Appendix A

AIPS Directory Structure and Software Management

A.1 Introduction

This appendix is based on AIPS Memo Number 39. The purpose of Memo 39 was to propose shareable images for AIPS under VMS. To this end, the authors proposed a revision of the directory structure and software management tools. This revision has been implemented, whereas shareable load modules in VMS have not. The model presented for the directory structure has also been adopted for Unix.

This appendix describes the directory structure and the software management tools that a programmer will need to work in a VMS or Unix environment. The original discussion of shareable load modules has been dropped, and the other discussion updated to reflect the current realities, especially the Unix implementation.

A.2 Directory Structure

A.2.1 Design Guidelines

The following are some of the guidelines used in devising this scheme.

1. Separate source code from all other system-specific files. This source code directory tree should contain no system-specific object libraries, command procedures etc., as these may well be implemented differently on different machines.
2. The source code areas should be clearly organized into standard AIPS areas and particular operating-system or device-specific areas. It is also convenient to allow the existence of a few generic areas for routines that are not standard, but are useful in various environments.
3. Clarify routine hierarchy to allow shareable images to be sensibly defined and to clearly reflect linking sequences.
4. The subroutine and program hierarchy should be independent of any object libraries or shareable images used on a particular system. The source code directories may be assembled into object libraries etc. in any manner convenient for the system being used.
5. Preserve non-standard areas so that we can keep track of programs which are, or use, non-standard code.
6. Define search paths to pick up the most suitable version of a routine automatically. For example, the search should begin with any device-specific routine, then with a generic routine, and finally with a standard routine. The first one found should be used. This ensures that the most efficient is used, while allowing less efficient, more general ones to be available.
7. Try to make the structure as logical and consistent as possible.

A.2.2 Directory Structure

The directory structure requires a hierarchical file system on the host computer. Given this restriction, it should be easy to implement on various operating systems. It attempts to divide up the files along the following lines.

1. Routine hierarchy — i.e., whether a routine makes use of the AP (or vector routines) or TV.

2. Routine type — whether a routine is a general library routine or specific to a single application program.

3. Routine version — whether a routine is standard and works with all implementations, generic and works with some, or specific and only works with one implementation.

The proposed directory structure uses the first of the above as the primary division of source code. All source code is contained in five top level areas i.e., areas one level below the AIPS version node (e.g., 15OCT85). These areas are labelled as follows:

1. APL — general utility routines

2. Q — AP (Vector) routines

3. Y — TV routines

4. QY — AP and TV routines (at present only application programs)

5. AIPS — POPS utility routines (may use TV also)

There are a few obvious omissions from this list, such as no attempt to formalize various graphics, terminal or network devices. These may also benefit from such a division, but at present AIPS has no suitably general model available. These may be added later.

These top level areas are each divided in an identical manner into three, although the third is omitted from the QY and AIPS areas:

1. Programs — application programs. Lower level areas are present for any device-specific programs. A non-standard area is also provided.

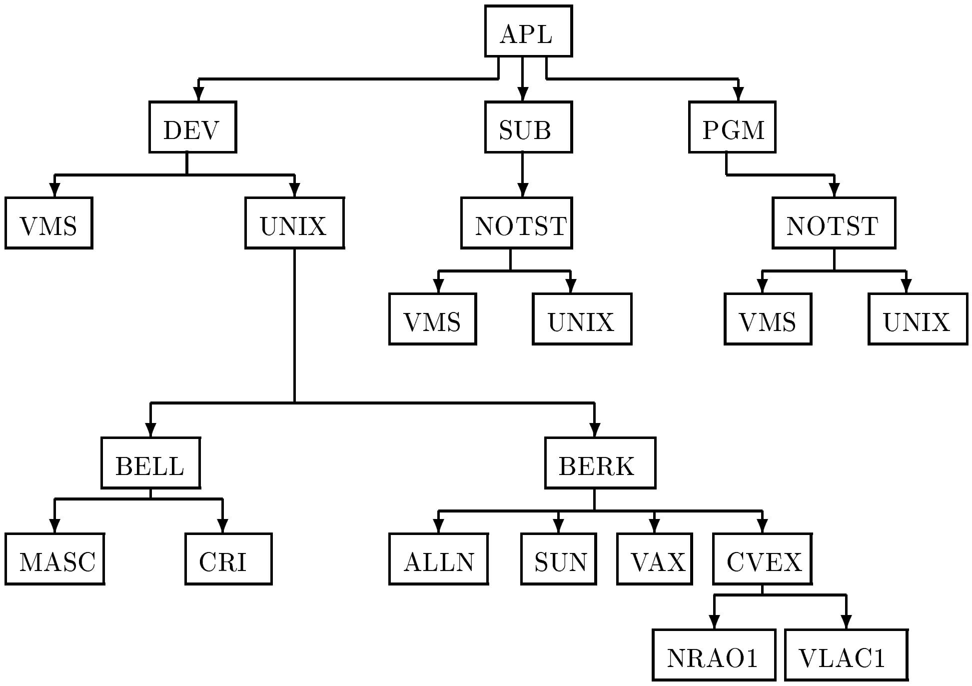
2. Utility routines — library subroutines that may call device-specific routines, but are themselves device independent. A non-standard area is also provided.

3. Device routines — library subroutines that are device specific. Various generic areas are also included.

In addition to these five source code areas, there are several other top level directory areas. All of these are now described in more detail. In this discussion, only three operating system branches are shown, but more can easily be added. Some of these low level areas may be further sub-divided, for example, to allow for different flavors/vendors of Unix systems.

APL

This area is for utility routines and programs that make no reference to an AP or TV device.

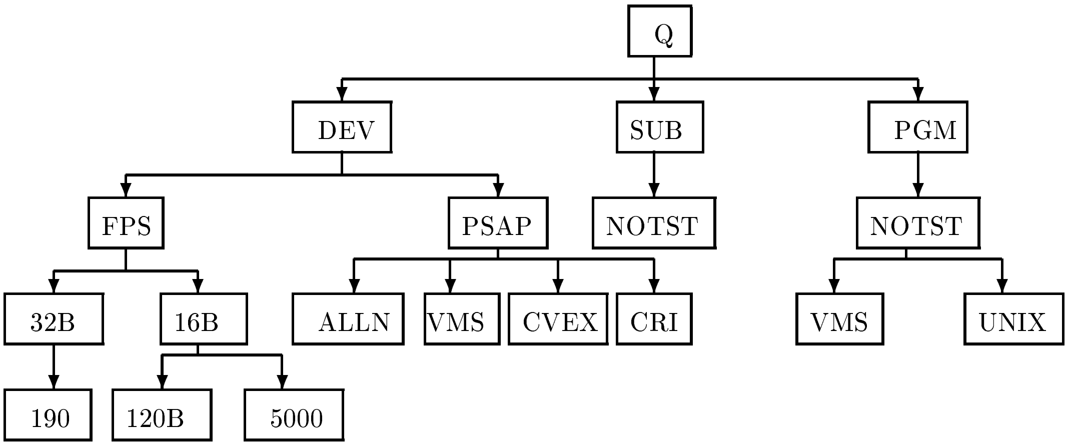


APL Directories

The DEV branch is for the standard set of Z routines. Many of these have now been made generic for some operating systems, and these are in the DEV area itself. The lower levels are for true system-specific versions. The SUB branch is for routines that are in principle system independent. There is a NOTST area for those which, while not fully following AIPS coding standards, stand a good chance of working on many systems. The system-specific areas on this branch are for peculiar non-standard routines that are not part of standard AIPS. The PGM branch is for task programs. It too has non-standard and system-specific areas.

Q

This area is for routines and programs that make use of the AP.

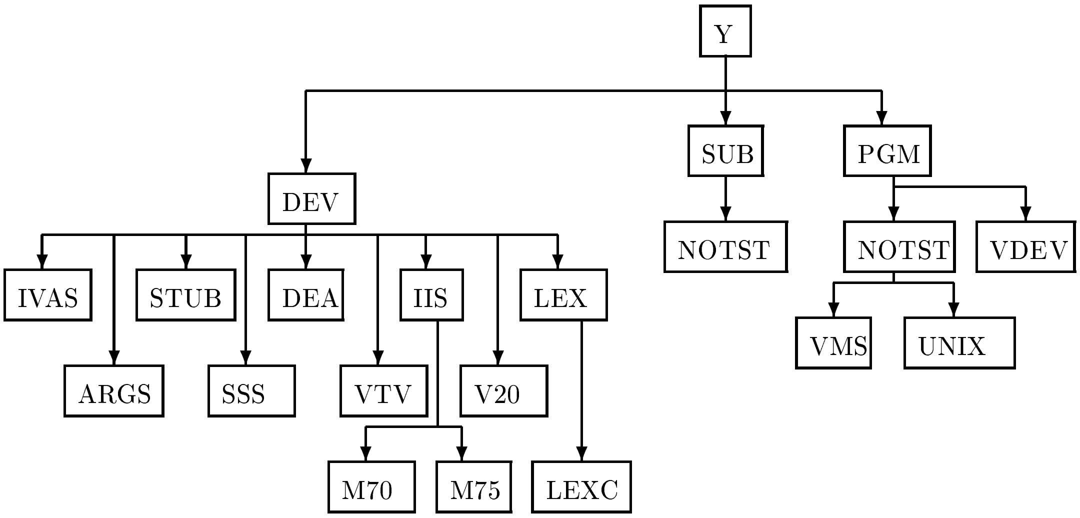


Q Directories

The DEV branch is for the various versions of the Q routines. The DEV area itself is for the most general version of these, i.e., the PSAP or "Pseudo AP" code. The lower level branches support a variety of different AP devices and vector hardware, in some cases with generic areas. Note that, because of the search path mechanism, these low level areas need not contain a full set of Q routines, generic ones from higher up the tree can be substituted. The SUB branch is for routines which make use of the Q routines, but are themselves device independent. This includes a non-standard area, but no system-specific ones. The PGM branch is for tasks which use the "Q" routines.

Y

This area is for routines and programs that make use of the TV.

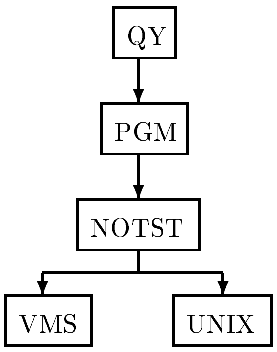


Y Directories

This tree is very similar to the Q tree. The only difference is in the device-specific DEV branch. The generic DEV area is for Y routines that really are implemented in device independent-ways. Note that there is a difference here between the Q and Y trees — all systems have some kind of "AP", while some systems do not have a TV. We therefore need to be able to distinguish generic routines from stubbed routines substituted when no TV is present. This is the purpose of the STUB area. Y routines for which no generic version is possible have stubbed versions in the generic DEV area. Those that do have generic versions have stubbed versions in the STUB area.

QY

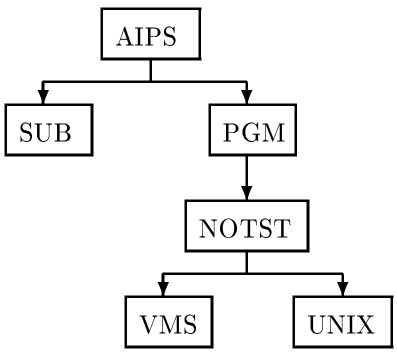
This area is for routines and programs that make use of the both the AP and TV. At present, this only occurs at the program level, so this tree is very simple.



QY Directories

AIPS

This area is for the POPS-level programs and related routines. Several of these make use of the TV device, but, as they are routines not accessible to tasks, they reside here. The stand-alone service programs are also stored in this area.

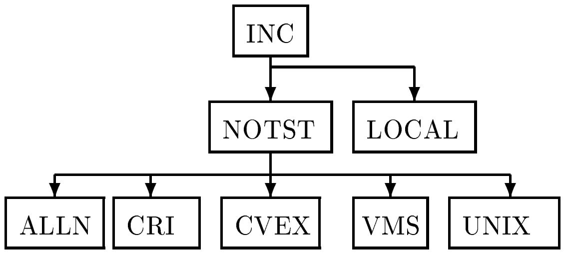


AIPS Directories

Notice that at present, there are no non-standard subroutines and no device- or system-specific subroutines.

