Going AIPS:
A Programmer's Guide to the NRAO Astronomical Image Processing System

Version 15 April 90

VOLUME 1

ABSTRACT

This manual is designed for persons wishing to write programs using the NRAO Astronomical Image Processing System (AIPS). It should be useful for a wide range of applications, from making minor changes in existing programs to writing major new applications routines. All basic aspects of AIPS programming are dealt with in some detail.

AIPS programmers:

Bill Cotton	— Image construction and applications routines
Phil Diamond	— Spectroscopy
Chris Flatters	— Polarization and VLBI
Eric Greisen	— TVs and AIPS
Kerry Hilldrup	— UNIX and Cray COS implementations
Gareth Hunt	— Data formats
Bill Junor	— VLBI
Glen Langston	— Imaging
Nancy Maddalena	— LaTeX conversion

Contents

Chapter 1

Introduction

1.1 Scope

This document is intended for programmers who are familiar with general programming practices and FORTRAN in particular and who are familiar with the common techniques for manipulating astronomical data. This manual is intended to be of use to casual as well as serious programmers wishing to program using the AIPS system. *Going AIPS* is not intended to be an exhaustive description of the functions and subroutines available in AIPS, but rather to illustrate general techniques.

1.2 Hey You, Read This

This manual is designed for a wide variety of users, ranging from those wishing to add 1 line of code to an existing task to the poor soul who has to assume the care and feeding of AIPS in the case all the current AIPS programmers are hit by a truck. While the weight of this manual would tend to bring on attacks of massive depression or homicidal mania in the lighter users from the above mentioned range, it should be noted that, for many purposes, only a small fraction of the material in this manual is necessary in order to program in the AIPS system. The following table suggests courses of action for various situations.

- "I want to get my data into AIPS."

  There are a number of skeleton tasks which make this relatively straightforward — frequently requiring several hours of effort. See the chapter on the skeleton tasks and ignore the rest of this manual unless you run into problems.

- "I just want to do something simple to my data."

  See the chapter on skeleton tasks. There are two tasks, FUDGE and TAFFY, which read uv data or an image, pass the data to a user-provided subroutine and write what comes back into a new file. All of the messy stuff is already taken care of.

- "I have this idea."

  This requires a bit more understanding about how AIPS works. Read the rest of this chapter, the chapter on the skeleton tasks, the chapter on tasks, and the chapter on disk I/O. Depending on the application, several other chapters may be relevant. Then find an existing task that is closest to your need and start from there. For a great many purposes the skeleton tasks are a good place to start. There is also a chapter describing various high-level utility routines such as making images from uv data or subtracting model values from uv data.

- "I have lots of ideas."

  Find a comfortable chair, open a six pack of beer and start reading.

- "We just bought the Whizbang 8000 computer and want to run AIPS on it."
  Read all of this manual especially chapters 10 and 15 and Appendix A, then give us a call.
- "Why didn't you %#&(*&! see that #&*@!% truck."
  Read it all, then write the parts left out. Lots of luck.

1.3 Philosophy

The NRAO Astronomical Image Processing System (AIPS) is designed to give the astronomer an integrated system of flexible tools with which to manipulate a wide variety of astronomical data. To be of maximum benefit to the general astronomical community and to increase the useful lifetime of the software, the AIPS system has gone to great lengths to isolate the effects of the particular computer and installation on which it is run. Needless to say, this portability requirement makes the programmer's life more difficult.

The routines which depend on the host machine or operating system are denoted by using a "Z" as the first character of the name; these are referred to as the "Z routines". No other "standard" routines should depend on the host machine or operating system to work properly. Routines which depend on the particular television display device are denoted with names beginning with a "Y"; these are the "Y routines". Routines which depend on the computing hardware (e.g., array processors, vector processors, or lack thereof) have names beginning with a "Q".

It has been argued that it is not worth the additional effort to isolate the machine dependencies. We are all aware of usable packages that have died because they were strongly tied to a particular computer. VAXes are currently losing their position of dominance in the astronomical computing community and those with a sufficiently long memory will recall that IBM 360s and 370s and CDC Cybers had a similar stranglehold during the 60s and early 70s. By not tying ourselves to a particular computer or even vendor, we have the freedom to buy hardware from the vendor who offers the most cost-effective models. This strategy should allow the AIPS system to last longer than previous systems, so we can spend more time investigating new algorithms and less time patching or recoding old programs every time we change computer.

In addition to isolating machine dependencies, we advocate modular program structure. By this we mean that the main program should be relatively short and should basically call routines each of which has a well defined and limited function. Modular coding is especially important for machines on which most programs must be overlaid (a dying species), but it also makes the code easier to debug, easier to maintain, and very importantly, easier from which to steal pieces. Routines which may be of use in other applications should be coded in as general a form as possible and placed in the appropriate AIPS subroutine library. This may take longer in the short run, but should pay off in the long run.

Another philosophical feature of AIPS is that the programs should run as quickly as possible without making the code too difficult to maintain. This is frequently a matter of judgment, but, in general, tricks and excessive cleverness should be avoided.

Since many of the most expensive AIPS tasks are I/O limited, the AIPS I/O system has been designed for maximum performance. In general, this means that I/O is done in a double-buffered mode, in as large blocks as possible, with fixed logical record size and programs work directly out of the I/O buffers. This makes many of the features of the I/O system, which are normally hidden from the programmer, much more obvious and allows the I/O to run as fast as the computer can manage.

The AIPS philosophy has always been that it should always be possible to determine what has been done to a data set. For this purpose, every permanent cataloged data file has an associated history file in which a permanent record is kept of the processing done to the data in that file. It is the responsibility of the programmer to insure the integrity of the history. In addition to the history files, most communications between the user and AIPS or tasks are logged in a file which can be printed.

1.4 An Overview of the AIPS System

The AIPS system consists of several distinct parts. First and most obvious to users is the program called AIPS. This program, based around the People Oriented Parsing System (POPS), interacts with the user,

performs many of the display functions, does some manipulation of data and initiates other programs which run asynchronously from AIPS. Functions built into AIPS are called verbs, the asynchronous programs are called tasks, and both are controlled by the values of parameters in the POPS processor known as adverbs. A third type of program in the AIPS system is the stand-alone utility program which is mostly of interest to the AIPS system manager.

1.4.1 Tasks

Communication between the AIPS program and the tasks it spawns is fairly limited. When a task is initiated from AIPS, an external file is read which specifies the number and order of adverbs whose values are sent to the task. These values, along with some "hidden" values, are written into a disk (TD) file. AIPS then initiates the requested task and begins looping, waiting for the task to either disappear or put a return code into the TD file. The task reads the TD file and depending on the value of a logical "hidden" adverb (DOWAIT in AIPS and RQUICK in the task) may immediately restart AIPS by returning the return code. The task then does the requested operation and before stopping, sends AIPS the return code if this was not done previously.

AIPS may communicate with a task after it has started running via the task communication (TC) file. A list of adverbs which are to be sent to the task is defined in the inputs file; in addition, other instructions such as "quit" may be sent. The task must read the TC file at relevant points. It is the responsibility of the programmer to check the TC file and take appropriate actions.

Tasks are used for operations which either require much computer memory or CPU time or both, whereas verbs are used for operations which take no longer than a few seconds to finish. Since the tasks run asynchronously from AIPS, the user may do other things while one or more tasks are running. Since there is a minimal interaction between AIPS and tasks, programming tasks is much simpler than programming verbs; AIPS does not need to be modified to install a new task. Tasks may communicate directly with the user.

1.4.2 Verbs

Verbs are the functions built into the AIPS program itself. Many of these involve the display of images and most of the interactive features of the AIPS system. POPS is a programming language itself, and complicated combinations of tasks and verbs may be assembled into POPS procedures. Verbs, but not tasks, may change the value of POPS adverbs.

The AIPS program is very modular and most verbs are implemented via a branch table contained in an external file. Most of the verbs are called from subroutines with names like AU1, AU2, AU5C etc. A table read from an external text file determines the subroutine and a function number for each function. The values of adverbs are contained in a common.

1.4.3 Data Files

Data is kept in files which are cataloged in AIPS. At present we have two kinds of data (more are possible): images and uv data. The internal structure is much like that of a FITS format tape. Associated with each main data file may be up to 20 types of auxiliary information files with up to 255 versions of each type. The basic information about the main data file and the existence of the auxiliary files (called extension files) is kept in a catalog file. Bookkeeping and other information is kept in the first record of most of the extension files. One example of the extension file is the HIstory file in which a record of the processing of the data is automatically logged by the AIPS tasks.

1.4.4 I/O

The AIPS system has three basic types of files and three types of I/O to access them. The main data files which are assumed to contain the bulk of the data are accessed in a double buffered mode with large blocks being transferred. The extension files are read by single buffered transfers of 256 integers. Both types are intrinsically random access; however, in practice the main data file access is sequential, but the extension

file access is frequently random. For the main data file, I/O tasks usually work directly from the I/O buffer. The third type of file is the text file. More details about the I/O routines can be found in the chapter on I/O.

1.5 Style

Since AIPS is a rather large package maintained by numerous people it is important that all of the software be written in a consistent style. The following sections describe the style in which AIPS software is to be written.

1.5.1 Precursor Comments

Precursor comments are the principal form of detailed programmer documentation in the AIPS system. These are comments placed immediately following the PROGRAM, SUBROUTINE, or FUNCTION statement which explain the purpose and methods of the routine, the input and output arguments, any use of variables in commons, and any special coding techniques or limitations in the transportability of the routine. Precursor comments do not need to be verbose, but they must explain most things which a programmer must know about calling the routine. Routines must have acceptable prologue comments before they will be accepted into the AIPS system.

The precursor section should begin with two coded comment lines which give the use of the routine and placing it in one of a number of categories. The first of these is a one line description of the function of the routine; this line begins with a comment delimiter followed by a "!". The second line lists the categories in which the routine fits; this line begins with a comment delimiter followed by a "#". These two lines allow the automatic generation of software documentation.

