures are the lowest common denominator solutions for all flavors of UNIX and shells. They are executable environment variables, something like symbols in VMS. The path can be restored to pre-`AIPS` level by setting `PATH` equal to the value of `TPATH`.

You may want to insert one of these `$CD...` in the `.login` or `.profile` of any accounts that will be doing significant programming in *AIPS*. However, this is only really needed for installation, programming and moving quickly around the directory tree. You will need to at least call `$CDOLD` or `$CDNEW` or `$CDTST` interactively before starting `INSTEP2` as it will fail otherwise.

### 2.4.4 Object Libraries and Source Code Search Paths

The *AIPS* system has re-invented the wheel in this area for Unix. This is largely due to the early emphasis on VMS systems which do not come by default with such a utility as the Unix `make`. So instead of editing a Makefile, the installer may have to edit files called `LIBR.DAT` and `INCS.SH` (in `$SYSLOCAL`) which perform much of the same purpose. It is *critical* that this is done right, as otherwise the resulting binaries (executable images) will not work correctly, if they even get made in the first place. These setup files will have been copied by `INSTEP1` to the `$SYSLOCAL` area.

Although it has been mentioned before in this document, it is worth repeating that *for most standard configurations, no editing of* `LIBR.DAT` *is necessary*. New ports, Convexes without IEEE floating point hardware, systems with IIS/IVAS/*etc.*, TV's, array processors and the like are the only reasonable candidates that you will have to modify this file.

### *2.4.5 Setting up LIBR.DAT*

This file has two sections. The first associates subroutine source code directories with one or more object libraries. The second section specifies one or more object library link lists for each program source code area. The entire *AIPS* directory hierarchy is designed so that programs that need the same sets of libraries are grouped in the same directories; this enables such a scheme to work.

There are several versions of `LIBR.DAT` provided with *AIPS*. If you are porting to a new system, you should look in the directories under `$ARCH/SYSTEM` for all architectures for a system you may be able to clone. If none of these are suitable, you can always use the generic `$SYSUNIX/LIBR.DAT` as your starting point, although this will need more work. It will have lines with `---something important---`; these are lines that you have to change, and such comments should be removed in your final version before starting `INSTEP2`.

The first part of `LIBR.DAT` is used to associate object libraries (actually Unix "archive" libraries, *i.e.*, `$LIBR/*/SUBLIB`) with the directories containing the different categories of subroutines. The directories will be searched in a somewhat peculiar order (*not* the order in which they are listed) to locate the proper module to be used in the construction of the associated object library. For example, on the NRAO/CV *AIPS* sun cluster (SunOS 4.1.2), the search path for Z-routine source code is:

```
Z-routines

$LIBR/APLSUN/SUBLIB:0:$APLSUN            (Sun specific)
$LIBR/APLSUN/SUBLIB:0:$APLBERK           (Berkeley (BSD) Unix)
$LIBR/APLSUN/SUBLIB:0:$APLUNIX           (Generic Unix)
$LIBR/APLSUN/SUBLIB:0:$APLGEN            (Generic)
```

The format here is `<aipslibrary>:<loadnumber>:<directory>`. The `<loadnumber>` is only intended for when a site needs multiple TV's (quite rare now, with workstations taking over from the IIS, IVAS, DeAnza and other devices) and/or multiple sets of Q routines (almost unheard of anymore, probably last used on a VMS VAX with AP and pseudo-AP versions). There must be *no leading, embedded or trailing blanks* in these search path definitions. For Suns that use dynamic libraries, the format is still the same; the `LIBR` and `LINK` sun-specific shell scripts will change, *e.g.*, `$LIBR/APLSUB/SUBLIB` to `$LIBR/APLSUB.so` on the fly.

The `MAKEAT` procedure has inbuilt assumptions about the way the directory hierarchy is maintained. Whenever a module is to be compiled with `COMRPL` or linked with `COMLNK`, all the subordinate directories are searched to see if there is a more system-specific version of the module. For instance, a `COMRPL $APLGEN/ZDATE` will call the `SEARCH` shell script which will, on a Sun4 system, check in:

```
APL/DEV/UNIX/BERK/SUN/ZDATE.*
APL/DEV/UNIX/BERK/ZDATE.*
APL/DEV/UNIX/ZDATE.*
APL/DEV/ZDATE.*
```

This list is derived from the `SEARCH<n>.DAT` files in `$SYSLOCAL` (`<n>` here corresponds to `<loadnumber>` and is usually zero). The net effect is that the most specific version of the routine is chosen. However, *the mechanism currently depends on maintaining the relation between directory depth and how machine specific each directory is.* It will fail if you do not follow the general scheme.

The `$LIBR` area itself contains only subdirectories (*e.g.*, `$LIBR/APLSUN/`) which serve as staging areas for object modules that are to be added or replaced in the object libraries (Unix archive files) using the host object librarian (*e.g.*, `ar`). All of the archive files have the same name, `SUBLIB`, but simply reside in different directories. *AIPS* only uses Unix archive files for object libraries, hence this is what they are referred to as in the remainder of this document. On Suns and HPs, the shared library `*.so` files are in, *e.g.*, `$LIBR/APLSUB.so`. This is partly to aid in the possible moving of the *AIPS* code, and a possible future binary distribution of the system. For Suns, see the documentation on the `LD_LIBRARY_PATH` variable.

For most installations, the `<loadnumber>` will be zero in all cases. In the unlikely event that there are two or more types of TV or AP/PseudoAP's to be supported on a single system, the same object module may be needed in multiple object libraries. Below is an example of this, where there are three TV's: an IIS model 70E, an IIS IVAS, and the XAS X-windows TV.

```
Y subroutine source code search paths and object libraries:

Standard routines

$LIBR/YSUB/SUBLIB:0:$YSUB

Non-standard routines

$LIBR/YNOT/SUBLIB:0:$YNOT

Standard Screen Server TV Y-routines

$LIBR/YSS/SUBLIB:0:$YSS
$LIBR/YSS/SUBLIB:0:$YGEN

IIS Model 70 Y-routines

$LIBR/YM70/SUBLIB:2:$YM70
$LIBR/YM70/SUBLIB:2:$YIIS
$LIBR/YM70/SUBLIB:2:$YGEN

IIS Model IVAS Y-routines

$LIBR/YIVAS/SUBLIB:4:$YIVAS
$LIBR/YIVAS/SUBLIB:4:$YGEN
```

In this example, whenever an object module is generated from source code common to the implementation of both the model 70 and IVAS, it gets added or replaced in both `$LIBR/YM70/SUBLIB` and `$LIBR/YIVAS/SUBLIB`. Similarly, object modules common to all three TV's are added/replaced in all three `SUBLIB` libraries. Note the fact that the `<loadnumber>` increments by 2 for TV's. In the linking stage, tasks that call the Y routines will be made into three separate binaries or executable files, one for each TV. The task for the TV server will be put in `$LOAD`, that for the IIS Model 70 will go in `$LOAD2` (same as `$LOAD/ALT2/`), and those for the IVAS will go in `$LOAD4` (same as `$LOAD/ALT4/`). This obviously has disk space consequences.

The normal action of a compile in *AIPS* using the standard `COMRPL` shell script is to simply place the `.o` or object file in the `$LIBR/whatever/` directory. They are later archived (and `ranlib`'d if needed) by other tools before any linking is performed (by `COMLNK`).

The following is the `$LIBR` directory structure as it existed on the NRAO-CV Convex C1 (before we retired it), from which the above segment of `LIBR.DAT` was extracted. The subroutine areas from which the object libraries were generated are given on the right:

```
$LIBR/AIPSUB/SUBLIB   $AIPSUB
$LIBR/APLCVEX/SUBLIB  $APLNRAO1, $APLCVEX, $APLBERK, $APLUNIX, $APLGEN
$LIBR/APLNOT/SUBLIB   $APLNOT
$LIBR/APLSUB/SUBLIB   $APLSUB
$LIBR/QNOT/SUBLIB     $QNOT
$LIBR/QSUB/SUBLIB     $QSUB
$LIBR/QVEX/SUBLIB     $QVEX, $QPSAP
$LIBR/YIVAS/SUBLIB    $YIVAS, $YGEN
$LIBR/YM70/SUBLIB     $YM70, $YIIS, $YGEN
$LIBR/YNOT/SUBLIB     $YNOT
$LIBR/YSUB/SUBLIB     $YSUB
$LIBR/YSS/SUBLIB      $YSS, $YGEN
```

There is nothing special about the `$LIBR` subdirectory names other than being as self-explanatory as possible. However, the library name *must* be `SUBLIB`. If the directories do not exist, `INSTEP2` will automatically create them based on the contents of `LIBR.DAT`. NOTE: *if for some reason a directory is not created,* `COMRPL` *will check for it and attempts to create it, crashing if this fails. If it does, it may mean that* `LIBR.DAT` *is not set up correctly.*

### *2.4.6 Program and Object Library Mappings*

While the first part of `LIBR.DAT` relates subroutine directories to one or more object libraries, the second part maps the program (task) source code areas to the object libraries that should be included in the argument list for the linker (loader). The separation between the two areas is delimited by the line:

```
AIPS stand alone program source code search paths and link libraries:
```

Object libraries are often repeated in these lists. This is to account for the single pass nature of some UNIX linker/loaders (usually `ld`). Libraries must often be cycled several times to resolve all external references. For example, the object library containing modules generated from the `$APLSUB` source code area need to be cycled at least twice in the lists for every program source code area. This is because routines used from the other object libraries specified in the lists often call `$APLSUB` routines. The required order of the link lists that appear in the various distributed versions of `LIBR.DAT` have been determined empirically and with rare exceptions, should apply to most systems. IBM/AIX (on RS/6000 machines) is an exception as the loader there uses multiple passes to resolve all undefined references; so the IBM version of `LIBR.DAT` does not have any duplicate libraries in these lists.

A default list is required in `LIBR.DAT` for each of the standard program source code areas. For Unix, These are:

| | | | |
|---|---|---|---|
| `$AIPGUNIX` | `$AIPNOT` | `$AIPPGM` | `$APGNOT` |
| `$APGOOP` | `$APGUNIX` | `$APLPGM` | `$QPGM` |
| `$QPGNOT` | `$QPGOOP` | `$QYPGM` | `$QYPGNOT` |
| `$YPGM` | `$YPGNOT` | `$YPGVDEV` | |

Different lists are distinguished by a number code (`<loadnumber>` above) inserted between the object library and area, and delimited with colons. Default lists are indicated by the number code `0` and executable files (binaries) generated using them are stored in the default load library, `$LOAD`.

If you have hardwired TV's and/or a real AP, you will need alternate lists with non-zero `<load-number>`s (from 1 to 9). As mentioned briefly above, the purposes of these additional numbers are to create alternative versions of certain programs that use the Y and/or Q routines for each alternative TV and/or AP. The binaries for these programs are stored in `$LOAD<n>` where `<n>` is the `<loadnumber>`. This environment variable will translate to `$LOAD/ALT<n>`. *Only* `$LOAD1` *through* `$LOAD3` *are defined by default on most configurations so you may have to add extra definitions to* `$AIPS_ROOT/AIPSASSN.*` *or* `$SYSLOCAL/CDVER.*`; see the sample section in these files for host `NRAO1` for an example (`LOAD4` is defined there).

The alternate lists and load module areas are only required on those systems where more that one type of array processor or TV device is to be implemented, or both. Most *AIPS* programs use neither Q-routines nor Y-routines (*e.g.*, `$APLPGM` and `$APGNOT`) and their load modules are always stored in the default load area. Other programs will use Q routines, Y routines or maybe both. There are areas for each of these possibilities, and additional areas for not-standard programs (those that don't quite comply with *AIPS* coding conventions but are too useful to do without), and the Object-Oriented Fortran areas. These areas have environment variables `$QPGM`,`$QPGNOT`, and `$QPGOOP`, `$YPGM` and `$YPGNOT`, and `$QYPGM` and `$QYPGNOT`.

As an example purely to illustrate this scheme, take a system with 2 different types of TV devices (*e.g.*, TV server and IVAS) as well as 2 different array processors (*real* and *pseudo*). On such a system, 4 different combinations of array processor and TV device dependency are possible:

- *real* AP with TV server (`$LOAD`)
- *pseudo* AP with TV server (`$LOAD1`)

- *real* AP with IVAS TV (`$LOAD2`), and
- *pseudo* AP with IVAS TV (`$LOAD3`)

The programs that depend on Q-routines will be linked once with the object libraries for the *real* array processor, creating one load module (binary or executable file), then linked again with the *pseudo* array processor object library, creating a second load module. The load areas that the resulting binaries are kept is indicated in parentheses in the itemized list above.

Programs that depended on the Y routines will similarly have two versions of each program built, and programs that depend on both Q and Y routines will have four versions built.

The convention of associating TV and AP device numbers with `<loadnumber>` is cast in stone. TV device type 1 is *always* associated with `<loadnumber>=0` and `1`, TV device type 2 with `<loadnumber>=2,3` and so on (do not confuse these TV device types with the TV *device number* that corresponds to a given image catalog). On the other hand, it does not matter whether the AP or PSAP is specified first, and on most unix systems only the PSAP routines will be used. So on such systems if you specify two types of TV device, you would use `<loadnumber>`s zero and two.

The way to implement this for the program section of `LIBR.DAT` is illustrated best by an example. Here is how it would be done for a system with an IIS model 70, an IVAS and the TV server:

```
QYPGM => Standard tasks that call both Q-routines and Y-routines

w/ IIS Model 70 Y-routines

$LIBR/QSUB/SUBLIB:0:$QYPGM
$LIBR/QVEX/SUBLIB:0:$QYPGM
$LIBR/YSUB/SUBLIB:0:$QYPGM
$LIBR/YM70/SUBLIB:0:$QYPGM
$LIBR/APLSUB/SUBLIB:0:$QYPGM
$LIBR/APLCVEX/SUBLIB:0:$QYPGM
$LIBR/APLSUB/SUBLIB:0:$QYPGM

w/ IIS Model IVAS Y-routines

$LIBR/QSUB/SUBLIB:2:$QYPGM
$LIBR/QVEX/SUBLIB:2:$QYPGM
$LIBR/YSUB/SUBLIB:2:$QYPGM
$LIBR/YIVAS/SUBLIB:2:$QYPGM
/wherever-you-keep-proprietary-libraries/XANTH.LIB:2:$QYPGM
$LIBR/APLSUB/SUBLIB:2:$QYPGM
$LIBR/APLCVEX/SUBLIB:2:$QYPGM
$LIBR/APLSUB/SUBLIB:2:$QYPGM

w/ TV Server Y-routines

$LIBR/QSUB/SUBLIB:4:$QYPGM
$LIBR/QVEX/SUBLIB:4:$QYPGM
$LIBR/YSUB/SUBLIB:4:$QYPGM
$LIBR/YSS/SUBLIB:4:$QYPGM
$LIBR/APLSUB/SUBLIB:4:$QYPGM
$LIBR/APLCVEX/SUBLIB:4:$QYPGM
$LIBR/APLSUB/SUBLIB:4:$QYPGM
```

In this case, when programs from the `$QYPGM` area are linked, they are linked three times, once with the IIS model 70 object library, then with the IVAS library (note the nonstandard location) and a third time with the TV Server library. The resultant executable modules are stored in the separate load libraries, $LOAD, $LOAD2 and $LOAD4. This is done automatically whenever a COMLNK is performed to compile and/or link a task.

Note that for IVAS TV's, a proprietary library is required. If you have an IVAS, you should edit LIBR.DAT to set its location appropriately. The Convex default version of this file has NRAO's location for this file in Charlottesville; this has to be changed before starting INSTEP3.

Many of the *AIPS* programming procedures make use of LIBR.DAT either directly or indirectly. These include COMRPL, COMLNK, LIBR, LIBS, LINK, MAKEAT and SEARCH. These are described briefly in the tables earlier (section 1.9) and also in the extensive comments in the shell scripts themselves.

Using LIBR.DAT, all of a given *AIPS* implementation's source code search paths (except for "INCLUDE" files), object code libraries, link lists and load module areas can be mapped in a single file. The only other files that need customization for compiling are the options files CCOPTS.SH, ASOPTS.SH where relevant, FDEFAULT.SH, and LDOPTS.SH.

### *2.4.7 INCS.SH*

The file called INCS.SH sets the search path for the INCLUDE files for Fortran modules by the source code preprocessor, PP. Any statement of the form

```
INCLUDE 'INCS:filename'
```

will trigger the preprocessor into searching a list of directories for the include files. Installers with VMS background will immediately see the origin of this usage, as on such systems INCS is a logical name implementing a search path. INCS.SH simply defines the environment variable $STDINCS as the search path.

The way this works in practice is that the PP script will call INCS.SH (for fortran files only, of course), and it will also set INCS=$STDINCS if the environment variable INCS is not already defined. Then PP searches each directory in INCS until the first include file matching the indicated filename is found. This way, it is possible to override the default definition, if you should ever have to do that.

For most standard configurations, the INCS.SH file is already set up and needs no modification. Usually the only include file in a system-specific include area will be PAPC.INC which sets the pseudo-AP array sizes. On older systems these were small, but the default (*e.g.*, for Suns) is now larger, typically 5 Megabytes. If your system has more than 128 megabytes of main memory, you should consider editing this file and increasing the PKPWD2 parameter so that the total AP size is 17 megabytes; this would enable a complete 2k square image to fit in the AP at one time with a little breathing room.

$INC is the root of the hierarchical include file directory structure. Like the Z-routine and SYSTEM trees, it is based on the operating system and for new ports it may be necessary to create a purely local directory for your system. (With recent ports, we have found this to be unnecessary, as both the $INCSOL and $INCHP areas are empty, and we never created areas for Dec/Alpha or Linux). System specific include files are mostly those used in the pseudo array processor library. Check for a subdirectory in the $INC/NOTST/ directory that matches your system. If found and it's not empty, check the INCS.SH file to make sure it defines STDINCS such that the system-specific include area is searched first.

You should also note the Pseudo Array Processor's size in the file PAPC.INC; this will automatically set the default number you see for the second AP memory in the RUN SETPAR menu (item 29). Older versions of *AIPS* had this set in DAPC.INC. This is truly an upper limit to the size of AP that the programs will use, though if you use SETPAR to set a value less than what is set in DAPC.INC, the programs will honor the smaller value.

The total number of kilowords in the regular AP is always 64 and is set by PKPWRD. The "second" AP array size is set by PKPWD2, but in the PAPC.INC file you simply set the variable APSIZE. As shipped the default is 1310720 words, corresponding to 5 megabytes. You will see that this causes the 2nd AP size in SETPAR to come up as 1216.

### *2.4.8 Compiler and Loader Options*

Each Unix system and compiler combination seems to have slightly different commands and options for compiling (Fortran and C) and loading/linking. To address this, the *AIPS* system includes shell scripts to set both the command and options as well as *AIPS*-specific default values. These files are:

- `ASOPTS.SH` for assembler (if you need it)
- `CCOPTS.SH` for the C compiler
- `FDEFAULT.SH` and `OPTIMIZE.LIS` for the Fortran compiler, and
- `LDOPTS.SH` for the loader

Three of these should have been copied to `$SYSLOCAL` by the `INSTEP1` shell script, and you should edit them there. The other two, `FDEFAULT.SH` and `OPTIMIZE.LIS` are in `$SYSUNIX`. The comments in these files make it relatively easy to edit and change them as appropriate. If you change the `FDEFAULT.SH` or `OPTIMIZE.LIS` files in `$SYSUNIX`, please keep backup copies of the originals, *e.g.*, `FDEFAULT.SH.NRAO`.

The issue of compiler optimization levels can be tricky. Faulty (*i.e.*, too much) optimization for only a few modules can cause big headaches. On some architectures (Sun, IBM RS/6000 for example), we have had success with turning optimization on just about everywhere. For most pre-configured systems, the options files will come set to suitable values, but you should check them anyway, if necessary, reading the manual pages on the Fortran, C and loader/linker (`ld`) commands if you are unsure about the options.

In the `$SYSUNIX` area, the files `FDEFAULT.SH` and `OPTIMIZE.LIS` help specify and centralize the Fortran compiler settings and optimization details. The former stores the default Fortran command name and standard options for a number of architectures. It also defines up to 9 levels of optimization. The latter stores the information on which programs, or subroutines, or areas, or architectures, or most combinations thereof should be compiled with what levels of optimization and debug settings. There are default values in here too.

It is desirable to throw whatever optimization your system allows at the Q routines. The most significant gains in performance will result from optimizing the contents of `$QPSAP` (see *AIPS* memo 71, for example). Watch out, though: compiler options and even their meanings may change from one system to the next, and even from one operating system version to the next. Some compilers, *e.g.*, IBM's, come with highly optimizing preprocessors that are enabled with a command line switch. In the case of IBM, the `FDEFAULT.SH` already has this information coded in the optimization definitions.

### *2.4.9 Preliminary Programs — NEWEST and PRINTENV*

There are three programs that are needed prior to starting the rebuilding of *AIPS*. One of these, the preprocessor, is described in the next section. The other two are `NEWEST` and `PRINTENV`. The purpose of the former is to determine which of a given set of files has been created most recently. The latter is functionally equivalent to the Berkeley `printenv` for systems that either don't have it or where the environment is too small. `INSTEP1` should have already built these for you; if it did this successfully — and they work — skip this and the next section (and treat yourself to a small banana split).

If you have to make these by hand, here is how:

```
% cd $SYSLOCAL
% ln -s $SYSUNIX/NEWEST.C newest.c                    (note lowercase!)
% cc -O -o NEWEST newest.c
```

If your system's version of `printenv` is either inadequate or just not there, then do this:

```
% ln -s $SYSUNIX/PRINTENV.C printenv.c                (again, lowercase)
% cc -O -o PRINTENV printenv.c
```

Then, after verifying that the programs work (`INSTEP2` will do this for you) you can delete the links `printenv.*` and `newest.*`.

There are two additional programs — `F2PS` and `F2TEXT` — that need to be built here, but `INSTEP1` should have been able to build these. They are not needed for the compile or link stages of the installation; only for use by *AIPS* itself. They have already been described in the previous sections.

### *2.4.10 Source Code Preprocessing*

*AIPS* currently uses a preprocessor on all systems to convert the Fortran code to generic Fortran 77. The preprocessor includes:

- locating `INCLUDE` statements, then searching for and inserting the text from the specified external file;
- detecting "local" include text at the beginning of the file, storing the text and its name away in memory, then inserting the text wherever specified via `INCLUDE` statements;
- substituting `REAL` for `HOLLERITH` type declarations, which are used for storing 4 (always) hollerith characters per element (regardless of host word size).
- Up to three levels of hierarchical include files can now be referenced; some of the code actually uses this. You will see the message "** New version of PP invoked: 3–level INCLUDEs" during each preprocessing.

`INSTEP1` should have attempted to generate the preprocessor. If it succeeded and produced a `PP.EXE` file that seems to work in `INSTEP2`, you can skip the rest of this section. However, if for some reason it failed and you have to build it manually, read on.

To generate the source code preprocessor binary `PP.EXE` which is in turn called by the source code preprocessor script `PP`, do the following:

```
% cd $SYSLOCAL
% ln -s $SYSUNIX/PP.FOR pp.f
% ln -s $APLUNIX/ZTRLOG.C ztrlog.c
```

`ZTRLOG.C` is a routine that translates environment variables that is needed by the preprocessor. If you are using AIX (IBM's version of Unix), or HP–UX, or UniCos (Cray), or any other system where an underscore is *not* appended to the name of C routines when they are called from Fortran, you will need to strip out the trailing underscore from `ztrlog.c` thus:

```
% mv ztrlog.c ztrlog.tmp
% cat ztrlog.tmp | sed -e 's/ztrlog_/ztrlog/g' >ztrlog.c
% rm ztrlog.tmp
```

This is what `INSTEP1` does to remove the underscore. Additionally, on UniCos systems, you need to convert the references to `ztrlog` to `ZTRLOG` (uppercase). This part may have worked so before you do this, check if it left a `ztrlog.c` file in `$SYSLOCAL`.

Additionally, on HP systems, you need to substitute a `STOP` statement in place of the `CALL EXIT` in `pp.f`:

```
% mv pp.f pp.tmp
% cat pp.tmp | sed -e 's/CALL EXIT.*$/STOP/g' >pp.f
% rm pp.tmp
```

Next, if your system is *not* Linux, compile the two source files with:

```
% cc -c -O ztrlog.c
% f77 -O -o PP.EXE pp.f ztrlog.o
```

You may have to substitute a different command in place of `f77` (*e.g.*, `fc` on convex, `xlf` on IBM/AIX, *etc.*). The `-O` is for optimization.

For Linux, a more complex workaround is needed. `INSTEP1` should have done this for you, but in case it failed, here are the steps needed. First, compile `ztrlog.c` as above (use `gcc`). Then:

```
% f2c pp.f
% mv pp.c pp.tmp
% cat pp.tmp | sed -e 's/exit_(/exit(/g' -e '/int exit/d' >pp.c
% rm -f pp.tmp
% f77 -O -o PP.EXE pp.c ztrlog.o
```

This little dance separates the Fortran-to-C conversion from the C compilation of the result, and replaces the external reference to `exit_` with the normal C `exit` call.

The procedure `PP` operates on fortran (`.FOR`), C (`.C`), and assembler (`.S`) files. It only runs the `PP.EXE` program on fortran files; for the other files, the operation is usually a simple copy to files with lowercase file "type", e.g. `FOO.c`. In the case of Fortran modules, `PP` sets `$INCS` to whatever is in the variable `$STDINCS` if it is not already set to something (see the description of `INCS.SH` above). If there is already a `FOO.f` or whatever, it will be overwritten by the new file that `PP` or `PP.EXE` creates.