Include

This area is for the various include files needed by routines in all the above trees.



Include Directories

The system-specific areas allow array sizes to change between systems, and also permit system-specific options, such as dependency directives needed by vectorizing compilers.

Help

The HELP tree is very simple, as all help files are in a standard format. This tree consists of a single area.

Load

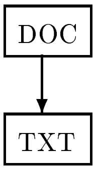
This area is for load modules, i.e., fully linked programs in a form ready to be run. This is split into a standard LOAD area and a few alternative areas immediately below (e.g., LOAD.ALT1). These alternate areas could, for example, be used to keep pseudo-AP versions of programs or versions linked for a second model of TV display.

Library

This area (LIBR) is for the various subroutine libraries used to build AIPS programs. Note that these have been moved out of the system-independent source code areas. We may in the future wish to include several libraries not of AIPS origin along with AIPS. These would enable AIPS programs to make use of some useful code that is available in the public domain. Such libraries will be included in this area.

Documentation

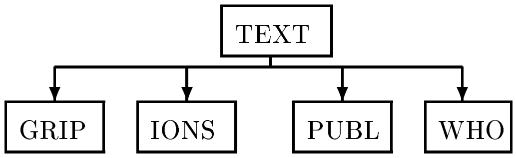
The DOC area is used to store documentation files (this manual, other coding descriptions) in DOCTXT. The directory structure is simple:



Documentation Directories

Text files

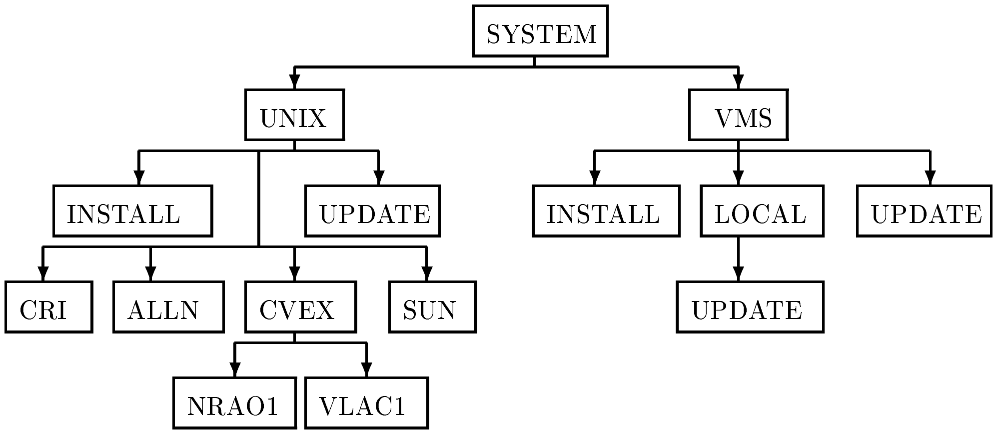
There are a number of directories which reside above the version specific portion of the AIPS directory tree. These include AIPSPUBL containing the *Cookbook* text, AIPSGRIP containing the AIPS gripe system, AIPSIONS containing ionospheric monotoring data and AIPSWHO containing mailing lists etc. The directory structure is shown in the following:



Text Directories

System

This area is used to store the various system-specific tools needed for programming, maintenance and execution of AIPS.



System Directories

The Unix, and VMS areas are for procedures and files describing system structures and link edit paths. The LOCAL areas are for local variants (site dependent) on the standard files. INSTALL areas are for source code shipping/installation procedures and UPDATE areas are for automatic update procedures (the so-called "midnight job").

A.2.3 Mnemonics – AREAS.DAT

Programmers always refer to the AIPS directory areas by means of mnemonics. These need to be implemented on various operating systems and it is convenient to store a list of them, complete with their associated areas, in a file which can be used by the operating system. Below is a copy of this file from SYSAIPS. It can be used to assign the appropriate mnemonics, or to create a complete directory tree.

```
! "-----------------------------------------------------------------"
! " AREAS.DAT                                                       "
! "-----------------------------------------------------------------"
! " This file defines the entire AIPS directory structure (relative to "
! " a 'version' directory node) plus the 'logical' associated with    "
! " each 'area' (i.e., directory).                                  "
! "                                                                 "
! " Unix: The script $SYSUNIX/AREAS uses $SYSAIPS/AREAS.DAT to create  "
! " $SYSLOCAL/AREAS.CSH and $SYSLOCAL/AREAS.SH, which contain C shell  "
! " and Bourne/Korn shell syntax, respectively, to define the area     "
! " logicals in terms of environment variables.  The definitions are   "
! " toggled between different versions of AIPS by executing $CDOLD,    "
! " $CDNEW or $CDTST, which are defined $AIPS_ROOT/LOGIN.CSH and/or    "
! " $AIPS_ROOT/LOGIN.SH.  The AREAS.* environment variables are NOT    "
! " required for the execution of AIPS programs, just for AIPS         "
! " programming.  In fact, having all the area logicals defined takes  "
! " up such a large fraction of the available environment space on     "
! " some systems that the execution of AIPS programs, Unix commands,   "
! " AIPS programming tools and system debuggers can be impaired.  To   "
! " avoid this, it may be necessary to comment out some or all of the  "
! " definitions that don't apply to the host implementation.          "
! "                                                                 "
! " VMS: AIPS_PROC:ASSNLOCAL.COM uses this file more or less directly  "
! " to establish the definitions of area logicals.  The definitions    "
! " are toggled between different versions of AIPS by executing the    "
! " AIPS defined 'VERSION' command (see AIPS_PROC:AIPSUSER.COM), e.g., "
! " 'VERSION NEW'.                                                   "
! "-----------------------------------------------------------------"
! " Top level directories of source code areas                      "
! "-----------------------------------------------------------------"
!
AIP             AIPS
APL             APL
Q               Q
QY              QY
Y               Y
!
! "-----------------------------------------------------------------"
! " AIPS program areas                                              "
! "-----------------------------------------------------------------"
!
!                                       "Standard programs"
```

```
AIPPGM          AIPS.PGM
!                                       "Non-standard programs"
AIPNOT          AIPS.PGM.NOTST
!                                       "Unix non-standard programs"
AIPGUNIX        AIPS.PGM.NOTST.UNIX
!                                       "VMS non-standard programs"
AIPGVMS         AIPS.PGM.NOTST.VMS
!
! "--------------------------------------------------------------------"
! " AIP subroutine areas - only referenced by AIP* programs            "
! "--------------------------------------------------------------------"
!
AIPSUB          AIPS.SUB
!


! "--------------------------------------------------------------------"
! " APL program areas - these reference only APL* routines             "
! "--------------------------------------------------------------------"
!
!                                       "Standard programs"
APLPGM          APL.PGM
!                                       "Non-standard programs"
APGNOT          APL.PGM.NOTST
!                                       "Unix programs"
APGUNIX         APL.PGM.NOTST.UNIX
!                                       "VMS programs"
APGVMS          APL.PGM.NOTST.VMS
!
! "--------------------------------------------------------------------"
! " APL subroutine areas - nothing here references Q or Y-routines     "
! "--------------------------------------------------------------------"
!
!                                       "Standard routines"
APLSUB          APL.SUB
!                                       "Non-standard routines"
APLNOT          APL.SUB.NOTST
!                                       "VMS non-standard routines"
APLNVMS         APL.SUB.NOTST.VMS
!
! "--------------------------------------------------------------------"
! " Z-routine areas                                                    "
! "--------------------------------------------------------------------"
!
!                                       "Generic"
APLGEN          APL.DEV
!                                       "Generic Unix"
APLUNIX         APL.DEV.UNIX
!                                       "Bell Unix"
APLBELL         APL.DEV.UNIX.BELL
!                                       "Cray Research Inc"
APLCRI          APL.DEV.UNIX.BELL.CRI
!                                       "Masscomp"
APLMASC         APL.DEV.UNIX.BELL.MASC
!                                       "Berkeley Unix"
```

```
APLBERK         APL.DEV.UNIX.BERK
!                                       "Alliant"
APLALLN         APL.DEV.UNIX.BERK.ALLN
!                                       "Convex"
APLCVEX         APL.DEV.UNIX.BERK.CVEX
!                                       "NRAO-VLA Convex local"
APLVLAC1        APL.DEV.UNIX.BERK.CVEX.VLAC1
!                                       "NRAO-CV Convex local"
APLNRAO1        APL.DEV.UNIX.BERK.CVEX.NRAO1
!                                       "Sun"
APLSUN          APL.DEV.UNIX.BERK.SUN
!                                       "VAX"
APLVAX          APL.DEV.UNIX.BERK.VAX
!                                       "Generic VMS"
APLVMS          APL.DEV.VMS
!
! "-------------------------------------------------------------------"
! " Documentation areas                                               "
! "-------------------------------------------------------------------"
!
DOC             DOC
DOCTXT          DOC.TEXT
!

! "-------------------------------------------------------------------"
! " Core dump area                                                    "
! "-------------------------------------------------------------------"
!
ERRORS          ERRORS
!
! "-------------------------------------------------------------------"
! " HELP file area                                                    "
! "-------------------------------------------------------------------"
!
HLPFIL          HELP
!
! "-------------------------------------------------------------------"
! " History area                                                      "
! "-------------------------------------------------------------------"
!
HIST            HIST
!
! "-------------------------------------------------------------------"
! " INCLUDE file areas                                                "
! "-------------------------------------------------------------------"
!
!                                       "Standard INCLUDEs"
INC             INC
!                                       "Non-standard INCLUDEs"
INCNOT          INC.NOTST
!                                       "Local"
INCLOC          INC.LOCAL
!                                       "Alliant"
INCALN          INC.NOTST.ALLN
```

```
!                                       "Cray Research Inc"
INCCRI          INC.NOTST.CRI
!                                       "Convex"
INCVEX          INC.NOTST.CVEX
!                                       "VMS"
INCVMS          INC.NOTST.VMS
!

! "-------------------------------------------------------------------"
! " Object module areas                                               "
! "-------------------------------------------------------------------"
!
!                                       "Subroutine object libraries"
LIBR            LIBR
!                                       "Executable modules"
LOAD            LOAD
!                                       "Alternate executable modules"
!                                       "Pseudo AP w/wo TV 1"
LOAD1           LOAD.ALT1
!                                       "TV 2 w/wo real AP"
LOAD2           LOAD.ALT2
!                                       "TV 2 w Pseudo AP"
LOAD3           LOAD.ALT3
!
! "-------------------------------------------------------------------"
! " POPS memory file area                                             "
! "-------------------------------------------------------------------"
!
MEMORY          MEMORY
!
! "-------------------------------------------------------------------"
! " Q-routine areas (real and pseudo array processor)                 "
! "-------------------------------------------------------------------"
!
!                                       "Generic"
QDEV            Q.DEV
!                                       "Generic FPS"
QFPS            Q.DEV.FPS
!                                       "16 bit FPS"
QFPS16          Q.DEV.FPS.16B
!                                       "Model 120B FPS"
Q120B           Q.DEV.FPS.16B.120B
!                                       "Models 5105, 5205 ... FPS"
Q5000           Q.DEV.FPS.16B.5000
!                                       "32 bit FPS"
QFPS32          Q.DEV.FPS.32B
!                                       "Model 190 FPS"
Q190            Q.DEV.FPS.32B.190
!                                       "Generic pseudo AP"
QPSAP           Q.DEV.PSAP
!                                       "Alliant pseudo AP"
QALN            Q.DEV.PSAP.ALLN
!                                       "Cray Research Inc pseudo AP"
QCRI            Q.DEV.PSAP.CRI
```

```
!                                       "Convex pseudo AP"
QVEX            Q.DEV.PSAP.CVEX
!                                       "VMS pseudo AP"
QVMS            Q.DEV.PSAP.VMS
!

! "-----------------------------------------------------------------------"
! " Programs that reference Q-routines                                    "
! "-----------------------------------------------------------------------"
!
!                                       "Standard programs"
QPGM            Q.PGM
!                                       "Non-standard programs"
QPGNOT          Q.PGM.NOTST
!                                       "VMS programs"
QPGVMS          Q.PGM.NOTST.VMS
!
! "-----------------------------------------------------------------------"
! " Subroutines that reference Q-routines                                 "
! "-----------------------------------------------------------------------"
!
!                                       "Standard routines"
QSUB            Q.SUB
!                                       "Non-standard routines"
QNOT            Q.SUB.NOTST
!
! "-----------------------------------------------------------------------"
! " Programs that reference both Q-routines and Y-routines                "
! "-----------------------------------------------------------------------"
!
!                                       "Standard programs"
QYPGM           QY.PGM
!                                       "Non-standard programs"
QYPGNOT         QY.PGM.NOTST
!                                       "VMS programs"
QYPGVMS         QY.PGM.NOTST.VMS
!

! "-----------------------------------------------------------------------"
! " System RUN file area - useful procedures for everyone                 "
! "-----------------------------------------------------------------------"
!
RUNSYS          RUN
!
! "-----------------------------------------------------------------------"
! " System manager areas                                                  "
! "-----------------------------------------------------------------------"
!
!                                       "Midnight job data area"
UPDATE          UPDATE
!                                       "Generic"
SYSAIPS         SYSTEM
!                                       "Generic Unix"
SYSUNIX         SYSTEM.UNIX
!                                       "Alliant"
```

```
SYSALLN         SYSTEM.UNIX.ALLN
!                                       "Cray Research Inc"
SYSCRI          SYSTEM.UNIX.CRI
!                                       "Convex"
SYSCVEX         SYSTEM.UNIX.CVEX
!                                       "NRAO-VLA Convex local"
SYSVLAC1        SYSTEM.UNIX.CVEX.VLAC1
!                                       "NRAO-CV Convex local"
SYSNRAO1        SYSTEM.UNIX.CVEX.NRAO1
!                                       "Sun"
SYSSUN          SYSTEM.UNIX.SUN
!                                       "UNIX installation"
INSUNIX         SYSTEM.UNIX.INSTALL
!                                       "Unix midnight and quarterly"
!                                       "update"
UPDUNIX         SYSTEM.UNIX.UPDATE
!                                       "NRAO-CV Convex midnight and"
!                                       "quarterly update"
UPDNRAO1        SYSTEM.UNIX.UPDATE.NRAO1
!                                       "NRAO-VLA Convex midnight and"
!                                       "quarterly update"
UPDVLAC1        SYSTEM.UNIX.UPDATE.VLAC1
!                                       "Generic VMS"
SYSVMS          SYSTEM.VMS
!                                       "Local VMS"
SYSLVAX         SYSTEM.VMS.LOCAL
!                                       "VMS installation"
INSVMS          SYSTEM.VMS.INSTALL
!                                       "VMS midnight and quarterly"
!                                       "update"
UPDVMS          SYSTEM.VMS.UPDATE
!                                       "NRAO-VLA VAXn midnight and"
!                                       "quarterly update"
UPDVLA          SYSTEM.VMS.UPDATE.VLA
!

! "-----------------------------------------------------------------------"
! " Y-routine areas                                                       "
! "-----------------------------------------------------------------------"
!
!                                       "Generic"
YGEN            Y.DEV
!                                       "ARGS pipe from M70"
YARGS           Y.DEV.ARGS
!                                       "Deanza"
YDEA            Y.DEV.DEA
!                                       "IIS generic"
YIIS            Y.DEV.IIS
!                                       "IIS Model IVAS"
YIVAS           Y.DEV.IVAS
!                                       "IIS Model 70"
YM70            Y.DEV.IIS.M70
!                                       "IIS Model 75"
YM75            Y.DEV.IIS.M75
```

```
!                                       "Lexidata"
YLEX            Y.DEV.LEX
!                                       "Lexidata C code"
YLEXC           Y.DEV.LEX.LEXC
!                                       "SUN Screen Server"
YSSS            Y.DEV.SSS
!                                       "SSS - SunView *.C"
YSVU            Y.DEV.SSS.SVU
!                                       "Stubbed"
YSTUB           Y.DEV.STUB
!                                       "Comtal Vision 1/20"
YV20            Y.DEV.V20
!                                       "Virtual TV"
YVTV            Y.DEV.VTV
!

! "------------------------------------------------------------------"
! " Programs that reference Y-routines                               "
! "------------------------------------------------------------------"
!
!                                       "Standard programs"
YPGM            Y.PGM
!                                       "Virtual TV program"
YPGVDEV         Y.PGM.VDEV
!                                       "Non-standard programs"
YPGNOT          Y.PGM.NOTST
!                                       "VMS programs"
YPGVMS          Y.PGM.NOTST.VMS
!
! "------------------------------------------------------------------"
! " Subroutines that reference Y-routines                            "
! "------------------------------------------------------------------"
!
!                                       "Standard routines"
YSUB            Y.SUB
!                                       "Non-standard routines"
YNOT            Y.SUB.NOTST
```