Following the coded routine description lines is a user agreement notice which is intented to discourage anyone from selling the AIPS software. Precursor comments describing the functionality of a routine should be indented three columns except for the indentation of new paragraphs. Sections describing input or output call arguments or major common variables should be set off by an "Inputs:" or "Outputs:" etc. line also indented three columns. Descriptions of individual variables should be indented 6 columns and consist of three parts: 1) variable name, 2) type and dimensionality and 3) function, units etc. These parts should be arrainged into columns. The declarations of the call arguments should be separated from the declarations of the local variables by a single (mostly blank) comment line.

As a simple example, consider:

```
      SUBROUTINE COPY (N, KFROM, KTO)
C-----------------------------------------------------------------------
C! copies integer words from one array to another
C# Utility
C   This software is the subject of a User agreement and is confidential
C   in nature. It shall not be sold or otherwise made available or
C   disclosed to third parties.
C-----------------------------------------------------------------------
C   COPY transfers N integer words from KFROM to KTO
C   Inputs:
C      N      I        number of words to be copied
C      KFROM  I(N)     input array
C   Outputs:
C      KTO    I(N)     output array
C-----------------------------------------------------------------------
      INTEGER   N, KFROM(*), KTO(*)
C
      INTEGER   I
C-----------------------------------------------------------------------
      IF (N.LE.0) GO TO 999
```

```
            DO 10 I = 1,N
               KTO(I) = KFROM(I)
 10            CONTINUE
C
 999  RETURN
      END
```

1.5.2 Body Comments

"Body" comments are placed at strategic locations throughout the body of the code. They act as sign posts to alert the reader to each logical block of code and also to clarify any difficult portions. Ideal places for body comments are prior to DO loops and IF clauses. Body comments within a routine must all begin in the same column and that column should be near column 41. Body comments (and precursor comments) should be typed in lower case letters. This helps to separate visually the comments from the program text (which must be all in upper case!!!).

1.5.3 Indentation

Another powerful tool to illustrate to the reader the logical structure of a routine is indentation. By indenting statements to indicate that they belong together, one can enhance greatly the readability of one's programs. Each step of indentation shall be three (3) spaces, beginning in column 7. Numbered CONTINUE statements should be employed to enhance the indentation pattern. DO loops and IF clauses should be indented. The final CONTINUE and END IF statements should be indented the same as the bulk of the loop or IF block. ELSE or ELSE IF statements should be indented the same as the corresponding IF statement. As an example, consider:

```
C                                        Multiply by transform matrix
      DO 20 I = 1,3
         VEC(I) = 0.0
         DO 10 J = 1,3
            VEC(I) = VEC(I) + TMATX(I,J)*VECO(J)
 10         CONTINUE
 20      CONTINUE
C                                        Unit vector to polar
C                                        Case at pole
      IF ((X.EQ.0.0) .AND. (Y.EQ.0.0)) THEN
         ALPHA = 0.0
         DELTA = 0.0
      ELSE
         ALPHA = ATAN (X, Y)
         DELTA = SQRT (X*X + Y*Y)
         END IF
      PDIST = ATAN2 (Z, DELTA)
C                                        Swap to increasing order
      IF (A.GE.B) THEN
         C = A
         A = B
         B = C
         END IF
      Z = Z ** (B-A)
```

1.5.4 CONTINUE Statements

All DO loops end with CONTINUE statements rather than some executable statement. This enhances legibility as well as preventing compilation errors on those statements which are not allowed, by some compilers, to be the last statement in a DO loop.

1.5.5 Statement Numbers

The use of GO TO statements is the cause of most logic errors in programming. Use of IF-THEN-ELSE constructions can frequently simplify the logic of a routine. Statement numbers must increase through the routine and should be integer multiples of 5 or 10. They should not exceed 999. Format numbers should have 4 digits with the low order 3 giving the nearest preceding statement number to the first statement using that format. All statement numbers are left justified beginning in column 2.

Statement numbers can help to clarify the logical structure of a routine. Let us consider the common example of a routine which begins with some setup operations (e.g., file opening), then does operation set A or B or C or D, and then does some close down operations (e.g., file closing) before returning. Where possible, such a routine should use statement numbers 5–95 for the setup, 100–195 for set A, 200–295 for set B, 300–395 for set C, 400–495 for set D, and 900–995 for the close down. All FORTRAN routines should end with a RETURN or STOP (main programs only) statment labeled 999.

1.5.6 Blanks

Blank spaces can improve the readability of the routine as can parentheses. Blanks should surround equals signs and separate multiple word statements. Parentheses are a great help in compound logical expressions. For example,

```
A = B
DO 10 I = 1,10
GO TO 999
CALL KPACK (IX, IY)
IF ((A.GT.B) .AND. (C.LE.D)) THEN
```

1.5.7 Modular Code

Modularity in program design is a very important asset for many reasons. Complicated tasks become clearer, to coder and reader alike, when constructed from a logical sequence of smaller operations performed by subroutine call. Such well-ordered tasks are far easier to design, to understand, and to make work correctly than vast monolithic single programs. Furthermore, the small operation subroutines will often turn out to be fairly general and useful to many other tasks as well. Programmers will have to remember that their tasks will have to run not only in the "unlimited" address space of 32-bit virtual computers, but also in the very limited address space of older computers. Although many modern computers have large ($\geq$ Mbyte) memories, these memories are often divided amongst many users. Also many older AIPS computers have limited memories. Therefore, programmers must remember that excessive page faulting is extremely expensive on most virtual memory computers.

1.5.8 Portability

The code of AIPS is intended to achieve a very high degree of portability between computers. The machine independent portion of the AIPS software must strictly conform, after preprocessing, to FORTRAN 77 rules. Vendor extensions to the language are not allowed. Extensions to FORTRAN allowed by the FORTRAN preprocessor are described later in the section on FORTRAN. In particular CHARACTER and numeric data are not to be equivalenced or mixed in a common.

All of the things mentioned in this chapter should be used in moderation. One can bury good code in a plethora of inane comments. One can inundate statements with parentheses or spread them out with blanks until they are no longer legible. Vastly elaborate indentation and numbering schemes can confuse rather

than aid the reader. The creation of large numbers of very short, special purpose subroutines will overburden linkage editors and AIPS's bookkeeping schemes. (In this regard, AIPS already contains a wide range of useful utility subroutines. Programmers should check to see if a function is already available before creating additional subroutines.) Basically, programmers should use good common sense in applying the standards described in this chapter.

1.6 Language

The magnitude of the AIPS project and the desire to achieve portability of the software require a high degree of standardization in the programming language and style. One must code in a language which can be compiled on all machines. One must follow strict rules in statement ordering and location so that simple preprocessors may, when necessary, locate and modify the standard code. Everyone must type code in the same way so that all programmers will be able to read it with as little effort and confusion as possible. All experienced programmers develop a personal typing style which they prefer. To them, the rules given in this chapter may seem arbitrary, capricious, and unworkable. Nonetheless, they are the rules to be followed when coding for the AIPS system. Routines which do not meet these standards will not be accepted. This project is too important and too large to allow compromise at this level. Also, we have found these rules to be fairly comfortable — after we got used to them.

1.6.1 FORTRAN

The programming language will be ANSI standard FORTRAN 77, except for the addition of INCLUDE, LOCAL INCLUDEs, and HOLLERITH declarations and the use of a minimum number of local assembly language (or C) Z routines when absolutely required. The extensions to FORTRAN 77 will be translated by the preprocessor to standard FORTRAN 77. The preprocessor will include the text of INCLUDE files, allowing the definition of "local INCLUDES" in the file and translate HOLLERITH declarations to a numeric data type.

Hollerith data is characters coded into a numeric type variable. The use of Hollerith data is required in some circumstances by the prohibition in FORTRAN of mixing CHARACTER and numeric data in a common or of EQUIVALENCEing them in any way. Since the length of a character string in terms of numeric data types is not defined character type data and numeric data may not be mixed in fixed length records.

Due to these restrictions on the use of CHARACTERs in FORTRAN, AIPS uses data type HOLLERITH in a limited set of circumstances, most notably in file catalog headers which are data structures containing numeric and character data. All translation between HOLLERITH and CHARACTER data types is done in the routines CHR2H and H2CHR which are described in Chapter 3. The only operation allowed for HOLLERITH type data is the assignment to another HOLLERITH variable. HOLLERITH variables must NEVER appear in DATA or WRITE statments.

AIPS FORTRAN requires that all variables be declared. This requirement, when enforced by the compiler, is a valuable tool for finding typos and related bugs.

A review of the entire language is inappropriate here, but programmers are urged to reread a basic reference. (Do not read your local VAX FORTRAN manual. Use a fundamental reference such as IBM's FORTRAN Language manual.) In particular, programmers are reminded that the names of commons, variables, functions, and subroutines must begin with a letter and contain no more than six (6) characters. In AIPS, program names may have no more than five characters because of the need to append the value of NPOPS. Comments are introduced by placing the capital letter C in column 1 of the card. No in-line comments are allowed. Continuation statements are formed by placing a non-blank character in column 6 of the card. In AIPS, this character shall be an asterisk (*). There may be no more than 19 continuations of a single statement. Only card columns 1–72 are used, even in comments. Executable statements at the first level of indentation begin in column 7. TAB characters must not be left in the code after it is typed and edited. The three non-standard statements have the forms:

1. INCLUDE 'INCS:<name> '

where INCLUDE begins in column 7, the first single quote is in column 15, the <name> is a left justified character string. and the second single quote follows <name> with no blanks. The conventions for <name> will be described later. The statement causes the file called <name> to be inserted in the routine in place of the INCLUDE statement. The INCS: indicates the standard include area or search path and should be omitted for "files" given by LOCAL INCLUDEs. Only a single level in INCLUDE is allowed.

2. LOCAL INCLUDE '<name> '

   where LOCAL starts in column 1 tells the preprocessor that the text following, up to the next "LOCAL END" also starting in column 1, is to be included when a "INCLUDE '<name> ' " line is encountered. LOCAL INCLUDEs are normally defined at the beginning of the file containing a task and should only contain text relevant to that task, e.g. defining it's internal commons.