## 2.5 Installation Step 2

### *2.5.1 Running INSTEP2*

For binary installations, you do —it not need to run the `INSTEP2` shells script. For source installations, change directory to `15JUL94/$ARCH/INSTALL`. This directory should have been created by `INSTEP1`; if not, create it and copy files `INSTEP2`, `INSTEP3`, and `INSTEP4` there from the `$INSUNIX` area (or make symbolic links). These are shell scripts that do most of the work for the remaining installation steps. `INSTEP2` will create any missing directories in which to store binary files (object libraries, executable modules, etc.) and compile all the subroutines required by *AIPS* on the host system based on the specifications given in `$SYSLOCAL/LIBR.DAT`. So it is important to have this edited *before* your first (and hopefully only) run of `INSTEP2`. To start things rolling, simply type:

```
% $CDTST
% INSTEP2
```

The first command is necessary to define the *AIPS* environment variables, if they have not already been defined. Depending on the version you want, you may substitute `$CDNEW` or `$CDOLD` here.

As this step can take a long time, you may want to shed it as a background task:

```
$ (INSTEP2 >/dev/null 2>INSTEP2.ERRS &)
```

(the above assumes a bourne, korn, or bash shell, not a c or tcsh shell). This will run the process as a grandchild, throw away the output (which is sent via "tee" to `INSTEP2.LOG` anyway) and captures any output to `stderr` in another file. You may see some messages in the error file, *e.g.*, `tsort: cycle in data` or messages about remaking `SEARCHO.DAT`. These are harmless. You can monitor progress of this process by:

```
$ tail -f INSTEP2.LOG
```

`INSTEP2` will first check for the existence of the directories in which binary files will be stored and if not found will attempt to create them. These existence/creation operations are very fast, but the source code compilation phase can take quite some time. On an unloaded Sun 3 system or a Convex C–1, this can mean the better part of a day, or more. On an IBM RS–6000/580 or a HP 9000/735, it will only be about two hours.

One thing to watch for is that if all of the `$LIBR/*` subdirectories are not created, `INSTEP2` can crash when it tries to create the directory (it's actually `COMRPL` that does this). This generally occurs if `LIBR.DAT` was not set up correctly the first time and `INSTEP2` has to be repeated. If you find yourself in this situation, you should either remove all the subdirectories and their contents from `$LIBR` and start `INSTEP2` from scratch, or create the missing directory by hand. In the latter case, you may have to mess with the list (`*.LIS`) files.

Another thing to watch for with `INSTEP2` (and the following steps) is that there have been reported problems with them when they are called from the Korn shell. These have not been verified recently, however, probably largely because the `INSTEPn` files are explicitly bourne shell scripts (or ksh or sh5 for Convex or Ultrix, respectively). If you come across this behaviour (and you shouldn't anymore) the simplest solution to this is to get into a generic bourne shell (or C shell) by typing `sh` (or `csh`) before starting the `INSTEPn` procedure (where `n` is 2, 3, or 4).

Sometimes the generation of the `.LIS` files goes wrong, and you end up with an empty file. If you find any of these, you should promptly interrupt the `INSTEP2` processing, remove the empty files and restart (after you have checked for sufficient disk space on the relevant filesystem — and in `/tmp`).

For new ports, the values of `BYTFLP`, `SPFRMT`, `DPFRMT`, `TTYCAR` as well as other system constants and parameters which are set in the system-specific version of `ZDCHI2` should be checked (see section 3.3). They are set correctly for the already-supported architectures, although you may want to set the Convex one accordingly; if you have IEEE hardware, use the `$APLNRAO1` or `$APLVLAC1` version of `ZDCHI2` instead of the generic `$APLCVEX` version (did you remember to change `LIBR.DAT`?).

`INSTEP2` will cycle through all the major subroutine source code areas in the following order: AIPSUB, APLSUB, APLOOP, APLGEN, YGEN, YSUB, YNOT, APLNOT, QDEV, QSUB, QNOT, QOOP. For each, it will execute the procedure `$SYSUNIX/MAKEAT` to generate "at" files containing lists of path names to all subroutine source code modules pertinent to the host implementation. For example, `MAKEAT $APLGEN` will ultimately produce the file `APLGEN.LIS` containing a list of all the subroutine source code found in the directory `$APLGEN` and below that is relevant to the host implementation. The path names selected are determined by the directory search paths defined in `$SYSLOCAL/LIBR.DAT`.

The "at" files are then used by the procedure `$SYSUNIX/COMRPL` to drive the mass compilation phase. As each subroutine is compiled, the resulting object module is staged to one or more `$LIBR` subdirectories, again based on `LIBR.DAT`. For example, object modules generated from `$APLGEN/`... subroutine source code is staged to the proper `$LIBR` subdirectory defined for such modules. After all subroutines are complied for, say `$AIPSUB`, the procedure `$SYSUNIX/LIBR` is invoked to build an object library (*archive* file in UNIX parlance) from all the object modules found in the directory `$LIBR/AIPSUB`. On Suns where you have chosen to use shared libraries (*i.e.*, created an empty file `$SYSLOCAL/USESHARED`), the `$SYSLOCAL` version of `LIBR` is used instead; this generated both static (`SUBLIB`) and shared or dynamic (`SUBLIB.so`) versions of the libraries.

If you are making debug libraries on a Sun, there is an extra area `$AIPS_VERSION/$ARCH/LIBRDBG` that contains a parallel set of directories to the `$LIBR` area. Then each compilation is done twice, first with any debug or no-opt settings overridden, then with no-opt and debug and nopurge forced on the compile. The first version of the object file generated goes in the regular staging area and the second (debug) version goes in the `LIBRDBG/*/` area. As the debug compile is performed last, it also leaves the source code in the `$PREP area` unless you override the settings in the usual configuration files. Note that the trigger for generating the separate set of debug libraries is the presence or absence of the file `DOTWOLIB` in the `$SYSLOCAL` area, and that it is only currently supported on Suns (including Solaris). Currently, `INSTEP2` will *not* make the debug versions of the libraries; it just leaves the object modules in the staging areas.

`COMRPL` marks each successfully compiled file in the "at" files by prefixing a dash (`-`) to the name. In the event of an interruption (system crash, whatever) `INSTEP2` can be restarted and will resume with the next module as if nothing had happened. `INSTEP2` will also stop whenever `COMRPL` fails on a given module. However, it waits until it has finished processing the particular "at" file before quitting, and then complains. This behaviour is different than in versions of *AIPS* before `15JAN94`.

This allows you to investigate and take corrective action and resume the process simply by running `INSTEP2` again. Alternatively, you could edit the respective "at" file, marking the problem routine as done (i.e., prefix `-` to its path name), restarting `INSTEP2` and work on the problem routine later. See also the *Known Problems* chapter for those problems that you can ignore. In particular, corrective action should be taken before `INSTEP2` finishes processing the respective "at" file since its next step is to build the respective object library.

The object libraries are built for single pass linking using unix commands `tsort` and `lorder`. That is, the ordering of the object code in the library is such that the loader can resolve all references in a single pass. A missing object module can perturb the single pass ordering.

The object libraries built by `INSTEP2` via the procedure `LIBR` all have the same name, `SUBLIB` (*e.g.*, `$LIBR/AIPSUB/SUBLIB`). For Suns and HPs, the shared libraries are located directly in the `$LIBR` area itself with names like, *e.g.*, `APLSUB.so`. `INSTEP2` will also check for the existence of the `SUBLIB` for a given major subroutine source code area and if found, will *completely* skip it, regardless of residual unprocessed path names in the respective "at" file. The reason for this is that, once the `SUBLIB` object libraries exist, the procedure LIBR merely replaces (or adds) object modules in the library without any consideration to ordering.

If you find yourself in a situation where a given SUBLIB was built prematurely, simply change to the respective $LIBR subdirectory, type ar x SUBLIB (or equivalent) to extract all object modules, and delete SUBLIB. Rerunning INSTEP2 will cause any residual unprocessed subroutine path names in the respective "at" file to be processed and subsequently invoke the procedure LIBR to rebuild the object library with single pass ordering.

The procedure COMRPL (compile and replace), despite its name, never actually puts the object modules in the object libraries. It only stages the object module to one or more of the $LIBR (and possibly $LIBRDBG) areas. Object library updating is only done either manually or by the procedure LIBRnews/ (which is called if needed by COMLNK; see INSTEP3). Recall that the object library lists for programs from the various program source code areas are also defined in $SYSLOCAL/LIBR.DAT. One of the features of LIBR is that it locks the target SUBLIB to avoid collisions that often result in corrupted object libraries, if there are multiple users performing compiles and links. Most sites will not need this feature.

The remainder of this subsection will cover certain aspects of the *AIPS* installation that you may have to deal with. These can usually be done in parallel with INSTEP2 and INSTEP3.

### *2.5.2 Device Names (Logicals)*

The files AIPSASSN.CSH and AIPSASSN.SH are used to define some *AIPS* system values in the form of environment variables. In older versions of *AIPS*, there was an important site- and host-specific section at the end of these files where tape drives and other things were defined. Since the 15JUL93 release, the tape drives are defined in the TPDEVS.LIST file and there will probably be no need to edit the AIPSASSN.* files.

However, they are still useful if you need to define any of the following: reserved terminals (optional), special task message terminals (optional), and dedicated graphics terminals (optional). These files are the best location for overriding default definitions for things like the FITS area, and temporary printing and plotting areas. But in a vanilla installation, you probably won't even have to look at them.

If for any reason you need to modify these files, look at the section at the end of the AIPSASSN files:

```
#                                   Host specific definitions
#                                   ~~~~~~~~~~~~~~~~~~~~~~~~~
#                                   LWPRINTER?, TASKTTn, RESSTTn,
#                                   TEKTKn, TAAC1, and modification
#                                   of any of the foregoing.
if [ "$SITE" = "NRAOCV" ] ; then
#                                   NRAO Charlottesville.  These
#                                   definitions are shown here only
#                                   for illustrative purposes.  You
#                                   should delete this section and
#                                   insert definitions appropriate
#                                   for your site.
#                                   --------------------------------
    : This is _the_ standard site
#
elif [ "$SITE" = NRAOAOC ] ; then
```

and so on. If you need to perform any customization (and this is unlikely), substitute your site name and host names for NRAOAOC and YUCCA, *etc.*. Also be sure that you make equivalent changes in both files, not just one.

### 2.5.3 Data Areas

There must be at least one *AIPS* user data area, or *AIPS* and its support tasks simply will not work. Since the 15APR92 release, these areas are defined at *AIPS* startup on the fly, depending on the contents of `$NET0/DADEVS.LIST` and the `da=` option on the command line. The former defines the data areas ("disks" in *AIPS* parlance) and whether they are required or optional. The latter selects which areas in this file the user wants to use for the duration of the *AIPS* session. Also the file `$NET0/NETSP` sets `TIMDEST` (expiry time) limits and user reservation numbers for the same disks.

The environment variable `DA00` will be defined (by the `AIPSASSN.*` files) as the full path name of a directory where *AIPS* "system" files are to be stored. These are distinct from the "user" files in the data areas mentioned above and should not be confused with them. As outlined in section 2.3.10, the `INSTEP1` procedure will create this area as `$AIPS_ROOT/DA00/$HOST` for each host.

The `DADEVS.SH` shell script is called by both `START_AIPS` at *AIPS* startup time, and by the `RUN` shell script whenever one of the standalone *AIPS* programs such as `FILAIP` is run. It (`DADEVS.SH`) is what takes the `da=` option, the current hostname, and the `DADEVS.LIST` file, and makes a list of data areas from these inputs. The method used to select disks assumes the hostname where the disks are physically mounted is embedded in its name somewhere, *e.g.*, `$AIPS_ROOT/DATA/FOOBAR_1` for data area number one on host `foobar`. Then, the disks selected for the user will be:

1) Any "required" areas indicated by a + in `DADEVS.LIST`
2) Any areas containing the name of the local host; and
3) Additional areas selected by name, *e.g.*, `da=foobar`

It is possible to set up host-specific versions of `DADEVS.LIST` by copying it to the `$DA00` area for a given host and changing it, and it is even possible for a user to copy it to their personal home area as `.dadevs` and customize it. The `DADEVS.SH` script looks first for the personal version, then for a host-specific version, and only then will it use the site-wide version.

However, these practices have two disadvantages: (1) the administrative overhead of changing the disk configuration increases linearly with the number of such files (it's easier if you stick with one site-wide file); and (2) users making their own `.dadevs` files will be asking the *AIPS* manager to change the system-wide `NETSP` file anytime they change their private data areas (otherwise they cannot reserve these areas for their own usernumbers and get the default `TIMDEST` or time destroy parameters). Of course there are obvious advantages as well, such as speed of startup where the `$NET0/DADEVS.LIST` has many disks and the `.dadevs` file may only have a few.

It is best *not* to put the user data areas on the same file system (partition, logical volume, disk, whatever you call it) as the *AIPS* distribution itself. The latter will incorporate the system files in `$AIPS_ROOT/DA00/$HOST/` and you do not want users filling up this same file system and preventing the *AIPS* accounting file from expanding; this would make `AIPS` difficult or impossible to run. Of course, you can always relocate the `$AIPS_ROOT/DA00` directory itself elsewhere, and put a symbolic link in its place pointing at the new location.

For the user data areas, it is good practice to keep separate *AIPS* disks on separate partitions (if you have more than one), except if you create additional areas for separate projects or other organizational reasons. The first user data "disk" (what `AIPS` says is "disk 1") is where certain user files are stored: message files, save/get files. For this reason, it may be desirable to specify one (or more) "required" disks in the `DADEVS.LIST` file — and make it clear to users who insist on their private `.dadevs` that they will be responsible for maintaining multiple copies of their message and save-get files if they change the required disk(s) to optional or remove them in the `.dadevs` file. Also, when (not if!) space on disk 1 is exhausted, *AIPS* usage can be impaired just as for the system file area mentioned in the previous paragraph.

The network scheme *AIPS* operates in has a convention where all data disks (from number 1 up) on a given host are defined as `$DATA_ROOT/$HOST_n`. For example, in NRAO's Charlottesville setup, the workstation called `lemur` has two disks, and the entries in `DADEVS.LIST` for it and the required `polaris` disk are:

```
+  /DATA/POLARIS_1
```

- /DATA/LEMUR_1
- /DATA/LEMUR_2

The automounter is used at this site and the `$DATA_ROOT` is of course `/DATA` which is also the name of the automounter map that enables these disks on different machines to be mounted on demand — and dismounted after several minutes of inactivity. Be warned, however: on some automounter versions, the auto dismounting can in fact cause long-running AIPS jobs to crash. The reason is that the *AIPS* process may not have an open file on a given data disk, and also will not have any of the data disks in the `PATH`. Hence the automounter will dismount it. In principle, when an open is attempted on a file, the automounter should intervene and mount the directory, but at least one site has reported that (for Solaris) this does not always happen and a "file not found" message is issued. Running the task again invariably gets things working again.

The `da=` option to the `START_AIPS` shell script will typically generate something similar to this:

```
baboon% aips da=lemur
(stuff about printer omitted here for clarity)
Data disk assignments:
   Disk 1 is /DATA/POLARIS_1
   Disk 2 is /DATA/BABOON_1
   Disk 3 is /DATA/BABOON_2
   Disk 4 is /DATA/LEMUR_1
   Disk 5 is /DATA/LEMUR_2
```

This example is run on a system called `baboon` which has two *AIPS* disks or data areas. The user has requested that the disks from host `lemur` be used as well, and in the NRAO–CV environment, the `polaris` disk is required and listed first in the `DADEVS.LIST` file. The requirement for selection based on the `da=` arguments is that they (`lemur` in the above example) be found in the disk name (uppercased). Also, the checking criterion is the existence of the empty `SPACE` lock file. If it is not detected, for whatever reasons, the procedure simply says that disk is not available and proceeds to try the next one. Suppose `BABOON_1` had been dismounted for some reason. Then there would have been 4 disks selected in the example above, and *AIPS* internally would know them as disks 1 through 4, instead of 1 through 5 and including the `BABOON_1` disk.

When running *AIPS*, users can set the adverb `BADDISK` to prevent scratch file creation on `$DA01` or any other disk. Also, there is an *AIPS* system parameter that allows only scratch files to be created on a given disk. Such a disk can be specified in `$NET0/NETSP` by setting the first reserved user number as -1 (values in this file override whatever is set via the `SETPAR` program). You may find it useful to establish one or more such scratch disks in your *AIPS* system. Furthermore, the same system parameter can be set such that only specified user numbers (up to 8 per disk) are allow to create files (of any type) on a particular disk. Note that currently `BADDISK` is restricted to at most 10 values; your users may want to restrict the number of disks they include with the `da=` option so they can make use of the adverb.

### *2.5.4 Tape Drives*

*AIPS* tape drives are numbered, not named, and referenced by the `AIPS` program via an integer parameter in an adverb such as `INTAPE` or `OUTTAPE`. The definition of all tape drives is centralized in one single file, `$NET0/TPDEVS.LIST`. A shell script (`TPDEVS.SH`) is called by various other scripts such as `START_AIPS` and `START_TPSERVERS` to define environment variables which contain the actual device names. Normally, a user of *AIPS* need never know these device names. Not so the installer, however...

Tape device names vary considerably from one system to another, and there are many conventions for encoding low/high density, raw mode, drive number *etc.*, on the many Unix systems *AIPS* runs on. As a particularly convoluted example, on a Convex, device `/dev/rmt12` means the first drive on the system, "raw" mode, no rewind on close (required modes for use with *AIPS* programs) and a density of 1600 BPI. The device alias for the same tape unit with the same properties except 6250 BPI, is `/dev/rmt16`. On the other hand, on an IBM RS–6000, the same device names would be `/dev/rmt0.5` and `/dev/rmt0.1` respectively.

The INSTEP1 procedure has fairly detailed commentary on the form of tape device names for the major systems likely to be encountered. For new ports, you should look up the manual pages or equivalent for such things as mtio. If all else fails, try man -k tape. Refer to the previous sections describing INSTEP1 for additional information about restrictions on the form of the device name, and the related density switching performed by ZMOUN2.C.

The sample TPDEVS.LIST file in $SYSUNIX (not the one copied to your $NET0 area) shows common device names for the architectures NRAO uses in-house. The *AIPS*routines ZTAP2 and ZMOUN2 in the various architecture-specific Z routine areas in general will know how to change device names on the fly to get the correct density requested (see the DENSITY adverb and its help file in AIPS). It is vital that you enter the basic device name in the TPDEVS.LIST file. For example, with an Exabyte 8500 on a Sun, use /dev/nrst0 instead of /dev/nrst8; and on an IBM RS/6000, you should use /dev/rmt0 instead of /dev/rmt0.1. The density selection algorithm inside *AIPS* can get confused easily if you fail to follow this advice.

Another feature that UNIX systems often lack is the notion of "allocating" a given tape drive to a specific user or process. That is, tapes can be read and/or written (unless write protected) by anyone on the system as long as the drive is not already opened by some other process. Convex systems offer a VMS-like tape mounting utility (tpmount and tpqueue) that we use to prevent collisions. AIPS uses file locking on a temporary disk file to attempt to implement this, but that does not protect against collisions with non-AIPS users.

In older versions of *AIPS*, the "verbs" to issue software mounts or dismounts were often unnecessary. However, that is no longer the case. The *AIPS* Z (machine specific) routines that handle tape mounts (ZMOUN2 in the various machine-specific directories) in most cases will query the device and report back what it thinks is being mounted. This feedback can be beneficial to the user who mistakenly tries to mount the wrong tape drive. This Z routine will undoubtably require local development if a tape allocation scheme (other than convex) is supported on your system and you want to or must use it. The Z-routines for Sun, Solaris, IBM, HP, Convex, and Dec (Ultrix and Alpha) should not require modification.

### *2.5.5 Remote Magnetic Tapes*

The concept of remote tapes has been built into *AIPS* itself in the form of two adverbs that appear in the inputs list for the MOUNT command. These are REMHOST, the name of the remote host, and REMTAPE, the *AIPS* tape number of the drive you want on that remote host. Note that the host name as input to *AIPS* is necessarily uppercased, but the Z routines will convert it to all lowercase before it gets anywhere near the network (*i.e.*, before the call to gethostbyname()). There are two additional "slots" for remote tape devices defined by default, *e.g.*, a system with one local tape unit will have *AIPS* tapes 2 and 3 for use as connections to remote tape drives.

To make this scheme work, two things are needed: a transport mechanism, and a dæmon or dæmons at the remote end. To this end, the TPMON task was created. This is a standalone task that is started by the START_TPSERVERS shell script (it calls an intermediate script but that is not relevant here). This task uses Berkeley Sockets to communicate over a TCP/IP network; the sockets are named AIPSMT0, AIPSMT1, and so on. Most if not all System V flavors of Unix that we have ported *AIPS* to in recent years do support some form of Berkeley sockets, hence the Z routines for dealing with them are in the $APLUNIX area.

To find out how many of these sockets you will need, pick the host with the most tape drives (say 4). Add one to this (for FITS disk access across the net) and this is the total number of service names you need, in the example here, 5. Then you want to add these to either the /etc/services file, or the YP/NIS master version of the file if that is how your network supplies it (we do). Here is a typical services list:

```
AIPSMT0        5010/tcp            # AIPS remote tapes
AIPSMT1        5011/tcp
AIPSMT2        5012/tcp
AIPSMT3        5013/tcp
AIPSMT4        5014/tcp
```

```
SSSIN          5000/tcp            # AIPS TV server inet sockets
msgserv        5008/tcp   MSGSERV  # AIPS Message Server
tekserv        5009/tcp   TEKSERV  # AIPS TekServer
```

(note the presence of the required TV server, tektronix graphics, and message server sockets too). We recommend that you try and use these service numbers if possible; this may prove useful if you ever need us to log in to your system remotely to fix a problem (we can then get your system to use our local TV and tek servers, or vice versa, which might be invaluable).

The way remote tapes now work is that there will be $n+1$ instances of `TPMON` started up on each host, where $n$ is the number of tape drives on the system. The first one (`TPMON1`) will handle remote requests for access to *FITS* disk files. The second one `TPMON2` handles requests for *AIPS* tape drive 1, the third for tape 2, and so on.

These `TPMON` "dæmons" will get started on the local host for the first time when *AIPS* is started for the first time. Thereafter, a startup of *AIPS* will cause the system to check if they are still running; if some are missing, they will be restarted. You can also have them start at boot time by having your sysadmin insert a call to `AIPS.BOOT` in the `/etc/rc.local` or equivalent file; this should be run as your *AIPS* account, *not* as root.

Users can, with the `tp=` command-line option, have the system issue a command (using `rsh`, or `remsh` on HP's) to a remote host to make sure the `TPMON` daemons are running. This is usually to give the user peace of mind that the dæmons on the remote host are indeed working, and that there won't be any breakdown in socket communication. As mentioned above, on each host there will be two additional tape "drives" defined as `REMOTE` listed in the startup messages. This allows up to two local users to access one remote tape drive each, or one user access to two remote tape drives.

As stated in the previous section, the use of the `MOUNT` command is **required**; you should inform the *AIPS* users of this (it was not always required in older versions, and old habits die hard). This applies to both local and remote tape drives. After a remote tape drive has been mounted, the I/O to it is performed via the sockets and `TPMON` which of course uses the *AIPS* Z routines on the host in question. So remote tapes are supported for any machine that runs *AIPS* and has a working set of tape Z routines.

In the `15OCT92` version of *AIPS*, the tape related Z routines for Suns (Sun-supplied SCSI Exabytes and HP DAT), Convexes (9-track), IBM RS/6000's (IBM Exabyte, IBM and HP 9-track, and HP DAT — with AIX 3.2), and DecStations (TK50, maybe Exabyte too?) were overhauled and many bugs have been eliminated. Even more work was performed with `15JUL93`, improving robustness, eliminating bugs, doing as much testing as we possibly could on the equipment at our disposal, and more. For `15JAN94` we smoothed out the wrinkles and made sure the Dec Alpha, Solaris and HP systems worked as we expected. On Suns with HP DAT drives, we had to patch the kernel to make sure the system knew that DAT drives were capable of backspacing; this was apparently not the default so be wary.

Also, be careful of third party drives; while many of these are perfect plug-compatible replacements for what the workstation vendors offer, sometimes they are marketed on the assumption that you will only ever be using `tar` or `dump` or `backup`*. *AIPS* is a *very* tough test of any tape drive and relies on being able to backspace by file and record (block).

* On IBM RS/6000 systems, some Exabyte 8500 systems from third party vendors cannot be recognized by AIX as dual-density drives, for example (SMIT will tell you it's an 8200). AIPS will allow you to force high density settings on 8200's on IBMs precisely for this reason. There are many other horror stories, so be careful out there!

### *2.5.6 Reserved Terminals*

An *AIPS* feature that is somewhat obsolete is the concept of reserved terminals. This is a terminal that is reserved for `AIPSn` (*e.g.*, `AIPS1`) use. Historically, such terminals were near an old-style TV display or graphics terminal and *AIPS* has the ability to increase the priority (or niceness, in Unix parlance) of the `AIPS` tasks running on such a reserved terminal. This feature (elevated priority) has not been used at NRAO for some time, and the use of dedicated *AIPS* reserved terminals died with our last remaining convex (yucca) at the AOC site.

The environment variables `RESSTT1`, `RESSTT2`, *etc.*, are used to specify which terminals are "reserved". These are purely optional and can be added later simply by defining them in your `AIPSASSN.CSH` and `AIPSASSN.SH` files. There are two examples of reserved terminals in the NRAO site-specific examples in these files in the `NRAOAOC` section for host `YUCCA`.

However, reserved terminals must have a specific and fixed device name, which preculudes the use of most windows on workstations as reserved terminals. It would be possible to implement a scheme similar to `TEKSRV` for this purpose, but there has been little interest expressed in this possibility and it will probably never happen.

### *2.5.7 Message Terminals*

As in the case of reserved terminals, the use of dedicated message terminals is also purely optional. These are terminals that are devoted to displaying the messages from *AIPS* tasks that are running as subprocesses ("child" processes). These are normally associated with reserved terminals. The environment variables `TASKTT1`, `TASKTT2`, *etc.*, specify the device names. They are the hardware equivalent of the `MSGSRV` task running in a window on an X11 display (see below).

This may be a luxury that you cannot afford since it requires that the terminal be connected to a port, but taken out of the interactive group. This makes it otherwise useless. However, it doesn't have to be a very smart terminal. At NRAO we used recycled and otherwise obsolete terminals as *AIPS* message terminals on our Convex systems while we had them. Like reserved terminals, message terminals can be added later by simply defining them in your `AIPSASSN.SH` and `.CSH` files, and they have to be fixed device names.