A.3 File Names For Data

As of the 15APR86 version of AIPS, the disk volume field for data files was replaced by a data format version code in the form of a letter. The letter used for 15APR86 was “A” and this changed to “B” for the 15JAN87 release. It should be quite sometime before we get to “Z”. As an example, the 15OCT85 format map file MA201501.221;1 was renamed to MAA01501.221;1 in the 15APR86 release. As of the 15OCT89 release the revision code has been “C”.

The change has a number of advantages:

1. Data backed up by system utilities (e.g., tar under Unix, BACKUP under VMS) can be restored to a different disk.
2. Multiple dismountable disk drives are now supported better. Previously, a disk written as AIPS disk 2 and then dismounted always had to be re-mounted as AIPS disk 2.
3. Data from different releases of AIPS with different data formats can coexist peacefully during data-format transitions. Data with different formats can be distinguished easily by filename.

4. An intelligent data file format update program (UPDAT) has been written. It can recognize what version of input data it is being fed and convert the format to the current version.

Files that are shared among users (and between different versions), such as system-parameter files, accounting files, batch files, etc. are found in the directory pointed to by logical device name DA00 and have a "1" in the AIPS version letter field (the "1" doesn't signify anything).

Memory files are stored in the version-specific area, $AIPS_VERSION/MEMORY under Unix and AIPS_VERSION:[MEMORY] under VMS. These also have a "1" in the AIPS version letter field.

A.4 VMS Details

The previous sections described the directories that are visible in all versions of AIPS. This section details the specifics of the VMS implementation.

A.4.1 Object libraries

With the source code directory structure, it is possible for AIPS to use different object library structures with different operating systems, as is convenient. Below is a list of object libraries suitable for VMS, together with a list of areas from which they are built. Note that the object library file names have been deliberately lengthened with the LIB string. This is to prevent any name conflicts with the directory-area mnemonics, which are listed below in search-path order.

1. APLSUBLIB.OLB from APLSUB
2. APLNOTLIB.OLB from APLNVMS, APLNOT
3. APLVMSLIB.OLB from APLVMS, APLGEN
4. QSUBLIB.OLB from QSUB
5. QNOTLIB.OLB from QNOT
6. QVMSLIB.OLB from QVMS, QPSAP
7. Q120BLIB.OLB from Q120B, QFPS16, QFPS
8. Q5000LIB.OLB from Q5000, QFPS16, QFPS
9. Q190LIB.OLB from Q190, QFPS32, QFPS
10. YSUBLIB.OLB from YSUB
11. YNOTLIB.OLB from YNOT
12. YSTUBLIB.OLB from YSTUB, YGEN
13. YM70LIB.OLB from YM70, YIIS, YGEN
14. YM75LIB.OLB from YM75, YIIS, YGEN
15. YDEALIB.OLB from YDEA, YGEN
16. YV20LIB.OLB from YV20, YGEN
17. YIVASLIB.OLB from YIVAS, YGEN
18. AIPSUBLIB.OLB from AIPSUB

When routines are modified, these object libraries are updated by means of a COMRPL procedure. There are a large number of directories; this means that programmers need to know precisely where a routine resides. It may be possible to reduce the impact of this by setting up logical names to implement search paths to find a particular routine. However, initially we have not done this, so as to help ensure that the programmers are aware of which version of a routine they are modifying, and any consequences it may have. Second, some routines find their way into more that one object library. This is done deliberately to simplify linking procedures while still maintaining a single copy of the ultimate source. The necessary intelligence to replace a routine in multiple libraries has been built into the COMRPL procedure, together with the intelligence to avoid replacing a device-specific routine in the library with a generic one. Appendix B is useful in determining in which directory a routine resides.

These object libraries serve two purposes. They can be used directly by a COMTST procedure for programs to link with directly. This is not the normal mode of operation, but is available for testing purposes. Normally the object libraries are used to build load modules with the COMLNK procedure. These procedures are described in detail in section 6.

A.5 A Tutorial for Programmers Using VMS

A.5.1 Initialization and Startup Procedures

LOGIN.PRG

The logical names and symbols needed to program in AIPS can be obtained by executing command procedure LOGIN.PRG. A programmer should put the following line (substituting the disk used for AIPS at his site for "AIPS_DISK_NAME") in his LOGIN.COM file:

```
$ @AIPS_PROC:LOGIN.PRG
```

where the logical is defined as

```
$ DEF AIPS_PROC AIPS_Disk_Name:[AIPS.date.SYSTEM.VMS]
```

At NRAO, this procedure makes TST the default AIPS_VERSION. Other sites may only have one AIPS_VERSION (NEW) and may have things set up differently.

AIPS "Version" "Option"

This procedure starts up a given version of AIPS. On NRAO Vaxes, "Version" can be one of OLD, NEW, or TST. One can also start up AIPS with the following options:

```
REMOTE - Used to run AIPS from a TEK graphics terminal.
DEBUG  - Run AIPS with the debugger.
LOCAL  - Run a private AIPS found in the current default directory.
```

The DEBUG option works only if the standard AIPS is linked with debug, or if you use the LOCAL option and you have an AIPS linked with debug in your current default directory.

A.5.2 Compiling and Linking

COMRPL "SubroutineSpec" "Option"

This procedure will preprocess, compile and replace a subroutine or set of subroutines in the proper AIPS libraries. The "Option" field, if present, MUST follow the "Subroutine Spec" field, rather than precede it. The parameter "SubroutineSpec" can be a single logical name and subroutine such as APLSUB:CTICS, or it can be a list of subroutines such as APLSUB:CTICS,COPY,APLNOT:CHKTAB, or it can be a wild-card such as APLSUB:CH*.*, or it can be a file containing a list or routines such as @MYLIST.TXT (the "@"

signifies a file). Note that, to specify the directory of the subroutine, you MUST use a logical name, such as APLSUB, rather than the full directory specification, such as [AIPS.15APR86.APL.SUB]. The procedure uses the standard AIPS defaults for the compile (FORTRAN) command. You may use any of the valid FORTRAN options listed at the end of this section. If you want to use more than one option, separate them with at least one blank. For example, the following command will compile subroutine CHCOPY, replace it in the standard AIPS library area, produce a listing, and produce no warning messages for undeclared variables, tabs, and lower case code (the highly deprecated DIRTY option).

```
$ COMRPL APLSUB:CHCOPY LIST DIRTY
```

The following examples show how multiple files can be compiled.

```
$ COMRPL APLSUB:MSGWRT,APLNOT:NXTFLG  ! Compile MSGWRT and NXTFLG.
$ COMRPL APLSUB:MP2*.FOR              ! Compile every routine whose
                                      ! name begins with MP2.
$ COMRPL @MYLIST.TXT                  ! Compile every routine listed
                                      ! in MYLIST.TXT
```

COMLNK "ProgramSpec" "Option"

This procedure will preprocess, compile and link a program or set of programs and put them in the AIPS "LOAD" area. If any alternate areas are set up, such as the pseudo AP area, then modules linked with alternate libraries will be put in the alternate areas. The "ProgramSpec" may be a list of programs, a wild-card, or a file containing a list of programs as described in the COMRPL explanation. The "Option" may be any of the list of options at the end of this section.