3. HOLLERITH <list>

   where <list> gives the list of variables to be declared as type HOLLERITH. The AIPS usage of HOLLERITH is that 4 characters may be stored in each element.

1.6.2 Statement Order

Statements must be ordered as follows. The PROGRAM, FUNCTION, or SUBROUTINE statement must occupy the first line and must begin in column 7. Then come the precursor comments, the declaration statements, the body of the program, the format statements, and the END statement. Each of these segments will be separated by a comment delimiter line (i.e., C followed by 71 minus signs). The last delimiters are omitted if there are no FORMATs. The last line of the body of the routine must have the statement number 999 and be a STOP (for programs) or RETURN (for functions and subroutines) statement. There must be no other STOP or RETURN statement in the routine.

Many computer systems allow declaration statements to occur in almost any order. However, FORTRAN and some of the simpler compilers do not. Therefore, in AIPS, we will use the following order:

1. Data type and dimension statements: HOLLERITH, INTEGER, LOGICAL, REAL, DOUBLE PRECISION and COMPLEX in any order. We prohibit DIMENSION and data types not allowed by FORTRAN 77 (excluding HOLLERITH). and any use of these statements for data initialization. Note: the use of COMPLEX arithmetic is discouraged as many compliers do not correctly compile statements involving complex arithmetic. PARAMETER statements should be included with (usually before) the declaration statements. Declaration, EQUIVALENCE and COMMON statments may be mixed.

   We prohibit use of the COMMON statement to give the types and dimensions of variables. Use of blank common must be reserved for cases where dynamic memory allocation is needed and the blank common can be changed in size.

2. Data initialization statements: DATA. We prohibit the use of DATA statements to initialize variables in commons (as do the FORTRAN standards and many compilers). The use of octal and hexadecimal numbers in data statements is forbidden.

3. Function definitions.

1.6.3 INCLUDEs

INCLUDE statements are used in AIPS primarily to provide a fixed and uniform set of declarations for commons and data structures. The naming conventions for such INCLUDEs is 'INCS:accc.INC', where INCS: is a logical directory name (which must be dealt with by the preprocessor), 'a' is P, D, or V for PARAMETER INCLUDEs (include files defining PARAMETERS), Declaration/EQUIVALENCE/COMMON includes and includes containing DATA statments. These INCLUDEs must be named in this order.

```
INCLUDE 'INCS:DBWT.INC'
```

causes the text:

```
C                                                          Include DBWT.
      INTEGER   BWTNUM, BWTLUN, BWTIND, BWTREC, BWTDAT(256)
      LOGICAL   WASERR
      CHARACTER BWTNAM*48
      COMMON /BWTCHC/ BWTNAM
      COMMON /BWTCN/ BWTDAT, BWTNUM, BWTLUN, BWTIND, BWTREC, WASERR
C                                                           End DBWT.
```

to be inserted. Note that CHARACTER variables are in a separate common from numeric variables.

1.6.4 Dimension Declaration

The declaration of the dimensionality of arrays should be done as accurately as possible. When arrays are passed as call arguments and the leading dimension is not passed, declare tha array "(*)" and never as "(1)". Arrays which are declared and equivalenced to other variables should be declared as accurately as possible.

1.6.5 SAVE Statements

If the value of a local variable in a subroutine or function is to be preserved between calls, it should be mentioned in a SAVE statement. Some but, not all, compilers do this automatically but it is not required by the definition of FORTRAN.

1.7 Documentation

Proper documentation for both users and programmers is vital to the success of any software system. In the AIPS system, this documentation is primarily the responsibility of the programmer. In the following sections the various categories of AIPS documentation are discussed.

1.7.1 User Documentation

HELP files

The primary source of user documentation is the HELP file. This information is available to the user on-line from the AIPS program. There are several types of help files: (1) task help files, (2) general help files, and (3) adverb help files. The general help files aid the user in finding the name of the task or verbs for a given operation. These entries consist of the name and a one line description of a task or verb. New tasks should be entered into the appropriate general help files. Task help files are the primary user documentation for a task or verb.

There are three parts of the task HELP file separated by a line of 64 -'s. Details about the format of the HELP file are found in the chapter on tasks.

1. INPUTS

   The INPUTS section of the help file is *required* for any task to run. AIPS uses this section to determine the number and order of adverbs to be sent to the task and can check on limits on the values. The INPUTS section also contains a short description of the use of the task and of each of the adverbs. A listing of the INPUTS section of the help file is displayed on the user's terminal showing the current values of the named adverbs when the user types "INPUT" to AIPS. The INPUTS section is also used to specify any adverbs which may be send to the task during its execution through the TC file.

2. HELP

   The HELP section of the help file gives a more detailed description of the function of the task and a more complete description of the meaning of each of the adverbs than the INPUTS section. This section should also explain the default values of the adverbs. The HELP section of the HELP file is listed on the users terminal when the user types "HELP name".

3. EXPLAIN

   The EXPLAIN section of the help file should describe the techniques for properly using the task; hints about reasonable values of the adverbs can be given here. A discussion of the interaction of the given task with other tasks is also appropriate. It is best if someone other than the programmer writes the EXPLAIN section of the help file. The HELP and EXPLAIN sections of the help file are written on the line printer when the user types "EXPLAIN name" to AIPS.

AIPS *Cookbook*

The AIPS *Cookbook* is the main User documentation for AIPS. However, many users are unaware of the existence of any feature in AIPS not advertised in the *Cookbook* and unfortunately, the *Cookbook* only covers the most elementary portions of the AIPS system.

1.7.2 Programmer Documentation

Precursor Comments

The most fundamental source of detailed programmer documentation in the AIPS system are comments in the source code, especially the precursor comments. The precursor comments for all routines should describe the use of the routine as well as the meaning, units, etc., of all call arguments. Many of the detailed descriptions of call sequences in this manual are essentially the precursor comments of the routines.

Shopping Lists

The precursor comments of routines contain one line descriptions of the routines. These are used to generate the shopping lists found in Appendix B.

CHANGE.DOC

Once source code, text files, etc. are entered into the AIPS libraries all changes should be documented in the CHANGE.DOC file. Installations outside of the main AIPS programming group are encouraged to adopt this system. The CHANGE.DOC file contains entries giving the date, name of the routine, and the name of the person making the change, with a short description of the changes. If a bug is being corrected, its symptoms should be described.

The Checkout 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 detailed description of the current check-out routines 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).

Chapter 2

Skeleton Tasks

By far the easiest way to write a new task is to find an old one that does something similar to what is desired and change it. With this thought in mind, we have written tasks whose sole purpose is to be changed into something useful. These tasks take care of most of the bookkeeping chores and make certain limited classes of operations quite simple. The source code for these tasks is heavily commented to aid the user in making the necessary modifications. The names and functions of these tasks are given in the following list.

- FUDGE This task modifies an existing uv data base and writes a new one.
- TAFFY This task modifies an existing image file and writes a new one.
- UVFIL This task creates, catalogs and fills a new uv data file.
- CANDY This task creates, catalogs and fills a new image file.
- PRPLn These tasks (PRPL1, PRPL2, PRPL3) are used to generate plots and are discussed in detail in the chapter on plotting.

Note: for many purposes task FETCH is adequate for reading an image into AIPS without modification. FETCH reads an image from a text file containing a description of the image. See the HELP file for FETCH for details.

Since these tasks contain most of the startup, shutdown, cataloging, etc. chores, they are a good place to start writing a new task. Many of the standard AIPS tasks are cloned from FUDGE or TAFFY. No one in the AIPS programming group has written a task from scratch in years. If the modified version of one of these tasks is to be of more than temporary use, the name of the task should be changed to avoid confusion. This chapter will describe in some detail the structure and use of the skeleton tasks.

2.1 Data Modification Tasks — FUDGE and TAFFY

There are two data modification tasks for the two types of data files, uv data (FUDGE) and images (TAFFY). The basic structure of these two tasks is very similar. The main routine in these tasks is very short and calls routines to do the basic functions:

1. Startup (FUDGIN in FUDGE, TAFIN in TAFFY)
   - initialize commons
   - get adverb values
   - restart AIPS (if DOWAIT is FALSE)
   - find input file in catalog
   - create and catalog output file
2. Process data (SENDUV in FUDGE, SENDMA in TAFFY)

3. write history (FUGHIS in FUDGE, TAFHIS called from OUTMA in TAFFY)
4. Shut down (DIE)
   - unmark catalog file statuses
   - restart AIPS if not done previously

Both FUDGE and TAFFY send one logical record (a visibility record in uv data or a row of an image) at a time to a user supplied subroutine. This subroutine can do some operation on the logical record and return the result. The result is then written to an output file. When all of the data has been processed, a final call is made to the user routine. In this call, the routine can record any entries to be made in the history file. In the history routine, the old history file is copied to the new file and some standard history entries are made. Then any user supplied entries are added. More detailed descriptions of FUDGE and TAFFY can be found in the following sections

2.1.1 FUDGE

FUDGE sends uv data records to a user supplied routine one at a time. The user routine performs some operation on the record and returns the record with a flag which says whether the result is to be kept or ignored. Many operations which require operating on several data records can be done by sorting the data with UVSRT so that records which are to be combined are adjacent in the data file. The structure of visibility records is described in detail in the chapter on disk I/O (Chapter 6).

If the size of the visibility record is unchanged, the only changes needed in FUDGE for most simple operations are in the user supplied routine DIDDLE. If the record size is changed, there must be changes made in FUDGIN so that the output file created has the correct size and catalog header information. SENDUV must also be modified so that it writes correct size records to the output file.