### *2.5.8 MSGSRV and TEKSRV, the Message and Graphics Servers*

In addition to the `XAS` imaging server, *AIPS* provides two additional windows in an X11 environment (including OpenWindows, Motif, DecWindows, HP VUE, *etc.*). These are `TEKSRV`†, a graphics server, and `MSGSRV`*, a server for *AIPS* messages from tasks. These are modelled after the early *AIPS* VAX and Convex setups, where there were dedicated terminals for messages and graphics.

These virtual counterparts of the early dedicated terminals can accept input from more than one *AIPS* session, and they are not restricted to either the host the underlying client programs are running on, or the display host that the X server runs on. The way in which this is achieved is via internet sockets using the `tekserv` and `msgserv` services which were shown in the above example of the `etc/services` file or YP/NIS services map.

`TEKSRV` serves as the graphics tablet for the *AIPS* sessions that a user originates on any hosts in the *AIPS* network from the workstation that the user sits in front of. Any tek-related commands such as `TKSL` will cause the graphics to appear in the window. `TEKSRV` by default uses the tektronix-emulation feature of the standard X11 `xterm` client (which also has vt100 emulation; text and graphics appear in two separate windows).

`MSGSRV`, the message server, serves as a collection point for the messages from *tasks* that are started from within *AIPS*. In fact, it can receive messages from many *AIPS* sessions simultaneously, and in most cases it is able to tag the messages with the host from which they originate.

When the `XAS` server is started for a given *AIPS* session, the message and tektronix servers are also started. Hence a given message server or tek server is always associated with the matching TV number as assigned by `TVDEVS.SH` based on the list in `HOSTS.LIST` (possibly supplemented with

† Originally called TEKSERVER and MSGSERVER

hardware TVs in TVDEVS.SH). There can only be one instance of each server running on each *AIPS* host at a time.

The servers are started by the XASERVERS shell script. By default they use the xterm terminal emulator, but of course any reasonable terminal emulator can be used for MSGSRV, *e.g.*, decterm, hpterm, cmdtool, or aixterm. For TEKSRV however, none of these other than xterm will do, as they lack the tektronix emulation and cannot display graphics. It is a good idea to get your *AIPS* users familiar with the two faces of xterm before they are exposed to TEKSRV by holding the control key down and pressing the middle mouse button to see the xterm menu relevant to tektronix graphics.

It is possible to specify, via environment variables, which terminal emulator you want. The relevant variables are:

- AIPS_TEK_EMULATOR, default xterm
- AIPS_MSG_EMULATOR, default xterm

You can set either of these to the string none and this will inhibit the indicated server from ever starting. If you decide on site-wide defaults other than the above for these variables, you should edit the AIPSASSN.* files and make the definitions there.

Unlike its sibling TEKSRV, there has been no attempt made to retrofit the MSGSRV functionality into SSSERVERS, given that SunView is not available in Solaris version 2 and the widespread use of OpenWindows and X11 on Suns. Sites that are still running SunView are strongly advised to consider using either generic X11 (preferably release 5 or 4), or OpenWindows version 2 or 3, in its place.

The TEKSRV and MSGSRV environment variables TKDEV, TKDEVnn, TTDEV and TTDEVnn, are set automatically at startup time by the *AIPS* scripts (as are the TVDEV and TVDEVnn variables). There is therefore no need to do anything (other than what INSTEP1 has already done in the HOSTS.LIST file) to configure these at this stage. There is support for up to 240 TV, tek, and message servers for any one *AIPS* site, as well as up to 15 "remote" (*i.e.*, tektronix-compatible graphics) terminals (241 through 255).

When a user starts up *AIPS* from a properly configured workstation running X, the XAS TV, TEKSRV graphics and MSGSRV message windows should pop up automatically. If you have not set any X resources for these windows, they will probably all pile on top of each other on the top left of the screen, though this will depend somewhat on your window manager. It is possible to override such things as position, font, geometry, foreground and background colors, icon location, and so on using resources AIPSmsg and AIPStek. Here is an example of these, and additional AIPStv resources, from the author's personal .Xdefaults file, with additional annotation explaining some features:

```
!
! AIPS: AIPStv for XAS, AIPSmsg for MSGSRV, AIPStek for TEKSRV.
!                                      inhibits shared memory if zero:
AIPStv*useSharedMemory: 1
!                                      Don't use as many colors
AIPStv*maxGreyLevel:    189
!                                      Initial location and size
AIPStv*geometry:        518x518-0-125
AIPStv*iconGeometry:   +635+100
AIPStv*cursorShape:     30
!                                      Cursor and Graphic plane colors
AIPStv*cursorR:          0
AIPStv*cursorG:        255
AIPStv*cursorB:        255
AIPStv*graphics1R:     255
AIPStv*graphics1G:     255
AIPStv*graphics1B:       0
AIPStv*graphics2R:      16
```

```
AIPStv*graphics2G:     255
AIPStv*graphics2B:       0
AIPStv*graphics3R:     255
AIPStv*graphics3G:     171
AIPStv*graphics3B:     255
AIPStv*graphics4R:       0
AIPStv*graphics4G:       0
AIPStv*graphics4B:       0
!                                      Message Server defaults
AIPSmsg*background: maroon
AIPSmsg*foreground: bisque
AIPSmsg.vt100.geometry: 81x24-0-0
AIPSmsg*iconGeometry: +780+100
AIPSmsg*iconName: MSGSERV
AIPSmsg*pointercolor: red
AIPSmsg*savelines: 500
AIPSmsg*scrollBar: True
AIPSmsg*tekInhibit: True
!                                      TekServer defaults
AIPStek*background: navyblue
AIPStek*foreground: yellow
AIPStek*scrollBar: off
AIPStek*iconGeometry:    +710+100
AIPStek*iconName:        TEKSERV
AIPStek*pointercolor:    red
AIPStek*tekGeometry:      768x585
```

Most of the keywords shown above are self-describing. For additional details, see the help files for `TEKSRV`, `MSGSRV`, and `XAS` in `$HLPFIL`, or just type, *e.g.*, `HELP MSGSRV` from within `AIPS`.

Finally, while it is possible to use `TVPL` to plot graphics in the overlay planes of the TV servers, there is an order of magnitude speedup in plotting that results from using `TKPL` and the `TEKSRV` window instead.

### 2.5.9 Old-style Graphics Terminals

In AIPS, a *graphics* terminal means a Tektronix 4010/4012 compatible device. Among the possible command line arguments to the *AIPS* startup procedure is the `REMOTE` option, which will define the user terminal as the device for graphics terminal I/O. If *AIPS* is started up with this option and any `TK*` verbs or tasks are executed, the user terminal will be used for graphics terminal I/O. Many Tektronix emulation terminals and terminal emulators, such as retrographics, visual 102G, Selanar, vterm–4010, *etc.*, can be used in this way. *AIPS* tek devices 241 through 255 are set aside for remote use and are pre-defined in the shell scripts as shipped; no customization is necessary.

Alternatively, independent graphics terminals may be defined. As in the case of message terminals, the physical graphics terminals defined in this way must be taken out of the interactive group and are otherwise dumb devices. The files `AIPSASSN.SH` and `.CSH` define these in the site- and host-specific section at the end. The relevant names are `TEKTKn` where the `n` corresponds to a reserved terminal; the scheme assumes that dedicated graphics terminals are associated with a dedicated reserved terminal for *AIPS* use (see above).

The use of these features will be "slim to none" in a workstation environment, as the various screen servers (TV, tek, message) will make their use unnecessary. However, they have their uses in setups where extensive use is still made of the types of graphics terminals mentioned above.

### *2.5.10 TV Display Devices*

All TV servers, including workstations that will use XAS, need to be defined in your `HOSTS.LIST` file. As `INSTEP1` will have configured this file for you, it should not be necessary to modify this file yourself, except to add additional *AIPS* or tv hosts, tv servers, or tv displays. (Versions of *AIPS* older than `15JUL93` had this information in a far less convenient form embedded in the `TVDEVS.SH` file.) The only exception to this is for "hardware" TV devices such as the IIS, IVAS, and DeAnza image displays. If you don't have any of these, you can safely skip the information that follows.

These devices need to be defined in a site- and host-specific section in `TVDEVS.SH`. There is a sample section in this file for `YUCCA` at the `NRAOAOC` site that defines the IIS model 70 tv for our last remaining Convex at the AOC. Refer to the file and its comments for the gory details. For such hardware devices, you will have to define two environment variables:

- `TVHOSTS` which defines the hardware TV's in the traditional *AIPS* sense, *e.g.*, `yucca+iis` for an IIS display `/dev/iis` on host `yucca`;
- `TVALTS` which lists all types of hardware TV device used, and the corresponding `<loadnumber>` (see section 2.4.5), *e.g.*`iis:2`.

For additional details, you should read the comments at the top of `TVDEVS.SH`. Note that you need to create image catalogs for all TVs defined in `HOSTS.LIST` *and* the `TVHOSTS` environment variable in `TVDEVS.SH`. These are set up for you when you `RUN FILAIP` and then call the shell script `SYSETUP`.

The maximum number of TV devices for a single *AIPS*site is 255, although the number of `TEKSRV` windows is restricted to 240 (see section above on graphics terminals). The scheme of having a double level of indirection (where the source code effectively does a `getenv(getenv("TVDEV"))` to get the actual TV name) has been preserved, but the installer no longer has to specify these apart from the `HOSTS.LIST` file; the `TVDEV` and `TVDEVxx` names are defined automatically on *AIPS* startup.

It may be worth consulting *AIPS* memo 74 to get an understanding of the ATNF setup, most of which has been incorporated into *AIPS* as of `15APR92`. At the time of writing, the old *AIPS* memo 66 is badly in need of a rewrite and should probably not be consulted as it can be very misleading in the context of the `15APR92` and later *AIPS* releases.

### *2.5.11 Batch Error Terminal/File*

If you elect to have *AIPS* batch streams (*i.e.*, where jobs run sequentially with lower execution priority), a provision should be made for catastrophic error logging. The environment variable `BATCH_OUT` should be defined as either a terminal or a simple disk file for this purpose. The system console may be a good choice, but a simple disk file will do just as well. This is defined in `AIPSASSN.SH` and `AIPSASSN.CSH` and defaults to `/dev/console`.

### *2.5.12 Printers and Plotters*

The printer setup has changed with the `15JUL94` version. Whereas the information was previously specified in a `PRDEVS.SH` file via direct shell definitions, now there is a `PRDEVS.LIST` file that will be created in your `$NET0` area by `INSTEP1`. See the section "Printer Setup" (2.3.9) for an example of what one looks like.

The `PRDEVS.SH` file is now generic and resides in the `$SYSUNIX` area; for backwards compatibility, a symbolic link to it is placed in your `$AIPS_ROOT` directory. Also, its behaviour appears as before to the remainder of the shell scripts and so it can safely be used — with the new `PRDEVS.LIST` file — with older versions of *AIPS* (at least back to `15APR92`).

`INSTEP1` will, if an older `PRDEVS.SH` file is found in the `$AIPS_ROOT` area (and the `PRDEVS.LIST` file is not in `$NET0`), attempt to create the `PRDEVS.LIST` file from the older definitions. If this is successful, no further questions will be asked.

If an older `PRDEVS.SH` file is not found, but there is an `/etc/printcap` file on this system, an attempt will be made to parse that file and extract the names of all printers defined therein. If an older `PRDEVS.SH` file is not found, and neither is `/etc/printcap`, then the script simply asks you how many printers you wish to define.

For each printer, you will be asked for its name (unless the `printcap` name was found), its type (one of tt PS for PostScript, `PS-CMYK` for color PostScript, `QMS` for QUIC printers, `TEXT` for plain text, or `PREVIEW` for screen previewers. Finally, you will be prompted for a description. You may also use type `REMOTE`; see the description of the file and this option earlier in this document for details.

The shell scripts `ZLASCL` and `ZLPCL2` have been made even more generic than before, with the use of the `F2PS` and `F2TEXT` filters. There should be no need to edit these files, unless you wish to change, *e.g.*, to A4 format paper. There are architecture-specific versions of `ZLPCL2` for IBM, Cray, and Solaris systems; the one for Solaris should be moved or removed as it does not use the new filters. These will be in the `$SYSLOCAL` area for these systems.

The printer selection is achieved in one of two ways. If there is only one printer specified in `PRDEVS.LIST`, then of course that has to be "the" printer. If there are more than one, however, the default action is to pop up a menu to the user at *AIPS* startup, show the descriptions, and ask for a number. Alternately, users can use the `pr=<n>` command line option when starting *AIPS* to select a printer by number and bypass the menu.

Once a printer has been selected, the shell scripts will handle text and graphics output in the most sensible way. Suppose the user selected a PostScript printer. Then any text (such as `PRTMSG` output) will be filtered through the text-to-postscript filter specified in `ZLASCL` and spooled to the appropriate printer. Any PostScript graphics (from `LWPLA`) will be sent directly without any filtering. Finally, if QMS output is detected (such as the output of `QMSPL`), the script emits an error message and looks at the list of printers to see if there is a QMS printer defined anywhere. If it finds one, it sends the QMS graphics file to that printer with another warning message sent to the user. This scheme also applies if the user has selected a text or QMS printer and some QMS or postscript output is detected.

The printing scripts have the ability to force large print jobs to go to a designated printer. This is used extensively at the AOC site to prevent too many hundred-page listings from tying up the smaller postscript printers and using a fast line printer for them instead. By setting the variables `BIGPRINT` and `BIGTHRESH`, you can enable this feature, and by *not* defining them, you can inhibit it. The former variable is the number of the "big" printer, and the latter is the number of lines (calculated by `wc`) that is the maximum for the other printers.

If you have a color PS printer, you should set its type to `PS-CMYK` in the `PRDEVS.LIST` file instead of just plain `PS`, and the script will then be able to catch color output (such as from the task `TVCPS` and send it directly to the color printer regardless of the printer specified by the user. However, the "feature" of diverting monochrome output from a color printer that was present in previous versions of *AIPS* has been discontinued. If your users attempt to do this, they will see a message:

```
Warning!  Monochrome file on color PS printer!
```

Postscript laser printers are the best supported plotting devices in *AIPS*. Tasks exist for generating graphics output for a variety of laser printers and graphics languages including `QMSPL` (QMS or Talaris, QUIC language), `LWPLA` (LaserWriter/PostScript) and `CANPL` (Canon LBP–8). Note that there is no support in the `ZLASCL/ZLPCL2` and `PRDEVS.SH` scheme for Canon printers. This would have to be added by a site with a Canon printer (obviously). `QMSPL` and `CANPL` make use of the `ZLAS*` C routines to create, open, close and dispose laser printer output files.

A task still exists that has fallen into disuse at NRAO but still may be useful at other *AIPS* sites. It is called `PRTPL` and it calls a subroutine `ZDOPRT`. Several versions of `ZDOPRT` can be found in `$APLVMS` for various printer-plotter devices (*e.g.*, Versatec, Printronix). The `ZDOPRT` routines *will require local development for Unix systems* since NRAO has never had such devices on any of its Unix *AIPS* systems, past or present.

Alternatively, if you have a (non-PostScript) laser printer with graphics capability, you may want to use the task `QMSPL` and the `ZLAS*` routines as models for a task to exploit it. Some modification to the `PRDEVS.SH` file and the `ZLASCL` and `ZLPCL2` scripts will then be needed.

## 2.6 Installation Step 3

### *2.6.1 Running INSTEP3*

The INSTEP3 procedure tends to be fairly short, typically taking from 10 minutes to over an hour, depending on the computer and compiler. It is quite significantly shorter than INSTEP2. At the risk of restating the obvious, it is unnecessary to run the procedure for binary installations. For source installations, simply type:

```
INSTEP3
```

As with INSTEP2, it is vital that the programming "logicals" be defined *before* you start it (*i.e.*, do a $CDTST first). The purpose of INSTEP3 is to compile a selected subset (about 20) of all the possible programs in the *AIPS* system (over 300 and still growing). The resulting executable modules are moved to the appropriate $LOAD (or $LOADn) directory. The subset represents:

- the stand-alone program FILAIP which creates the *AIPS* system files and initializes all but the *POPS* memory files;
- the stand-alone program POPSGN which initializes the *POPS* memory files that are created by FILAIP;
- the stand-alone programs SETPAR (general) and SETTVP (TV oriented) for setting values in the system parameter file created by FILAIP;
- the system specific *AIPS* startup program ZSTRTA, and AIPS itself;
- the tasks DISKU and NOBAT;
- a subset of tasks originally known as the *Dirty Dozen* which includes FITLD, UVSRT, UVDIF, UVMAP, COMB, APCLN, SUBIM, CCMRG, CALIB, MX, VTESS, PRTAC and FITTP; and
- network tasks TPMON, TEKSRV, and MSGSRV.

INSTEP3 begins by checking that the source code for these programs exists, and building an "at" file called INSTEP3.LIS containing the source code path names. It then drives the shell script COMLNK which in turn compiles the programs and links the resulting object module with the $LIBR/*/SUBLIB files as prescribed in $SYSLOCAL/LIBR.DAT.

### *2.6.2 Running FILAIP*

Once INSTEP3 has completed, you will need to execute (in this order) FILAIP, POPSGN and SETPAR for source-only installations. If you are performing a binary installation, you have the option of using the ready-to-run system files on the tape, or to regenerate them. INSTEP1 will have presented a list of the parameters used to generate the files on tape:

```
INSTEP1:  #disks, #catalog entries per disk:  15 -100
INSTEP1:  #interactive AIPS, #batch queues:  4 2
INSTEP1:  #TV, #TK devices:  10 255
INSTEP1:  #tape drives:  4
```

These should be adequate for most installations. If you wish to increase any of the parameters other than number of disks and catalog entries, you will have to regenerate the files. INSTEP1 will run the program if you indicate you wish to change the numbers.

For those systems with hardware TV display devices it will be necessary to execute SETTVP as well (it's not necessary for the screen servers); however, this can wait until you are ready to actually load images to the device (but should be done before running the SYSETUP script). To execute FILAIP, simply type:

```
RUN FILAIP
```

The generic procedure RUN, which resides in the directory $SYSUNIX, is simply an aid to running (via an exec command) the various standalone tasks outside of AIPS and BATER. This procedure will call the

HOSTS.SH, AIPSPATH.SH, and AIPSASSN.SH files in the $AIPS_ROOT area. It also calls the DADEVS.SH file in $SYSUNIX so that all the data areas are defined. For most programs, it's just the first data disk that matters, but RUN does support the da= argument in exactly the same way as the START_AIPS script, if you should need this.

It uses the environment variable $AIPS_VERSION to find the file (in the LOAD subdirectory of same). To change *AIPS* versions, use $CDOLD, $CDNEW, and $CDTST prior to running these programs.

It is essential that you make sure the *AIPS* "disk 1" area, reported by RUN as it starts up, exists and is writable by the installation account. While most of the files will be created in the host-specific system area $DA00, one (the message file for user 1) has to be created in $DA01. If there is already a message file for user 1 there, FILAIP will detect and not overwrite it.

If you are running FILAIP for the first time, it will emit a brief copyright notice. Then it asks you for the number of disks and catalog entries per disk. The number of disks gets overridden via the DADEVS.SH file (it sets NVOL as the number of disks), so just enter 15 (the maximum you can have in any one *AIPS* session). It is highly recommended that you use private catalogs by negating the number of catalog entries. Otherwise, all *AIPS* users will share one (very large) catalog which is a single point of failure. Also, catalogs are extensible so the number itself is not critical.

```
# disks, # cat entries/disk (<0 => private catlgs) (2 I)
15 -100
```

Next, you will be asked the maximum number of simultaneous *AIPS* users allowed (up to 12), as well as the number of *AIPS* batch queues (if any) you want.

```
# interactive AIPS, # batch queues (2 I)
8 2
```

Each *AIPS* session, interactive or "batch", has its own unique memory file to store values of adverbs and other things, and FILAIP needs to know how many of these to create. They are created in $DA00 which is of course host-specific. Once you create them for one system, you can use SYSETUP to copy them to all other hosts of the same architecture, after you copy some of them to a template area. See the section below on "*Setting up a Template Area*"). The *POPS* number of your *AIPS* session will match the memory file you get. The number created is always the sum of the interactive *AIPS* and batch queues +1, so in the above example 11 memory files will be created. You can have a maximum of 15 (hex F) memory files. It is hard to expand the number of memory files later so be generous with these parameters. You can always use SETPAR (below) to restrict these numbers later.

Since the 15APR92 release, the memory file has been split in two. The first part is independent of *POPS* number or *AIPS* session and is stored in $AIPS_VERSION/$ARCH/MEMORY/MEC00001;1. The other part is dependent on the *POPS* number and so there are more than one of them. They are stored in the host-specific system area $AIPS_ROOT/DA00/$HOST/MEC0000n;1 where the n will be the hex number from one to whatever. *Both of these areas and the files in them must be writable by anyone who will use your AIPS system.*

The next two questions ask how many TV (image display) and graphics (tek or TK) devices you want to set up, and how many tape drives you want to use. These should correspond to the numbers of (a) AIPS hosts plus TV servers defined in HOSTS.LIST, (b) 255 for the number of graphics devices, and (c) the maximum number of tape drives on any of your hosts plus 1 (for now):

```
# TV, # TK devices (2 I)
10 255
# tape drives (I)
4
```

It is again far easier to specify a larger number of TV's than you currently have as it makes adding new TV servers a lot easier. Otherwise you will have to go through a somewhat tortuous recipe in order to create new image catalogs for each new system you add. If you specified one or more TV's, you will be asked for some basic information on each TV, *e.g.*:

```
Image device No. 1 #gray, #graph, #img/gray, #ISUs (4 I)
```

```
4 4 64 0                        (this is for an IIS)
Image device No. 2 #gray, #graph, #img/gray, #ISUs (4 I)
3 4 256 0                       (this is for an IVAS)
Image device No. 3 #gray, #graph, #img/gray, #ISUs (4 I)
2 4 256 0                       (this is for a TV server)
Image device No. 4 #gray, #graph, #img/gray, #ISUs (4 I)
2 4 256 0                       (this is for a TV server too.)
Image device No. 5 #gray, #graph, #img/gray, #ISUs (4 I)
2 4 256 0                       (so is this.  Yeah, it's tedious)

(inputs for 6 through 10 omitted for sanity)
```

For installations with large numbers of TV servers, this procedure is a bit painful; our apologies, the *AIPS* group will try to make it a bit easier for the next release.

These four quantities are: The number of grayscale image planes (*not* bitplanes), the number of auxiliary graphics planes, the number of gray or color levels per image plane, and the number of image storage units. Don't worry about what this last item is; you probably don't have one and are not likely to see one except at the VLA/AOC sites of NRAO. Finally, `FILAIP` terminates with this message:

```
FILAI1: Init POPS memory files 1 through 11 with program POPSGN
FILAI1: Done!
```

When `FILAIP` completes, on some versions of SunOS you may see what appears to be the rather scary looking message:

```
Note: Following IEEE floating-point traps enabled; see ieee_handler(3M):
Overflow;  Division by Zero;  Invalid Operand;
Sun's implementation of IEEE arithmetic is discussed in
the Numerical Computation Guide.
```

Do not be concerned about this, as it says the traps are *enabled*, not that any occured. If you see the message here, you will probably see it everytime you finish `AIPS` or any of the *AIPS* standalone tasks.

`FILAIP` will create a number of system files in the `$DA00` area. The table below summarizes these files and their purpose:

| File type | Number | Description |
|---|---|---|
| ACf00000;1 | 1 | Accounting file |
| BAf00q0n;1 | 1–? | Batch text files; q is queue number, n is *POPS* no. |
| BQf00000;1 | 1 | Batch queueing file |
| ICf00000;1 | 1 | "Image" catalog file for Tek graphics |
| ICf0000n;1 | #TV's | Image catalog for each tv, $1 <= n <= \#TV's$ |
| IDf0000n;1 | #TV's | TV lock file (for who currently has it open) |
| PWf00000;1 | 1 | Password file |
| SPf00000;1 | 1 | System parameter file (use `SETPAR` to change) |
| TCf00000;1 | 1 | "Show and Tell" file for running tasks |
| TDf00000;1 | 1 | Task data file; communication from spawned tasks to *AIPS* |
| TPf0010n;1 | #tapes | Tape lock file to avoid contention |

In the above table, the third letter of each filename (indicated by `f`) indicates the data format version; the current letter for this is `C`, hence the accounting file will be `ACC00000;1`, and likewise for the other files. Format type "A" was introduced as of `15OCT86`, "B" at `15JAN87`, and "C" at `15OCT89`. Prior to that numbers were used.

`FILAIP` will also create the file `MSf00100.001;1` in the `$DA01` area. This is the message file for user 1 (the *AIPS* system manager). As mentioned at the start of this section, the first *AIPS* "data disk" area `$DA01` needs to exist and be writable prior to running `FILAIP`.

### *2.6.3 Passwords*

Among the files created by **FILAIP** is a password file. All passwords are initially blank except for user #1 which is reserved for the AIPS manager. The password for user #1 is initially **AMANAGER** (upper case; the checking is case sensitive). Since the password entry for user #1 is initially non-blank, whenever programs run as user #1, the password will be required. The stand-alone utility programs such as **FILAIP**, **POPSGN**, **SETPAR**, **SETTVP**, *etc.*. all run as user #1 by default. For example, if after successfully running **FILAIP** as described above, you were to re-run it, the first prompt you would get would be for the password (since the password file now exists). When typing the password, it will not be displayed on the terminal and you will only be given a few chances to get it right before the program will give up and exit.

Passwords may be changed for the current user via the `PASSWORD` verb in *AIPS*. User #1 is permitted to set the password for any other user number. If the password is forgotten for user #1, you will have to delete or move the entire password file and recreate it via the program **FILINI**. Make sure you know how big the password file needs to be before deleting the old one.

The default setup for the network scheme is to have one instance of the password file created, and have it hard-linked in the `$DA00` areas of the other hosts. Thus, a change of password for one user on a given host will automatically change it on all hosts.