COMTST "ProgramSpec" "Option"

This is a version of COMLNK designed for preprocessing, compiling and linking experimental AIPS programs in a programmer's own area. This procedure will compile and link a program or set of programs and put the executable module in the current default directory. This routine also uses an option file "ProgramName".OPT, if it exists, or LOCAL.OPT, if it does not. One of these option files MUST be found in the default directory. Option files are used to specify which libraries and routines to link with a program. A programmer will usually copy the appropriate COMLNK option file to his own area for use with COMTST. COMLNK finds its option files in AIPS_PROC by following this rule: If a program is found in a directory XYZ, then its option file is AIPS_PROC:XYZOPT.OPT. If an alternate LOAD area exists for a program, such as the pseudo AP area, then COMLNK also uses AIPS_PROC:XYZOPTn.OPT (n = 1 to 6) to link the alternate executable module(s). A programmer working with MX (which is found in QYPGNOT) will copy AIPS_PROC:QYPGNOTOPT.OPT to his own area and rename it LOCAL.OPT or MX.OPT. If a programmer wants to use the pseudo AP libraries instead, then he will copy AIPS_PROC:QYPGNOTOPT1.OPT to his area and rename it LOCAL.OPT or MX.OPT. These option files can also be used as a means of specifying experimental subroutines or libraries. For instance, a programmer working on MX may copy AIPS_PROC:QYPGNOTOPT.OPT into MX.OPT and then put the names of any experimental subroutines or libraries in MX.OPT. A full example is given in the section "COMPILING AND LINKING, AN EXAMPLE".

Options

The following options can be used with the compile and link procedures:

```
Option       Minimum
           Abbreviation    Comments

DEBUG          DE          LINK with DEBUG (compile is always debug)
```

```
NODEBUG      NODE     LINK without DEBUG (Default)
LIST         LI       produce compiler listing
NOLIST       NOLI     no listing (Default)
MAP          MA       produce LINKER map.
NOMAP        NOMA     no linker map (Default)
OPTIMIZE     OP       compile optimized and NODEBUG.
NOOPTIMIZE   NOOP     compile no-optimized (Default)
DIRTY        DI       no warnings for undeclared variables, tabs
NODIRTY      NODI     warnings for undeclared var, tabs (Default)
PURGE        PU       purge executable after link (Default)
NOPURGE      NOPU     do not purge executable
```

A.5.3 Miscellaneous routines

VERSION "Version"

This command will set the default version (release) to "Version", i.e., all logicals will point to the "Version" version of the directories. "Version" can be either OLD, NEW or TST. The version will stay in effect until the programmer changes it, or logs off. Note that, when starting up the AIPS program, this command is executed to select the version of AIPS to be used. This procedure should be used (with "Version" NEW) before checking out programs from NEW, or compiling and linking NEW routines. To again use the TST version, use the procedure with "Version" set to TST.

FORK "command"

FORK is useful for running things, such as links and compiles, as a subprocess. It is defined to be

```
SPAWN/NOWAIT/NOTIFY/INPUT=NLA0:/OUTPUT=FORK.LOG"
```

The following example shows how to compile and link IMLOD in a subprocess:

```
$ FORK COMLNK IMLOD
```

FLOG

This command is defined to be "TYPE FORK.LOG" and will type the latest FORK log file in the current directory.

A.5.4 Compiling and Linking: An Example

This example shows how we can compile and link an experimental version of program MX with experimental versions of subroutines GRDAT and DSKFFT, and keep the executable image in our own directory.

First, we set our default to some work directory and copy the current versions of MX, DSKFFT, and GRDAT from QYPGNOT and APLNOT. NRAO programmers should copy the routines using the code checkout system.

Next, we need an option file to tell the linker what subroutines and libraries to use. MX is found in QYPGNOT, so we copy over the option file for the QYPGNOT programs and rename it to LOCAL.OPT or MX.OPT. This can be done using the following command:

```
$ COPY AIPS_PROC:QYPGNOTOPT.OPT LOCAL.OPT
```

QYPGNOTOPT not only works for MX, but, since it has every library (except for the POPS language processor stuff) in it, it can also be used to link any task with the standard AIPS subroutines.

To link MX with our experimental version of GRDAT and DSKFFT, we can use the text editor to change LOCAL.OPT which looks like this:

```
LIBR:QNOTLIB/LIB,LIBR:APLNOTLIB/LIB,-
LIBR:QSUBLIB/LIB,-
LIBR:Q120BLIB/LIB,-
LIBR:YSUBLIB/LIB,LIBR:YM70LIB/LIB,-
LIBR:APLSUBLIB/LIB,LIBR:APLVMSLIB/LIB,LIBR:APLSUBLIB/LIB,-
FPS:HSRLIB/LIB,FPS:FPSLIB/LIB
```

to make it look like this:

```
GRDAT,DSKFFT,-
LIBR:QNOTLIB/LIB,LIBR:APLNOTLIB/LIB,-
LIBR:QSUBLIB/LIB,-
LIBR:Q120BLIB/LIB,-
LIBR:YSUBLIB/LIB,LIBR:YM70LIB/LIB,-
LIBR:APLSUBLIB/LIB,LIBR:APLVMSLIB/LIB,LIBR:APLSUBLIB/LIB,-
FPS:HSRLIB/LIB,FPS:FPSLIB/LIB
```

The "-" is the line continuation indicator in option files.

To preprocess and compile subroutines in a private directory use the following procedure:

```
$!COMPILE.COM
$!-------------------------------------------------------------------
$! Use:
$! @COMPILE subroutine_name   option ! option equals LIST or CROSS or
$! DIRTY or nothing
$! Only does a compile, leaves .OBJ
$!--------------------------------------------------------------------
$!
$!                                      Determine if LIST option.
$ OPTION := "/NOLIST"
$ IF (P2.EQS."LIST") THEN OPTION := "/LIST"
$ IF (P2.EQS."CROSS") THEN OPTION := "/LIST/CROSS/SHOW=INCLUDE"
$ OPT1 := "/STANDARD=(SYNTAX,SOURCE_FORM)/WARNINGS=(DECLARATIONS)"
$ IF (P2.EQS."DIRTY") THEN OPT1 := ""
$ OPT2 := "/DEBUG/NOOPTIMIZE"
$ ON ERROR THEN GOTO FINI
$!                                      Preprocess
$ WRITE SYS$OUTPUT " Preprocess "'P1'".FOR  to "'P1'".f"
$ @AIPS_PROC:PP 'P1'.FOR 'P1'.f
$!                                      Compile subroutine
$ WRITE SYS$OUTPUT "compile in ",F$DIRECTORY(),":"'P1'".f"
$ FOR 'OPT1' 'OPT2' 'OPTION' 'P1'.f
$ PURGE 'P1'.OBJ
$ PURGE 'P1'.f
$ FINI:
$ EXIT
```

Now we make the changes to GRDAT, DSKFFT and MX. Then we compile and link them with the following commands (the DEBUG on the COMTST command is optional):

```
$@COMPILE GRDAT
$@COMPILE DSKFFT
$ COMTST MX DEBUG
```

Suppose we want to link MX with debug and have the link run as a subprocess. Then we can type in

```
$ FORK COMTST MX DEBUG
```

We will be notified when COMTST finishes (or aborts!). We should type FORK.LOG (we can use the FLOG command) to make sure our task compiled and linked correctly.

A.5.5 Debugging under VMS

To run the VMS debugger the task and any relevant routines should have been compiled and link edited with the DEBUG option. Use of the debugger on optimized code can be confusing so is best avoided.

If a private directory is to be used during the debugging phase then the .EXE and the .HLP file should be in the same directory. Then inside AIPS set adverb VERSION to point to this directory, e.g:
> VERSION = 'mydisk:[mydir.aips]'
and INPUTS, HELP and GO will use the right versions of the files. To cause a task to be run under the debugger in AIPS use pseudo verb SETDEBUG, e.g.:
> setdebug = 20

Using a value of 0 turns off initiating tasks under the debugger. It is also useful to type "WAIT" after "GO" to AIPS to prevent both AIPS and the debugger from trying to talk to the terminal at the same time.

A.5.6 Check out system

The AIPS group has instituted a check-out system for the text files in the master version of the AIPS system (including CHANGE.DOC). The purpose of this check out system is to prevent different programmers from destroying each others changes to code by trying to work on the same routines at the same time. There are occasionally changes made in AIPS which require changes in most or all tasks; frequently the original programmer of a task will be unaware of these changes. For these reasons, modifications or additions to the the master version of AIPS should (are required to):

1. Check out the relevant files. A brief description of the checkout system is given in a later section; a detailed description of the check-out system may be found in DOCTXT:CHKOUT.RNO.
2. Modify the files.
3. Check the files back in.
4. Document the changes in CHANGE.DOC (which must itself be checked out).

All directories should be specified using the logical names instead of the full directory names. The programmer must make sure that AIPS_VERSION is set correctly. AIPS_VERSION will be TST after a programmer executes LOGIN.PRG, but AIPS_VERSION can be set to NEW if the programmer runs the NEW version of AIPS or sets the version to NEW using the VERSION command.

To check things out of NEW, the programmer should use the command

```
$ VERSION NEW
```

to set the programmer's current working version to NEW. The version can be reset to TST with the command

```
$ VERSION TST
```

A file that is still checked out of NEW cannot be checked out of TST, or vice versa.

A brief description of the functions of the checkout system is given in the following:

- CHKOUT < file name > Allows a programmer to checkout a file. A copy of the file will be written into the current default directory. The file name must include the logical defining the directory. CHKOUT will ask for a one line reason for checking out the file. Example: CHKOUT APLPGM:IMEAN.FOR.
- PUTBCK < file name > Returns a modified file to the appropriate directory; the file must reside in the default directory. The file name must include the logical defining the directory. Example: PUTBCK APLPGM:IMEAN.FOR.
- REMOVE < file name > Deletes all versions of a file. The file name must include the logical defining the directory. Example: REMOVE APGVMS:VBAD.FOR.
- FORGET < file name > Cancels the CHKOUT of a file. The file name must include the logical defining the directory. Example: FORGET APLSUB:MDISK.FOR.
- NAMCHK < file name > Reserves a name for a routine being developed. The file name must include the logical defining the directory. Example: NAMCHK APLPGM:HMEAN.FOR.
- CPURGE < file name > Purges files in the standard AIPS source code directories matching file name. Executing a VMS PURGE command will not be allowed to delete these files.
- OUTPRT Prints a list of files currently checked out.
- HISPRT Prints the checkout history.

A.6 Unix Details

This section describes the details the for the Unix implementation. In many cases, the Unix implementation is the same as for VMS.

A.6.1 Mnemonics

Programmers always refer to the AIPS directory areas by means of mnemonics. These need to be implemented on various operating systems and it is convenient to store a list of them, complete with their associated areas in a file which can be used by the operating system. A copy of this file appears in section A.2.3 above. It can be used to assign the appropriate mnemonics and/or to create a complete directory tree.