The source code for DIDDLE contains precursor comments explaining the use of the routine; these comments are reproduced below.

```
      SUBROUTINE DIDDLE (NUMVIS, U, V, W, T, IA1, IA2, VIS, RPARM,
     *   INCX, IRET)
C-----------------------------------------------------------------------
C   This is a skeleton version of subroutine DIDDLE which allows the
C   user to modify a UV data base.  Visibilities are sent one at a time
C   and when returned are written on the output file if so specified.
C
C   Up to 10 history entries can be written by using WRITE to
C   record up to 64 characters per entry into array HISCRD.  Format:
C         WRITE (HISCRD(entry #),format) history information
C   The history is written after the last call to DIDDLE.
C
C   Messages can be written to the monitor/logfile by encoding
C   the message (up to 80 char) into array MSGTXT in INCLUDE DMSG.INC
C   and then issuing a call:
C            CALL MSGWRT (priority #)
C
C   If IRET > 0, then the output file will be destroyed iff
C   it was created in the current execution.
C
C   If the size of the vis record is to be changed, appropriate
C   modifications should be made to CATBLK in FUDGIN before the call
C   to UVCREA and LRECO in SENDUV should reflect the correct size of
C   the output record.
C
C   See the precursor comments for UVPGET for a description
```

```
C   of the contents of COMMON /UVHDR/ which allows easy access to
C   much of the information from the catalog header (CATBLK) and
C   which describes the order in which the data is given.
C
C   After all data has been processed a final call will be made to
C   DIDDLE with NUMVIS = -1.  This is to allow for the completion of
C   pending operations, i.e. preparation of HIstory cards.
C
C   LUN's 16 and 17 are open and not available to DIDDLE.
C
C   The current contents of CATBLK will be written back to the
C   catalog after the last call to DIDDLE.
C
C   Inputs:
C      NUMVIS  I     Visibility number, -1 => final call, no data
C                    passed but allows any operations to be completed.
C      U       R     U in wavelengths
C      V       R     V in wavelengths
C      W       R     W in wavelengths
C      T       R     Time in days since 0 IAT on the first day for
C                    which there is data, the julian day corresponding
C                    to this day can be obtained in D   form by:
C                    CALL JULDAY (CATH(KHDOB),XDAY) where XDAY will
C                    be the Julian day number.
C      IA1     I     First antenna number
C      IA2     I     Second antenna number
C      RPARM   R(*)  Random parameter array which includes U,V,W etc
C                    but also any other random parameters.
C      VIS     R(INCX,*)  Visibilities in order real, imaginary, weight
C                    (Jy, Jy, unitless).  Weight <= 0 => flagged.
C                    NOTE: INCX may be any value .GE. 2
C   Inputs from COMMON:
C      NAME2       C*12     Name of the aux. file
C      CLAS2       C*6      Class of the aux. file.
C      SEQ2        I        Sequence number of the aux. file.
C      DISK2       I        Volumn number of the aux. file.
C      APARM(10)   R(10)    User array.
C      BPARM(10)   R(10)    User array.
C      BOX(4,10)   R(4,10)  User array.
C      RA          D        Right ascension (1950) of phase center. (deg)
C      DEC         D        Declination (1950) of phase center. (deg)
C      FREQ        D        Frequency of observation (Hz)
C      NRPARM      I        # random parameters.
C      NCOR        I        # correlators
C      CATBLK      I(256)   Catalog header record.
C
C   Output:
C      U           R     U in wavelengths
C      V           R     V in wavelengths
C      W           R     W in wavelengths
C      T           R     Time in same units as input.
C      RPARM       R     Modified random parameter array.  N.B.  U,V,W,
C                        time, baseline should not be modified in RPARM
C      VIS         R     Visibilities
```

```
C      IRET       I    Return code  -1 => don't write
C                                    0 => OK
C                                   >0 => error, terminate.
C
C   Output in COMMON:
C      NUMHIS     I         # history entries (max. 10)
C      HISCRD     C(NUMHIS) History records
C      CATBLK     I         Catalog header block
C-----------------------------------------------------------------------
```

There are a number of adverbs already included in FUDGE to pass user information to the user routine; these are specifications for a second input file and the arrays CPARM, DPARM and BOX. More or different adverbs are readily added.

FUDGE will automatically compress the output file if the number of visibility records in the file is reduced. The source code for FUDGE can be found in the standard program source area; this is usually assigned the logical name "APLPGM:" whose value is AIPS_VERSION:[APL.PGM] on VMS systems.

2.1.2 TAFFY

TAFFY reads a selected subset (or all) of an image, sends the image one row at a time to a user supplied routine (DIDDLE) which operates on the row. The user routine sends back the result which may be of arbitrary length; in particular the input row may be reduced to a single value. The values sent back from the user supplied routine are written into the new cataloged file. DIDDLE can defer returning the next row; this allows the use of scrolling buffer. TAFFY can handle multi-dimensional, blanked images. The task TRANS may be used before a TAFFY clone to transpose which ever axis is necessary to the first axis. The returned value of a row may be deferred for those cases when a scrolling buffer of the input is needed.

If the size or format of the output file is to be different from the input file, or if it is necessary to check that the proper axis occurs first in the data array, or if there are several possible operations to be specified by the adverb OPCODE, then the routine NEWHED needs to be modified. The main purpose of NEWHED is to form the catalog header record for the output file. For many purposes the only modifications needed to NEWHED are to modify the values in DATA statements from the default values supplied. The beginning portion of NEWHED is reproduced below.

```
      SUBROUTINE NEWHED (IRET)
C-----------------------------------------------------------------------
C   NEWHED is a routine in which the user performs several operations
C   associated with beginning the task.  For many purposes simply
C   changing some of the values in the DATA statments will be all that
C   is necessary.  The following functions are/can be performed
C   in NEWHED:
C       1) Modifying the catalog header block to represent the
C   output file.  The MINIMUM modifications required here are those
C   required to define the size of the output file; ie.
C      CATBLK(KIDIM)   = the number of axes,
C      CATBLK(KINAX+i) = the dimension of each axis, and
C   Other changes can be made either here or in DIDDLE; the
C   catalog block will be updated when the history file is
C   written.
C       2) Checking the input image and/or input parameters.
C   For example, if a given first axis type such as
C   Frequency/Velocity is required this should be checked.  The
C   routine currently does this and all that is required to
C   implement this is to modify the DATA statments.
C   A returned value of IRET .NE. 0 will cause the task to terminate.
```

```
C   A message to the user via MSGWRT about the reason for the
C   termination would be friendly.  This can be done by encoding
C   the message into MSGTXT, setting IRET to a non-zero value
C   and issuing a GO TO 990.
C       3) Default values of some of the input parameters
C   (OUTNAME, OUTCLASS, OUTSEQ, OUTDISK, TRC and BLC defaults are
C   set elsewhere).  As currently set the default OPCODE is the
C   first value in the array CODES which is set in a data statment.
C
C    Input in common:
C       CATBLK    I(256)  Output catalog header, also CATR, CATD
C       CATOLD    I(256)  Input catalog header, also OLD4, OLD8
C    Output:
C       CATBLK    I(256)  Modified output catalog header.
C       IRET      I       Return error code, 0=>OK, otherwise abort.
C-----------------------------------------------------------------------
      INTEGER   IRET
C
      CHARACTER ATYPES(10)*8, FCHARS(3)*4, BLANK*8, CODES(10)*4,
     *   UNITS(10)*8, CTEMP*8
      HOLLERITH OLD4(256)
      DOUBLE PRECISION    OLD8(128)
      INTEGER   NCODE, NTYPES, IOFF, IERR, INDXI, INC, INDEX,
     *   NCHTYP(10), LIMIT, I, FIRSTI, FIRSTO
      LOGICAL   LDROP1
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'TAFFY.INC'
      INCLUDE 'INCS:DCAT.INC'
      EQUIVALENCE (CATOLD, OLD4, OLD8)
      DATA FCHARS /'FREQ','VELO','FELO'/
      DATA BLANK /'        '/
C                                     User definable values
C                                     # and value of OPCODEs
      DATA NCODE /0/
      DATA CODES /10*'    '/
C                                     Output units for each OPCODE.
      DATA UNITS /'UNDEFINE',9*'    '/
C                                     Allowed number of axis types
C                                     and types.
      DATA NTYPES /0/
      DATA ATYPES /10*'        '/
      DATA NCHTYP /10*4/
C                                     If LDROP1 is .TRUE. then the
C                                     first axis will be dropped,
C                                     (ie, one value results from
C                                     the operation on each row.)
      DATA LDROP1 /.FALSE./
```

The data modification routine in TAFFY is DIDDLE which contains numerous precursor comments describing its use; these precursor comments follow.

```
      SUBROUTINE DIDDLE (IPOS, DATA, RESULT, IRET)
```

```
C-----------------------------------------------------------------------
C   This is a skeleton version of subroutine DIDDLE which allows
C   operations on an image one row at a time (1st dimension).
C   Input and output data may be blanked. The calling routine keeps
C   track of max., min. and the occurence of blanking.  If DROP1 is
C   .TRUE., the calling routine expects 1 value returned per call;
C   otherwise, CATBLK(KINAX) values per call are expected returned.
C   NOTE: blanked values are denoted by the value of the common variable
C   FBLANK.
C       DIDDLE may accumulate a scrolling buffer by returning a negative
C   value of IRET.  This tells the calling routine to defer writing the
C   next row.  If rows are deferred then an equal number of calls to
C   DIDDLE will be made with no input data; this allows reading out any
C   rows left in DIDDLEs internal buffers.  Such a "no input call" is
C   indicated by a value of IPOS(1) of -1.  The writing of the returned
C   values of these "no input calls" may NOT be deferred.
C       Up to 10 history entries can be written  to
C   record up to 64 characters per entry into array HISCRD. Ex:
C         WRITE (HISCRD(entry #), format) list
C   TRC, BLC and OPCODE are already taken care of.
C   The history is written after the last call to DIDDLE.
C       Messages can be written to the monitor/logfile by encoding
C   the message (up to 80 char) into array MSGTXT in COMMON /MSGCOM/
C   and then issuing a call:
C        CALL MSGWRT (priority #)
C
C       If IRET .GT. 0 then the output file will be destroyed.
C
C       After all data have been processed a final call will be made to
C   DIDDLE with IPOS(1)=-2.  This is to allow for the completion of
C   pending operations, i.e. preparation of HIstory cards.
C
C       AIPS LUN's 16-18 are open and not available to DIDDLE.
C
C       The current contents of CATBLK will be written back to the
C   catalog after the last call to DIDDLE.
C
C   Inputs:
C      IPOS   I(7)    BLC (input image) of first value in DATA
C                     IPOS(1) = -1 => no input data this call.
C                     IPOS(2) = -2 => last call (no input data).
C      DATA   R(*)    Input row, magic value blanked.
C   Values from commons:
C      ICODE   I      Opcode number from list in NEWHED.
C      FBLANK  R      Value of blanked pixel.
C      CPARM   R(10)  Input adverb array.
C      DPARM   R(10)  Input adverb array.
C      CATBLK  I      Output catalog header (also CATR, CATD)
C      CATOLD  I      Input catalog header (also OLD4, OLD8)
C      DROP1   L      True if one output value per call.
C   Output:
C      RESULT  R(*)   Output row.
C      IRET    I      Return code   0 => OK
C                                  >0 => error, terminate.
```

```
C   Output in COMMON:
C     NUMHIS  I          # history entries (max. 10)
C     HISCRD  C(NUMHIS)  History records
C     CATBLK  I          Catalog header block
C-----------------------------------------------------------------------
```

In addition to the adverb OPCODE to specify the desired operation and the adverbs BLC and TRC to specify the window in the input map, there are several user defined adverbs sent to TAFFY. These are the arrays CPARM and DPARM; more and/or other adverbs can be added.