### *2.6.4 Running POPSGN*

*POPS* is the command language of *AIPS*. The *POPS* memory files created by `FILAIP` must be initialized as a separate step. If you fail to do this, *AIPS* won't have the foggiest idea what you're talking about, not even the `EXIT` command. This is the sole purpose of the program `POPSGN` which prompts you for its inputs. To execute `POPSGN`, simply type:

```
RUN POPSGN
```

The following is an example execution of `POPSGN` that will initialize the memory files created from the `FILAIP` example above. What you type is in **BoldFace**, computer response in `typewriter`:

```
% RUN POPSGN
Data disk assignments:
   Disk 1 is /DATA/ORANGUTAN_1

Starting up POPSGN (RELEASE OF 15JAN94)
Enter Idebug, Mname, Version (1 I, 2 A's) (NO COMMAS)
0 POPSDAT TST
>                                   (press RETURN when you see this)
POPSG1:  Popsgen complete
POPSG1:  UNKNOW  15JAN94 NEW: Cpu=    1.53  Real=   12.0  IOcount=    49
```

The `Idebug` response should always be zero. Your response for `Mname` should always be `POPSDAT` (yes, uppercase). A file called `POPSDAT.HLP` is stored in the directory `$HLPFIL` which contains the initialization values for the various parameters and adverbs that will be stored in the memory files. `POPSGN` will accept a different name and version for this file, but until you know what you're doing, don't try it*.

The final input to `POPSGN` is the version name of the memory files to initialize. This should be given as `TST`.

After entering this information, `POPSGN` will grind away for a short while and eventually issue a > prompt. Simply type a carriage return at this point.

The next step is to run the program `SETPAR` and perhaps `SETTVP` as well to set any parameters not initialized correctly by `FILAIP` during its creation of the system parameter file. Then copying various files to a template directory should be done for network sites, and the `SYSETUP` shell script can then be run to set up the system (`$DA00`) areas for all the other hosts.

* In order to add adverbs and/or verbs to the *AIPS* command set, `$HLPFIL/POPSDAT.HLP` (or a copy of it) must be modified to incorporate the new definitions and `POPSGN` must be rerun to re-initialize the *POPS* memory files; See the "Going AIPS" programming manuals for more on adding verbs and adverbs.

### *2.6.5 Running SETPAR and SETTVP (and maybe SETSP)*

Even though `FILAIP` creates the system parameter file and initializes some of the parameters, it does not initialize all of them. For the moment, it is not particularly crucial to set the remaining parameters, as there are halfway-sensible defaults already set.

In test-installing *AIPS* on many systems, the author has found that item 19 in the `SETPAR` list, *i.e.* the local system identification (name), is probably the only one that definitely should be set before using *AIPS*. It is a 20-character string that describes the host you are installing *AIPS* on, the first 6 of which get printed out whenever an *AIPS* task completes (more of it appears on some printouts).

Another interesting parameter is item 29, the "2nd AP size". It now defaults to whatever value the variable `PKPWD2` has in the file `PAPC.INC`. This file is found in either `$INC`, the include file area, or a system-specific version of this (*e.g.*, `$INCALLN`). The default value is 1216 which gives a total pseudo-AP size of 5 Megabytes. On systems with significant amounts of memory, you may want to raise this limit to, *e.g.*, 17 or 65 megabytes (these would allow a full 2k or 4k image, respectively, to fit in memory).

The following is an example execution of SETPAR and the parameter values are from our *AIPS* environment on a typical NRAO-CV SparcStation IPX `orangutan`:

```
$ RUN SETPAR
Data disk assignments:
  Using host-specific file /AIPS/DA00/BABOON/DADEVS.LIST
  for AIPS data area definitions ("disks")
   Disk 1 is /DATA/BABOON_1
   Disk 2 is /DATA/BABOON_4

Starting up SETPAR (RELEASE OF 15JUL94)
Enter:  1=Start Over, 2=Change parameters, 3=Change DEVTAB, 4=Quit
2
  1  No. of AIPS data disks                 15
  2  No. of tape drives                      3
  3  No. of lines per CRT page              24
  4  No. of lines per print page            97
  5  No. of batch queues                     2
  6  Plotter no. of X dots per page       2112
  7  Plotter no. of Y dots per page       1600
  8  Plotter no. of X dots per character    20
  9  Plotter no. of Y dots per character    25
 10  No. of interactive AIPS                 7
 11  No. of words in REAL AP (in 1024s)     64
 12  No. of TV devices available            31
 13  No. of graphics devices available     240
 14  No. of X dots per mm on printer     7.830
 15  No. of X dots per mm on tektronix   5.000
 16  No. of POPS allowed access to TVs      15
 17  No. of POPS allowed access to TKs      15
 18  No. entries in private catalogs       500
 19  Local system identification (A20)  Baboon (NRAO-CV SS2)
 20  Maximum user number (<= 4095)        4095
 21  TIMDEST minima disks  1 -  3 days   14.  14.  14.
 21  TIMDEST minima disks  4 -  6 days   14.  14.  14.
 21  TIMDEST minima disks  7 -  9 days   14.  14.  14.
 21  TIMDEST minima disks 10 - 12 days   14.  14.  14.
 21  TIMDEST minima disks 13 - 15 days   14.  14.  14.
 22  TIMDEST limit: SAVE/GET files       365.
```

```
 23  TIMDEST/EXIT limit: messages          3.
 24  TIMDEST limit: scratch files          3.
 25  TIMDEST limit: empty CA files       0.250
 26  No AP batch starts: weekend hours   0.000 0.000
     No AP batch starts: weekday hours   0.000 0.000
 27  AP roll interval minutes            5.000
     AP patience: (1)*N+(2)*N*N minutes  5.000 1.000
 28  Line printer width (72 - 132)         132
 29  Pseudo-AP 2nd memory (1024s)         1216
 30  Max length of "short" vector            0
 31  Graphics (TK) screen size: x, y      1024   780
 32  Graphics (TK) character size: x, y     14    22
 33  Disk & reserved users or -1 scratch (9 I)
     Disk  2 Users    0
     Disk  3 Users    0
     Disk  4 Users    0
     Disk  5 Users    0
     Disk  6 Users    0
     Disk  7 Users    0
     Disk  8 Users    0
     Disk  9 Users    0
     Disk 10 Users    0
     Disk 11 Users    0
     Disk 12 Users    0
     Disk 13 Users    0
     Disk 14 Users    0
     Disk 15 Users    0
 34  Delay time print file delete (secs)  1200
Enter number to change or  0 = Print, -1 = Return
-1
Enter:  1=Start Over, 2=Change parameters, 3=Change DEVTAB, 4=Quit
4
```

Many of the parameters are self-explanatory. Others are largely irrelevant, *e.g.*, the AP parameters if you do not configure batch queues. In general the figures above are what you should use, at least for the more obscure items.

New to this release is the presence of item 34, "*Delay time print file delete*". This quantity governs the time delay between submission of a print file to be printed and its deletion by AIPS. If set to zero, it means the temporary files created by any and all printing/plotting verbs and tasks (*e.g.*, LWPLA, PRTMSG) *will not get deleted*. If set to a non-zero positive number, the units are seconds, and the recommended value is at least 300 (5 minutes). While some Unix spooling systems permit a "delete on printed" flag to the lpr or lp command, this option is not available in some systems, *e.g.*, Solaris. The time-delayed deleting of print files is a compromise, and while it solves the problem for most cases, there are some times when it will not work. If the time interval is too short and the print queue very busy, files may get deleted before they get printed. If it is too long, the print spool area may fill up.

The disk parameters shown above are mostly irrelevant, as they get overridden. The number of disks is obviously variable, hence item 1 above is meaningless; it gets overridden by the NVOL environment variable set by DADEVS.SH on AIPS startup. Likewise, item 21 — the TIMDEST parameters — are dictated by the values in the NETSP file. If users at your site use the personal .dadevs file to point at areas that are *not* referenced in NETSP, then the values shown by SETPAR for the number that corresponds to that particular data area will be used instead, so if you anticipate users defining personal data areas that will not be referenced in NETSP, it may be wise to set some suitable default parameters in item 21. Finally, item 33 (reserved users on the data areas) is again overridden by the values in NETSP.

Check that item 12, the number of TV devices, is set to the number of TV servers (*AIPS* hosts and TV hosts) and TV devices actually used at your site, or better still set it higher than this so you don't need to increase the number when adding a new host.

There are two parameters related to the "AP" in the list above. Items 11 and 29 will always reflect the parameters the system was built with when you run `SETPAR` for the first time. The value for *No. of words in AP (in 1024s)* should be `64` for any installation using the Pseudo-AP Q routines (*i.e.*, most if not all modern installations). If you have a real AP, set the size accordingly (and drop us a line; we are unaware of any Unix systems with such a device).

The *Pseudo-AP 2nd memory (1024s)* parameter will be set at `1216` for most systems; this gives a total pseudo AP memory size of 5 Megabytes. If you changed the value in the `PAPC.INC` file to define a larger AP array size before `INSTEP2`, then the size you chose will be reflected in what you see in `SETPAR`. You can of course adjust this value downwards by using `SETPAR`, but not upwards beyond the pre-compiled limit you have set. If you want to change the AP size upwards, you will need to change the corresponding values in the `PAPC.INC` include file, and also recompile all routines in the `$QPSAP` and `$QOOP` areas, as well as rebuilding all tasks that call them (areas `QPGM`, `QPGNOT`, `QPGOOP`, `QYPGM`, and `QYPGNOT`).

If you want to support an AIPS environment where one host has a large AP size, perhaps because of its large memory size, but the remaining AIPS hosts do not, the use of `SETPAR` is recommended. This has been used at NRAO to provide an 80-megabyte AP for large-memory systems via `PAPC.INC`, but reduce the parameter to 20 or 5 megabytes on smaller systems that use the same binaries. No obvious problems relating to memory or the AP were observed.

`SETTVP` is used to set TV specific parameters, and normally should never be needed. The default values are appropriate for the XAS TV server, and as long as your users remember to give `tvinit` as the first TV command when they start up. This will initiate a handshake type protocol between `XAS` and `AIPS` which will set the TV parameters to appropriate values as dictated by the parameters compiled into `XAS` and the X server under which it finds itself running. If you have not edited or changed the TV server's source code, and you do not have any hard-wired TV devices, skip past the examples to the description of `SETSP`.

The following examples are probably obsolete, as they show the parameters for a couple of "hard-wired" TV displays as they were on NRAO's Convex C1 systems almost two years ago. There were three types of TV supported on this system: an IIS model 70E, an IIS IVAS, and the AIPS TV server on remote workstations. Here are the values for the IIS as reported by the last released version of *AIPS* to run on that Convex:

```
% RUN SETTVP
Starting up SETTVP (RELEASE OF 15OCT92)
Enter 1=Init, 2/3=Change parms, 4=ISU, 5=quit and the TV # (2I2)
2 1
1  No. of gray-scale planes         (I)     4
2  No. of graphics overlay planes   (I)     4
3  No. of images / gray-plane       (I)    64
4  X,Y size of TV planes (pixels) (2 I)   512   480
5  Maximum gray-scale intensity     (I)   255
6  Peak intensity out of LUT        (I)   255
7  Peak intensities in/out of OFM (2 I)  1023  1023
8  X,Y min. scroll increments     (2 I)     1     1
9  Maximum zoom: (>0) power of 2    (I)     3
      (< 0) Max factor = 1 - MAXINT
10 Type of split-screen allowed     (I)     4
      1=Vert, 2=Hori, 3=Either, 4=Both
11 # X,Y pixels in TV characters  (2 I)     7     9
12 X-axis image write mode(s)       (I)     1
      0 - None, 1 -> Right, 2 -> Left
```

```
13 Y-axis image write mode(s)       (I)     2
      0 - None, 1 -> Up, 2 -> Down
Enter number to change, 0=print, -1=return (I2)
```

The X,Y size of TV planes is set here to $512 \times 480$ because the IIS was set in hardware to only display 480 lines (we had a film recorder connected that required this). Other IIS TV's had 512 lines configured.

Before showing the values of the parameters for the other TV's, some parameters should be explained further. The first three have already been described in the section on `FILAIP`. The terms LUT and OFM stand for Look-up table and Output Function Memory, both of which can be regarded as color tables (in a simplified way).

The following shows the parameters for an IIS IVAS display:

```
1  No. of gray-scale planes         (I)     3
2  No. of graphics overlay planes   (I)     4
3  No. of images / gray-plane       (I)   256
4  X,Y size of TV planes (pixels) (2 I)  1024  1024
5  Maximum gray-scale intensity     (I)   255
6  Peak intensity out of LUT        (I)   255
7  Peak intensities in/out of OFM (2 I)  1023  1023
8  X,Y min. scroll increments     (2 I)     1     1
9  Maximum zoom: (>0) power of 2    (I)     3
      (< 0) Max factor = 1 - MAXINT
10 Type of split-screen allowed     (I)     4
      1=Vert, 2=Hori, 3=Either, 4=Both
11 # X,Y pixels in TV characters  (2 I)    13    19
12 X-axis image write mode(s)       (I)     1
      0 - None, 1 -> Right, 2 -> Left
13 Y-axis image write mode(s)       (I)     2
      0 - None, 1 -> Up, 2 -> Down
```

And finally, here are the parameters for the *AIPS* TV servers:

```
1  No. of gray-scale planes         (I)     2
2  No. of graphics overlay planes   (I)     4
3  No. of images / gray-plane       (I)   256
4  X,Y size of TV planes (pixels) (2 I)  1142   800
5  Maximum gray-scale intensity     (I)   199
6  Peak intensity out of LUT        (I)   255
7  Peak intensities in/out of OFM (2 I)   255   255
8  X,Y min. scroll increments     (2 I)     1     1
9  Maximum zoom: (>0) power of 2    (I)   -15
      (< 0) Max factor = 1 - MAXINT
10 Type of split-screen allowed     (I)     0
      1=Vert, 2=Hori, 3=Either, 4=Both
11 # X,Y pixels in TV characters  (2 I)     7     9
12 X-axis image write mode(s)       (I)     3
      0 - None, 1 -> Right, 2 -> Left
13 Y-axis image write mode(s)       (I)     3
      0 - None, 1 -> Up, 2 -> Down
```

The only item not set in the handshake between the TV server and `AIPS` is item 3, and this will always be 256 unless you make drastic changes to the source code of the TV server and have an X display with more than 8 bits.

Finally, the `SETSP` program may be useful to large sites. It uses a file called `$NET0/SPLIST` to query the system parameter files on a large number of hosts. Here is an excerpt from ours:

```
AIPS_ROOT:DA00/ATEN/SPC00000;1
AIPS_ROOT:DA00/BABOON/SPC00000;1
AIPS_ROOT:DA00/BACH/SPC00000;1
AIPS_ROOT:DA00/ORANGUTAN/SPC00000;1
```

Basically, there is one entry for each host listing in (a weird hybrid of VMS and Unix) *AIPS* syntax the filename of the system parameter file for that host. When you `RUN SETSP`, it reads this list and looks at all these files. You can then look at one of the `SETPAR` parameters and see the values for *all* the hosts in one go. You can also set a value (same one across all hosts). The only disadvantage is that the `SPLIST` file must be maintained by hand. However, sites like NRAO and the ATNF have found it to be well worth the slight inconvenience.

The `SETSP` program is *not* automatically compiled by `INSTEP3`; if you want it, you must either complete `INSTEP4` or manually compile it with a `COMLNK $AIPPGM/SETSP.FOR` command.

### *2.6.6 Setting up TEMPLATE areas and running SYSETUP*

Once you have a set of system files that is suited to your installation for one host of a given architecture, you should create a `TEMPLATE/` directory under `$AIPS_VERSION/$ARCH/` if it does not already exist, and copy a subset of the files into it (while they are "virginal", *i.e.*, untouched by *AIPS*). While this has to be done by hand for the source-only installation, the binary installation gives you a choice: either use the pre-generated system files, or re-run `FILAIP` and `POPSGN` to modify the parameters. If you need to populate the template area by hand, the following line should do it (for Bourne-like shells):

```
$ cp $DA00/[ABMST]* $AIPS_VERSION/$ARCH/TEMPLATE/
$ cp $DA00/I[CD]C0000[01]* $AIPS_VERSION/$ARCH/TEMPLATE
```

You of course only need to go through this if you have more than one host of the same architecture. If you only have one system, but plan on adding one or more later, it is still best to set up the area at this point.

Now you are almost ready to run the `SYSETUP` script. An attempt has been made (with the `15JUL94` version) to insert the knowledge of IC and ID files into this script, but early reports indicate that it may not work perfectly; check the patch area on the *AIPS* anonymous ftp server, or contact the *AIPS* group if in doubt.

If you need to fix up the image catalogs by hand, first make sure you have enough image catalogs by counting the number of `IDC000*` files in your first `$DA00` area. Then compare this to the number of AIPS and TV hosts in `HOSTS.LIST` (this discussion assumes no "hardwired" TV's). If this sum exceeds the number of ID files, copy the `ICC00001;1` and `IDC00001;1` files in the template area to, *e.g.*, `ICC0001A;1` and `IDC0001A;1`*, in the `$DA00` area, if the next TV would be number 26 (hex 1A).

When you run the `SYSETUP` script, it will ask you for the *AIPS* manager account name (default `aipsmgr`), and the *AIPS* user group name (default `aipsuser`). You should be installing *AIPS* from the manager account, although you may elect to have the manager and `aips` accounts as one and the same. If installing *AIPS* from a personal account, set yourself up as the *AIPS* manager, at least for now. For simplicity you may want to select the default group for the account you installed *AIPS* from as the group. If you created a group specifically for *AIPS* users earlier in the installation, you should use that group.

Next it asks for the master host name. This should be the host name of the system you just installed *AIPS* on. If you have correctly configured the `HOSTS.LIST` file and extracts all hosts from it that match the current value of `$SITE`. Then it operates in a loop on all hosts in the list (skipping the one you indicated as the master). If you call it with, *e.g.*, `SYSETUP FOOBAR PLUGH`, it will only process hosts called foobar and plugh. Whether these are new or old, they should be already entered in `HOSTS.LIST`.

* You will have to escape the semicolons with a backslash for most shells.

For each host, it will check for the existence of several directories and set appropriate protections (user and group allowed write access) and setgid bits on them. If the `$DA00` area for that host does not exist, it gets created. It checks what architecture the host belongs to and sets that in the variable `$arch` (lowercase). Then it copies the files `$AIPS_VERSION/$arch/TEMPLATE/[ABMST]*` to the `$DA00` area for that host. This gets the accounting file, the batch files, memory files, system parameter and space lock file, task control file and tape files. These have to be host-specific by their nature. It also checks how many TV devices are defined (including TV servers) and makes sure there are enough IC and ID files in the master area. If not, it copies more from the template area.

Then it will insert hard links in the host's `$DA00` area to the master machine's gripe, password, and image catalog/lock files (`GR`, `PW`, `IC` and `ID` types). Note that it does *not* create new image catalogs; you have to do this yourself as indicated above if you add new hosts.

Finally it checks the architecture-specific area for that host to make sure the memory file is there.

Obviously, the strategy to adopt in a network *AIPS* setup is to run the installation procedure to at least `INSTEP3` on one system from each architecture you plan on supporting. Then make sure the template areas for all architectures are set up and (from any of the hosts that you just installed *AIPS* on) call the `SYSETUP` script and let it do the work.

If you have multiple "sites" in the same `$AIPS_ROOT` to support two or more incompatible architectures, you need to go through the template area creation and `SYSETUP` for each of them. The script will only process hosts within one `$SITE` variable and so limits itself to compatible architectures (if you have multiple sites defined in your `HOSTS.LIST`).

Once it is finished, you will need to run `SETPAR` on each host in turn to at least set the site name. Most other parameters can either be set via `SETSP` later (after `INSTEP4` which builds it) or are set now from the `DADEVS.LIST` or `NETSP` files.

### *2.6.7 Exercising AIPS*

The sections that follow will describe the various aspects of *AIPS* that will need to be tested to ensure that your new installation performs correctly. It also includes some information that will be helpful if any debugging needs to be performed.

In addition to checking for the usual things (see if it starts up, try `print 2 + 2`, look at a few inputs, try `GO DISKU` and so on), you can try the so-called DDT tests (if you're feeling ambitious). This will definitively tell you how well your installation of *AIPS* is working, both in terms of accuracy and timing. If you are porting *AIPS* to a new system previously untried by NRAO or anyone else you know, it is *especially* important that you perform this step. For already-ported-to systems, if the simple tests above work, you probably don't need to bother with the DDT.

However, please note that the DDT test underwent significant changes with the `15JAN94` release of *AIPS*. See *AIPS* memo 85 for the full details. The master DDT images are available via anonymous ftp from `baboon.cv.nrao.edu` in the directory `pub/aips/DDT/15JAN94`. Please exercise discretion and "netiquette" if you wish to retrieve the large ones!

When you have new DDT timing/accuracy results, *please* communicate the results to the *AIPS* group. We strive to keep those results in our memos and databases (and give you credit for it, of course!) Also if you are interested in timing tests, be sure to be aware of the various factors that can affect this: NFS versus local disks, system load or other users, 2nd AP memory size, slow v. fast SCSI disks, amount of memory, OS version, and so on. See *AIPS* memo 85 for more details.

### 2.6.8 AIPS Start-up Procedure

After all this work you are *almost* finally poised to start up *AIPS* itself. If `INSTEP1` did its work, you should just be able to type `aips` and have it start up for you. If you have not yet set up the network services required by *AIPS*, you can expect lots of error messages to be displayed on your window or terminal, but the basic system will still work. You just won't have things like the TV server, tek and message servers, and remote tape capability.

The way the `aips` command is made available is via the `LOGIN.CSH` or `LOGIN.SH` command; users who want access to `AIPS` should include the appropriate one in their login/profile files (see "*Login Procedures*" above). Then no additional work is necessary on their part (or hopefully yours) to enable them to use *AIPS*. It may be convenient for people in "uppercase mode" to make a symbolic link in `$SYSLOCAL` called `AIPS`, in addition to the one already there called `aips`.

You may find it useful to copy the *AIPS* "man" or manual page to your local "man" area (your sysadmin will probably know where this is), *e.g.*:

```
# cp $SYSUNIX/AIPS.L /usr/local/man/man1/aips.1
```

This will enable people to type `man aips` and get some useful information on the startup options, provided they have their `MANPATH` environment variable set up to include your local area, whether it be as indicated above or elsewhere.

The `START_AIPS` procedure goes through many steps, calling some of the other files in the `$AIPS_ROOT` and `$SYSUNIX` areas, before it finally performs an "exec" to the `$SYSUNIX/AIPSEXEC` procedure. This in turn is a stripped-down version of the old (pre-`15APR92`) `AIPS` shell script that used to start *AIPS* all by itself.

To initiate a `BATER` (*AIPS* batch job submission program), there is a procedure called `BATER` (and `bater`). This takes a subset of the command line arguments that `START_AIPS` takes, *i.e.*, a version name, `REMOTE`, `DEBUG`, or `LOCAL`; the `REMOTE` option has already been mentioned (graphics terminal). One problem with `BATER` is that it does not support the `da=` option.

When you invoke these procedures, after some setup the respective startup program is executed, `ZSTRTA` for `AIPS` or `ZSTRTB` for `BATER`. The major purpose of the `ZSTRT*` programs is to assign the lowest available *POPS* number to the current job that is not already in use (and not reserved for another terminal if reserved terminals are in use) and to invoke the actual program as a process with a name such as `AIPSx`, where `x` is the assigned *POPS* number. Tasks initiated from an `AIPSx` session inherit this *POPS* number and have similar process names (*e.g.*, `MX3`). To startup *AIPS* in the simplest case, simply type:

```
% aips
```

or for an example that shows some of the network features,

```
% aips pr=2 da=sparky,zippy tp=astro2 tv=foo:bar
```

The options in the latter case mean the following: Use printer number 2 from the menu (leave out the `pr=x` to get the menu), include *AIPS* data areas from remote hosts sparky and zippy, make sure the `TPMON` daemons are running on remote host astro2, and override the default TV settings, using (X11) display on host foo while running the `XAS` TV server itself on host bar. As you can see, the system is quite flexible. The only caveat is that most of the above hosts need to be *AIPS* hosts. The exception is machine foo, which just has to have its X server grant permission to display X11 images from machine bar. For additional details on these options, see the *AIPS* manual page in `$SYSAIPS` (or wherever you installed it), or type `HELP AIPS` inside `AIPS` itself.

The first thing you will see in the former case (if you have more than one printer configured) is something like this:

```
$ aips

You have a choice of 10 printers.  These are:
```

```
      1.  Library FAST PostScript printer, Duplex
      2.  Library Fast PostScript printer, normal
      3.  Postscript printer downstairs
      4.  Postscript printer in room 215
      5.  Color Postscript printer, AIPS Cage
      6.  Postscript 35mm slides
      7.  GNU Ghostview X-windows previewer
      8.  Sun OpenWindows previewer
      9.  QMS1725 Postscript printer in GREEN BANK
      10.  Printer not.available, assume PostScript

Enter your choice:
```

This is of course specific to NRAO's Charlottesville setup but your output should be similar and reflect your `PRDEVS.LIST` file. The tenth printer would have indicated something other than `not.available` had the `PRINTER` environment variable been defined. At this point, you enter a number for the printer you want, and continue:

```
The output print queue is ps0dup, type PS

Data disk assignments:
  Using site-wide default file /AIPS/DA00/DADEVS.LIST
  for AIPS data area definitions ("disks")
   Disk 1 is /DATA/POLARIS_1
   Disk 2 is /DATA/BABOON_1
   Disk 3 is /DATA/BABOON_2

Tape assignments:
   Tape 1 is Exabyte 8500 (Sun) on BABOON
   Tape 2 is REMOTE
   Tape 3 is REMOTE

You seem to be at a workstation called baboon

===========================  AIPS NEWS  ============================

   *   This is the AIPS login notice.  If this file is in $AIPS_ROOT,
       it will be printed out each time AIPS is started up.  You may
       also want to have it printed out by the .login or .profile if
       you have a dedicated AIPS account.

======================================================================

Starting up 15JUL94 AIPS with normal priority
Begin the one true AIPS number 1 (release of 15JUL94) at priority =   0
AIPS 1: You are assigned TV device   5
AIPS 1: You are assigned graphics device   5
AIPS 1: Enter user ID number
?
```

You will probably see (asynchrously) the message:

```
STARTPMON: Starting TPMON1 on BABOON...
```

and also (later):

```
TPMON1:  begins
```

There will be additional messages that may come out asynchrously for additional `TPMON` dæmons, and also some from the XAS TV server and the `TEKSRV` and `MSGSRV` servers:

```
XASERVERS: Starting TEKSRV on baboon, DISPLAY :0
XASERVERS: Starting MSGSRV on baboon, DISPLAY :0
XASERVERS: Starting XAS on baboon, DISPLAY :0
```

The startup procedure tries to identify the username and `DISPLAY` variable if a server is found to be already running.