A.6.2 Object Libraries

With the source code directory structure shown above, it is possible for AIPS to use different parts of the directory infrastructure with different operating systems and peripherals. Under Unix, the mapping of source code area search paths, the mapping of subroutine source code area to object libraries, and the mapping of object library link lists to program source code areas are all maintained in a single file called LIBR.DAT. The paraform LIBR.DAT provided in the generic Unix system area (i.e., $SYSUNIX) is listed below. This paraform should be copied to $SYSLOCAL and modified to reflect the host implementation. Note that the object library file names are always SUBLIB and that they are each stored in a subdirectory of $LIBR, the name of which reflects the source code area from which the object code is derived. In the case of libraries generated from multiple source code areas, the name reflects the most vendor/model/version specific area used (e.g., YIVAS, APLCVEX). Under Unix, the mechanics of adding/replacing object code in an object library are rather expensive. For this reason, object libraries are maintained in separate subdirectories of $LIBR so that new object modules may be staged there. These are added/replaced en masse whenever the target object library is included as part of a link operation (see COMLNK below).

```
                     --- Begin $SYSUNIX/LIBR.DAT ----
AIPS subroutine source code search paths and object libraries:

$LIBR/AIPSUB/SUBLIB:0:$AIPSUB

APL subroutine source code search paths and object libraries:

Standard routines

$LIBR/APLSUB/SUBLIB:0:$APLSUB

Non-standard and routines

$LIBR/APLNOT/SUBLIB:0:$APLNUNIX
$LIBR/APLNOT/SUBLIB:0:$APLNOT

Z-routines

$LIBR/APLALLN/SUBLIB:0:---Your local Z-routine directory goes here---
$LIBR/APLALLN/SUBLIB:0:$APLALLN---For example---
$LIBR/APLALLN/SUBLIB:0:$APLBERK---For example---
$LIBR/APLALLN/SUBLIB:0:$APLUNIX
$LIBR/APLALLN/SUBLIB:0:$APLGEN

Q subroutine source code search paths and object libraries:

Standard routines

$LIBR/QSUB/SUBLIB:0:$QSUB

Non-standard routines

$LIBR/QNOT/SUBLIB:0:$QNOT

Q-routines

$LIBR/QVEX/SUBLIB:0:$QVEX---For example---
$LIBR/QVEX/SUBLIB:0:$QPSAP---For example---
$LIBR/QVEX/SUBLIB:0:$QDEV

Y subroutine source code search paths and object libraries:

Standard routines

$LIBR/YSUB/SUBLIB:0:$YSUB

Non-standard routines

$LIBR/YNOT/SUBLIB:0:$YNOT
```

```
Y-routines

$LIBR/YSTUB/SUBLIB:0:$YSTUB---For example---
$LIBR/YSTUB/SUBLIB:0:$YGEN

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

AIPGUNIX => Unix specific stand alone programs

$LIBR/AIPSUB/SUBLIB:0:$AIPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$AIPGUNIX
$LIBR/APLSUB/SUBLIB:0:$AIPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$AIPGUNIX
$LIBR/APLSUB/SUBLIB:0:$AIPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$AIPGUNIX

AIPPGM => Standard stand alone programs

$LIBR/AIPSUB/SUBLIB:0:$AIPPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$AIPPGM
$LIBR/YSUB/SUBLIB:0:$AIPPGM
$LIBR/YSTUB/SUBLIB---For example---:0:$AIPPGM
$LIBR/APLSUB/SUBLIB:0:$AIPPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$AIPPGM
$LIBR/APLSUB/SUBLIB:0:$AIPPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$AIPPGM

YVTVPGM => TV by wire control program

$LIBR/AIPSUB/SUBLIB:0:$YVTVPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$YVTVPGM
$LIBR/YSUB/SUBLIB:0:$YVTVPGM
$LIBR/YSSS/SUBLIB---For example---:0:$YVTVPGM
$LIBR/APLSUB/SUBLIB:0:$YVTVPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$YVTVPGM
$LIBR/APLSUB/SUBLIB:0:$YVTVPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$YVTVPGM

APL-task source code search paths and link libraries:

APGUNIX => Unix specific tasks that call neither Q nor Y-routines

$LIBR/APLNOT/SUBLIB:0:$APGUNIX
$LIBR/APLSUB/SUBLIB:0:$APGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$APGUNIX
$LIBR/APLSUB/SUBLIB:0:$APGUNIX

APGNOT => Non-standard tasks that call neither Q nor Y-routines

$LIBR/APLNOT/SUBLIB:0:$APGNOT
$LIBR/APLSUB/SUBLIB:0:$APGNOT
$LIBR/APLALLN/SUBLIB---For example---:0:$APGNOT
$LIBR/APLSUB/SUBLIB:0:$APGNOT
$LIBR/APLNOT/SUBLIB:0:$APGNOT
```

```
$LIBR/APLALLN/SUBLIB---For example---:0:$APGNOT

APLPGM => Standard tasks that call neither Q nor Y-routines

$LIBR/APLSUB/SUBLIB:0:$APLPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$APLPGM
$LIBR/APLSUB/SUBLIB:0:$APLPGM

Q-task source code search paths and link libraries:

QPGUNIX => Unix specific tasks that call Q-routines but not Y-routines

$LIBR/QNOT/SUBLIB:0:$QPGUNIX
$LIBR/APLNOT/SUBLIB:0:$QPGUNIX
$LIBR/QSUB/SUBLIB:0:$QPGUNIX
$LIBR/QVEX/SUBLIB---For example---:0:$QPGUNIX
$LIBR/APLSUB/SUBLIB:0:$QPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$QPGUNIX
$LIBR/APLSUB/SUBLIB:0:$QPGUNIX

QPGNOT => Non-standard tasks that call Q-routines but not Y-routines

$LIBR/QNOT/SUBLIB:0:$QPGNOT
$LIBR/APLNOT/SUBLIB:0:$QPGNOT
$LIBR/QSUB/SUBLIB:0:$QPGNOT
$LIBR/QVEX/SUBLIB---For example---:0:$QPGNOT
$LIBR/APLSUB/SUBLIB:0:$QPGNOT
$LIBR/APLALLN/SUBLIB---For example---:0:$QPGNOT
$LIBR/APLSUB/SUBLIB:0:$QPGNOT

QPGM => Standard tasks that call Q-routines but not Y-routines

$LIBR/QSUB/SUBLIB:0:$QPGM
$LIBR/QVEX/SUBLIB---For example---:0:$QPGM
$LIBR/APLSUB/SUBLIB:0:$QPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$QPGM
$LIBR/APLSUB/SUBLIB:0:$QPGM

Y-task source code search paths and link libraries:

YPGUNIX => Unix specific tasks that call Y-routines but not Q-routines

$LIBR/YNOT/SUBLIB:0:$YPGUNIX
$LIBR/APLNOT/SUBLIB:0:$YPGUNIX
$LIBR/YSUB/SUBLIB:0:$YPGUNIX
$LIBR/YSTUB/SUBLIB---For example---:0:$YPGUNIX
$LIBR/APLSUB/SUBLIB:0:$YPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$YPGUNIX
$LIBR/APLSUB/SUBLIB:0:$YPGUNIX

YPGNOT => Non-standard tasks that call Y-routines but not Q-routines

$LIBR/YNOT/SUBLIB:0:$YPGNOT
$LIBR/APLNOT/SUBLIB:0:$YPGNOT
```

```
$LIBR/YSUB/SUBLIB:0:$YPGNOT
$LIBR/YSTUB/SUBLIB---For example---:0:$YPGNOT
$LIBR/APLSUB/SUBLIB:0:$YPGNOT
$LIBR/APLALLN/SUBLIB---For example---:0:$YPGNOT
$LIBR/APLSUB/SUBLIB:0:$YPGNOT

YPGM => Standard tasks that call Y-routines but not Q-routines

$LIBR/YSUB/SUBLIB:0:$YPGM
$LIBR/YSTUB/SUBLIB---For example---:0:$YPGM
$LIBR/APLSUB/SUBLIB:0:$YPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$YPGM
$LIBR/APLSUB/SUBLIB:0:$YPGM

QY-task source code search paths and link libraries:

QYPGUNIX => Unix specific tasks that call both Q-routines and Y-routines

$LIBR/QNOT/SUBLIB:0:$QYPGUNIX
$LIBR/APLNOT/SUBLIB:0:$QYPGUNIX
$LIBR/QSUB/SUBLIB:0:$QYPGUNIX
$LIBR/QVEX/SUBLIB---For example---:0:$QYPGUNIX
$LIBR/YSUB/SUBLIB:0:$QYPGUNIX
$LIBR/YSTUB/SUBLIB---For example---:0:$QYPGUNIX
$LIBR/APLSUB/SUBLIB:0:$QYPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:0:$QYPGUNIX
$LIBR/APLSUB/SUBLIB:0:$QYPGUNIX

QYPGNOT => Non-standard tasks that call both Q-routines and Y-routines

$LIBR/QNOT/SUBLIB:0:$QYPGNOT
$LIBR/APLNOT/SUBLIB:0:$QYPGNOT
$LIBR/QNOT/SUBLIB:0:$QYPGNOT
$LIBR/QSUB/SUBLIB:0:$QYPGNOT
$LIBR/QVEX/SUBLIB---For example---:0:$QYPGNOT
$LIBR/YSUB/SUBLIB:0:$QYPGNOT
$LIBR/YSTUB/SUBLIB---For example---:0:$QYPGNOT
$LIBR/APLSUB/SUBLIB:0:$QYPGNOT
$LIBR/APLALLN/SUBLIB---For example---:0:$QYPGNOT
$LIBR/APLSUB/SUBLIB:0:$QYPGNOT

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

$LIBR/QSUB/SUBLIB:0:$QYPGM
$LIBR/QVEX/SUBLIB---For example---:0:$QYPGM
$LIBR/YSUB/SUBLIB:0:$QYPGM
$LIBR/YSTUB/SUBLIB---For example---:0:$QYPGM
$LIBR/APLSUB/SUBLIB:0:$QYPGM
$LIBR/APLALLN/SUBLIB---For example---:0:$QYPGM
$LIBR/APLSUB/SUBLIB:0:$QYPGM
                     --- End $SYSUNIX/LIBR.DAT ----
```

There are two major procedures called COMRPL and COMLNK used in the programming of AIPS under Unix.

COMRPL, given the name of an AIPS subroutine and a reasonable starting point, will search the directory structure for the version of the source code most appropriate to the host implementation, preprocess it (if necessary), compile it (if necessary) and stage the resulting object module for replacement in the proper object library or libraries.

Under some implementations, it is necessary that the object module from a given routine be stored in more than one object library. For example, if a system has the luxury of two TV display devices that are not of the same make and model (e.g., IIS model 70 and IIS model M75), it is possible that the object module generated from a given subroutine source code area (e.g., $YIIS) is the same for both devices. In this case, a copy of the object module is staged for replacement in each of the object libraries appropriate to the different devices (e.g., $LIBR/YM70/SUBLIB and $LIBR/YM75/SUBLIB).

COMLNK, given the name of an AIPS program and a reasonable starting point, will search the directory structure for the version of the source code most appropriate to the host implementation, preprocess it (if necessary), compile it (if necessary), determine from $SYSLOCAL/LIBR.DAT the appropriate object libraries to include in its link list, perform the link and move the resulting executable to the appropriate load library.

Similar to the case of COMRPL, under some implementations, it is necessary that the object module from a given program be linked with more than one list of object libraries. Each link produces a distinct executable module. For example, given the same hypothetical system described above, where there are two TV display devices that are not of the same make and model (e.g., IIS model 70 and IIS model M75), the object module generated from a given TV oriented program source code area (e.g., $YPGM) needs to be linked once with the object library list including the library appropriate for one of the devices and then again with the object library list appropriate for the other device. The resulting executables are moved to the appropriate load libraries (e.g., $LOAD and $LOAD2). In multiple TV device environments, the desired TV must be selected by the user at the beginning of an AIPS session. The AIPS startup procedure will query the user for this, if a definition for the environment variable TVDEV2 exists.