More details about TAFFY can be found in the comments in the source version of the program. The source code for TAFFY can be found in the standard program source area; this is usually assigned the logical name "APLPGM:" whose value is AIPS_VERSION:[APL.PGM] on VMS systems.

2.2 Data Entry Tasks (UVFIL and CANDY)

There is a pair of skeleton tasks for entering data into AIPS, UVFIL for uv data and CANDY for images. These tasks are used to enter either observational or model data into the AIPS system. CANDY especially has been used a number of times and usually takes a couple of hours to produce a working program. (Use of task FETCH is useful in many cases for entering an image into AIPS).

These tasks each have two subroutines which may need to be supplied or modified. The first routine is the one to create the new header record and, for UVFIL, to enter information about the antennas. Most of the modifications required are changes to DATA statements from the supplied default values. The beginning portion of these routines will be given with the detailed descriptions of UVFIL and CANDY. Details about the catalog header record are given in the chapter on catalogs.

The second routine, to be supplied by the user, generates the data to be written to the output file. This may be done by reading an external disk or tape file or by any other means.

The basic structure of UVFIL and CANDY are very similar. The main routine in these tasks is very short and calls routines to do the basic functions:

1. Startup (UVFILN in UVFIL, CANIN in CANDY)
   - initialize commons
   - get adverb values
   - restart AIPS (If DOWAIT is FALSE)
2. Create new catalog header record (NEWHED)
   - create and catalog output file
   - Enter antenna information (In UVFIL only)
3. Read/generate data (FIDDLE in UVFIL, MAKMAP in CANDY)
4. Write history (and antenna file) (FILHIS in UVFIL, CANHIS in CANDY)
5. Shut down (DIE)
   - Unmark catalog file statuses
   - Restart AIPS if not done previously

2.2.1 UVFIL

UVFIL creates, catalogs and fills an AIPS uv data file. It can be used either to translate uv data from another format or generate model data.

UVFIL comes with specific example code reading a file. The first routine, NEWHED, which the user may need to modify is used to enter information required to create the catalog header block and to enter information about the antennas. The beginning portion of this routine follows:

```
      SUBROUTINE NEWHED (IRET)
C---------------------------------------------------------------------
C   NEWHED is a routine in which the catalog header is constructed.
C   Necessary values can be read in in the areas markes "USER CODE
C   GOES HERE".
C
C   NOTE: the AIPS convention for the coordinate reference value
C   for the STOKES axis is that 1,2,3,4 represent I, Q, U, V
C   stokes' parameters and -1,-2,-3,-4 represent RR, LL, RL and
C   LR correlator values.  Currently set for R and L polarization
C   ie Ref. value = -1 and increment = -1.
C
C   The MINIMUM information required here is that
C   required to define the size of the output file; ie.
C      CATBLK(KIGCN) = Number of visibility records
C      CATBLK(KIPCN) = Number of random parameters.
C      CATBLK(KIDIM) = Number of axes,
C      CATBLK(KINAX+i) = the dimension of each axis.
C   Other changes can be made either here or in FIDDLE; the
C   catalog block will be updated when the history file is
C   written.
C      The antenna information can also be entered in this
C   routine.  It is possible to put much more information in the
C   ANtenna file.
C
C   Input in common:
C      CATBLK(256)    I     Output catalog header, also CATR, CATH, CATD
C                           The OUTNAME, OUTCLASS, OUTSEQ are entered
C                           elsewhere.
C   Output in common:
C      CATBLK(256)    I     Modified output catalog header.
C      IRET           I     Return error code, 0=>OK, otherwise abort.
C    Also the antenna information can be filled into a common.
C---------------------------------------------------------------------
      INTEGER   IRET
C
      CHARACTER RTYPES(7)*8, TYPES(7)*8, UNITS*8, TELE*8, OBSR*8,
     *   INSTR*8, OBSDAT*8, LINE*80
      INTEGER   I, NAXIS, NRAN, NCHAN, NPOLN, NDIM(7), INDEX, XCOUNT,
     *   LUN, FIND
      LOGICAL   APPEND
      REAL      CRINC(7), CRPIX(7), EPOCH, BANDW
      DOUBLE PRECISION CRVAL(7)
      INCLUDE 'UVFIL.INC'
      INCLUDE 'INCS:DCAT.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
```

```
      INCLUDE 'INCS:DUVH.INC'
C                                       User definable values
C                                       Random parameters.
C                                         No. random parameters.
      DATA NRAN /5/
C                                         Rand. parm. names.
      DATA RTYPES /'UU-L-SIN','VV-L-SIN','WW-L-SIN',
     *   'TIME1   ','BASELINE',2*'        '/
C                                       Uniform axes.
C                                         No. axes.
      DATA NAXIS /5/
C                                         Axes names.
      DATA TYPES /'COMPLEX ','STOKES  ','FREQ    ',
     *   'RA      ','DEC     ',2*'        '/
C                                         Axis dimensions
      DATA NDIM /3,1,1,1,1,0,0/
C                                         Reference values
      DATA CRVAL /1.0D0, -1.0D0, 5*0.0D0/
C                                         Reference pixel.
      DATA CRPIX /7*1.0/
C                                         Coordinate increment.
      DATA CRINC /1.0, -1.0, 0.0, 0.0, 0.0, 2*0.0/
C                                       Epoch of position.
      DATA EPOCH /1950.0/
C                                       Units
      DATA UNITS /'JY      '/
C-----------------------------------------------------------------------
```

The user supplied routine FIDDLE returns visibility records which are written into the cataloged output file. The precursor comments describing the use of FIDDLE follow.

```
      SUBROUTINE FIDDLE (NUMVIS, U, V, W, T, IA1, IA2, VIS, RPARM, IRET)
C-----------------------------------------------------------------------
C  This is a skeleton version of subroutine FIDDLE which allows the
C  user to create a UV data base.  Visibilities are returned one at
C  a time and are written on the output file.
C
C       Up to 10 history entries can be written by using WRITE to
C  record up to 64 characters per entry into array HISCRD. Ex:
C     WRITE (HISCRD(entry #),format #) list
C  The history is written after the last call to FIDDLE.
C
C       Messages can be written to the monitor/logfile by writing
C  the message (up to 80 char) into array MSGTXT in INCLUDE DMSG.INC
C  and then issuing a call:
C     CALL MSGWRT (priority #)
C
C       If IRET .GT. 0 then the output file will be destroyed.
C   A value of IRET .lt. 0 indicates the end of the data.
C
C       See the precursor comments for UVPGET for a description
C  of the contents of COMMON /UVHDR/ which allows easy access to
C  much of the information from the catalog header (CATBLK) and
C  which describes the order in which the data is being written.
```

```
C
C       After all data has been processed a final call will be made to
C  FIDDLE with NUMVIS = -1.  This is to allow for the completion of
C  pending operations, i.e. preparation of HIstory cards.
C
C     AIPS I/O  LUN 16 is open and not available to FIDDLE.
C  FORTRAN unit numbers greater than 50 will probably not get the
C  AIPS routines confused.  (Any unit numbers other than 1, 5, 6 and 12
C  will probably also work.)
C
C       The current contents of CATBLK will be written back to the
C  catalog after the last call to FIDDLE.
C
C  Inputs:
C     NUMVIS      I     Visibility number, -1 => final call, no data
C                       passed but allows any operations to be completed.
C
C  Inputs from COMMON:
C     IN2FIL  C*48  Name of the aux. file
C     APARM   R(10) User array.
C     BPARM   R(10) User array.
C     RA          D  Right ascension (1950) of phase center. (deg)
C     DEC         D  Declination (1950) of phase center. (deg)
C     FREQ        D  Frequency of observation (Hz)
C     NRPARM      I  # random parameters.
C     NCOR        I  # correlators
C     CATBLK(256)I  Catalog header record. See Going AIPS for details.
C
C  Output:
C     U       R       U in wavelengths at the reference frequency.
C     V       R       V in wavelengths
C     W       R       W in wavelengths
C     T       R       Time in days since the midnight at the start of
C                     the reference date.
C     IA1     I       Antenna number of the first antenna.
C     IA2     I       Antenna number of the second antenna.
C                     NOTE: IA2 MUST be greater that IA1
C     RPARM   R       Modified random parameter array. NB U,V,W,
C                     time and baseline should not be modified in RPARM
C     VIS     R(3,*)  Visibilities.  The first dimension is the COMPLEX
C                     axis in the order Real part, Imaginary part,
C                     weight. The order of the following visibilities is
C                     defined by variables in COMMON /UVHDR/ (originally
C                     specified in NEWHED).  The order number for Stokes
C                     parameters is JLOCS and the order number for
C                     frequency is given by JLOCF.  The lower order
C                     number  increases faster in the array.
C                     See precursor comments in UVPGET for more details.
C     IRET    I       Return code  -1 => End of data.
C                             0 => OK
C                            >0 => error, terminate.
C
C  Output in COMMON:
C     NUMHIS   I          # history entries (max. 10)
```

```
C     HISCRD   C(NUMHIS)  History records
C     CATBLK   I          Catalog header block
C---------------------------------------------------------------------
```

The user defined array adverbs APARM and BPARM are sent to UVFIL; more and/or other adverbs can easily be added. The source code for UVFIL can be found in the non-standard program source area; this is usually assigned the logical name "APGNOT:" whose value is AIPS_VERSION:[APL.PGM.NOTST] on VMS machines.