Unfortunately, these asynchronous messages may come out just as the poor user is trying to enter the user number, and it can be a bit confusing.

If you tried this command from a workstation running X11 (including OpenWindows, Motif, VUE, *etc.*), three extra windows should have appeared (two of them iconified). One will be the TV server; the XAS icon has a picture of a gorilla on it with "new XAS" in tiny letters below. Another (also iconified) will be an `xterm` that runs the `TEKSRV` program and displays graphics. The final one will be the `MSGSRV` message server and will probably *not* appear as an icon, but as an open `xterm` instead. You can, of course, customize the startup location and characteristics of all these as indicated above via the `.Xdefaults` file.

If your windowing system is already displaying a lot of colors (*e.g.*, a background color image on the root window), XAS may start up with a virtual color map. When you open the XAS window and move the cursor into it (or click on it if you use click-to-type), the colors on your display will probably change suddenly as the virtual color map becomes the default. When you move the focus (move the mouse or click on a different window), the default color map will be restored. This is normal behaviour. It can be avoided by putting as few colors as possible into your windowing system before `XAS` is started, or lowering the number of colors available for `XAS` via the X resources; again, see the example earlier in this document.

Note also the "AIPS NEWS" message; this is from the file `AIPS.MSG` which is in the `$AIPS_ROOT` area. It is a good place to put messages concerning the *AIPS* system for all your users to see. The original is in `$SYSUNIX`.

You should see the `>` prompt at the end of the session after entering your AIPS (VLA) user number (improvise if you don't have one but leave 1 for the *AIPS* manager). If this is not the first time starting, you will see the "Recovered POPS environment" message.

If you have an older version of *AIPS* and are sharing the data areas, you may see the message:

```
VERS TOO OLD
ABORT!
```

when you first start the new version up. This is not a significant problem; it is simply the result of a failed attempt to read the `LASTEXIT` "save-get" file that contains various settings from a given user number's last *AIPS* session. The `15JUL94` save-get files are incompatible with some older versions. Conversely, if you switch from the new version to the old, it will complain about `VERS TOO NEW`. These messages will not hamper you in any other way.

### 2.6.9 Task Initiation

Assuming *AIPS* in fact starts up, you're ready to start exercising it a bit. To make sure tasks are activated properly, type the following inside *AIPS*:

```
> GO DISKU
```

This should result in a fairly uninteresting report on disk space usage (unless there is some real data on the disks in which case it may be enlightening, or even embarrassing). You may want to check the inputs* before running it. You may also try the `NOBAT` task; this is simply designed to sleep for the specified time, preventing batch jobs from running.

If your message server is running, you will see the task output come out in its window, whereas output from the main `AIPS` program comes out in whichever terminal emulation window where you started `AIPS`.

### 2.6.10 Tape Drives

The major tasks that use the tape drives other than `AIPS` itself are `FITLD` and `FITTP`; the former reads FITS data and the latter writes it. Both can handle either UV or Image data, and both accommodate tape or disk as the medium. There are two older tasks: `UVLOD` (reads UV data only), and `IMLOD` (reads images) but these each only have a subset of the functionality of `FITLD`. The latter also has more VLBI capabilities than its predecessors.

Getting tape I/O to work correctly in *AIPS* is often extremely difficult and frustrating. Any tape device that *AIPS* uses *must support backspace operations on a per-file and per-record basis*. This eliminates all quarter-inch cartridge (QIC) devices as the nature of the format they use prohibits such operations. In addition, the tape driver must support variable record-length read and write operations.

The success or failure of *AIPS* tape I/O depends critically on the tape interface routines `ZTAP2` and `ZMOUN2`. These will require local development for new ports, and they may require modification for existing ports that have tape systems other than the specific types NRAO has direct experience with. Versions of `ZTAP2` already exist in the various subdirectories of `$APLUNIX` and these should work without modification on tape drives as follows:

- Exabyte 8mm 8200 and 8500 drives on a Sun, SCSI bus (SunOS 4.1, 5.1, 5.2, 5.3) Vendor: Sun
- Exabyte 8mm 8200 on Sun (SunOS 4.1, SCSI) Vendor: CoComp
- Exabyte 8500 on Sun (SunOS 5.3, SCSI) Vendor: ZZYZX
- 4mm DAT drive on a Sun, SCSI bus (SunOS, possibly with patches for backspace, variable records, and knows-end-of-data). Vendors: Exabyte, ZZYZX (HP 35470A)

- IBM-supplied Exabyte 8200 and 8500 drives, SCSI bus (AIX 3.1, 3.2)
- IBM-supplied 9-track drive (9348-012), SCSI bus (AIX 3.1, 3.2)
- HP 9-track 88780B drive, SCSI bus (AIX, SunOS too?)
- HP DAT drive on IBM, SCSI bus (AIX 3.2 or later)
- ZZYZX 1.3Gb 4mm DAT drive on IBM, SCSI, 150Mb personality
- ZZYZX Exabyte 8500 on IBM, SCSI bus (AIX 3.2 or later)

- Convex 9-track drives (Convex-OS 8.0 or later)
- Storage-Tek 9-track drives on a convex (Convex-OS 8.0 or later)

- TK50 on SCSI bus on DecStation/System (Ultrix 4.0/4.3); Exabyte also?
- DEC 4mm DAT drive on Alpha OSF/1 systems (OSF/1 1.2) and Intel/Linux systems with SCSI card.

- HP 4mm DAT drive on HP 9000/7xx series (HP-UX 9.x).

* just type INPUTS DISKU to see them

For anything else, you should do considerable testing with both reading, writing and tape motion (verb `AVFILE`) to verify that the system works correctly. If not, you will have to modify the `ZTAP2` and possibly the `ZMOUN2` routines, both of which have extensive debug `printf` statements in comments ready for your use.

The selection of tape drives above reflects those NRAO had in routine operation with `AIPS` at the time this document was last updated; it does not endorse or recommend any particular vendor or model. If you are selecting a vendor for a tape drive and are concerned whether it will work with *AIPS*, you should impress on your sales representative the requirement that the device provide status information (current file, record number), perform backspaces over files and records, and reliably report end of information and beginning of tape.

The way in which tape "mounting" in *AIPS* works is as follows. The file `TPDEVS.LIST` should contain the tape definitions for all systems within a given site. The tape names therein may need to be in a specific format; comments in the file and in `INSTEP1` should be a guide for the format needed. The reason for this need will be obvious momentarily. When a user issues the `MOUNT` command from within `AIPS`, the `ZMOUN2` routine is called and is expected to do the following:

1. Check what sort of tape drive it actually is, if possible, via system calls.
2. Retrieve the value of the `TAPEn` environment variable; `TPDEVS.SH` will have defined this at `AIPS` startup (`n` is a number).
3. Check the value of the `DENSITY` adverb from `AIPS`.
4. Modify the device name if the density and tape drive type permit it, *i.e.*, dual density exabytes or multi-density 9-track.
5. Store the modified device name in a special environment variable `AMT0n`.

Subsequent access to the tape device will be through this hidden `AMT0n` environment variable. The user should normally not ever know about this variable, and it should *only* need to be set in the `ZMOUN2.C` routine.

The `$APLGEN` directory itself contains stubbed versions of `ZTAP2` and `ZMOUN2`. If these are the versions that actually get processed by the `INSTEP2` phase and added to your Z-routine object library, programs linked with it will tell you that one or both of the routines "requires local development" whenever any tape manipulation is attempted. If you need to develop versions for a new architecture, use either the `$APLSUN` or `$APLSOL` versions as a starting point.

Remember to have the network services implemented if you are attempting to use remote tape access.

### *2.6.11 "Dirty Dozen" Tasks (DDT) testing*

The other programs generated by `INSTEP3` will allow you to do quite a range of typical *AIPS* processing. After you're satisfied that the system is functioning properly, you can generate the remaining executable modules of the *AIPS* system via `INSTEP4` for source installations (see below). For details on the DDT suite of programs, see *AIPS* memos 73 (*AIPS DDT History*) and 85 (*DDT Revised and AIPSMark(93) Measurements*. You may want to request a DDT tape from the *AIPS* group in Charlottesville, or get them over the internet network via ftp or world-wide web.

To obtain the small, medium, or large datasets for the DDT via internet anonymous ftp, connect to `baboon.cv.nrao.edu` (192.33.115.103). The files are in directory `pub/aips/DDT/15JAN94` and have names beginning with `DDTS` for the small set, `DDTM` for medium and `DDTL` for large. Preserve the names if you want the `DDTEXEC` procedure to read the files from a FITS disk area. There should be 11 files for each set. Please be considerate about getting the large data set; we will appreciate it if you restrict such access to off hours as the files are at least 4 megabytes apiece.

The world-wide web location of the *AIPS* home page is given on the front page of this document. The DDT page is referenced therein, and can be directly addressed via:

```
http://info.cv.nrao.edu/aips/ddt.html
```

The master DDT images were recomputed (for the first time in years) for the `15JAN94` release, and the procedure itself was also overhauled. `ASCAL` was omitted and replaced with `CALIB`, and likewise `UVLOD` and IMLOD were replaced with the generic FITLD. The ability to read the DDT files from FITS disk instead of tape was also added. See *AIPS* memo 85 for the full details. Also note that you cannot use the older DDT images with the new procedures, and vice versa.

Memos are also available on the baboon.cv.nrao.edu anonymous ftp server in the directory pub/aips/TEXT/PUBL, and via WWW through the *AIPS* home page. This is not the master area for memos, but we attempt to keep it reasonably up to date with copies of relevant documents. It is a mirror for the $AIPSPUBL area you will find on your *AIPS* installation, with the difference that the ftp/WWW area is constantly being updated. Most TeX memos will have a PostScript version as well. For the DDT, the relevant memos are numbers 85 and 73.

### *2.6.12 Running under a Debugger*

If DEBUG is one of the command line options used to start up the AIPS task, the relevant routine (ZACTV9) in the ZSTRTA program will start up the AIPS.EXE binary under control of the specified debugger (dbx, adb, dbxtool, debugger, csd, xde, or some other); you get prompted for its name.

The *AIPS* group have found that the environment variable LM_LICENSE_FILE may have to be explicitly defined as pointing to the correct license file and available in the environment before running AIPS in debug mode with the newer Sun debuggers. If your site runs a license manager on Suns, you may or may not have to define this variable as we do. The symptom of needing it defined is that the debugger starts up but refuses to do anything as it can't get a license to run.

You will also get asked if you want to run AIPS itself under the control of the specified debugger. If not, then only the other tasks that AIPS sheds will start up in debug mode†. Running a process like AIPS under the control of a debugger as well as its tasks (subprocesses) can be very confusing, if not simply undesirable; hence this choice.

When a task, or AIPS itself, is run in debug mode, the program name — including the *POPS* digit — will be embedded in the debugger prompt on most systems. In addition, for tasks, the AIPS parent process will be put into a wait state (regardless of the value assigned to the *AIPS* adverb DOWAIT).

### *2.6.13 The VERSION Adverb*

In porting to a new system, or even when trying to debug certain parts of *AIPS* or new tasks, it is often useful to have private versions of certain tasks. The adverb VERSION in *AIPS* is used to point to task executables in different directories. The intention is that, in addition to the normal selection (OLD, NEW, and TST), the user should be able to point to a private version of a task via the use of an UPPERCASE environment variable that specifies the full name of the directory containing the desired binary (executable).

A version of the task help file (*e.g.*, UVLOD.HLP) must reside in the same directory as the private executable module. As all string adverbs in *AIPS* are automatically converted to uppercase, your environment variable name should also be entirely uppercase. Of course, the area it points to need not be uppercased. The names of the files must also be entirely upper case. When you start up *AIPS*, you can then specify this variable as the value of VERSION. For example, if we want to run a private version of the task FOO, and FOO.EXE and FOO.HLP reside in the area /u2/pmurphy/aips/test/, then:

```
% setenv MYSTUFF /u2/pmurphy/aips/test
```

or

```
$ MYSTUFF=/u2/pmurphy/aips/test; export MYSTUFF
```

Then in *AIPS*, after setting VERSION='MYSTUFF', the system will look at the private version of FOO whenever the inputs, help, explain, or go verbs are called for task FOO.

† only if SETDEBUG(n) is issued in *AIPS* with $n > 10$

## 2.7 Installation Step 4

The final stage in the installation process is the one that builds all tasks in *AIPS*. It will remake those already made in `INSTEP3` but this causes very little overhead. For the most part it is completely automatic, and if you have tested *AIPS* out with the procedures outlined in the previous section, it should go smoothly.

### *2.7.1 Running INSTEP4*

The `INSTEP4` procedure can take as long or longer than `INSTEP2`. Once again, if you have a binary tape, *you do not need to run* `INSTEP4`*; the task binaries were copied from the tape*. For source installations, to initiate `INSTEP4`, simply type:

```
% INSTEP4
```

It will cycle through all the major program source code areas in the following order:

| | |
|---|---|
| AIPPGM | *AIPS* Program itself and related beasts |
| AIPGUNIX | Unix-related *AIPS* tasks |
| APLPGM | Main area for application tasks |
| APGNOT | Non standard application tasks |
| APGOOP | Tasks that use object-oriented fortran routines |
| AIPNOT | Non standard *AIPS* tasks |
| QPGM | AP-specific tasks |
| QPGNOT | Non standard AP specific tasks |
| QYPGM | AP- and TV-specific tasks |
| QYPGNOT | as above but not standard |
| QPGOOP | as APGOOP but AP-specific |
| YPGM | TV-specific tasks |
| YPGNOT | as above but not standard |

and build "at" files for all the programs found, then use the "at" files to drive the execution of `COMLNK`, as in the case of `INSTEP3` but on a much larger scale (approximately 300 programs in all). Like `INSTEP2`, there may be failures especially on a new port. Also, some may fail to link if some of the `INSTEP2` subroutines failed to compile (see the *Known Problems* chapter).

# 3. Z-Routines and Z-Programs

The following is a collection of notes regarding Z-routines for the Unix version of *AIPS*. Some of the information is no doubt quite dated.

## 3.1 Introduction

The original implementation of *AIPS* under Unix occurred as a result of the pioneering efforts of David Garrett in the Astronomy Department at the University of Texas, Austin. At NRAO, the Unix implementation has evolved under a number of systems beginning with Amdahl's UTS implementation of Version 7 UNIX. A Masscomp MC 500 at NRAO Greenbank running Bell's System III UNIX had also served as a testbed from time to time although never as a production machine.

All that is ancient history. We now have eight IBM RS/6000 model 560 or 580 workstations running AIX 3.2.5 as our main *AIPS* workhorses. These have completely replaced the functionality of our older Convex C1's (we no longer have any Convex systems in-house). The main Scientist and Programmer systems are mostly SparcStation IPC and IPX workstations at SunOS 4.1.2 (with some at 5.3). In addition, we have one Sparc–10 (SunOS 5.3, with 4 "heads"), a few additional smaller IBM RS/6000s (AIX 3.2), one DecStation (Ultrix 4.3/DEC Fortran), a Dec Alpha 3000 model 300 workstation (OSF/1), and our newest member of the *AIPS* family, a Gateway 2000 486DX2/66V PC clone (Linux 1.1.47, OS updated almost daily). VAX-en have totally disappeared from our computer rooms insofar as *AIPS* use is concerned. The ports of *AIPS* to all the Unix machines mentioned above are relatively mature. In addition, through the courtesy of the people at NASA's Jet Propulsion Laboratory in Pasadena, California, and the Goddard Space Flight Center in Greenbelt Maryland, we made a full port to the HP 9000 series 700 systems running HP–UX 8 and 9, and also to a Silicon Graphics Indigo 2 system based on the 150MHz MIPS R4400 chip, running Irix version 5. A port to Apple's A/UX is in progress at the time of writing, again, through the courtesy of outside users and a guest account accessible to NRAO staff via internet.

In addition, *AIPS* has been ported in the (somewhat distant) past to the Alliant FX architecture and to Cray's UNICOS. Non-NRAO sites have also ported (or attempted to port) *AIPS* to a variety of UNIX environments, though not all these attempts worked, and we have never heard from some of you who initially told us you were starting a port (are you still there???) The result has been the evolution of a rather robust, portable implementation. However, NRAO must continue to rely on sites that are running flavors of UNIX or that have hardware different from ours to contribute the special code that these systems may require. This has already produced a subset of routines that is quite generic for Unix/AIPS, but we need to know about variations, improvements and most important, cases where what we think is generic is not in fact the case at all. So *please* let us know!

## 3.2 Standards

The *AIPS* group continues to make the Unix Z-routines as generic as possible. What we are up against is the vendor extensions that, while useful in the short term, are non-portable and/or non-standard*. Wherever routines can be written in ANSI standard Fortran 77, they should be, so that they can be used by any system that has a compliant compiler. The real authority for this is the American National Standard document *Programming Language FORTRAN* (ANSI publication X3.9–1978).

The situation for Z-routines that must be written in C was less clear but things have improved. The ANSI standard for C has been available for some time now, and most systems have or offer such a compiler. There are still some compilers that only support the older (so-called "K&R" C, after Kernighan and Ritchie, the authors of *The C Programming Language*, considered by many to be the original standard work).

During the recent port to SunOS 5 (Solaris 2), most of the existing Z routines were overhauled and converted so that they will work either with ANSI C or K&R C. On Suns with the ANSI C Compiler (called `acc` in SunOS 4), you should use this rather than the older compiler. The differences amount to function prototypes and possibly some different include files.

* For example, many systems will allow you to `EQUIVALENCE` data items of type `CHARACTER` with data items of other types; this is an ANSI (f77) standard violation.

## 3.3 Defining your System: the ZDCHI2 Routine

The system parameters defined in the routine `ZDCHI2` are critical to the successful exchange of data between systems, primarily via tape or FITS disk. These include the values assigned to the `/DCHCOM/` common variables `BYTFLP`, `SPFRMT` and `DPFRMT`. The values assigned to several other, less critical system parameters in this routine also warrant attention. These include the values assigned to `NSHORT`, `TTYCAR`, `SYSTYP` and `SYSVER`.

### *3.3.1 BYTFLP — Byte order: big or little Endian?*

The value assigned to `BYTFLP` is used to indicate whether the host uses byte or word flip, or both, or neither (most common) for its internal representation of data. Byte/word flipped data is where the order of the bytes in data items is *flipped* ("little endian" ordering) compared to the more common order ("big endian"; at least it's more common in the *AIPS* world). *AIPS* has been designed such that it converts its input from the outside world (*i.e.*, tape and FITS disk data) into the byte order of the host, and on output converts it back to big-endian order. So on most machines, the conversion routines are no-ops. Here a byte is considered to be 8 bits, and a word is 16 bits.

For the "big-endian" case, `BYTFLP` should be left as `0`. This covers Suns, IBM RS/6000's, HP 9000 series 700, and Convex. For "little-endian" systems such as VAX-en, DecStations (MIPS/RISC), Dec Alpha (OSF/1), SGI (?) and Intel (*e.g.*, 80486) based processors, the value for `BYTFLP` should be set to `3` to indicate both byte and word flipping. Not all MIPS-based processor use little endian ordering; like the Motorola 88000 and AMD 29000 RISC processors, MIPS-based processors actually support both ordering schemes, select-able only at power-up. However, the highest profile MIPS-based processor is easily the DecStations which uses little-endian ordering. The systems you get direct from MIPS itself however do not.

The following Z-routines are sensitive to byte/word flip:

| | | | | |
|---|---|---|---|---|
| `ZBYTFL` | `ZBYTF2` | `ZI16IL` | `ZI32IL` | `ZILI16` |
| `ZILI32` | `ZIPACK` | `ZR32RL` | `ZR64RL` | `ZRLR32` |
| `ZRLR64` | | | | |

### *3.3.2 Floating Point Formats: SPFRMT and DPFRMT*

`SPFRMT` and `DPFRMT` are floating-point format indicator codes used to indicate the host single and double precision floating-point formats. *AIPS* uses these to determine how to convert the host floating-point data format to IEEE format before writing it to tape or FITS disk and, conversely, how to convert IEEE floating-point data read from tape to the host format.

Most processors brought to market in recent years use IEEE float-point format (Sun, IBM RS6000, HP/Apollo, Convex, Dec (non-VAX), Intel ...). Versions of the IEEE/host floating-point format conversion Z-routines exist already for hosts that use IEEE (trivial, of course) and VAX F-floating-point. The routines that perform such conversions (`ZR32RL`, `ZRLR32`, `ZR64RL` and `ZRLR64`) are designed to stop dead in their tracks if they see an unsupported floating-point indicator code.

### *3.3.3 Other Miscellaneous Parameters*

There are four parameters that may need to be set or changed for a new system. These are:

- `NSHORT` should be the number of iterations that characterizes a loop as "short". Some loop terminating values cannot be known at compile time. `NSHORT` provides a means by which we might be able to code alternate, scalar code to which execution would branch upon detecting that the loop is "short". This is of course for `$QPSAP` routines mainly.
- `TTYCAR` (0 or 1) is simply used to toggle whether or not carriage control is required for terminal I/O. If you aren't sure, try a few simple experiments to see if formatted terminal I/O requires a leading blank (see `$APLGEN/ZTTYIO.FOR`).
- `SYSTYP` and `SYSVER` are `CHARACTER` variables used to indicate the operating system (*e.g.*, `'UNIX'` versus `'VMS'`) and `SYSVER` is used to indicate the version (*e.g.*, `'SunOS 4.1.2'`, `'5.4'`). For

DEC/Ultrix systems, do not alter the SYSTYP variable as it is checked in other Z routines when opening disk files for direct access (*e.g.*, ZTPOPD); the record length convention on VMS and Ultrix systems is that the units of record length are longwords (4 bytes each), whereas on most other systems the units are bytes.

## 3.4 Routines that may Require Local Development

**For new ports**, some Z routines may require local development. These are in the $APLGEN and/or $APLUNIX areas and may be "stubbed". The stubbed versions will print an error message and either return with an error code or stop the program. In many cases, working versions of routines exist that are suited to a particular vendor's environment (*e.g.*, $APLIBM, $APLSOL and $APLSUN). These are environments with which we have either had first hand experience or for which non-NRAO sites have submitted code.

The routines that potentially require local development include:

- tape manipulation (ZTAP2 and ZMOUN2); look at Sun and IBM versions for examples;
- line printer output spooling (ZLPCL2.C
- IEEE to local floating-point conversion (ZR32RL, ZR64RL, ZRLR32 and ZRLR64)
- VLA archive tape data handling (ZDM2DL, ZMCACL, ZRDMF and ZRM2RL)
- Other data conversion routines (*e.g.*, ZI16IL, ZILI16, *etc.*) if you are installing *AIPS* on a 64-bit machine
- Getting accumulated CPU time (ZCPU; make sure the units are right)
- Getting free space (ZFRE2; the $APLUNIX version works for most systems)
- TV oriented Z-routines (*e.g.*, ZM70*) if your system has such a device.

In the early stages of a new port, the most important Z-routines to work on are ZDCHI2, ZTAP2 and ZMOUN2. In ZDCHI2, you want to make sure that the values for BYTFLP, SPFRMT, DPFRMT and TTYCAR are correct for the target system. ZTAP2 and ZMOUN2 may take a bit of work to determine how to detect such things as file marks and beginning or end of tape. The other Z routines are not so critical and can be worked on as the need arises.

A list of most (maybe even all) of the Z routines relevant for Unix is presented in Appendix 1 of this document. This is unfortunately quite out of date; refer to the source code itself if in doubt. A brief explanation of its function is also given for each routine. If you need more than this, consult the two-volume *Going AIPS* document set. The list in the appendix here may not include updated information on the latest versions (*caveat emptor*!).

## 3.5 Unix Z programs

The only Z programs developed for Unix/*AIPS* at NRAO are ZSTRTA (interactive *AIPS* startup) and ZSTRTB (batch *AIPS* startup). Generic versions of these two can be found in $AIPGUNIX. ZSTRTA is executed as part of the *AIPS* startup procedure $SYSUNIX/AIPSEXEC. The purpose of ZSTRTA is to determine the lowest available/appropriate *POPS* number. This is determined from the existence of processes with names of the form AIPSn where n ranges from 1 to the maximum number of interactive *AIPS* allowed as specified via SETPAR in the system parameter file ($DA00/SP*). This value is also limited by the number of memory files generated via FILAIP (or FILINI) and initialized by POPSGN. Like ZSTRTA, ZSTRTB is executed as part of the BATER startup script $SYSUNIX/BATER. BATER is an *AIPS*-like program for preparing and submitting *AIPS* batch jobs.

There may be problems with the process determination part of these routines, particularly when different accounts are used to run *AIPS*. There also may be file locking problems associated with this, especially on NFS-mounted file systems. In general, an effort has been made to steer clear of the older flock() system call and use the locking mechanisms available in fcntl() instead. The problem with using flock() is that it seems to work fine if the locked file is on a NFS file system, *i.e.*, it reports success, but the file is not in fact locked. Conversely, fcntl() does support locking on remote NFS file systems, but it has problems, notably at very large installations such as NRAO's AOC network.

*AIPS* makes extensive use of file locking, and as one of the file-systems was used by *AIPS* users on over 40 workstations rather heavily, the limit in the lock dæmon of 256 locks did on occasion get exceeded. Other sites are not as likely to run into this problem, however, as the AOC is probably the largest *AIPS* installation at this time.