Unlike the programming environment for AIPS under VMS, the procedure COMTST does not exist. The Unix version of COMLNK has been designed to detect whether the directory of the specified program module is one of the official AIPS source code areas. If not, it moves the resulting executable module to the current working directory (if necessary) instead of the official AIPS load library. This also requires that the user provide a filename with the extension ".OPT" (or ".opt") containing a suitable object module/library link list. Similarly, if such a link list is provided and the program module resides in one of the official AIPS source code areas, COMLNK will assume that this is a non-standard link and will simply move the resulting executable to the current working directory (if necessary). All of these intended protections against corruptions of the official load library can be easily circumvented. They are mostly intended to protect against inadvertent corruptions. Such link list files are specified as command line arguments to the COMLNK procedure, e.g.,

```
COMLNK $APLPGM/UVSRT APLPGM.OPT
```

A utility exists called LIBS that will display the required link list for the programs which reside in a given AIPS source code area. For example,

```
LIBS $QYPGNOT > FOO.OPT
```

would generate the object library link list required for all programs that reside in the source code area defined as $QYPGNOT (i.e., non-standard programs that depend on both Q-routines and Y-routines) and redirect the list to FOO.OPT, i.e.,

```
$LIBR/QNOT/SUBLIB
$LIBR/APLNOT/SUBLIB
$LIBR/QSUB/SUBLIB
```

```
$LIBR/QVEX/SUBLIB
$LIBR/YSUB/SUBLIB
$LIBR/YM70/SUBLIB
$LIBR/APLSUB/SUBLIB
$LIBR/APLCVEX/SUBLIB
$LIBR/APLSUB/SUBLIB
```

FOO.OPT could then be used as is, or edited to include non-standard object code as full pathnames of either object libraries or individual object modules. The pathnames can contain any combination of literals, wild-carding and environment variables (i.e., whatever you can keep straight). For example,

```
$MYAREA/mymod.o
$myarea/[a-z]*.o
/aippgmr/khilldru/DEBUG/ZSUBLIB
$KCHJUNK/[X-Z]*/ZQ*
$MYLIBS/*.LIB
$LIBR/QNOT/SUBLIB
$LIBR/APLNOT/SUBLIB
$LIBR/QSUB/SUBLIB
$LIBR/QVEX/SUBLIB
$LIBR/YSUB/SUBLIB
$LIBR/YM70/SUBLIB
$LIBR/APLSUB/SUBLIB
$LIBR/APLCVEX/SUBLIB
$LIBR/APLSUB/SUBLIB
```

The contents of the ".OPT" files are be evaluated at link time.

The search process as executed by COMRPL and COMLNK is designed to substitute the most appropriate version and form of the routine specified, regardless of what the user types. The appropriate version is determined by the search path as defined in $SYSLOCAL/LIBR.DAT. Actually, for the sake of speed, the environment variable definitions of $SYSLOCAL/LIBR.DAT are evaluated and stored as pathnames in $SYSLOCAL/SEARCH.DAT and this file is used instead. $SYSLOCAL/SEARCH.DAT is regenerated whenever any of the programming tools which depend on it detect that $SYSLOCAL/LIBR.DAT is newer. Concomitant to the search process for the most appropriate version of a given module for the host implementation (e.g., Unix versus VMS Z-routine) is a search process for the most up to date "form" of the module (e.g., unpreprocessed, preprocessed or object module). This is determined by the most recent modification date of the various extant forms. In the case of Fortran oriented modules, this also includes the modification dates of any included source text (i.e., source text stored in different modules but "included" as part of the preprocessing step).

A.7 A Tutorial for Programmers Using Unix

A.7.1 Initialization And Startup Procedures

LOGIN.CSH or LOGIN.SH

The logical names and symbols needed to program in AIPS (and to run AIPS) can be obtained by executing the script LOGIN.CSH for those whose default login shell is the C shell or LOGIN.SH for those whose default login shell is either the Bourne (System V only) or Korn shell. Very early in the AIPS installation process, the LOGIN.∗ files that come on the installation tape should be moved to the home directory of the login designated as the repository for the AIPS system. Those who want to program in AIPS should add the execution of the appropriate LOGIN.∗ file to their private login procedures. Those programmers whose default login shell is the C shell should add the line

```
source AIPS_account_home_directory/LOGIN.CSH
```

and those programmers whose default login shell is either the Bourne or Korn shell should add the line

```
. AIPS_account_home_directory/LOGIN.CSH
```

substituting the local pathname for the "AIPS_account_home_directory". At NRAO this procedure defaults $AIPS_VERSION to $TST. The versions of the LOGIN.∗ files that come on the installation tape default $AIPS_VERSION to $NEW. The LOGIN.∗ files only define the means by which the AIPS programming "logicals" (i.e., environment variables) can be defined and toggled between the $OLD, $NEW and $TST versions. Unlike VMS, redefining the programming logicals entails redefining all of the individual logicals, not just AIPS_VERSION. Also, since child processes cannot change the environment of their parent, this cannot be done via a procedure. There is the notion of aliases under the C shell and functions under the Bourne and perhaps Korn shells (System V Unix only). However, the only universal solution seems to be the notion of an "executable" environment variable. This is something we have never seen used anywhere else, or even discussed in the Unix literature, but it works. The LOGIN.∗ files define three environment variables named CDOLD, CDNEW and CDTST. These will redefine AIPS_VERSION as $OLD, $NEW or $TST, respectively and execute the commands in $AIPS_VERSION/SYSTEM/UNIX/LOCAL/AREAS.CSH via the "source" command or $AIPS_VERSION/SYSTEM/UNIX/LOCAL/AREAS.SH via the "." command depending on whether LOGIN.CSH or LOGIN.SH was used to define the CD∗ environment variables. To define or redefine the AIPS programming logicals, the user need only type:

```
$CDOLD (or $CDNEW, or $CDTST)
```

This is not required for the execution of AIPS, but is crucial for the AIPS programming tools to work. Programmers may prefer to include the execution of one of $CDOLD, $CDNEW or $CDTST to their login procedure as well. However, their execution will substantially slow down the login process.

AIPS "Version" "Option"

This procedure is used to start up an interactive AIPS session. The following text is taken from the comments found at the beginning of the AIPS start-up procedure as stored in $SYSUNIX:

```
: "-----------------------------------------------------------------"
: " Usage: AIPS [NEW, OLD or TST] [REMOTE] [DEBUG] [LOCAL]          "
: "-----------------------------------------------------------------"
: " Procedure to start up an AIPS session with process name AIPSn,  "
: " then disappear (i.e., exec without fork).                       "
: "                                                                 "
: " Inputs:                                                         "
: "    OLD,                                                         "
: "    NEW or                                                       "
: "    TST    to select version of AIPS to run (default is NEW)     "
: "    REMOTE to indicate a remote terminal i.e., no TV and TEK output "
: "           is directed to the user's terminal (i.e., it better be  "
: "           a Tektronix 4010/4012 compatible terminal if any TK*    "
: "           verbs or tasks are executed)                           "
: "    DEBUG  to run with debugger                                   "
: "    LOCAL  to run a local version of AIPS (assumes AIPS.EXE is in  "
: "           current working directory)                             "
: "                                                                 "
: " Generic Unix version.                                           "
: "-----------------------------------------------------------------"
```

BATER "Version" "Option"

This procedure is used to start up an interactive BATER session. BATER can be used to prepare and submit jobs to the AIPS batch queue. The following text is taken from the comments found at the beginning of the BATER start-up procedure as stored in $SYSUNIX:

```
: "-------------------------------------------------------------------"
: " Usage: BATER [NEW, OLD or TST] [DEBUG] [LOCAL]                    "
: "-------------------------------------------------------------------"
: " Procedure to start up an BATER session with process name BATERn,  "
: " then disappear (i.e., exec without fork).                         "
: "                                                                   "
: " Inputs:                                                           "
: "    OLD,                                                           "
: "    NEW or                                                         "
: "    TST    to select version of BATER to run (default is NEW)      "
: "    DEBUG  to run with debugger                                    "
: "    LOCAL  to run a local version of BATER (assumes BATER.EXE is in "
: "           current working directory)                              "
: "                                                                   "
: " Generic Unix version.                                             "
: "-------------------------------------------------------------------"
```

RUN "Program"

This is a general purpose startup procedure for any of the stand-alone utility programs in AIPS (e.g., SETPAR, RECAT, etc.). This is normally only used by the local AIPS manager(s). The following text is taken from the comments found at the beginning of the RUN procedure as stored in $SYSUNIX:

```
: "-------------------------------------------------------------------"
: " Usage: RUN program                                                "
: "-------------------------------------------------------------------"
: " A script to facilitate the execution of AIPS stand-alone programs  "
: " (e.g., FILAI*, SETPAR, POPSGN, RECAT, SETTVP, etc.).  AIPS and     "
: " BATER sessions should be initiated via the procedures AIPS and     "
: " BATER (what else?).  The version of the program started is         "
: " determined by $AIPS_VERSION as set upon login or by the execution  "
: " of $CDOLD, $CDNEW or $CDTST (or manually, of course).              "
: "                                                                   "
: " Generic Unix version.                                             "
: "-------------------------------------------------------------------"
```

COMRPL

This procedure will preprocess (if necessary) and/or compile (if necessary) subroutines, then stage the resulting object modules for replacement in the proper object module library or libraries (if any). It takes a variety of options which are described below. Arguments to COMRPL can appear in any order and in any combination. At least one subroutine should be specified. However, if it is invoked with no arguments, or otherwise incorrectly, it will display a terse synopsis of its usage. The following text is taken from the comments found at the beginning of the COMRPL procedure as stored in $SYSUNIX:

```
: "-------------------------------------------------------------------"
```

```
: " Usage: COMRPL [directory-path/][@]routine[.FOR,.f,.C,.c,.S,.s,.o]  "
: "               [AIPS-style-options] [Unix-style-options] [file.LOG] "
: "-----------------------------------------------------------------"
: " Drives the preprocessing of, and/or compilation of, and/or library "
: " replacement of AIPS routines.  Any source code generated as the    "
: " result of preprocessing is left in the same directory as the       "
: " un-preprocessed source code.  Object modules that are the result   "
: " of the compilation of source code which resides in a subdirectory  "
: " of $AIPS_VERSION are moved to the proper $LIBR subdirectory as     "
: " defined in $SYSLOCAL/LIBR.DAT (unless NORE[PLACE] is specified).   "
: "                                                                    "
: " Inputs (can appear in any order):                                  "
: "                                                                    "
: "    1) [directory-path/][@]routine[.FOR,.f,.C,.c,.S,.s,.o]          "
: "                                                                    "
: "       At least one (uppercase) routine module name with or without "
: "       an extension.  If not a pathname, the current working        "
: "       directory is assumed and prepended.  Pathnames can be given  "
: "       either literally or using environment variables defined as   "
: "       directory paths (e.g., $APLSUB/[@]routine[.FOR,.f,.C,.c,.S,  "
: "       .s,.o]).  The special character '@', if prepended to the     "
: "       filename, denotes the name of a file containing a list of    "
: "       such routine module pathnames.  If extensions are given with "
: "       simple filenames, (i.e., 'no directory-path/' prefix), it    "
: "       speeds up the command line parsing somewhat.  This is        "
: "       because filename versus AIPS-style option ambiguities are    "
: "       resolved by first testing for AIPS-style option recognition  "
: "       then assuming the argument is a simple filename.  In any     "
: "       case, the extension is effectively ignored since SEARCH      "
: "       strips it and tries to determine the fastest up-to-date      "
: "       module form.  SEARCH will also search "directory-path/" and  "
: "       below for the existence of a routine module more appropriate "
: "       to the host implementation and, if necessary, substitute the "
: "       proper 'directory-path/' and/or filename extension.  In the  "
: "       case where the starting 'directory-path/' is not a           "
: "       subdirectory of $AIPS_VERSION, the search is restricted to   "
: "       that directory.  Otherwise, the directory search path is     "
: "       determined from $SYSLOCAL/SEARCH.DAT.                        "
: "                                                                    "
: "    2) [AIPS-style-options]                                         "
: "                                                                    "
: "       Recognizable AIPS-style options.  These are translated into  "
: "       local syntax based on the definitions in the host specific   "
: "       $SYSLOCAL/*OPTS.SH files invoked by the appropriate compiler "
: "       procedure (i.e., FC, CC, or AS).  Recognized AIPS-style      "
: "       options include:                                             "
: "                                                                    "
: "       (NO)DE[BUG]   - generate code suitable for execution under   "
: "                       host debugger(s).                            "
: "       (NO)DI[RTY]   - compile letting declarations default (not    "
: "                       recommended)                                 "
: "       (NO)LI[ST]    - generate line numbered listing of source     "
: "                       code as part of compilation process (if no   "
: "                       compilation is necessary, no listing is      "
```

```
: "                      generated)                                   "
: "       (NO)MAP       - generate link map                            "
: "       (NO)OPTn      - optimization level (n = 0 to 9)              "
: "       (NO)PR[OFILE] - generate code suitable for profiling         "
: "       (NO)PU[RGE]   - delete preprocessed source code after        "
: "                      successful compilation and delete             "
: "                      automatically generated log files if all     "
: "                      goes well                                     "
: "       (NO)RE[PLACE] - move object module to proper $LIBR           "
: "                      subdirectory (procedure LINK does any         "
: "                      necessary replacements in and randomizations "
: "                      of $LIBR/.../SUBLIB's prior to linking)       "
: "                                                                    "
: "       where                                                        "
: "          (NO)  = alternate form (e.g., NODEBUG is the opposite of  "
: "                  DEBUG)                                            "
: "          [...] = additional letters of option not required but     "
: "                  recognized                                        "
: "                                                                    "
: "    3) [Unix-style-options]                                         "
: "                                                                    "
: "       Unix-style options which are passed on to the local compiler "
: "       involved.                                                    "
: "                                                                    "
: "    4) [file.LOG]                                                   "
: "                                                                    "
: "       Optional log filename of the form '*.LOG'.  If not given,    "
: "       log files are automatically generated (or appended to) for   "
: "       each routine being processed.  If purging is enabled either  "
: "       by default or by specifying PURGE on the command line and    "
: "       all goes well, these automatic log files as well as          "
: "       preprocessed forms of the routine involved are deleted.  If  "
: "       the user specifies a '.LOG' file on the command line, it is  "
: "       either generated or appended to but never deleted.           "
: "                                                                    "
: " Generic Unix version.                                              "
: "--------------------------------------------------------------------"
```