2.2.2 CANDY

CANDY is similar to TAFFY except there is no AIPS input data file. This is a good routine to use to generate an AIPS image from either a model or an external data file. CANDY has example code (mostly commented out) in the text which gives an example of reading a formatted disk file. (Note this function is also done in a general way in routine FETCH).

The routine in CANDY in which the values necessary for the catalog header must be entered is named NEWHED. The beginning, heavily commented, portion of NEWHED follows.

```
      SUBROUTINE NEWHED (IRET)
C---------------------------------------------------------------------
C   NEWHED is a routine in which the user performs several operations
C   associated with beginning the task.  For many purposes simply
C   changing some of the values in the DATA statments will be all that
C   is necessary.  The following functions are/can be preformed
C   in NEWHED:
C       1) Creating the catalog header block to represent the
C   output file.  The MINIMUM information required here is that
C   required to define the size of the output file; ie.
C      CATBLK(KIDIM)= the number of axes,
C      CATBLK(KINAX+i) = the dimension of each axis.
C   Other changes can be made either here or in MAKMAP; the
C   catalog block will be updated when the history file is
C   written.
C       2) Setting default values of some of the input parameters
C   As currently set the default OPCODE is the first value in the
C   array CODES which is set in a data statment.
C
C    Input:
C     CATBLK    I(256)  Output catalog header, also CATR, CATD
C                       The OUTNAME, OUTCLASS, OUTSEQ are entered
C                       elsewhere.
C    Output:
C     CATBLK    I(256)  Modified output catalog header.
C     IRET      I       Return error code, 0=>OK, otherwise abort.
C---------------------------------------------------------------------
      INTEGER   IRET
C
      CHARACTER FCHARS(3)*4, BLANK*8, CODES(10)*4, UNITS(10)*8,
     *   ATYPES(7)*8, LINE*80
      INTEGER   I, MAXIS, IROUND, NCODE, IERR, NX, NY, INDEX
      INCLUDE 'CANDY.INC'
      INCLUDE 'INCS:DCAT.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DMSG.INC'
```

```
      INCLUDE 'INCS:DHDR.INC'
      DATA FCHARS /'FREQ','VELO','FELO'/
      DATA  BLANK /'        '/
C                                     User definable values
C                                     # and value of OPCODEs
      DATA NCODE /0/
      DATA CODES /10*'    '/
C                                     Output units for each OPCODE.
      DATA UNITS /'UNDEFINE',9*'        '/
C                                     Number of axes and types.
C                                     (Set for two axes = Ra, Dec.)
      DATA NAXIS  /2/
      DATA ATYPES /'RA---SIN', 'DEC--SIN',
     *   'STOKES  ', 'FREQ    ', 3*'        '/
C-----------------------------------------------------------------------
```

The user supplied routine that reads or generates the image is MAKMAP. This routine returns the image one row at a time. The precursor comments describing the use of this routine follow.

```
      SUBROUTINE MAKMAP (IPOS, RESULT, IRET)
C-----------------------------------------------------------------------
C  This is a skeleton version of subroutine MAKMAP which allows
C  the user to create an image, one row at a time.
C  Output values may  be blanked.
C  The calling routine keeps of max., min. and to occurence of blanking.
C  CATBLK(KINAX) values per call are expected returned.
C  NOTE: blanked values are denoted by the value of the common variable
C  FBLANK
C
C       Up to 10 history entries can be written by using WRITE to
C  record up to 64 characters per entry into array HISCRD. Ex:
C     WRITE (HISCRD(entry #),format #,) list
C  TRC, BLC and OPCODE are already taken care of.
C  The history is written after the last call to MAKMAP.
C
C       Messages can be written to the monitor/logfile by writing
C  the message (up to 80 char) into array MSGTXT in COMMON /MSGCOM/
C  and then issuing a call:
C     CALL MSGWRT (priority #)
C
C       If IRET .GT. 0 then the output file will be destroyed.
C
C       After all data has been processed a final call will be made to
C  MAKMAP with IPOS(1)=-1.  This is to allow for the completion of
C  pending operations, i.e. preparation of HIstory cards.
C
C       LUN's 16-18 are open and not available to MAKMAP.
C
C       The current contents of CATBLK will be written back to the
C  catalog after the last call to MAKMAP.
C
C  Inputs:
C   IPOS   I(7)    BLC (input image) of first value in DATA
C  Values from commons:
```

```
C   ICODE   I       Opcode number from list in NEWHED.
C   FBLANK  R       Value of blanked pixel.
C   CPARM   R(10)   Input adverb array.
C   DPARM   R(10)   Input adverb array.
C   CATBLK  I(256) Output catalog header (also CATR, CATD)
C  Output:
C   RESULT   R(*)   Output row.
C   IRET     I      Return code   0 => OK
C                                 >0 => error, terminate.
C  Output in COMMON:
C  NUMHIS  I           # history entries (max. 10)
C  HISCRD  C(NUMHIS)   History records
C  CATBLK  I           Catalog header block
C-----------------------------------------------------------------------
```

Pixel blanking is supported through magic value blanking, i.e., the value of FBLANK is recognized to mean no value is associated with the pixel. The source code for CANDY is fairly heavily commented and can be found in the non-standard program source area; this is usually assigned the logical name "APGNOT:" whose value is AIPS_VERSION:[APL.PGM.NOTST] on VMS systems.

2.3 Modifying a Skeleton Task

To make a modified version of one of the skeleton tasks, first copy the source code and the help file to the area in which you intend to work on the task. Then rename the task to avoid confusion (only five characters are allowed in an AIPS task name). In addition to changing the name of the files, it is crucial to change the name of the task entered in a DATA statement in the main program. You should also change the task name referenced in the help file. (If there is a chance that your new task will become part of the standard AIPS package, and we welcome contributions, rename the names of the subroutines as well.)

The next step is to modify the source code to taste. If the adverbs which the task uses are changed, the help file should also be changed to reflect this. If the task is to be of more than temporary use, then it is friendly to put sufficient documentation into the help file to assist other users in understanding the use of the input adverbs; besides, you will also forget just what it is that BPARM(3) does. Once the source code is modified, see Appendix A for details about compiling, linking and debugging a task

Chapter 3

Getting Started — Tasks

3.1 Overview

This chapter will describe both the general structure of AIPS tasks and the operations which are needed for the smooth startup and shutdown of most tasks. Following chapters will describe in detail other aspects of AIPS tasks. The principal steps of a "typical" task are illustrated in the following. The names of relevant AIPS utility subroutines are given in parentheses.

1. Startup
    - initialize commons (ZDCHIN, VHDRIN etc.)
    - get adverb values (GTPARM)
    - restart AIPS (RELPOP)
2. Setup data files
    - find input file in catalog (MAPOPN, CATDIR, CATIO)
    - create and catalog output file (MCREAT, UVCREA)
    - create scratch files (SCREAT)
3. Process data
    - Check task communication (TC) file for any further instructions (GTTELL)
4. Write history (HISCOP, HIADD, HICLOS)
5. Shut down (DIETSK, DIE)
    - destroy scratch files
    - unmark catalog file statuses
    - restart AIPS if not done previously

The programmer specifies the adverbs to be used for a task in the first section of the help file. The AIPS user specifies the values of the adverbs used to control a task and AIPS writes these values into a disk file (TD). The task must read these values from the TD file. After AIPS has started up a task, it suspends itself until either, (1) the task returns a return code in the TD file, or (2) the task disappears. It is the responsibility of the task to restart AIPS. This is usually done either at the beginning or at the end of the task, depending on the value of the adverb DOWAIT (usually called RQUICK in tasks).

After a task has started, the user may send further instructions — mainly changed adverb values or instructions to quit. This communication is through the task communication (TC) file; the task reads this file using the routine GTTELL. The adverb values to be sent to the task are indicated in the INPUTS section of the help file.

AIPS tasks use commons extensively to keep various system and control information. Since many of these commons are in many hundreds of routines, their declarations are kept in INCLUDE files. This allows relatively simple system-wide changes in these basic commons.

Most of the details of the installation on which a task is running is kept in a disk text file. These details include, how many tape drives, how many disk drives, etc. The parameters characterizing the system are kept in a common which must be initialized by a call to the routine ZDCHIN. Several other commons may be used in a given task, and many of these need to be initialized at the beginning of the program.

There is an accounting file which keeps track of various bookkeeping details of tasks. Calls to the accounting routines are hidden from the programmer of the standard startup and shutdown routines.

Data in the AIPS system are kept in cataloged disk files. Information about the main data file is kept in a catalog header record and only data values are kept in the main data file. Auxiliary data may be kept in one or more "extension" files associated with a cataloged file. Many AIPS tasks modify a data file and write the results into a new cataloged file, although the user is frequently allowed to specify the input file as the output file.

Each cataloged AIPS data file should have an associated HIstory extension file in which as complete as possible a record of the processing is kept. It is the responsibility of the programmer of a task to copy old history files to a new file, if necessary, and to update the history information. In general, the values of the adverbs after defaults have been filled in are kept in the history file. There are usually other extension files which should also be copied if a new output file is being generated. These include ANtenna files for UV data and CLEAN components (CC) files for images. These may be convienently copied using routine ALLTAB.