# 4. Release Notes

## 4.1 Introduction

The purpose of this chapter is to give installers some notion of the system dependent changes in *AIPS* that may affect local development. The same change information can usually be extracted from the *AIPS*letter, but its impact may not be obvious. If this is your first installation of AIPS, these notes are probably of little interest, but they may give you some idea what kind of future changes might trip you up when you update your system.

The sections for the `15JUL89` and earlier versions may be found in the `15APR91` version of this document.

## 4.2 Notes for the 15JUL94 Release

### *4.2.1 Binary "Load and Go" Tapes*

The major difference in the `15JUL94` release — as far as the installer is concerned — is the availability of a binary installation tape for several architectures. Using such a tape, there should be no need to compile or link any of the source code. However, the complete source code is included on all binary tapes for completeness, and also to allow additional architectures to be added to the installation (see the section on Multiple Architectures near the beginning of this document).

With so many possible variations in Operating System versions, compilers, *etc.*., it is not realistic for NRAO to even try to provide a separate binary tape for each combination. What has been possible is to perform the test installs prior to the release in as generic a way as possible, using few or none of the "add-on" tools or packages that exist locally, and create a binary tape based on the end product of the test installation. This has been performed, disk space and access to the relevant systems permitting, for the following systems:

- SparcStation IPX, SunOS 4.1.2
- IBM RS/6000 model 580, AIX 3.2.5
- Dec Alpha AXP 3000/300, OSF/1 1.2
- SparcClassic LX (SunOS 5.3)
- Gateway 2000 486DX2/66V (Linux 1.1.23)

In many cases, if your hardware/OS combination does not *quite* match the above configurations, the binary installation should still work. In particular, it is highly likely that SunOS 4.1.3, OSF/1 1.3 and 2.x, and later versions of Linux will accept the binaries on the distribution tapes and run them with no problems. However, this cannot be guaranteed.

There is no special procedure to accommodate the binary installations. It is still necessary to run `INSTEP1` which in turn has been modified to detect and correctly handle (we hope) both binary and source installations. Of course, the `INSTEP2` et al. procedures are unnecessary for the binary case.

#### *4.2.2 Changes to how AIPS handles Printers*

The definition of printers is now handled in a way that is consistent with that for tape drives, disks, and hosts. A brand new generic `PRDEVS.SH` script has been written which gets its information from `PRDEVS.LIST` in the `$NET0` area. The external actions of the script appear identical to the older `PRDEVS.SH` files that had to be hand-edited, so it is possible to use the newer version of the script with older *AIPS* versions.

The format of the `PRDEVS.LIST` file is very similar to that of `TPDEVS.LIST`, *i.e.*, a printer name as the first column, ancillary information in following columns, and a description at the end. If you have an existing `PRDEVS.SH` file with printer definitions, the `INSTEP1` script will attempt to use it to automatically generate the `PRDEVS.LIST` file. Failing this, it checks to see if `/etc/printcap` exists and if so, gets a list of printer names from it. If there is no printcap file, it asks how many printers you want and what their names are.

In addition to this change, two new programs were added: `F2PS` and `F2TEXT`. The former takes the place of the third party filters such as `lwf` and `nenscript` that were required in older versions if you wanted to convert plain text to PostScript. You no longer need these third party filters for *AIPS* printing. The latter essentially does the same job as the `fpr` or `asa` programs (Fortran carriage control converters). These are built automatically during `INSTEP1`.

#### *4.2.3 File Delete-on-Print delay Change*

Whereas in the last release, the parameter controlling the time delay between submitting a file to the print queue and deleting it was a `# define` in the source code, it is now a configurable parameter that you can set via `SETPAR`.

#### *4.2.4 Linux "hack" Removed*

The kludge alluded to in the `15JAN94` release notes below is no longer needed. The `MAX` function in the `PARAMETER` statements was removed from the two tasks and they now compile and link without additional help under Linux.

### 4.3 Notes for the 15JAN94 Release

Refer to the *AIPS*Letter for `15JAN94` for lots of interesting details about this release. Some of the highlights are pointed out here.

#### *4.3.1 File Deletion before the darn thing is printed*

One rather nasty problem with previous releases of *AIPS* was the "feature" that the *AIPS* tasks themselves wanted to delete a file after it was submitted to the print queue. If the Berkeley `-s` option to `lpr` was used in the `ZLPCL2` shell script, a symbolic link is made by the spooler to the file and the file itself is not copied to the spool area. This also happens by default on some systems, often triggered by file sizes in excess of 1 megabyte. However, if there are other entries in the queue and the file is not processed right away, then the *AIPS* task will delete the file before the spooler can open it.

The ideal way around this would be to pass on an option to the `lpr` or `lp` command to delete the file after printing. Unfortunately this feature is not in general available on System V spoolers. So the compromise arrived at in `ZLPCL2.C` and `ZLASC2.C` was to remove the `unlink()` call and replace it with a set of commands that in effect delay the deletion by a fixed amount. The default delay is 300 seconds and is set as a `#define` in the source code. This feature works nicely on Suns but seems to have some problems on AIX; the subject is under investigation at the time of writing. Watch in the patch area of the *AIPS* anonymous ftp server (see above) or the web *AIPS* home page for updates on this story.

### 4.3.2 AIPS on a PC with Linux

For many years, people within and without the *AIPS* group talked and sometimes joked about the idea of running *AIPS* on a personal computer. In recent years with the development of full 32-bit Intel 80386, 486, and pentium based systems, the idea became more than just a fantasy.

Towards the end of 1993, Jeff Uphoff who was then a grad student at Virginia Tech, contacted NRAO through his supervisors with the idea of porting *AIPS* to a 386-based PC that ran Linux. This is a complete operating system, available in the public domain, that is very Unix-like and completely replaces the restrictive *DOS* or *MS-Windows* environment. See the Vendor-specific notes for more details on Linux and what bits and pieces one needs to put together for an *AIPS* machine.

Within a few weeks, Jeff was running the DDT tests on his home PC and getting acceptable numbers — at least in terms of accuracy! Shortly afterwards, Jeff visited NRAO with a tape, and installed Linux and his port of *AIPS* on a 486DX2/66V system within an afternoon. The DDT results on this indicated performance at about the level of an IPC. Since then, Jeff's work has been incorporated into the main *AIPS* distribution and `tarsier`, an identical machine to the above, is now served by a midnight job just as our IBM, Dec Alpha, and other in-house systems.

The only compiler needed for this port is Gnu C. The public domain `f2c` (Fortran-to-C) translator is used on all the Fortran source in *AIPS*. Its use caused us to discover a veritable rat's nest of coding errors in certain calling sequences that other compilers had completely missed. There are two programs that use `PARAMETER` statements with a function embedded (`MAX`) and a kludge has to be applied to get the converter and compiler to work correctly with this (see the `INSTEP1` description earlier in this document).

Additional details can be seen in the "AIPS on a Personal Computer!" article in the `15JAN94` *AIPS*letter (available online via ftp or web).

### 4.3.3 AIPS on Dec/Alpha

The port to the Digital Equipment Corp. "Alpha" AXP architecture running the OSF/1 operating system is also complete. The fact that the compiler technology used here and that used for Ultrix is very similar made this quite an easy port (by *AIPS* standards). The `$APLDEC` area is now shared by both OSF/1 and Ultrix, with an additional Ultrix-specific `$APLULTRX` area for a few routines. To really confuse you, the `$SYSDEC` area still refers to Ultrix, however, and `$SYSALPHA` is the Dec/Alpha system area.

### 4.3.4 Improved HELP Facilities

Two new verbs, `APROPOS` and `ABOUT`, make their debut in `15JAN94`. The former works pretty much like `man -k` in Unix, except its database is derived from the *AIPS* help files. In other words, `APROPOS CLEAN` will show a 1-line summary for any help file that contain words beginning with "CLEAN" in either the 1-line summary or the keyword section of the help file. It does this via a file `LSAPROPO.HLP` which is analogous to the "whatis" database for the Unix manual pages. The `LSAPROPO.HLP` file is in turn generated by the `HLPB.EXE` program. If you modify a help file other than this one, you can update `LSAPROPO.HLP` as follows:

```
% cd /tmp; cp $HLPFIL/LSAPROPO.HLP .
% $LOAD/HLPB.EXE FOOBAR.HLP
```

This will create a LSAPROPO.OUT in the same area. If it differs from `LSAPROPO.HLP`, you should rename the `.OUT` file to be `LSAPROPO.HLP` and put it back in the help file area.

On the master *AIPS* machine in Charlottesville, the above is performed anytime someone checks in a help file; this keeps the "apropos" database current at all times.

The `ABOUT` pseudo-verb is similar, but it takes a restricted "category" as its argument. Each help file in *AIPS* has one primary and one or more secondary keywords. The allowed values for these keywords can be found simply by saying `HELP ABOUT` in *AIPS*, and they are also listed in `PRIMARY.HLP` and `SECONDRY.HLP`, respectively.

There is a new shell script `SHOPH` in `$SYSUNIX` that automates the generation of the `ZZ*.HLP` files used by `ABOUT`. It has not been extensively tested but has worked for us on the few occasions we used it. You should not have to worry about this, as the `ZZ*.HLP` files are distributed with *AIPS*and do not have to be regenerated.

## 4.4 Notes for the 15JUL93 release

Two new ports make their debut with this release, one for "Solaris 2" on Sparc systems, the other for HP–UX on Hewlett-Packard 9000 series 700 systems. As both fall into the `BELL` side of the tree, the support within *AIPS* for the System V flavor of Unix has improved significantly. In addition, several corrections and contributed modules were received for the Cray (UniCos) and Silicon Graphics (Irix) systems. On the Berkeley side of the universe, support was improved for the DEC/Ultrix port, with the upgrade of NRAO's DecStation to Ultrix 4.3 and DEC Fortran. A port is in progress at the time of writing to DEC's Alpha architecture running the OSF/1 (Unix-like) operating system.

The behaviour of *AIPS* and specifically `FITTP` has been changed. Prior to this release, two end of file (EOF) marks were always written at the end of data. This is correct for 9 track tapes but not for Exabyte or DAT drives. All tapes written with *AIPS* prior to March 30, 1993 in `15JUL93` or written with earlier releases will need special treatment with this released version. See the 15JUL94 *AIPS*letter for details on how to work around or fix the problem.

Several speedup improvements and bug fixes found their way into the `XAS` TV server. Notable among the bug fixes was the elimination of a large memory leak when the MIT Shared Memory extension was used. Again, see the *AIPS*letter for the details.

The Introduction of the Message Server allows task messages to be shown in a separate window on a Workstation screen. It can accept messages from many different *AIPS* sessions as long as they all use the same *AIPS* TV number (*i.e.*, like the `TEKSRV` graphics server, `MSGSRV` is tied to the TV server).

The tape routines were overhauled (again), and now when the users `mount` a tape, *AIPS* will report back what it thinks was really mounted. This will in most cases require that the tape be physically in the drive before the `mount` command is issued. The tape density selection now really sets the density on most systems, including Exabyte 8500's on Suns and IBMs.

There is a good section in the *AIPS*letter on the Object Oriented Fortran code. If you are considering writing a new *AIPS* task, it will probably save you a lot of grief if you read this section and the *AIPS* memo (#78) that describe the interface. Most of the new tasks in this release are written using this OOP interface.

The *AIPS* preprocessor now allows nesting of include files up to three levels deep. It announces this each time it is run.

The ability to have two sets of libraries — debug and non-debug/optimized — on Sun systems (SunOS 4 and 5) is introduced in this release. Also, the ability to use shared libraries was extended to HP systems and SunOS 5.

Most, if not all, of the *AIPS* C code is now capable of being compiled by an ANSI C compiler. In addition, as much of the code as possible was made to comply with the POSIX–1 standard. Most Berkeley flavored code has been eliminated in the generic area, although the use of the socket library is an exception; the Z routines that operate on these now resides in `$APLUNIX` as we have found them to be available on even the least Berkeley-like Unix systems.

Many of the shell scripts that used the `ps` command to check for processes had to be changed, as its syntax is different for System V Unix. Also, the printing scripts had to accommodate both the `lpr/lpq` Berkeley printing commands and the `lp/lpstat` System V commands.

## 4.5 Notes for the 15OCT92 release.

Most of this is summarized from the `15OCT92` AIPSLETTER; refer to it for details.

### *4.5.1 The XAS Screen Server*

The TV server `XAS` was overhauled and improved considerably for this release. It is now considerably faster, and will try to use the shared memory extension if the X server running supports it. This, and the ability to store up a given number of commands before updating the screen, can be controlled via the `.Xdefaults` file. Finally, unix sockets (`DISPLAY` set to `:0` instead of `<hostname>:0`) are used for `XAS` and `TEKSERVER` windows where possible. See *AIPS* memo 81 for details.

### *4.5.2 Tape Access*

Various buffer sizes were increased and considerable extra work went into making remote tape access faster. The gory details are to be found in *AIPS* memo 80. Now, accessing a remote tape should be just as fast as a local tape.

### *4.5.3 Fortran Compilation*

The manner in which the `FC` and associated scripts (`COMLNK`, `COMRPL`) determine the optimization and debug level for a given compilation has been radically changed. Now, the default compilation options for all architectures are stored in `$SYSUNIX/FDEFAULT.SH` and the optimization/debug level determination is performed by `FCLEVEL.SH` using the text file `OPTIMIZE.LIS`. This last file is essentially a database of what *AIPS* modules need what level of optimization and/or debugging. It allows you to select a single routine, an area, or all files; also you can select any of these on a single architecture. Look at the file (in `$SYSUNIX`, or `$SYSLOCAL` on Suns) for details; it's largely self-explanatory. The details of what the various *AIPS* optimize levels `OPT0` through `OPT9` are is set in the `FDEFAULT.SH` file.

### *4.5.4 ZABOR2 and ZACTV9*

These C language routines control exception handling and task activation, respectively. The latter now uses POSIX standards and creates tasks as grandchildren; this cuts down on the occurrence of "zombie" processes. The former now allows tasks being run in the presence of a debugger to trap exceptions normally, unless they are actually being debugged.

## 4.6 Notes for the 15APR92 Release

Many of the changes described in *AIPS* memo 74 (*AIPS at the ATNF*) have been implemented in the general release of the system (mostly thanks to Mark Calabretta). The shell scripts that set up the *AIPS* environment and start the *AIPS* program have been overhauled so that networks are more of an integral part of the system instead of an afterthought. The way TV servers are handled has been radically changed, a tektronix window server has been added, and (most important to the overworked and underpaid *AIPS* installer) `INSTEP1` has been introduced. On top of this, the command-line options for starting *AIPS* have changed so that disk selection (on a per-host basis, *i.e.* all disks on a given hosts or none), printer selection, and tv selection (to override the default) is possible. A Unix "man" page for `aips` is included.

The remote tape mechanism has been completely redone. There is a new tape daemon `TPMON` that handles requests from remote hosts for local tape drives — and uses *AIPS* Z routines. These Z routines have also been overhauled and work well with all the tape drives we could get our hands on (Sun/Exabyte, Convex/9trk, IBM/Exabyte/9trk, DecStation/TK50/Exabyte?).

XAS is now the preferred (and best-supported) TV server. It has all the features of the others (SSS and XVSS) but only relies on the basic Xlib routines (it doesn't need any extra toolkits or libraries).

There's more, but you will have to read the `15APR92` *AIPS* newsletter for the gory details.

## 4.7 Notes for the 15APR91 Release

No major system-level changes to report. `ZFRE2.C` in the `$APLSUN` area has been written so that it should run on Convex and IBM RS/6000 systems unmodified; a test version ran with no errors on these two systems as well as Sun3 and Sun4. The only version tested within a full-blown *AIPS* environment, however, was for Suns. If you have a Convex or an IBM RS/6000 system, please try it out and let us know if the `$APLSUN` version works. Do NOT try it on a DecStation! There is a version in the works for that as the system call in question (`statfs()`) is significantly different on Dec/Mips systems.

### 4.8 Notes for the 15OCT89 Release

The `15OCT89` release is the first publicly available release of the overhauled system.

No more `INTEGER*2` (and hardly any hollerith data anymore). Lots of other changes too numerous to mention (see memo titled *Conversion of Old AIPS Software to Run Under* `15OCT89` *and Later Releases* which should accompany installation kits for the next few releases). There is *only* one way to convert data on disk generated from previous releases to the new data format required by the `15OCT89` release. This is by writing the old data to tape using `FITTP` under the old system, then read the data from tape into the new system using `IMLOD` or `UVLOD`. The overhauled code runs about as fast as the pre-overhauled code. The *AIPS* group has been reorganized.

## 5. Updating User Data Formats

If you have no previous installation of *AIPS* in production, you can totally ignore this chapter. However, if you have existing data on disk that was created with a version of *AIPS* older than `15OCT90`, the formats of some of the data files will be incompatible with the newly installed version of AIPS. Before that release, changes of *AIPS* data formats were done with a program called `UPDAT`. However, the changes involved in the conversion to the `15OCT89` and later releases are so great that no special purpose translation task was prepared. Older files must be written to and read from FITS files to be used in `15OCT90` and later versions.

## 6. Vendor Specific Notes

### 6.1 Introduction

No two Unix systems are identical. In fact, on the same Unix system, no two Operating System (OS) levels are identical when it comes to *AIPS*. NRAO supports some better than others, largely because we can't afford one of everything (even though we'd like to, especially the faster ones...). Many of the notes below are quite outdated and some of the problems will have been resolved as a result of the source code overhaul (`15OCT89`), the network retrofit (`15APR92`) and later releases. However, you may still be able to find solutions to problems you encounter from the notes below.

### 6.2 Sun Systems

The primary workstation at all of NRAO's sites is Sun's Sparc. At the time of writing, the mix includes IPX, IPC, Sparc2, Sparc10, and SparcClassic systems. There are one or two Sun–3's hanging on and used only as X terminals. The following notes are based on our own experiences as well as that of others, under a variety of OS versions, from SunOS 3.5 up through 4.1.3. See the separate section on Solaris below for versions SunOS 5.x.

1) There are some SunOS 4.x releases that have caused some problems, in particular 4.1 with version 1.2 of the Sun Fortran compiler (try to get Fortran version 1.3.1 at least). We never got a working *AIPS* system with this combination.
2) Most Sun Fortran versions up to 1.4 have a bug that causes random numbers to be stored in variables initialized with DATA statements, when the `TMPFS` option is used in the kernel (*i.e.*, when the `/tmp` partition is mounted on swap; a `df /tmp` will reveal this). *We have not observed this behaviour in SunOS 5.* If you have `/tmp` mounted on the swap partition for older OS versions, *AIPS* will not build correctly. You must either (a) Apply the Sun patches 100174–01 (and 100175?); (b) remake your kernel without the `TMPFS` option and mount `/tmp` on its own partition; (c) if you can, just mount `/tmp` elsewhere anyway; or (d) use the `-temp=` option of the fortran compiler to force it to use some directory other than `/tmp` for storage of temporary files during compilation.

3) There is a problem with `$APLUNIX/ZACTV9.C` for SunOS versions *earlier than 4.1*; it calls `waitpid()` which was introduced at 4.1. If you have an earlier version of the OS, use the generic Unix version (`$APLUNIX/ZACTV9.C`) *from an older release of AIPS, e.g.*`15OCT92` instead by moving it to your `$ZLOCAL` area.
4) If you have a really old OS version, 3.5 to be specific, `f77 -O` compilations can get into infinite loops. Compilations that apparently succeed can nevertheless yield execution errors (*e.g.*, infinite loops, bus errors). What little performance improvement you get from `-O` optimization (about 10–20%) is probably not worth either the installation or execution problems it can cause. Compiling `f77 -P` also fails on a handful of modules and is probably even less worthwhile to pursue. A very important exception to the above is the Q-routine library which should always be compiled at the highest optimization level possible. Modules written in C are also an exception and can be safely compiled using `cc -O`.
5) On Sun 3 systems, special care should be taken to properly define the default floating point compiler option for `f77` (Sparc systems in general don't need any special command-line option for this). This may be done in two ways:
   - by defining the environment variables `COMP` in the section of `$SYSUNIX/FDEFAULT.SH` relevant to your architecture, and `LINK` in your local version of `LDOPTS.SH`, to include the the desired floating point option, or:
   - by defining the environment variable `FLOAT_OPTION` as part of your login procedure.

   The documentation for `f77` describes the various floating point options. On Sun 3's you will want to use `-f68881` and *AIPS* has ported fairly well under this option in the past. The default option is `-fsoft` which has problems. This is not the preferred option anyway. Most Sun–3's are configured with at least the Motorola MC68881 floating-point co-processor. No serious problems have been reported with the `ffpa` option. This option is, of course, highly desirable for *AIPS* code, in particular for the Q-routines, but is only meaningful for systems configured with a floating point accelerator. Such sites have reported performance enhancements of 4-5x on compute intensive tasks such as `MX`. Our experience with the `-fswitch` option indicates that it is probably safe, but generates slow code (about 2x slower).

   It should be emphasized that NRAO no longer has any Sun-3 systems in its *AIPS* environment and thus *cannot support these systems* at the level it has in the past. Sites with such equipment are strongly urged to investigate upgrade/trade-in possibilities, if at all possible.
6) Under OS 4.0, program `LINK`s may fail with `_units undefined`. This does not seem to be a problem with later OS versions; if you can, try to upgrade to at least 4.1. The following will probably suffice as a work around. Create `$SYSLOCAL/SYSTEM.C` containing:

```
system_(s,n)
char *s;
int n;
{
    system(s);
    return;
}
```

Then `CC $SYSLOCAL/SYSTEM.C` and modify `$SYSLOCAL/LDOPTS.SH` so that the link options line looks like: `LINK="$DEBUG $SYSLOCAL/SYSTEM.o /usr/lib/libc.a"`.

7) For `SUN3`, there was a change in the fpa floating-point exception handling as of OS 4.0. In `$APLSUN/ZABORS.C`, the call to `sigfpe_handler` should be changed to a call to `fpa_handler` for systems with floating-point accelerators. *No modifications are necessary for Sparc systems.*
8) On some systems (perhaps only those with Xylogics 472 tape controllers), the routine `ZTPMI2` had been known to fail on partial record reads (*e.g.*, `TPHEAD`). The workaround was to always read 32K bytes into a temporary buffer (*i.e.*, despite the actual number of bytes requested), then copy the actual number of bytes requested to the output buffer. It also seems to require compilation using the `-g` option (*e.g.*, `COMRPL DEBUG ...`).

   We have no evidence or indication that this is still the case, and the Z routines for tapes have undergone so many changes, that the "hack" listed in previous versions of this document is no longer being included. However, it is not difficult to retrofit the technique above into the modern version of the routine.

9) If you will be rebuilding the `AIPS.EXE` program while people are using your *AIPS* installation, or any other commonly used program that may be active for long periods of time (days perhaps), be warned. When `COMLNK` replaces the binary file, the process running the old version will inevitably get a `bus error`. You can alleviate this to some extent by specifying `SAVE=TRUE` in your local `LDOPTS.SH` file; this moves the `AIPS.EXE` file to `AIPS.EXE.OLD` before inserting a brand new `AIPS.EXE` (similarly for any `COMLNK`'d task).

   Although Suns trap `SIGBUS` by default, apparently the error handler code never gets called because the disk file changed "underneath" it. Fortunately, Suns handle this fairly well and the task just aborts. You will make fewer enemies if you are aware of this and warn people before rebuilding anything, or use the SAVE option.

10) Also if you use shared libraries, you may see some "unresolved references" errors from the link phase of `INSTEP3` and `INSTEP4`. As of `15JAN94`, **this should not happen**. It is a fatal error under Solaris 2 (see below), and we have tried to eliminate all causes for this error. In previous *AIPS* versions, the advice given here was to ignore the warnings. Don't; instead, let us know the details.

11) One more thing about shared libraries: their use may force you to increase swap space on your systems. At NRAO, most of our Suns have anywhere from 75 to 105 Megabytes of swap. 80 is a good value to shoot for. In large installations, the savings in disk space from the smaller *AIPS* binaries in `$LOAD` is more than offset by the additional space one has to reserve for swap on many workstations.

12) If you have a very recent version of the Fortran compiler, you may want to use `-libmil` in the Fortran `FDEFAULT.SH` settings (and an appropriate `-cg` setting in `LDOPTS.SH`) for a slight speed advantage. Otherwise, the use of `-libmieee` may be advantageous. Check your `f77` manual page for details.

13) If you are using GNU C in place of the Sun C compiler, be sure to include the GNU C library in the link options in `LDOPTS.SH`. If you do not, tasks will fail to link with an unresolved reference to `___eprintf`.

## 6.3 Solaris (Sparc) Notes

1) `INSTEP1` will probably warn you that the tape devices you specify, *e.g.* `/dev/rmt/0ln` is not a character-special device. Ignore these warnings.

2) `PATH`!!! Make sure your path variable is set to include all the wonderful new places where compilers and other tools have hidden. You will want `/usr/ccs/bin` and `/opt/SUNWspro/bin` in there at a minimum in addition to the default.

3) There may be file locking problems in `/tmp` when using the `fcntl()` type locks that *AIPS* now uses. As part of the installation testing, a data disk (#1) was made in `/tmp` on our Solaris 2.1 machine, and this was in turn mounted on swap space. With this configuration, an unexpected lock on a tape lock file got stuck for no apparent reason. We do not recommend using `/tmp` on swap to hold the first *AIPS* data area. It *might* be OK for a scratch disk, but you may be safer avoiding it altogether. This test install was for `15JUL93`. For later releases, the data disk was on a regular disk and no problems were found.

4) The Solaris version of the `ZLPCL2` shell script did not get upgraded or removed when the generic version was modified to use the new `F2PS` and `F2TEXT` printers. You should rename this script to be something else (or `chmod -x` it to make it not executable), so that the generic version in `$SYSUNIX` is used instead. The older Solaris version uses something called `postprint` which we found on our machine in the `/usr/lib/lp/postscript/` directory.

5) Some Solaris sites have reported problems with the Solaris 2.3 automounter. If an *AIPS* data area has not been accessed for 5 minutes (or whatever the automounter timeout is), that area is unmounted and the next access by *AIPS* may get a "file not found" error. Repeating the *AIPS*

task is usually enough to get around the problem. Another possible solution is to have a shell process sitting with its current working directory in the relevant data area. Yet another is to increase the automounter timeout (*e.g.*, to 200,000 seconds).