For example, the following command will preprocess (if necessary) the subroutine $APLSUB/CHCOPY.FOR, compile the preprocessed source code using the default compiler options as defined in the corresponding $SYSLOCAL/*OPTS.SH compiler options files, and stage the resulting object module for replacement in the object library appropriate for subroutines from $APLSUB.

```
COMRPL $APLSUB/CHCOPY
```

The following examples show how multiple files can be compiled.

Process the subroutines MSGWRT and NXTFLG:

```
COMRPL $APLSUB/MSGWRT $APLNOT/NXTFLG
```

Process all routines in $APLSUB whose name begins with MP2:

```
COMRPL $APLSUB/MP2*.FOR
```

Process every routine pathname listed in FOO.LIST:

```
COMRPL @FOO.LIST
```

Simply for the purpose of illustration, the next example does everything above, but with the debug compiler option enabled, the replacement option disabled (i.e., object modules will be left in the same directory as the source code) and with a ".LOG" file specified in which all actions are to be recorded (i.e., as well as displaying them on the terminal):

```
COMRPL $APLSUB/MSGWRT DeBug $APLNOT/NXTFLG WIERD.LOG $APLSUB/MP2*.FOR \
@FOO.LIST NORepLACe
```

COMLNK "ProgramSpec" "Option"

This procedure will preprocess (if necessary) and/or compile (if necessary) a program or set of programs and/or link them with an appropriate object library link list. The resulting executable modules are moved to the proper AIPS load libraries (if any). Any necessary replacements of object modules in object libraries are performed prior to any links that include such libraries. Recall that COMRPL does not actually replace object modules in object libraries, it only stages them for replacement. This way, the price of replacements and the subsequent required "randomizations" of object libraries is only paid at link time rather than in each COMRPL. Like COMRPL, COMLNK takes a variety of options which are described below. Arguments to COMLNK can appear in any order and in any combination. At least one program should be specified. However, if it is invoked with no arguments, or otherwise incorrectly, it will display a terse synopsis of its usage. The following text is taken from the comments found at the beginning of the COMRPL procedure as stored in $SYSUNIX:

```
: "-----------------------------------------------------------------"
: " Usage: COMLNK [directory-path/][@]program[.FOR,.f,.C,.c,.S,.s,.o]  "
: "               [AIPS-style-options] [Unix-style-options]           "
: "               [file.OPT] [file.LOG]                               "
: "-----------------------------------------------------------------"
: " Drives the preprocessing of and/or compilation of and/or linking of"
: " AIPS programs.  Object modules that are the result of compilations "
: " are left in the same directory as the source code.  Executable     "
: " modules that are the result of linking modules which all reside in "
: " subdirectories of $AIPS_VERSION are moved to $LOAD (unless         "
: " NORE[PLACE] is specified, in which case, the executable module is  "
: " left in the same directory as the source code).  Otherwise,        "
: " executable modules are moved to or left in the current working     "
: " directory.                                                         "
: "                                                                    "
: " Inputs (can appear in any order):                                  "
: "                                                                    "
: "    1) [directory-path/][@]program[.FOR,.f,.C,.c,.S,.s,.o]          "
: "                                                                    "
: "       At least one (uppercase) program module name with or without "
: "       an extension.  If not a pathname, the current working        "
: "       directory is assumed and prepended.  Pathnames can be given  "
: "       either literally or using environment variables defined as   "
: "       directory paths (e.g., $APLPGM/[@]program[.FOR,.f,.C,.c,.S,  "
```

```
: "       .s,.o]).  The special character '@', if prepended to the     "
: "       filename, denotes the name of a file containing a list of    "
: "       such program module pathnames.  If extensions are given with "
: "       simple filenames, (i.e., 'no directory-path/' prefix), it    "
: "       speeds up the command line parsing somewhat.  This is        "
: "       because filename versus AIPS-style option ambiguities are    "
: "       resolved by first testing for AIPS-style option recognition  "
: "       then assuming the argument is a simple filename.  In any     "
: "       case, the extension is effectively ignored since SEARCH      "
: "       strips it and tries to determine the fastest up-to-date      "
: "       module form.  SEARCH will also search "directory-path/" and  "
: "       below for the existence of a program module more appropriate "
: "       to the host implementation and, if necessary, substitute the "
: "       proper 'directory-path/' and/or filename extension.  In the  "
: "       case where the starting 'directory-path/' is not a           "
: "       subdirectory of $AIPS_VERSION, the search is restricted to   "
: "       that directory.  Otherwise, the directory search path is     "
: "       determined from $SYSLOCAL/SEARCH.DAT.                        "
: "                                                                    "
: "    2) [AIPS-style-options]                                         "
: "                                                                    "
: "       Recognizable AIPS-style options.  These are translated into  "
: "       local syntax based on the definitions in the host specific   "
: "       $SYSLOCAL/*OPTS.SH files invoked by the respective steps     "
: "       (i.e., FC, CC, or AS and LINK).  Recognized AIPS-style       "
: "       options include:                                             "
: "                                                                    "
: "       (NO)DE[BUG]    - generate code suitable for execution under  "
: "                        host debugger(s).                           "
: "       (NO)DI[RTY]    - compile letting declarations default (not   "
: "                        recommended)                                "
: "       (NO)LI[ST]     - generate line numbered listing of source    "
: "                        code as part of compilation process (if no  "
: "                        compilation is necessary, no listing is     "
: "                        generated)                                  "
: "       (NO)MAP        - generate link map                           "
: "       (NO)OPTn       - optimization level (n = 0 to 9)             "
: "       (NO)PR[OFILE] - generate code suitable for profiling         "
: "       (NO)PU[RGE]    - delete preprocessed source code after       "
: "                        successful compilation and delete           "
: "                        automatically generated log files if all    "
: "                        goes well                                   "
: "       (NO)RE[PLACE] - move executable module to $AIPS_VERSION/LOAD "
: "                        if appropriate                              "
: "                                                                    "
: "       where                                                        "
: "          (NO)  = alternate form (e.g., NODEBUG is the opposite of  "
: "                  DEBUG)                                            "
: "          [...] = additional letters of option not required but     "
: "                  recognized                                        "
: "                                                                    "
: "    3) [Unix-style-options]                                         "
: "                                                                    "
: "       Unix-style options which are passed on to the local compiler "
```

```
: "       involved (the compiler is also invoked for the linking step  "
: "       rather than invoking the loader directly).                   "
: "                                                                    "
: "    4) [file.OPT]                                                   "
: "                                                                    "
: "       Semi-optional link list file of the form '*.OPT'.  If not    "
: "       given and the program object module passed to LINK resides   "
: "       in a subdirectory of $AIPS_VERSION, the procedure LINK will  "
: "       try to determine the default link list from the definitions  "
: "       in $SYSLOCAL/LIBR.DAT.  Otherwise, a '*.OPT' file must be    "
: "       specified.  The routine LIBS, given the pathname of an AIPS  "
: "       program area will print out the default link list (e.g.,     "
: "       LIBS $APLPGM will print out the default link list for all    "
: "       $APLPGM programs).  Its output can be redirected to a        "
: "       'file.OPT' to simplify the construction of these files.      "
: "                                                                    "
: "    5) [file.LOG]                                                   "
: "                                                                    "
: "       Optional log filename of the form '*.LOG'.  If not given,    "
: "       log files are automatically generated (or appended to) for   "
: "       each program being processed.  If purging is enabled either  "
: "       by default or by specifying PURGE on the command line and    "
: "       all goes well, these automatic log files as well as          "
: "       preprocessed forms of the program involved are deleted.  If  "
: "       the user specifies a '.LOG' file on the command line, it is  "
: "       either generated or appended to but never deleted.           "
: "                                                                    "
: " Generic Unix version.                                              "
: "--------------------------------------------------------------------"
```

COMTST

Use COMLNK.

Options

The following AIPS-style options can be used with the compile and link procedures:

```
Option      Minimum
          Abbreviation    Comments

DEBUG         DE          Compile or link with debug option enabled
NODEBUG       NODE        Compile or link without debug option enabled
LIST          LI          Produce a line numbered source code listing
NOLIST        NOLI        No line numbered source code listing
MAP           MA          Produce a link map
NOMAP         NOMA        No link map
OPTn          OPTn        Compile with optimization level n = 0 to 9
NOOPTn        NOOPTn      Disable optimization level n = 0 to 9
DIRTY         DI          Let declarations default
NODIRTY       NODI        Treat undeclared items as fatal errors
PURGE         PU          Delete preprocessed source code and
                          auto-logs if all goes well (also program
```

```
                             object module after successful links)
   NOPURGE       NOPU        No deletion

Unix-style options are passed on to the compiler involved.  The local
definitions of the AIPS-style options and the default modes are setup
in ASOPTS.SH (assembler), CCOPTS.SH (C compiler), FCOPTS.SH (Fortran
compiler) and LDOPTS.SH (linker).  These files are stored in _$SYSLOCAL.
```

A.7.2 Miscellaneous Routines

VERSION "Version"

See $CDOLD, $CDNEW and $CDTST under "LOGIN.CSH or LOGIN.SH" above.

FORK

The FORK procedure makes no sense under Unix (use &). The following example shows how to compile and link AIPS as a background process:

```
COMLNK $AIPPGM/AIPS &
```

FLOG

The FLOG procedure makes no sense under Unix. Log files can be specified on the command line. Otherwise they are automatically generated for each module as it is processed. In either case, the user can examine the log files at any time using any number of different Unix commands.

A.7.3 Compiling and linking, an example

This example shows how we can link a private, experimental version of the program MX with private copies of the subroutines GRDAT.FOR and DSKFFT.FOR. We will use the standard version of MX.FOR as found in $QYPGNOT.