Most communication between the user and AIPS or tasks is done through a single routine (MSGWRT) which logs most of the communications in a disk file which can be printed. A major difference between the message file and history files is that history files are permanent, whereas message files are not. User interaction with a task is allowed; see the section below on communicating with the user via ZTTYIO.

The simplest way to write a program is to find a program that is close to the one desired and make the necessary changes. In this spirit, there are two tasks available which read data, send it to a routine, and write the result back to a new cataloged disk file. Two others will create and catalog a new disk file and fill it with data generated in a subroutine. These routines (FUDGE, CANDY, TAFFY, and UVFIL) allow the simplest access to the AIPS data files, and even for fairly complicated tasks, one of these programs is a good place to start (a great many AIPS uv tasks were cloned from FUDGE). The chapter on skeleton tasks describes these tasks in more detail. Three skeleton tasks for plotting (PFPL1, PFPL2, and PFPL3) are described in the plotting chapter.

3.2 The Cost of Machine Independence

There are a number of general programming aspects which are seriously affected by the requirement of machine independence. Most of these problems are alleviated by strict adherence to the standards of Fortran 77. The most serious problem is due to inadequate definition of CHARACTER variables in the Fortran 77 standards; this issue is discussed below. When the specifics of the machine/OS on which the software is running MUST be taken into account this dependency must be isolated into an explicitly machine dependent routine ("Z", "Y" or "Q" routines).

3.2.1 Character Strings

The definition of CHARACTER type variables in Fortran 77 does not explicitly give the relationship of the size of a given CHARACTER variable to that of numeric variables. The result of this is that CHARACTER and numeric data cannot be EQUIVALENCEd in any way or mixed in binary records of known length. For this reason there are two types of variables in AIPS which contain character information. These are: CHARACTER and HOLLERITH. CHARACTER variables are the standard Fortran 77 data type and are used in AIPS wherever possible. In some circumstances character information cannot be stored in CHARACTER variables and in these cases the data is declared type HOLLERITH which the preprocessor redeclares as a numeric data type. AIPS HOLLERITH variables are defined to contain 4 characters per element.

HOLLERITH data in AIPS is never to be initialized using DATA statements and is never to be used in READ or WRITE statements. All conversion between HOLLERITH and CHARACTER type variables is through the routines H2CHR and CHR2H. This allows the use of data structures such as the AIPS catalog header without violation of the Fortran 77 rules. The cases in which HOLLERITH data is used is summarized in the following:

1. any file containing mixed numeric and character data in binary form,
2. the I/O buffer used to read or write a file with mixed numeric and character data in binary form,
3. character data in the POPS processor,
4. AIPS string adverb values passed via GTPARM,
5. the catalog header records (CATBLK),
6. any other data structures containing mixed numeric and character data.

There are a number of AIPS utility routines for dealing with CHARACTER and HOLLERITH strings. These are briefly described in the following and are described in detail at the end of this chapter.

- CHCOPY moves characters from one HOLLERITH string to another
- CHCOMP compares two HOLLERITH strings
- CHFILL fills portion of HOLLERITH string with a specified character
- CHLTOU converts a CHARACTER string to all upper case letters
- CHMATC searches one HOLLERITH string for the occurrence of another
- CHR2H converts a Fortran CHARACTER variable to an AIPS HOLLERITH string
- CHWMAT matches a pattern string containing "wild-card" characters with a test string. The wild-cards "*" for any number and "?" for exactly one of any character are supported.
- H2CHR convert AIPS Hollerith string to Fortran CHARACTER variable

3.3 Task Name Conventions

The number of characters allowed in task names is limited in many operating systems to six characters. AIPS uses the last character of the name to indicate the AIPS number of the initiating process, in hexadecimal, leaving five characters for a task name. It is most helpful to the bewildered user looking through the mass of AIPS tasks if the name is at least vaguely mnemonic. For example, most tasks whose principal output is to the line printer are named 'PRT..'; many tasks manipulating uv data are named 'UV...' etc.

3.4 Getting the Parameters

3.4.1 In AIPS (Help file)

The adverbs to be used by a task are defined by the programmer in the beginning portion of the help file. This portion of the HELP file lists the adverbs in order, can give limits on the range of acceptable values, and gives a short description of the use of the adverb. If the limit fields for an adverb are left blank, then no limits are enforced. When AIPS receives the GO command, it reads the associated help file for the list of adverbs and places the current values of these adverbs as well as a few "hidden" adverbs into the task data (TD) file. Entries with a "?" in column 10 are ignored by GO. AIPS then starts the requested task. An example, the help file for PRTIM follows:

```
; PRTIM
;---------------------------------------------------------------
;! Task displays a map on line-printer or terminal
;# TASK PRINTER
;  This software is the subject of a User agreement and is
;  confidential in nature. It shall not be sold or otherwise
;  made available or disclosed to third parties.
;---------------------------------------------------------------
PRTIM     LLLLLLLLLLLLUUUUUUUUUUUU CCCCCCCCCCCCCCCCCCCCCCCCCCCC
PRTIM:  Task to print the image intensities in digital form
USERID      -32000.0      32000.0  User ID. 0 => current user,
                                     32000 => any user.
INNAME                               Image name(name).
INCLASS                              Image name(class).
INSEQ            0.0        9999.0  Image name(seq. #). 0 => high
INDISK           0.0           9.0  Disk drive #.       0 => any
BLC              0.0        4096.0  Bottom left corner of image
                                     0 => entire image
TRC              0.0        4096.0  Top right corner of image
                                     0 => entire image
NDIG             0.0           7.0  Digits in display.     0 => 1
FACTOR      -99999.0      999999.0  Multiplication factor. 0 => 1
XINC             0.0         100.0  Display every XINC col. 0=> 1
YINC             0.0         100.0  Display every YINC rows. 0=>1
DOCRT           -1.0         132.0  >0 -> use CRT, else printer
                                    >72 => CRT width in chars
---------------------------------------------------------------
PRTIM
Type: Task
Use:  PRTIM displays an image on the line printer or the user's
      terminal.  The input parameters specify (1) any
      rectangular solid in the image, (2) the number of digits
      used for the display, (3) the skipping of pixels and (4)
      the multiplication of the display after normalization.
      Normalization is done using the larger of Datamax and
      -10.0*Datamin rounded up to the next higher power of 10.
      For NDIG=1, Datamin is used instead of 10 * Datamin.  This
      default scaling (FACTOR = 1) causes the numbers to be
      printed in "natural" units (e.g. Jy/beam) scaled by some
      power of 10.  The output shows the scaling (e.g. 1000 =
      1.0 Jy/beam on NDIG = 3).
Adverbs:
  USERID......User ID of owner of image.  0 => current user,
                32000 => any user.
  INNAME......Image name(name).        Standard defaults.
  INCLASS.....Image name(class).       Standard defaults.
  INSEQ.......Image name(seq. #).      0 => highest.
  INDISK......Disk drive # of image.   0 => any.
  BLC.........The Bottom Left-hand pixel of the subarray of
              the map to be displayed. The value (0,0) means
              start at the bottom left of the entire image.
  TRC.........The Top Right-hand pixel of the subarray of
              the map to be displayed.  The value (0,0) means
              go to the top right of the entire image.
```

```
  NDIG........The number of digits in the display.  If NDIG <=
              0, an NDIG of 1 is used.
  FACTOR......The multiplication factor for the display.  If
              FACTOR <= 0.0, a FACTOR of 1.0 is used.
  XINC........Display every XINC column(s) in the display.
              If XINC <= 0, an XINC of 1 is used.
  YINC........Display every YINC row(s) in the display.
              If YINC <= 0, a YINC of 1 is used.
  DOCRT.......True (> 0.0) means to use the terminal, otherwise
              use the line printer.  If 72 < DOCRT <= 132, the
              task will assume that the terminal is DOCRT
              characters wide.
------------------------------------------------------------
```

The first few lines of the HELP file are precursor comment lines that give the classification and a description of the function of the item described in the HELP file and a statement designed to discourage the sale of AIPS to third parties. On the first line after the precursor lines, the name of the task is given. The "L", "U" and "C" are guides showing the fields for the lower and upper limit for the value of the adverb and for the comment field. These symbols mark fields in columns 11–22 (lower limit, if any), 23–34 (upper limit, if any) and 36–64 (comment). No text should extend beyond column 64. The next line gives the name of the task and a short explanation of the task. Following this is the list of adverbs, their limits and a short description the use of each. The descriptions should be in lower case.

Column 10 in the first line of an adverb in the inputs section is used to indicate when the adverb is to be used. If column 10 is blank or "*" then the adverb is used by the adverb GO and is written into the TD file. If column 10 is "*" or "?" then the adverb will be used by the verb TELL and written into the TC file.

Following the inputs section of the HELP file and separated by a line of 64 "-" signs comes the help section. This is the text which is displayed on the users terminal when he types "HELP name" to AIPS. This section gives more details about the use of the task and its adverbs. The HELPs section should have the format shown in the example above; explanations should be in lower case, where appropriate, and text should not extend beyond column 64.

Following the helps section of the HELP file and separated from it by a line of 64 "-" is the explain section. This text, preceded by the help section, is printed when the user types EXPLAIN ... to AIPS. This section, which is unfortunately absent from the example above, describes in detail how to use the task and its relation to other tasks. The general method the task uses should be described in the explain section.

3.4.2 At Task Startup (GTPARM)

When the task comes alive it must read the Task Data (TD) file to get the values of the adverbs. This is done via a call to GTPARM. (Details of the call sequence to GTPARM can be found at the end of this chapter).

A convenient way to access the values returned by GTPARM is to declare a common in a task LOCAL INCLUDE which has the variables in order and pass the name of the first variable in place of RPARM. The values can then be obtained by name. Note that all values are as REAL or HOLLERITH variables. Characters are in HOLLERITH strings and require (NCHAR+3)/4 storage elements and, in general, these HOLLERITH variables need to be converted to CHARACTER variables using H2CHR before use.