6) See items 10, 12, and 13 above in the Sun4 section.

## 6.4 IBM RS/6000 Notes

There are three flavors of C compiler available with AIX (version 3 and above). These are `xlc`, `c89`, and `cc`. The first is the recommended compiler for AIPS, while the last is usually used for compatibility with IBM RT and PS/2 applications.

One of the big problems in mixing C and Fortran code on AIX systems is that, unlike SunOS, Convex-OS, Ultrix, VMS, *etc.*, the linker does *not* append an underscore to the C routine names when they are called from a fortran main program. Fortunately, the `-qextname` option takes care of this rather nicely. Older versions of *AIPS* used a rather tortuous and occasionally unreliable method of preprocessing to correct the problem.

In addition, there exists a Fortran *and* C version of some well-known system calls such as `getenv()` with conflicting parameters. The workaround for this so far has been to use the IBM-specific `LINK` shell script which explicitly includes the `libc` and `libbsd` runtime libraries. No action is needed; `INSTEP1` sets things up so this is the default `LINK` on all IBM RS/6000 systems.

In the early days of AIX *AIPS*, some users reported that they had to add `-D_POSIX_SOURCE` and `-D_ALL_SOURCE` to `CCOPTS.SH` to properly get the include files. We have not found this to be needed with AIX 3.2 on any of our many IBM systems. If you find this to be necessary, please let us know the details.

## 6.5 DEC/MIPS (DecStation 3100 *etc.*) Notes

1. The AIPS port to the DEC/Mips architecture (DecStations and DecSystems) was initially for Ultrix version 4.0 and the Mips-supplied Fortran compiler (version 2.1). Since the `15JUL93` release, Ultrix 4.3 and DEC Fortran are the tools of choice. After we took into account the various bug fixes kindly reported to us by several concerned *AIPS* Ultrix users outside NRAO, the installation test went remarkably well on this system (apart from pilot error).
2. When using the DEC version of the `dbx` debugger, be careful in believing what it says when printing Fortran `LOGICAL` variables. It led the author on a merry wild goose chase because it almost always prints "false" when the logical value is in fact true. It says "false" when it really is false too. Look at the value in hex if you really want to know what it is.
3. As some DEC-specific file routines like `ZTPOPD.FOR` rely on the value of `SYSVER` which is in turn set in `ZDCHI2`, do not mess with this variable. Also note that `SYSTYP` cannot be used as another Z routine sets it to `UNIX`.

## 6.6 Dec Alpha AXP, OSF/1

The similarities between the compilers on this system and those available under Ultrix were so great that there are few differences between the ports. The default Bourne shell is a lot better on OSF/1 but `INSTEP1` still needs to run as a Korn shell script; it will modify itself and ask to be restarted.

As this system is fundamentally a 64-bit architecture, the memory requirements for it seem to be considerably higher than for typical 32-bit systems. We have run DDT tests on an AXP 3000 model 300 system under OSF/1 1.2 with both 32 and 64 megabytes of memory. In the former case, the elapsed time was dominated by OS disk activity, whereas there was a notable improvement in the latter case. We suspect that 128 megabytes of memory would have been a lot more comfortable. For more on these tests, refer to AIPS memo 85.

Certain older versions of the compilers and operating system seem to cause floating point underflows to generate an exception, regardless of `-fpe` options used for the compiler. This has been seen with OSF/1 version 1.2 and DEC-Fortran 3.3 at NRAO. Reports from outside users indicate that an upgrade to OSF/1 2.0 and DEC-Fortran 3.4 solves this problem; as the problem existed in the system shared libraries, a re-compilation is not necessary (unless you used additional directives in `LDOPTS.SH` to force completely static binaries to be created).

## 6.7 HP 9000/700 Series

The initial port to these systems was performed on HP–UX 08.09 on two systems kindly made available to us via internet and guest accounts by the people at JPL. Later, the OS there was upgraded to 09.01 with apparently no ill effects. The installation testing for `15JUL93` was done on this newer system; at the time of writing, a test install has not been performed on `15JAN94` or `15JUL94`.

It may be that the shared library support does not work as well in the newer OS version than in the older one. At least one beta tester reported this. Caution is recommended, and if you run into problems with the shared libraries (or any other part of the HP port), let us know (and switch to non-shared for your sanity). Conversely, if you find shared libraries work well for you, *please* let us know the details!

## 6.8 Intel/Linux

Currently the only PC operating system *AIPS* runs on is Linux, which is a free (public domain) UNIX-like operating system available by anonymous ftp from **sunsite.unc.edu** (preferred) and **tsx-11.mit.edu**. See below for more details on versions of the bits and pieces you will need. In the future, alternate freely-available Unix-like operating systems may also be supported, but we can make no promises.

Initial benchmarks in November 1993 put *AIPS* on an Intel *486DX2/66V* at roughly the same speed as a Sun IPC (roughly 0.5 AIPSMARKS as defined by the old "large DDT"). A 486DX/33 runs the DDT (small) just over half as fast as a DX2/66, and a 386DX/33 (even with a 80387 installed) is so slow as to be nearly unusable in any serious applications (about 45 minutes for the Small DDT; compare this to 10 minutes on the 486DX2/66V). Tests with a 486SX/33 (The author's home system!) reveal that the DX processor (which has hardware floating point support) is almost a pre-requisite for running *AIPS*. It works with only the SX processor, but it is painfully slow mainly due to the floating point emulation.

The official AIPSMark$^{(93)}$ for the *486DX2/66V* system is 0.51 (see *AIPS* memo 85). The minimal and "comfortable" system requirements are as follows:

- A 386 or better (486, pentium) processor. A 486DX at 33 MHz is probably the lowest "usable" system. On 486SX or any 386 systems, make sure you have the math co-processor.
- 8 Megabytes of RAM (more if you plan on running X windows and/or AIPS heavily; our benchmarks on the 486 were with 16 and 32 megs). 16 Megabytes of swap space is also recommended; you can have multiple swap partitions/files and this will improve performance for largish problems.
- If you want to run X windows, a color monitor is recommended. Most SVGA card/monitor combinations are supported by the Linux X server. The NRAO system we run has an ATI Mach_32 chip-set accelerated SVGA card.
- For a full Linux/X11/*AIPS* installation, you will need about 100 Megabytes for the Linux/X11 part, and about 170–200 Megabytes for all of AIPS, assuming you strip the `*.EXE` files (they are stripped on the binary Linux tape and take up 103 megabytes). Add to that whatever you want for local user data. A 400 megabyte disk would give you 100+ Megabytes for AIPS data after you are done with everything else.
- All software needed is in the public domain or under a GNU Copy-Left (except *AIPS* which is of course freely available under the usual user agreement to academic/education/research sites).

The software pieces that you will need include:

- Any Linux kernel since about 0.99.12 (except 0.9.13 which had file locking problems) with `gcc` (GNU C.

- The `f2c` Fortran to C converter (versions from Oct 1993 or more recent). The `f77` shell script front-end to `f2c` was modified locally to perform some *AIPS*-specific functions, and the modified version is included in the *AIPS* distribution, making its way to `$SYSLOCAL` via the installation procedure `INSTEP1`.

All kernel sources, libraries, and compilers that you will need are in the public domain, covered by the GNU public license. They are available on the sites mentioned above. There are also several Linux "packages" available on these archives that can be used to build a complete, working system. These are not necessarily needed for *AIPS* but you may want them for other purposes.

Some of the details of the port follow.

1. The `/usr/lib/libbsd.a` library was used for the socket library, because of BSD-style `ioctl()` calls used by *AIPS*. This should be transparent to both the installer and the end user, as it is included in `LDOPTS.SH`. This also necessitated using the include file `<bsd/sgtty.h>` as the definitions of `TIOCGETP` and `TIOCSETP` are here. `sgtty.h` redefines `ioctl()` as `bsd_ioctl()`, which is only in `libbsd.a`. Again, everything should be taken care of and no code changes should be needed.
2. `INSTEP2` was observed to hang when compiling large libraries (those containing a large number of object files). This may have been due to an old version of `bash` and has not happened recently. It may have also been due to the passing of a greater number of command line parameters to COMRPL than the environment can handle. A simple `CONTROL-C` and re-execution of `INSTEP2` will restart it where it left off. Likewise for `INSTEP4`.
3. Local tape access has been investigated by using an Adaptec (?) SCSI card and borrowing the DEC Alpha's 4mm tape drive (vendor: Digital). We have managed to get some functionality with this system, but it is still not as robust as we would like. Currently it supports writing but reading does not work. The workaround for FITS tapes is to use `dd bs=28800` to dump files from tape to disk and thence into `AIPS` via `FITLD`. Most PC's have either no tape drive or have a QIC (quarter inch cartridge) or similar drive. The latter is incapable of backspace operations (other than rewind) so is of little interest to *AIPS*, other than for backup of FITS disk files. Remote tape access, and FITS disk file access both work.

## 6.9 Silicon Graphics

The port to Irix version 5 on a Silicon Graphics workstation was performed by Jeff Pedelty at NASA's Goddard Space Flight Center (GSFC) in Greenbelt, Maryland. It went quite smoothly, and was probably made easier by the improved System V ("Bell") support obtained from previous ports (Solaris, HP).

Once again, the weak point of this port is mag-tape access. Work is continuing (at the time of writing) on improving this, but without an actual machine for us to "beat on" in-house, progress on sorting out the magnetic tape access problems will be unavoidably slow.

## 6.10 Alliant Systems

The last port to the Alliant FX/series was in September 1988 using *AIPS* release `15APR88` with Fortran 3.1.33, libmath˙v4.0, striped disk, ACEs and System Configuration for FX80 supermkt (serial # 109 ):

```
Memory Size    = 96 Mb
CE-Cache Size = 512 Kb
User Memory    = 90.6 Mb
Paging File    = 134.1 Mb
Swapping File = 109.7 Mb
Number of IPs = 6
M012/M020/VME = 0/0/6
Number of CEs = 8
Detach/Attach = 0/8
```

Obviously this is quite dated, so installers of 15JUL94 should proceed with caution (We would welcome additional notes from *AIPS* managers with Alliant systems; please e-mail to `aipsmail@nrao.edu` if you can help).

1) For single CE configurations, Q-routines and those modules listed in the obsolete file `$SYSCVEX/-OPT2.LIS` (yes, `$SYSCVEX`) should probably be compiled with optimization level `-Ogv -DAS`. All other Fortran code should probably be compiled `-Og -DAS`.
2) For multi-CE configurations, Q-routines and those modules listed in `$SYSCVEX/OPT2.LIS` (yes, `$SYSCVEX`) should probably be compiled with optimization level `-O -DAS -alt`. All other Fortran code should probably be compiled `-Ogv -DAS`. Compiling all code `-O -DAS -alt` does not seem to improve performance significantly and results in execution errors in a few routines (*i.e.*, `$APLSUB/PASENC`, `ZR32RL`, `ZRLR32`, `ZR64RL ZRLR64` and probably others that were not exercised by our "Dirty Dozen Test").
3) In multi-CE configurations with detached singleton CEs, those programs NOT involving concurrent or vector/concurrent code (*e.g.*, no Q-routines or `$SYSCVEX/OPT2.LIS` modules) should probably be linked using the `-nc` option which prevents them from executing on the CE cluster. It may be easiest to make `OPTO="-nc"` with `LINK="-v $OPTO $DEBUG''` in `LDOPTS.SH` and relink the smaller number of programs you DO want to execute on the CE cluster via `LINK NOOPTO <module>.o`.
4) The `WHNALT.o` file found in `$QALN` was derived from proprietary Alliant source code that we cannot distribute. `WHNALT.o` should always be incorporated into your pseudo AP object library. `WHNALT` is only called by `$QALN/QCLNSU.FOR`.
5) The `WHNALT` routine has been known to fail in a data-dependent fashion (*e.g.*, the DDT large case `APCLN` problem) on Alliant systems configured with ACEs. `WHNALT` has worked for years without problems on CE configured Alliants. The modified version of `QCLNSU` below can be used as a workaround (*i.e.*, in place of `$QALN/QCLNSU.FOR`) until the problem can be resolved. The problem is that sometimes one of the values in the array of indices returned by `WHNALT` is zero. The workaround simply checks for this condition and if so, recalls `WHNALT`. The section of code around statement 110 should read:

```
      SUBROUTINE QCLNSU (COMP, LMAP, L1MAP, L2MAP, IBX, IBY, JNDEX,
     *   INDEX)

      ....
C                                         Compress x window
      L21MAP = L2MAP - L1MAP + 1
 110  CALL WHNALT (L21MAP, IDXRAY(L1MAP), N1, IBX, IB, NIB)
C                                        Kludge to get around Alliant
C                                        ACE "bug" where WHNALT may
C                                        return a zero value in the array
C                                        of indices (\ie, IB).  If so,
C                                        recall WHNALT.
      IRCALL = 0
      DO 120 LOOP = 1,NIB
         IF (IB(LOOP).EQ.0) IRCALL = 1
 120     CONTINUE
      IF (IRCALL .EQ. 1) GO TO 110
C                                         Subtraction loop
      ...
```

6) The call of `$QALN/SCFFT.o` by `$QALN/QCFFT.FOR` has been replaced by a call to `CFFT1D` from Alliant's `libmath.a`. Both `QCLNSU.FOR` and `QMULCL.FOR` in `$QALN` still call `ISAMAX`, but Alliant's `libmath.a` should now contain this BLAS level 1 LINPACK routine. The line `-lmath: ...`

should appear in the link lists (as defined in **$SYSLOCAL/LIBR.DAT**) for all programs linked with the Q-routine object library (see **$SYSALLN/LIBR.DAT**). The former **ISAMAX.o** and **SCFFT.o** as well as the versions of **QCFFT.FOR** that call **CFFT** and **SCFFT** have been saved in **$QALN** as **ISAMAX.O**, **fSCFFT.O**, **QCFFT.CFFT** and **QCFFT.SCFFT**, respectively, just in case your Alliant's **libmath.a** does not contain **ISAMAX** or **CFFT1D** yet. Let us know, otherwise we will eventually delete **ISAMAX.O**, **SCFFT.O**, **QCFFT.CFFT** and **QCFFT.SCFFT**.

## 6.11 Convex Systems

In the heyday of the Convex era at NRAO, we had 3 Convex C1s that served as *AIPS* work horses, one in Charlottesville configured with both an IIS model 70E and IVAS image display device; and two in Socorro, each configured with an IIS model 70F and one also with an IVAS. All three had several Gigabytes of striped disk.

However, all of these Convex systems have been de-commissioned and removed. Our last, **yucca**, was turned off for the last time in early 1994. Thus, our support for Convex systems in the future will not be at the level found in older releases.

The last release of *AIPS* to be run in production mode on a NRAO Convex was **15JUL93**. The configuration was Convex-OS 9.0 with IEEE floating-point selected via the **-fi** compiler switch. *AIPS* is still in production use elsewhere on other Convex models, including the C2 and C3 series. However, as the *AIPS* code and the Convex OS both continue to change, the possibility of encountering bugs/problems in the Convex version will no doubt increase.

1) The **$SYSCVEX** directory contains a number of files used in *AIPS* programming that are setup for Convexes. These files will be copied to your **$SYSLOCAL** directory by the **INSTEP1** procedure and override the versions found in **$SYSUNIX**. The **$SYSNRAO1** and **$SYSVLAC1** directories are the **$SYSLOCAL** directories for the now-retired NRAO-CV, and NRAO-AOC Convex C1s, respectively. These directories may also contain files of interest to other Convex sites. The files may also be dated and possibly obsolete, some having survived from the pre-**15APR92** epoch. You should try to rely as much as possible on the generic **$SYSCVEX** files where possible.
2) The **$APLGEN** versions of **ZR32RL**, **ZR64RL**, **ZRLR32** and **ZRLR64** should suffice for either "native" (quasi-VAX F) or IEEE floating point. Just make sure **SPFMRT** and **DPFRMT** are set properly in **ZDCHI2.FOR**. The Generic **$APLCVEX** version of this last file is set up for "native" floating point, whereas the **$APLNRAO1** and **$APLVLAC1** versions (identical) both use IEEE format.
3) The routine **QRECT.S** in the directory **$QVEX** will work for either "native" or IEEE floating-point.
4) You should use **/dev/rmtxx** for any tape definitions in the **TPDEVS.LIST** file if tape allocation is available on your system (*i.e.*, **tpmount** and **tpunmount**). These were referred to as **tpalloc/tpdealloc** in earlier Convex-OS releases. If not, the **$APLCVEX** version of **ZMOUN2.C** should be copied to the local Z routine directory and modified to eliminate the use of **tpmount/tpdismount** and the definition of the **AMTOn** environment variables via **ZCRLOG**.
5) The Convex Fortran compiler **fc** contained a bug in version 5.0 which involves a feature called *global register allocation*. This bug is also found in version 5.1, but was (we think) fixed in version 5.2 and subsequent versions. If you have the older version of the compiler (check with **vers /usr/convex/fc**), you should use the undocumented compiler flag **-ngr** when compiling code with the optimization flags **-O1** or **-O2** level, for example, Q-routines. Otherwise, incorrect results are possible.
6) Version 9.0 and later of Convex-OS broke the old version of **ZDELA2.C**. The symptoms were that tasks refuse to go away when they complete. The file **$APLCVEX/ZDELA2.C** will work fine for this version, but pre-9.0 systems should use the more general **$APLBERK** version (otherwise tasks won't even get started!).
7) **$APLBERK/ZCREA2.C** failed to compile on Convex-OS 9.0 because of problems in the included file **<sys/mount.h>**. The version in **$APLCVEX** has this include removed.
8) Some larger Convex systems have equally large disk partitions. If any of your partitions is more than about 2 Gigabytes, the algorithm used in **$APLBERK/ZFRE2.C** to calculate disk space will

fail because of the limitations of a 32-bit integer. A more robust calculation method is needed; the current one converts Unix blocks to bytes and thence to *AIPS* blocks, hence the potential for overflow. One user had a 10Gb partition and `FREE` reported -2 Gbytes free (yes, minus two)! Also, `$APLUNIX/ZCREA2.C` will need some attention as it also uses 32-bit integers to calculate the number of free *bytes* on the relevant file-system before creating a new file. On systems where the free space can exceed 2 Gigabytes, this will cause overflow and tasks will die.

9) The method used in `INSTEP1` to convert the shell scripts to the Korn shell is not perfect. It seemed to work on the test installation, despite stopping silently. A restart of the procedure appeared to work without other intervention. Also, the `LIBR` step seemed not to work in `INSTEP2` on some libraries, but it worked fine when done manually.

Finally, a reminder: Check the settings of the compiler options in the files `CCOPTS.SH`, `LDOPTS.SH`, `ASOPTS.SH`, and `FDEFAULT.SH`, adding `-fi` if you have the IEEE hardware, and removing it if not. Also, remember to edit the default Convex version of `LIBR.DAT`; it is set up for NRAO's old Convex at Charlottesville and unlikely to be correct for other sites.

# APPENDIX 1. Unix Z Routines

The following are brief (and possibly outdated) descriptions of the current Z-routines. They come from the headers of the files in **$APLGEN** and were generated in an automated manner. They are split by category as in the *Going AIPS* manuals.

The following "categories" are defined in the headers of the source code:

| | | | |
|---|---|---|---|
| BINARY | FITS | FORMAT | GRAPHICS |
| IO-BASIC | IO-REMOTE | IO-TV | IO-UTIL |
| MATH | MESSAGES | OOPS | PLOT |
| PLOT-UTIL | PRINTER | SERVICE | SYSTEM |
| TAPE | TERMINAL | TEXT | TV-IO |
| TV-UTIL | UTILITY | WINDOWS | Z |
| Z2 | | | |

Many of the descriptions below will appear in more than one category if the routine in question fits into two or more categories.

**Category: BINARY**

| | |
|---|---|
| `ZBYMOV.FOR` | move 8–bit bytes from in-buffer to out-buffer |
| `ZBYTFL.FOR` | interchange bytes in buffer if needed to go between local & standard |
| `ZC8CL.FOR` | convert packed ASCII buffer to local character string |
| `ZCLC8.FOR` | convert local character string to packed ASCII buffer |
| `ZDHPRL.FOR` | convert 64-bit HP floating buffer to local DOUBLE PRECISION values |
| `ZGETCH.FOR` | get a character from a REAL word |
| `ZI16IL.FOR` | convert FITS-standard 16-bit integers to local integers |
| `ZI32IL.FOR` | convert FITS-standard 32-bit integers from buffer into local integers |
| `ZI8IL.FOR` | convert 8-bit unsigned integers in buffer to local integers |
| `ZILI16.FOR` | convert local integers to 16-bit FITS integers in a buffer |
| `ZILI32.FOR` | convert local integer into FITS-standard 32-bit integers |
| `ZPUTCH.FOR` | inserts 8-bit "character" into a word |
| `ZR32RL.FOR` | convert 32-bit IEEE floating buffer to local REAL values |
| `ZR64RL.FOR` | convert 64-bit IEEE floating-point buffer to local "DOUBLE PRECISION" |
| `ZR8P4.FOR` | converts pseudo I*4 to double precision - for tape handling only |
| `ZRDMF.FOR` | convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers |
| `ZRHPRL.FOR` | convert 32-bit HP floating buffer to local REAL values |
| `ZRLR32.FOR` | converts buffer of local REAL values to IEEE 32-bit floating-point |
| `ZRLR64.FOR` | convert buffer of local double precision values to IEEE 64-bit float. |
| `ZRM2RL.FOR` | convert Modcomp to local single precision floating point |
| `ZTAB.FOR` | return the character form for horizontal TAB |
| `ZUVPAK.FOR` | Pack visibility data, 1 correlator per real with magic value blank. |
| `ZUVXPN.FOR` | Expands packed visibility data and adds weight |

**Category: FITS**

| | |
|---|---|
| `ZBYTF2.FOR` | interchange bytes in buffer if needed to go between local & standard |
| `ZBYTFL.FOR` | interchange bytes in buffer if needed to go between local & standard |
| `ZTPMID.FOR` | pseudo-tape disk read/write for 2880-bytes records |
| `ZTPOPD.FOR` | open a pseudo-tape, sequential disk file for FITS |
| `ZTPWAD.FOR` | "wait" for IO operation to complete on pseudo-tape disk file (ZTPMID) |
| `ZX8XL.FOR` | convert FITS table bit array to AIPS bit array |

**ZXLX8.FOR** convert AIPS bit array to FITS binary table bit array

**Category: FORMAT**

**ZDM2DL.FOR** convert Modcomp REAL*6 and REAL*8 to local double precision

**Category: GRAPHICS**

**ZTKBUF.FOR** flush TK buffer if needed, then store 8-bit byte in buffer
**ZTKCL2.FOR** close a Tektronix device
**ZTKCLS.FOR** close the TK device
**ZTKFI2.FOR** read/write from/to a Tektronix device
**ZTKOP2.FOR** read/write from/to a Tektronix device
**ZTKOPN.FOR** open a TK device

**Category: IO-BASIC**

**ZCLOSE.FOR** closes open devices: disk, line printer, terminal
**ZCMPR2.FOR** truncate a disk file, returning blocks to the system
**ZCMPRS.FOR** release space from the end of an open disk file
**ZCREA2.FOR** create the specified disk file
**ZCREAT.FOR** creates a disk file
**ZDACLS.FOR** close a disk file
**ZDAOPN.FOR** open the specified disk file
**ZDEST2.FOR** destroy a closed disk file
**ZDESTR.FOR** destroy a closed disk file
**ZEXIS2.FOR** return size of disk file and if it exists
**ZEXIST.FOR** return file size and, consequently, whether file exists
**ZEXPN2.FOR** expand an open disk file
**ZEXPND.FOR** expand an open disk file — either map or non-map now allowed
**ZFI2.FOR** read/write one 256-integer record from/to a non-map disk file
**ZFIO.FOR** reads and writes single 256-integer records to non-map disk files
**ZFRE2.FOR** return AIPS data disk free space information
**ZMI2.FOR** read/write large blocks of data from/to disk, quick return
**ZMIO.FOR** random-access, quick return (double buffer) disk IO for large blocks
**ZMKTMP.FOR** convert a "temporary" file name into a unique name
**ZMSGCL.FOR** close Message file or terminal
**ZMSGDK.FOR** disk IO to message file
**ZMSGOP.FOR** open a message file or message terminal
**ZMSGXP.FOR** expand the message file
**ZOPEN.FOR** open binary disk files and TTY devices
**ZPATH.FOR** convert a file name 'Logical:file' to full path name
**ZPHFIL.FOR** construct a physical file or device name from AIPS logical parameters
**ZPHOLV.FOR** construct a physical file - version for UPDAT
**ZRENA2.FOR** rename a file
**ZTFILL.FOR** zero-fill, initialize a file IO table (FTAB) entry
**ZTPOP2.FOR** open a tape device for double-buffer, asymchronous IO
**ZTPWA2.FOR** wait for read/write from/to a tape device
**ZWAI2.FOR** wait for read/write large blocks of data from/to disk
**ZWAIT.FOR** wait for asynchronous ("MAP") IO to finish

**Category: IO-REMOTE**

| | |
|---|---|
| `ZRCLOS.FOR` | close a connection to a server |
| `ZRCLS2.FOR` | close a connection to a server – generic version |
| `ZROPEN.FOR` | request a connection to a server |
| `ZROPN2.FOR` | request a connection to a server – generic version |
| `ZRRD2.FOR` | read a message from a server – generic version |
| `ZRREAD.FOR` | read a message from a server |
| `ZRWR2.FOR` | write a message to a server – generic version |
| `ZRWRIT.FOR` | write a message to a server |
| `ZRXDW2.FOR` | write one byte of expedited data to a remote process – generic |
| `ZRXDWR.FOR` | write expedited data to a server |