First, we change to some work directory and copy the current versions of DSKFFT.FOR and GRDAT.FOR from $APLNOT. Now we make any changes as desired to GRDAT.FOR and DSKFFT.FOR and COMRPL them with the following command:

```
COMRPL DSKFFT GRDAT
```

COMRPL will recognize that DSKFFT and GRDAT reside in the current working directory (which is presumably not an AIPS directory defined in $SYSLOCAL/LIBR.DAT). In this case, COMRPL will go through all its normal actions, but will make no attempt to stage the resulting object modules for replacement in an AIPS object library. Instead, the object modules will be left in the same directory as the source code.

For example, if we executed the COMRPL command line above on the NRAO-CV Convex with $AIPS_VERSION defined as /AIPS/15APR87 and did this from the directory /aippgmr/khilldru where DSKFFT.FOR and GRDAT.FOR had been copied, COMRPL would display the following on the user's terminal:

```
COMRPL    : Date       Fri Feb 13 04:16:53 EST 1987
COMRPL    : Substitute /aippgmr/khilldru/DSKFFT.FOR
COMRPL    : for        /aippgmr/khilldru/DSKFFT
PP        : Preprocess /aippgmr/khilldru/DSKFFT.FOR
PP        : into       /aippgmr/khilldru/DSKFFT.f
```

```
FC        : Date       Fri Feb 13 04:17:26 EST 1987
FC        : Interpret  FC  \
FC        :            /aippgmr/khilldru/DSKFFT.f
FC        : as         LIST=FALSE PURGE=TRUE
FC        : plus       fc -V -c -O0 \
FC        :            /aippgmr/khilldru/DSKFFT.f
CONVEX FPP VERSION V2.2
CONVEX FSKEL VERSION V2.2
CONVEX FC VERSION V2.2
FC        : Compile of /aippgmr/khilldru/DSKFFT.f
FC        : ends successfully.
FC        : Delete     /aippgmr/khilldru/DSKFFT.f
COMRPL    : Module     /aippgmr/khilldru/DSKFFT.o
COMRPL    : not        /AIPS/15APR87/...
COMRPL    : Not replaced!
COMRPL    : Date       Fri Feb 13 04:17:47 EST 1987
COMRPL    : Substitute /aippgmr/khilldru/GRDAT.FOR
COMRPL    : for        /aippgmr/khilldru/GRDAT
PP        : Preprocess /aippgmr/khilldru/GRDAT.FOR
PP        : into       /aippgmr/khilldru/GRDAT.f
FC        : Date       Fri Feb 13 04:18:21 EST 1987
FC        : Interpret  FC  \
FC        :            /aippgmr/khilldru/GRDAT.f
FC        : as         LIST=FALSE PURGE=TRUE
FC        : plus       fc -V -c -O0 \
FC        :            /aippgmr/khilldru/GRDAT.f
CONVEX FPP VERSION V2.2
CONVEX FSKEL VERSION V2.2
CONVEX FC VERSION V2.2
FC        : Compile of /aippgmr/khilldru/GRDAT.f
FC        : ends successfully.
FC        : Delete     /aippgmr/khilldru/GRDAT.f
COMRPL    : Module     /aippgmr/khilldru/GRDAT.o
COMRPL    : not        /AIPS/15APR87/...
COMRPL    : Not replaced!
COMRPL    : Ends successfully
```

As you can see, COMRPL is rather verbose and didactic. It invokes various subordinate procedures to accomplish its mission. The procedure responsible for each action is listed in the left margin. Each of these is designed so that it can be used stand-alone, if so desired. A description of their usage can be found at the beginning of the text of each. Most are stored in $SYSUNIX, but a few are system specific and reside in $SYSLOCAL. However, using COMRPL affords the best protection against foul ups.

Next, we need an option file to tell the linker what object modules and object libraries to use. The name of the options file can be anything that you please, except it must have an extension of ".OPT" (or ".opt"). We can use the procedure LIBS to create an initial version of an options file for programs found in $QYPGNOT (like MX). To do this, we type:

```
LIBS $QYPGNOT > MYMX.OPT
```

This will extract the normal library link list from $SYSLOCAL/LIBR.DAT for programs that reside in $QYPGNOT and store this list in MYMX.OPT. To link our private versions of GRDAT and DSKFFT with $QYPGNOT/MX, we need to use a text editor to change this version of MXMY.OPT from:

```
$LIBR/QNOT/SUBLIB
$LIBR/APLNOT/SUBLIB
$LIBR/QSUB/SUBLIB
$LIBR/QVEX/SUBLIB
$LIBR/YSUB/SUBLIB
$LIBR/YM70/SUBLIB
$LIBR/APLSUB/SUBLIB
$LIBR/APLCVEX/SUBLIB
$LIBR/APLSUB/SUBLIB
```

to:

```
DSKFFT.o
GRDAT.o
$LIBR/QNOT/SUBLIB
$LIBR/APLNOT/SUBLIB
$LIBR/QSUB/SUBLIB
$LIBR/QVEX/SUBLIB
$LIBR/YSUB/SUBLIB
$LIBR/YM70/SUBLIB
$LIBR/APLSUB/SUBLIB
$LIBR/APLCVEX/SUBLIB
$LIBR/APLSUB/SUBLIB
```

With a suitable ".OPT" file prepared, we are ready to create our private version of an MX executable. To do this, we need only type:

```
COMLNK $QYPGNOT/MX.FOR MYMX.OPT
```

For example, if we executed the COMLNK command line above on the NRAO-CV Convex with $AIPS_VERSION defined as /AIPS/15APR87 and did this from the directory /aippgmr/khilldru where our private DSKFFT.o and GRDAT.o reside, COMLNK would display the following on the user's terminal:

```
COMLNK    : Date       Fri Feb 13 05:12:59 EST 1987
COMLNK    : Substitute /AIPS/15APR87/QY/PGM/NOTST/MX.o
COMLNK    : for        /AIPS/15APR87/QY/PGM/NOTST/MX.FOR
LINK      : Date       Fri Feb 13 05:15:10 EST 1987
LINK      : Interpret  LINK MYMX.OPT \
LINK      :            /AIPS/15APR87/QY/PGM/NOTST/MX.o
LINK      : as         PURGE=FALSE REPLACE=TRUE
LINK      : plus       /usr/convex/fc -V -g \
LINK      :            /AIPS/15APR87/QY/PGM/NOTST/MX.o \
LINK      :            DSKFFT.o \
LINK      :            GRDAT.o \
LINK      :            /AIPS/15APR87/LIBR/QNOT/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/APLNOT/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/QSUB/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/QVEX/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/YSUB/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/YM70/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/APLSUB/SUBLIB \
LINK      :            /AIPS/15APR87/LIBR/APLCVEX/SUBLIB \
```

```
LINK      :            /AIPS/15APR87/LIBR/APLSUB/SUBLIB \
LINK      :            -o /AIPS/15APR87/QY/PGM/NOTST/MX.EXE
CONVEX FC VERSION V2.2
LINK      : Moved      /AIPS/15APR87/QY/PGM/NOTST/MX.EXE
LINK      : to         /aippgmr/khilldru/MX.EXE
LINK      : Link of    /AIPS/15APR87/QY/PGM/NOTST/MX.o
LINK      : ends successfully.
COMLNK    : Delete     /AIPS/15APR87/QY/PGM/NOTST/MX.LOG
COMLNK    : Ends successfully
```

Note that, in this case, no preprocessing or compiling was performed. Despite the fact that the command line specified $QYPGNOT/MX.FOR, the search process found an extant version of $QYPGNOT/MX.o (i.e., the MX program object module) which it determined was up to date. It therefore substituted $QYPGNOT/MX.o for $QYPGNOT/MX.FOR and COMLNK dutifully proceeded directly to the link step. If we had known this a priori, we could have instead invoked the procedure LINK via "LINK $QYPGNOT/MX.o MYMX.OPT". However, this can be dangerous since LINK makes no attempt to determine whether the specified object module is up to date. In any case, it has become the preferred practice to leave program object modules around, since it is much faster than preprocessing and compiling the the same source code again. The object modules occupy about the same disk space as a second copy of the unpreprocessed source code and, as long care is taken (e.g., using COMLNK with its search process), the practice is safe. Also note that, whereas the executable module was originally generated in the $QYPGNOT directory, it was ultimately moved to the current working directory.

Suppose we wanted to compile and/or link $QYPGNOT/MX with execution profiling enabled and have the link run as a background process. For this, we type:

```
COMLNK $QYPGNOT/MX PROFILE &
```

In the above, we used the AIPS-style option to enable execution profiling. Alternatively, we could have specified the local compiler option for execution profiling explicitly, for example:

```
COMLNK $QYPGNOT/MX -p &
```

The "-p" would have been passed on to the compiler assuming that it had some meaning. Once this is known, this is the practice that most knowledgeable Unix users will probably adopt. The AIPS-style options are merely preserved for those who don't know any better.

The actions of COMLNK will be displayed on the terminal as well as recorded in a log file whose name defaults to MX.LOG (unless otherwise specified). If all goes well, MX.LOG will be deleted. If not, it will be available for post mortem examination. If we really want to, we can redirect the terminal output from COMLNK to the "bit bucket" by typing:

```
COMLNK $QYPGNOT/MX > /dev/null &
```

In any case, unless we logout and login again, the shell will notify us when any of our background processes finish, successful or not. If we've redirected our COMLNK output to /dev/null, the existence of MX.LOG will also tell us that the COMLNK failed. This is not true if the user specifies a log file on the COMLNK command line. If a log file is specified, COMLNK assumes that the user must want this information for some reason and will leave it around. Furthermore, if the user-specified log file already exists, new text is simply appended.

A.7.4 Non-standard INCLUDE files

The source code preprocessor must naturally have a mechanism for handling included source text in Fortran modules. These are used exclusively in AIPS code to insert variable declarations, COMMON definitions, EQUIVALENCE statements, DATA initialization statements, PARAMETER statements and special compiler directives. Since there is no industry standard for such included text, the AIPS coding practice is to use VMS-style INCLUDE statements. These take the following form:

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

The "INCS:" portion refers to an AIPS programming logical. In Unix, this takes the form of an environment variable defined as a search path. This search path consists of a blank-separated list of directory pathnames. If another directory is to be added to the search path (e.g. /mnt/myname/aips) then define INCS:
% setenv INCS "/mnt/mydir $INCXXX $INCNOT $INC"
Note: you *must* use double quotes in defining $INCS. The actual value of $INCXXX should be depends of the type of computer you are using. Determine the one to use from examining the listing of AREAS.DAT earlier in this chapter. Usually for developing new routines only $INC of the AIPS standard INCLUDE libraries are needed. If $INCS is undefined the preprocessor will set it to a standard value for your installation.

A.7.5 Running Tasks from Private Directories

If a private directory is to be used then the .EXE and the .HLP file should be in the same directory. To use the executables in a private directory during a session with AIPS it is first necessary to define an environment variable to point to this directory, e.g.:
setenv MYVAR /mnt/myname/aips (C shell)
or
MYVAR=/mnt/myname/aips (Bourne or Korn shell)
export MYVAR

Then, inside AIPS set adverb VERSION to point to this directory, e.g:
> VERSION = 'MYDIR'
and INPUTS, HELP and GO will use the right versions of the files.

A.7.6 Debugging under Unix

To run the debugger the task and any relevant routines should have been compiled and link edited with the DEBUG option. Use of the debugger on optimized code can be confusing so is best avoided. If you are using executables in a private directory see the previous section.

DEBUG must be specified on the command line when starting AIPS:
The startup procedure will then ask you which debugger (e.g., dbx, csd, adb) and if you wish to run AIPS itself under the debugger. To cause a task to be run under the debugger in AIPS use pseudo verb SETDEBUG, e.g.:
> setdebug = 20
Using a value of 0 turns off initiating tasks under the debugger. AIPS will not resume until after the task has completed.

A.7.7 Check out system

Programmers at NRAO must use the checkout procedures on CVAX to change AIPS code. Please remember to specify directories using their logical names instead of the full directory names. Otherwise, the automatic procedures for updating other NRAO machines each night will fail.

THIS PAGE DELIBERATELY BLANK