**Category: IO-TV**

| | |
|---|---|
| `ZARGC2.FOR` | close an ARGS TV device |
| `ZARGCL.FOR` | close an ARGS TV device |
| `ZARGMC.FOR` | issues a master clear to an ARGS TV |
| `ZARGO2.FOR` | open ARGS TV device |
| `ZARGOP.FOR` | open ARGS TV device |
| `ZIPACK.FOR` | pack/unpack long integers into short integer buffer |
| `ZM70CL.FOR` | close an IIS Model 70 TV device, flushing any buffer |
| `ZM70M2.FOR` | issues a master clear to an IIS Model 70 TV |
| `ZM70MC.FOR` | issues a master clear to an IIS Model 70 TV |
| `ZM70OP.FOR` | open IIS Model 70 TV device |
| `ZM70XF.FOR` | read/write data to IIS Model 70 TV with buffering |
| `ZV20CL.FOR` | close a Comtal Vision 1/20 TV device |
| `ZV20OP.FOR` | open Comtal Vision 1/20 TV device |
| `ZV20XF.FOR` | read/write data to Comtal Vision 1/20 TV device |

**Category: IO-UTIL**

| | |
|---|---|
| `ZDIR.FOR` | build a full path name to files in AIPS-standard areas (HE, RU, ...) |
| `ZFULLN.FOR` | convert file name to full pathname with no logicals |
| `ZRENAM.FOR` | rename a disk file |

**Category: MATH**

| | |
|---|---|
| `ZHEX10.FOR` | Compute the decimal equivalent of a hexadecimal number. |

**Category: MESSAGES**

| | |
|---|---|
| `ZMSGCL.FOR` | close Message file or terminal |

**Category: OOPS**

| | |
|---|---|
| `ZOINTD.C` | Initialize AIPS Object directory |

**Category: PLOT**

| | |
|---|---|
| `ZTKSRV.FOR` | Provides an X-Windows tektronix graphics terminal |

**Category: PLOT-UTIL**

| | |
|---|---|
| `ZDOPRT.FOR` | reads bit file and causes it to be plotted on printer/plotter |

`ZLASC2.FOR` spool a closed laser printer print/plot file
`ZLASCL.FOR` close and spool a laser printer print/plot file
`ZLASIO.FOR` open, write to, close and spool a laser printer print/plot file
`ZLASOP.FOR` open a laser printer print/plot file
`ZLWIO.FOR` open, write to, close and spool a PostScript print/plot file
`ZLWOP.FOR` open a PostScript (LaserWriter) print/plot file

**Category: PRINTER**

`ZDOPRT.FOR` reads bit file and causes it to be plotted on printer/plotter
`ZENDPG.FOR` advance printer if needed to avoid electrostatic-printer "burn-out"
`ZLASC2.FOR` spool a closed laser printer print/plot file
`ZLASCL.FOR` close and spool a laser printer print/plot file
`ZLASIO.FOR` open, write to, close and spool a laser printer print/plot file
`ZLASOP.FOR` open a laser printer print/plot file
`ZLPCL2.FOR` queue a file to the line printer and delete
`ZLPCLS.FOR` close an open printer device
`ZLPOP2.FOR` open a line-printer text file - actual OPEN call
`ZLPOPN.FOR` open a line-printer text file
`ZLWIO.FOR` open, write to, close and spool a PostScript print/plot file
`ZLWOP.FOR` open a PostScript (LaserWriter) print/plot file

**Category: SERVICE**

`ZADDR.FOR` determine if 2 addresses inside computer are the same
`ZDELA2.FOR` delay current process a specified interval
`ZDELAY.FOR` delay current process a specified interval
`ZDM2DL.FOR` convert Modcomp REAL*6 and REAL*8 to local double precision
`ZERRO2.FOR` return system error message for given system error code
`ZGTBIT.FOR` get array of bits from a word
`ZHEX.FOR` encode an integer into hexadecimal characters
`ZKDUMP.FOR` display portions of an array in various Fortran formats
`ZMSGWR.FOR` call MSGWRT based on call arguments - for C routines to call MSGWRT
`ZMYVER.FOR` returns OLD, NEW, or TST based on translation of logical AIPS_VERSION
`ZPTBIT.FOR` put array of bits into a word
`ZTIME.FOR` return the local time of day
`ZVERSN.FOR` returns code for OLD,NEW,TST based on matching RLSNAM to logicals

**Category: SYSTEM**

`ZABOR2.FOR` establishes or carries out (when appropriate) abort handling
`ZABORT.FOR` establishes or carries out (when appropriate) abort handling
`ZACTV8.FOR` activate the requested program, returning process ID information
`ZCPU.FOR` return current process CPU time and IO count
`ZCRLOG.FOR` (re)define a logical name
`ZDATE.FOR` return the local date
`ZDCHI2.FOR` initialize device and Z-routine characteristics commons - local vals
`ZDCHIC.FOR` set more system parameters; make them available to C routines
`ZDCHIN.FOR` initialize message, device and Z-routine characteristics commons
`ZFREE.FOR` display available disk space

`ZGNAME.FOR` get name of current process
`ZPRI2.FOR` raise or lower the process priority
`ZPRIO.FOR` raise or lower the process priority
`ZPRPAS.FOR` prompt user and read 12-character password (invisible) from CRT
`ZSETUP.FOR` performs system-level operations after VERNAM, TSKNAM, NPOPS known
`ZSTAI2.FOR` does any system cleanup needed at the end of interactive AIPS session
`ZSTAIP.FOR` does any system cleanup needed at the end of interactive AIPS session
`ZTACT2.FOR` inquires if a task is currently active on the local computer
`ZTACTQ.FOR` inquires if a task is currently active on the local computer
`ZTKILL.FOR` deletes (or kills) the specified process
`ZTQSP2.FOR` display AIPS account or all processes running on the system
`ZTQSPY.FOR` display AIPS account or all processes running on the system
`ZTRLOG.FOR` translate a logical name
`ZWHOMI.FOR` determines AIPSxn task name; sets NPOPS, assigns TV and TK devices

**Category: TAPE**

`ZBKLD1.FOR` initialize environment for BAKLD
`ZBKLD2.FOR` does BACKUP operation: load images from tape to directory
`ZBKLD3.FOR` clean up system things for BAKLD ending
`ZBKTP1.FOR` initialize BACKUP to tape operation for BAKTP
`ZBKTP2.FOR` write a cataloged file plus extensions to BACKUP tape in BAKTP
`ZBKTP3.FOR` clean up host environment at end of BAKTP
`ZBYTF2.FOR` interchange bytes in buffer if needed to go between local & standard
`ZBYTFL.FOR` interchange bytes in buffer if needed to go between local & standard
`ZMCACL.FOR` convert Modcomp compressed ASCII to Hollerith characters (for FILLR)
`ZMOUN2.FOR` mount or dismount magnetic tape device
`ZMOUNR.FOR` mount or dismount remote magnetic tape device
`ZMOUNT.FOR` mount or dismount magnetic tape device
`ZR8P4.FOR` converts pseudo I*4 to double precision - for tape handling only
`ZRDMF.FOR` convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers
`ZRM2RL.FOR` convert Modcomp to local single precision floating point
`ZTAP2.FOR` position (forward/back record/file), write EOF, etc. for tapes
`ZTAPE.FOR` mount, dismount, position, write EOF, etc. for tapes
`ZTAPIO.FOR` tape operations for IMPFIT (compressed FITS transport tape)
`ZTAPR.FOR` mount, dismount, position, write EOF, etc. for remote tapes
`ZTPCL2.FOR` close a tape device
`ZTPCLD.FOR` close pseudo-tape disk file
`ZTPCLR.FOR` closes a remote tape device (real or pseudo-tape disk)
`ZTPCLS.FOR` closes a tape device (real or pseudo-tape disk)
`ZTPMI2.FOR` tape read/write
`ZTPMID.FOR` pseudo-tape disk read/write for 2880-bytes records
`ZTPMIO.FOR` read/write tape devices with quick return IO methods
`ZTPMIR.FOR` read/write tape devices with quick return IO methods
`ZTPOP2.FOR` open a tape device for double-buffer, asymchronous IO
`ZTPOPD.FOR` open a pseudo-tape, sequential disk file for FITS
`ZTPOPN.FOR` open tape or pseudo-tape device

`ZTPOPR.FOR` open remote tape or pseudo-tape device
`ZTPWA2.FOR` wait for read/write from/to a tape device
`ZTPWAD.FOR` "wait" for IO operation to complete on pseudo-tape disk file (ZTPMID)
`ZTPWAR.FOR` wait for asynchronous IO to finish on remote tape or pseudo-tape disk
`ZTPWAT.FOR` wait for asynchronous IO to finish on tape or pseudo-tape disk
`ZVTPC2.FOR` close virtual tape connection to remote, real-tape computer
`ZVTPC3.FOR` close connection in real-tape computer to client, virtual-TP computer
`ZVTPGC.FOR` close & reopen connection in server (real-TP) to client (virtual-TP)
`ZVTPO2.FOR` open connection in client (virtual-tape) to server (remote&real-tape)
`ZVTPO3.FOR` open connection in server (real-tape) to client (virtual-tape)
`ZVTPRC.FOR` closes channel in server (real-tape) to client (virtual-tape)
`ZVTPRO.FOR` open socket in server (real-tape) to any client (virtual-tape)
`ZVTPRX.FOR` does IO for server (real tape) to client (Virtual-tape), close/reopen
`ZVTPX2.FOR` writes/reads to/from server for the client (virtual tape) machine
`ZVTPX3.FOR` reads/writes from/to client (virtual tape) for the server (real tape)

**Category: TERMINAL**

`ZPRMPT.FOR` prompt user and read 80-characters from CRT screen
`ZPRPAS.FOR` prompt user and read 12-character password (invisible) from CRT
`ZTTBUF.FOR` reads terminal input with no prompt or wait - simulates TV trackball
`ZTTCLS.FOR` close a terminal device
`ZTTOP2.FOR` open a message terminal
`ZTTOPN.FOR` open a terminal device
`ZTTYIO.FOR` read/write buffer to terminal

**Category: TEXT**

`ZDIR.FOR` build a full path name to files in AIPS-standard areas (HE, RU, ...)
`ZTCLOS.FOR` close text file opened with ZTOPEN
`ZTOPE2.FOR` open text file for ZTOPEN
`ZTOPEN.FOR` open text file - logical area, version, member name as arguments
`ZTREAD.FOR` read next 80-character record in sequential text file (ZTOPEN type)
`ZTXCLS.FOR` clos text file opened via ZTXOPN
`ZTXIO.FOR` read/write a line to a text file
`ZTXMA2.FOR` find all file names matching a given wildcard specification
`ZTXMAT.FOR` return list of files in specified area beginning with specified chars
`ZTXOP2.FOR` translate the file name and open a text file
`ZTXOPN.FOR` open a text file for read or write
`ZTXZAP.FOR` delete text file openable via ZTXOPN

**Category: TV-IO**

`ZARGS.FOR` sends command to/from the ARGS TV device
`ZARGXF.FOR` translates IIS Model 70 commands into calls to ZARGS for ARGS TV
`ZDEAC2.FOR` close DeAnza TV device
`ZDEACL.FOR` close DeAnza TV device
`ZDEAMC.FOR` issue a master clear to the TV - for DeAnzas this is a No-Op
`ZDEAO2.FOR` opens DeAnza TV device
`ZDEAOP.FOR` opens DeAnza TV device

`ZDEAX2.FOR` do actual read/write from/to DeAnza device
`ZDEAXF.FOR` do IO to DeAnza TV
`ZIVSO2.FOR` opens IVAS TV device - using the IIS IVAS package
`ZIVSOP.FOR` opens IVAS TV device - using the IIS package
`ZM70C2.FOR` close IIS Model 70/75 TV device
`ZM70O2.FOR` opens IIS Model 70.75 TV device
`ZM70X2.FOR` read/write from/to IIS Model 70/75 device
`ZSSSC2.FOR` Close SUN Screen Server device, flushing any buffer
`ZSSSCL.FOR` close the SUN-Screen-Server device
`ZSSSMC.FOR` issue a Master Clear operation to SUN-Screen-Server devbice
`ZSSSO2.FOR` Open SUN-Screen-Server device, set-up any buffer
`ZSSSOP.FOR` open SUN-Screen-Server (TV) device
`ZSSSX2.FOR` write to/ read from SUN-Screen-Server device
`ZSSSXF.FOR` write/read to/from the SUN-Screen-Server (TV) device
`ZTTBUF.FOR` reads terminal input with no prompt or wait - simulates TV trackball
`ZV20C2.FOR` close Comtal Vision 1/20 TV device
`ZV20MC.FOR` issue a master clear to the TV - for Comtal this is a No-Op
`ZV20O2.FOR` opens Comtal Vision 1/20 TV device
`ZV20X2.FOR` does I/O to Comtal Vision 1/20 TV device
`ZVTVC2.FOR` close virtual TV connection to remote, real-TV computer
`ZVTVC3.FOR` close connection in real-TV computer to client, virtual-TV computer
`ZVTVCL.FOR` close connection in client (virtual-TV) to server (remote, real-TV)
`ZVTVGC.FOR` close & reopen connection in server (real-TV) to client (virtual-tv)
`ZVTVO3.FOR` open connection in server (real-TV) to client (virtual-TV)
`ZVTVOP.FOR` opens connection from client (virtual-TV) to server (real-TV)
`ZVTVRC.FOR` closes channel in server (real-TV) to client (virtual-TV)
`ZVTVRO.FOR` open scket in server (real-TV) to any client (virtual-TV)
`ZVTVX2.FOR` writes/reads to/from server for the client (virtual TV) machine
`ZVTVX3.FOR` reads/writes from/to client (virtual TV) for the server (real TV)
`ZVTVXF.FOR` sends data from the client (virtual TV) to server (real TV)

**Category: TV-UTIL**

`ZVTVO2.FOR` open connection in client (virtual-TV) to server (remote, real-TV)

**Category: UTILITY**

`ZERROR.FOR` prints strings associated with system error codes for Z routines
`ZMSGER.FOR` prints strings associated with system error codes for ZMSG routines
`ZMSSRV.FOR` Provides an X-Windows message terminal
`ZTKSRV.FOR` Provides an X-Windows tektronix graphics terminal

**Category: WINDOWS**

`ZMSSRV.FOR` Provides an X-Windows message terminal
`ZTKSRV.FOR` Provides an X-Windows tektronix graphics terminal

**Category: Z**

`ZABORT.FOR` establishes or carries out (when appropriate) abort handling
`ZACTV8.FOR` activate the requested program, returning process ID information

| | |
|---|---|
| `ZADDR.FOR` | determine if 2 addresses inside computer are the same |
| `ZARGCL.FOR` | close an ARGS TV device |
| `ZARGMC.FOR` | issues a master clear to an ARGS TV |
| `ZARGOP.FOR` | open ARGS TV device |
| `ZARGXF.FOR` | translates IIS Model 70 commands into calls to ZARGS for ARGS TV |
| `ZBKLD1.FOR` | initialize environment for BAKLD |
| `ZBKLD2.FOR` | does BACKUP operation: load images from tape to directory |
| `ZBKLD3.FOR` | clean up system things for BAKLD ending |
| `ZBKTP1.FOR` | initialize BACKUP to tape operation for BAKTP |
| `ZBKTP2.FOR` | write a cataloged file plus extensions to BACKUP tape in BAKTP |
| `ZBKTP3.FOR` | clean up host environment at end of BAKTP |
| `ZBYMOV.FOR` | move 8-bit bytes from in-buffer to out-buffer |
| `ZBYTFL.FOR` | interchange bytes in buffer if needed to go between local & standard |
| `ZC8CL.FOR` | convert packed ASCII buffer to local character string |
| `ZCLC8.FOR` | convert local character string to packed ASCII buffer |
| `ZCLOSE.FOR` | closes open devices: disk, line printer, terminal |
| `ZCMPRS.FOR` | release space from the end of an open disk file |
| `ZCPU.FOR` | return current process CPU time and IO count |
| `ZCREAT.FOR` | creates a disk file |
| `ZDATE.FOR` | return the local date |
| `ZDCHIN.FOR` | initialize message, device and Z-routine characteristics commons |
| `ZDEACL.FOR` | close DeAnza TV device |
| `ZDEAOP.FOR` | opens DeAnza TV device |
| `ZDEAXF.FOR` | do IO to DeAnza TV |
| `ZDELAY.FOR` | delay current process a specified interval |
| `ZDESTR.FOR` | destroy a closed disk file |
| `ZDHPRL.FOR` | convert 64-bit HP floating buffer to local DOUBLE PRECISION values |
| `ZDM2DL.FOR` | convert Modcomp REAL*6 and REAL*8 to local double precision |
| `ZENDPG.FOR` | advance printer if needed to avoid electrostatic-printer "burn-out" |
| `ZERROR.FOR` | prints strings associated with system error codes for Z routines |
| `ZEXIST.FOR` | return file size and, consequently, whether file exists |
| `ZEXPND.FOR` | expand an open disk file — either map or non-map now allowed |
| `ZFIO.FOR` | reads and writes single 256-integer records to non-map disk files |
| `ZFREE.FOR` | display available disk space |
| `ZGETCH.FOR` | get a character from a REAL word |
| `ZGNAME.FOR` | get name of current process |
| `ZGTBIT.FOR` | get array of bits from a word |
| `ZHEX.FOR` | encode an integer into hexadecimal characters |
| `ZIPACK.FOR` | pack/unpack long integers into short integer buffer |
| `ZKDUMP.FOR` | display portions of an array in various Fortran formats |
| `ZLASIO.FOR` | open, write to, close and spool a laser printer print/plot file |
| `ZLPCLS.FOR` | close an open printer device |
| `ZLPOPN.FOR` | open a line-printer text file |
| `ZLWIO.FOR` | open, write to, close and spool a PostScript print/plot file |
| `ZM70MC.FOR` | issues a master clear to an IIS Model 70 TV |
| `ZM70OP.FOR` | open IIS Model 70 TV device |

`ZM70XF.FOR` read/write data to IIS Model 70 TV with buffering
`ZMIO.FOR` random-access, quick return (double buffer) disk IO for large blocks
`ZMKTMP.FOR` convert a "temporary" file name into a unique name
`ZMOUNT.FOR` mount or dismount magnetic tape device
`ZMSGCL.FOR` close Message file or terminal
`ZMSGDK.FOR` disk IO to message file
`ZMSGER.FOR` prints strings associated with system error codes for ZMSG routines
`ZMSGOP.FOR` open a message file or message terminal
`ZMSGXP.FOR` expand the message file
`ZMSSRV.FOR` Provides an X-Windows message terminal
`ZOPEN.FOR` open binary disk files and TTY devices
`ZPHFIL.FOR` construct a physical file or device name from AIPS logical parameters
`ZPHOLV.FOR` construct a physical file - version for UPDAT
`ZPRIO.FOR` raise or lower the process priority
`ZPRMPT.FOR` prompt user and read 80-characters from CRT screen
`ZPRPAS.FOR` prompt user and read 12-character password (invisible) from CRT
`ZPTBIT.FOR` put array of bits into a word
`ZPUTCH.FOR` inserts 8-bit "character" into a word
`ZRDMF.FOR` convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers
`ZRENAM.FOR` rename a disk file
`ZRLR64.FOR` convert buffer of local double precision values to IEEE 64-bit float.
`ZRM2RL.FOR` convert Modcomp to local single precision floating point
`ZSTAIP.FOR` does any system cleanup needed at the end of interactive AIPS session
`ZTAB.FOR` return the character form for horizontal TAB
`ZTACTQ.FOR` inquires if a task is currently active on the local computer
`ZTAPE.FOR` mount, dismount, position, write EOF, etc. for tapes
`ZTAPIO.FOR` tape operations for IMPFIT (compressed FITS transport tape)
`ZTIME.FOR` return the local time of day
`ZTKSRV.FOR` Provides an X-Windows tektronix graphics terminal
`ZTOPEN.FOR` open text file - logical area, version, member name as arguments
`ZTPMIO.FOR` read/write tape devices with quick return IO methods
`ZTPWAT.FOR` wait for asynchronous IO to finish on tape or pseudo-tape disk
`ZTQSPY.FOR` display AIPS account or all processes running on the system
`ZTTBUF.FOR` reads terminal input with no prompt or wait - simulates TV trackball
`ZTXCLS.FOR` clos text file opened via ZTXOPN
`ZTXIO.FOR` read/write a line to a text file
`ZTXMAT.FOR` return list of files in specified area beginning with specified chars
`ZTXOPN.FOR` open a text file for read or write
`ZTXZAP.FOR` delete text file openable via ZTXOPN
`ZV20CL.FOR` close a Comtal Vision 1/20 TV device
`ZV20MC.FOR` issue a master clear to the TV - for Comtal this is a No-Op
`ZV20OP.FOR` open Comtal Vision 1/20 TV device
`ZV20XF.FOR` read/write data to Comtal Vision 1/20 TV device
`ZVERSN.FOR` returns code for OLD,NEW,TST based on matching RLSNAM to logicals
`ZWHOMI.FOR` determines AIPSxn task name; sets NPOPS, assigns TV and TK devices

**Category: Z2**

| | |
|---|---|
| ZABOR2.FOR | establishes or carries out (when appropriate) abort handling |
| ZARGC2.FOR | close an ARGS TV device |
| ZARGO2.FOR | open ARGS TV device |
| ZARGS.FOR | sends command to/from the ARGS TV device |
| ZBYTF2.FOR | interchange bytes in buffer if needed to go between local & standard |
| ZCMPR2.FOR | truncate a disk file, returning blocks to the system |
| ZCREA2.FOR | create the specified disk file |
| ZCRLOG.FOR | (re)define a logical name |
| ZDACLS.FOR | close a disk file |
| ZDAOPN.FOR | open the specified disk file |
| ZDCHI2.FOR | initialize device and Z-routine characteristics commons - local vals |
| ZDCHIC.FOR | set more system parameters; make them available to C routines |
| ZDEAC2.FOR | close DeAnza TV device |
| ZDEAO2.FOR | opens DeAnza TV device |
| ZDELA2.FOR | delay current process a specified interval |
| ZDEST2.FOR | destroy a closed disk file |
| ZDIR.FOR | build a full path name to files in AIPS-standard areas (HE, RU, ...) |
| ZERRO2.FOR | return system error message for given system error code |
| ZEXIS2.FOR | return size of disk file and if it exists |
| ZEXPN2.FOR | expand an open disk file |
| ZFI2.FOR | read/write one 256-integer record from/to a non-map disk file |
| ZFRE2.FOR | return AIPS data disk free space information |
| ZIVSO2.FOR | opens IVAS TV device - using the IIS IVAS package |
| ZLASC2.FOR | spool a closed laser printer print/plot file |
| ZLASCL.FOR | close and spool a laser printer print/plot file |
| ZLASOP.FOR | open a laser printer print/plot file |
| ZLPCL2.FOR | queue a file to the line printer and delete |
| ZLPOP2.FOR | open a line-printer text file - actual OPEN call |
| ZM70C2.FOR | close IIS Model 70/75 TV device |
| ZM70O2.FOR | opens IIS Model 70.75 TV device |
| ZMI2.FOR | read/write large blocks of data from/to disk, quick return |
| ZMOUN2.FOR | mount or dismount magnetic tape device |
| ZMOUNR.FOR | mount or dismount remote magnetic tape device |
| ZMSGWR.FOR | call MSGWRT based on call arguments - for C routines to call MSGWRT |
| ZPATH.FOR | convert a file name 'Logical:file' to full path name |
| ZPRI2.FOR | raise or lower the process priority |
| ZRCLS2.FOR | close a connection to a server – generic version |
| ZRENA2.FOR | rename a file |
| ZROPN2.FOR | request a connection to a server – generic version |
| ZRRD2.FOR | read a message from a server – generic version |
| ZRWR2.FOR | write a message to a server – generic version |
| ZRXDW2.FOR | write one byte of expedited data to a remote process – generic |
| ZSTAI2.FOR | does any system cleanup needed at the end of interactive AIPS session |
| ZTACT2.FOR | inquires if a task is currently active on the local computer |
| ZTAP2.FOR | position (forward/back record/file), write EOF, etc. for tapes |
| ZTAPR.FOR | mount, dismount, position, write EOF, etc. for remote tapes |

`ZTKCL2.FOR` close a Tektronix device
`ZTKOP2.FOR` read/write from/to a Tektronix device
`ZTOPE2.FOR` open text file for ZTOPEN
`ZTPCL2.FOR` close a tape device
`ZTPCLR.FOR` closes a remote tape device (real or pseudo-tape disk)
`ZTPMI2.FOR` tape read/write
`ZTPMID.FOR` pseudo-tape disk read/write for 2880-bytes records
`ZTPOP2.FOR` open a tape device for double-buffer, asymchronous IO
`ZTPOPD.FOR` open a pseudo-tape, sequential disk file for FITS
`ZTPOPR.FOR` open remote tape or pseudo-tape device
`ZTPWA2.FOR` wait for read/write from/to a tape device
`ZTPWAR.FOR` wait for asynchronous IO to finish on remote tape or pseudo-tape disk
`ZTQSP2.FOR` display AIPS account or all processes running on the system
`ZTTCLS.FOR` close a terminal device
`ZTTOP2.FOR` open a message terminal
`ZTTOPN.FOR` open a terminal device
`ZTXMA2.FOR` find all file names matching a given wildcard specification
`ZTXOP2.FOR` translate the file name and open a text file
`ZV20C2.FOR` close Comtal Vision 1/20 TV device
`ZV20O2.FOR` opens Comtal Vision 1/20 TV device
`ZV20X2.FOR` does I/O to Comtal Vision 1/20 TV device
`ZVTPC2.FOR` close virtual tape connection to remote, real-tape computer
`ZVTPC3.FOR` close connection in real-tape computer to client, virtual-TP computer
`ZVTPGC.FOR` close & reopen connection in server (real-TP) to client (virtual-TP)
`ZVTPO2.FOR` open connection in client (virtual-tape) to server (remote&real-tape)
`ZVTPO3.FOR` open connection in server (real-tape) to client (virtual-tape)
`ZVTPX2.FOR` writes/reads to/from server for the client (virtual tape) machine
`ZVTPX3.FOR` reads/writes from/to client (virtual tape) for the server (real tape)
`ZWAI2.FOR` wait for read/write large blocks of data from/to disk