3.4.3 While a Task is Running (GTTELL)

While a task is running in an interactive (non-batch) mode the user may send further instructions to the task. This is done using verb TELL which writes instructions in the task communications (TC) file. The task may read its instructions in the TC file using routine GTTELL. (Details of the call sequence to GTTELL are given at the end of this chapter.)

3.5 Restarting AIPS

When AIPS starts a task, it suspends itself until either (1) the task returns a return code in the TD file or (2) the task disappears. It is therefore the responsibility of the task to restart AIPS. The timing of this is determined by the value of RQUICK returned by GTPARM (set by the user as the AIPS adverb DOWAIT). If RQUICK is true, then AIPS should be restarted as soon as possible (after perhaps some error checking on the inputs). This is done by the routine RELPOP (the call sequence is given at the end of this chapter). If the task has an interactive portion, it should be completed before restarting AIPS; this will keep the task and AIPS from trying to talk to the user terminal at the same time.

RELPOP returns to AIPS a return error code RETCOD. A non-zero value of RETCOD indicates that the task failed, in which case AIPS will terminate the current line of instructions, procedure or RUN file. If RQUICK is false, then AIPS is not to be restarted until the task terminates. In this case RELPOP is called by either DIETSK or DIE and the programmer only has to be sure the correct value of RQUICK is sent to DIETSK.

3.6 INCLUDE files

AIPS tasks make extensive use of commons to keep system constants and to communicate between subroutines. Many of these commons are in hundreds of routines. To make these commons manageable, they are declared in INCLUDE files which are filled into the source code by the AIPS preprocessor.

The INCLUDE files names have the form nxxx.INC where n indicates the type of include file: P indicates that PARAMETER statements are included, D indicates that type declarations and/or COMMONs and/or EQUIVALENCES are included, V indicates that DATA statements are included, Z indicates that machine dependent declarations are included. In general, the ordering of the includes is in order Pxxx.INC, Dxxx.INC then Vxxx.INC. Fortran specifies that all declarations come before any executable statements and DATA statements are considered executable. The directory containing the INCLUDE files is specified via a logical name. The word INCLUDE must start in column 7 and the entire name of the file must be bracketed in single quotes. An example:

```
      INCLUDE 'INCS:DDCH.INC'
```

In current VMS and UNIX implementations INCS: is a search path specifying a list of directories to search. These directories are ordered from the most machine specific to the most general. For development and test purposes, it is possible to modify the search path to search the programmer's directory first. This is done with an

```
$ASSIGN (search path) INCS
```

in VMS and by assigning a search path to the environment variable INCS in UNIX:

```
%setenv INCS "/mnt/mydir $INCXXX $INCNOT $INC"
```

where /mnt/mydir is the directory to be added and $INCXXX should be replaced with the include directory specific to the local machine (e.g. $INCVEX for Convexes).

Many tasks also have their own includes; this greatly reduces the problems in developing and maintaining tasks. In order to facilitate task INCLUDEs the AIPS preprocessor allows the defination of LOCAL INCLUDEs. These are segments of text which are defined in the file in which they are to be INCLUDEd. By convention these are given at the beginning of the file and have the syntax illustrated in the following example:

```
LOCAL INCLUDE 'MYTASK.INC'
C                                          Local include for MYTASK
      HOLLERITH XSTR1, XSTR2
      REAL      X, Y, Z
      INTEGER   I, J, K
      CHARACTER STR1*8, STR2*4
```

```
      COMMON /MYCOM/ XSTR, XSTR2, X, Y, Z, I, J, K
      COMMON /MYCHR/ STR1, STR2
LOCAL END
```

The text segment defined in this example can then be INCLUDEd by the preprocessor with a statement INCLUDE 'MYTASK.INC' beginning in column 7.

3.7 Initializing Commons

In order for the commons mentioned in the previous section to be of use, their values must be filled in. For this purpose there are a number of common initialization routines. These commons and their initialization are discussed in the following sections.

3.7.1 Device Characteristics Common

The most important commons are the Device Characteristics Commons; these are obtained from the INCLUDE file DDCH.INC. The text of this INCLUDE is to be found at the end of this chapter. The contents of the Device Characteristics commons are initialized by a call to ZDCHIN. Details of the call sequence can be found at the end of this chapter. Many of the values in the Device Characteristics common are read from a disk file. The values in this file can be read and changed using the stand-alone utility program SETPAR. The constants kept in this common are described in the following:

```
SYSNAM  C*20      System name
VERNAM  C*4       Version ID
RLSNAM  C*8       Release name
DEVNAM  C(10)*48  Names of files using non-FTAB I/O currently open.
NONNAM  C(8)*48   Names of files using non-map I/O currently open.
MAPNAM  C(12)*48  Names of files using map I/O currently open.
SYSTYP  C*4       system type: 'VMS ' or 'UNIX' or ???
SYSVER  C*8       system version: '4.5', 'BSD 4.2', 'SYS 5', ...
XPRDMM  R         Printer points per millimeter
XTKDMM  R         Graphics points per millimeter
TIMEDA  R(15)     Min. TIMDEST time for each disk (days)
TIMESG  R         Min. TIMDEST time for SAVE/GET files (days)
TIMEMS  R         Min. automatic destruction time for messages
TIMESC  R         Min. automatic destruction time for scratch
TIMECA  R         Min. destruction time for empty catalogs.
TIMEBA  R(4)      Times during which AP Batch jobs cannot start.
                  1, 2 start, stop times (hrs) on weekends
                  3, 4 start, stop times (hrs) on weekdays
TIMEAP  R(3)      1 => time between rolls (min)
                  2,3 polynomial terms for determining how long
                  a job must wait before grabbing the AP.
FBLANK  R         REAL value used to indicate blanking
DBLANK  D         DOUBLE PRECISION value used to indicate blanking.
HBLANK  H         HOLLERITH blank string (4 char)
RFILIT  R(14)     Spare
NVOL    I         Number of disk drives available to AIPS
NBPS    I         Number of bytes per disk sector
NSPG    I         Number of disk sectors per allocation granule
NBTB1   I         Number bytes in FTAB / non-FTAB device
NTAB1   I         Max number of non-FTAB devices open at once
NBTB2   I         Number bytes in FTAB / slow I/O device
NTAB2   I         Max number of slow I/O devices open at once
```

```
NBTB3   I       Number bytes in FTAB / fast I/O device
NTAB3   I       Max number of fast I/O devices open at once
NTAPED  I       Number of tape drives available to AIPS
CRTMAX  I       Number lines / CRT terminal page
PRTMAX  I       Number lines / printer page
NBATQS  I       Number batch AIPSs in system
MAXXPR  I(2)    Number of plotter dots / page in X, Y
CSIZPR  I(2)    Number of plotter dots / character in X, Y
NINTRN  I       Maximum # simultaneous interactive AIPSs
KAPWRD  I       # 1024s of words of array processor memory
NWDPDP  I       # words / double-precision floating point
NBITWD  I       # bits / word
NWDLIN  I       # words in a POPS input line
NCHLIN  I       # characters in a POPS input line
NTVDEV  I       # television display devices available
NTKDEV  I       # graphics display devices available
BLANKV  I       Integer magic value => blanked pixel
NTVACC  I       Number POPS programs allowed access to TV devices
NTKACC  I       Number POPS programs allowed access to graphics
UTCSIZ  I       Private catalog size (0=>public)
BYTFLP  I       Byte flip, 0=none, 1=bytes, 2=words, 3=both
USELIM  I       Maximum user number
NBITCH  I       # bits per character
NCHPRT  I       Width of line printer in characters.
KAP2WD  I       # 1024s words of secondary AP memory.
MAXXTK  I(2)    Graphics screen size x,y
CSIZTK  I(2)    Graphics character size x,y
DASSGN  I(8,15) Lists of allowed users, 8 per disk for up to 15
                disks.
SPFRMT  I       Single precision floating-point format code
                0 => OTHER
                1 => IEEE
                2 => VAX F
                3 => VAX G
                4 => IBM (not supported yet)
DPFRMT  I       Double precision floating-point format code
                (see codes for SPFRMT)
NSHORT  I       Shortest vector length to vectorize
TTYCAR  I       1 => TTY i/o uses carriage control characters.
DEVTAB  I(50)   Device type code numbers
FTAB    I(*)    I/O driving tables
```

3.7.2 Catalog Pointer Common

The catalog header record for an AIPS data file is a data structure containing characters, integers, and single and double precision reals. The size of the record is fixed at 512 bytes where a byte is defined as half an integer. Values in the catalog header record are accessed from a number of arrays of different data types equivalenced together. Since different computers have different sizes for different data types, we use pointers in these equivalenced arrays. These pointers are kept in a common invoked with the INCLUDE DHDR.INC and are initialized by a call to VHDRIN. VHDRIN has no arguments, but should be called after ZDCHIN. For more details, see the chapter on the catalog header. The catalog header can contain arbitrary keyword/value pairs to allow storage of information not currently allocated space in the header. Access to these keyword/value pairs is throught routine CATKEY.

3.7.3 History Common

The routines that write HIstory files carry information in pointers in commons invoked with the INCLUDE DHIS.INC and are initialized by a call to HIINIT; the details of the call sequence are given at the end of this chapter.

3.7.4 TV Common

The routines that talk to the television display use information from the commons obtained by the INCLUDEs DTVC.INC and DTVD.INC. If a task uses the TV, there must be an initializing call to YTVCIN which has no call arguments.

YTVCIN initializes the common which describes the characteristics of the interactive display devices and the common which has the current status parameters of the TV. The values set are default values only. They are reset to the current true values by a call to TVOPEN. YTVCIN resets the common values of TVZOOM and TVscroll, but does not call the TV routines to force these to be true. See the chapter on the television devices for more details.

3.7.5 UV data pointer common

The format in which uv data is stored is relatively flexible and is described in the chapter on disk I/O. Since it is rather flexible, the location in a logical record of a given value must be determined from the catalog header. In order to make it easier to find values in a uv data record, we use a common containing pointers; this common is obtained by using the INCLUDE DUVH.INC. This common is filled in by a call to UVPGET which analyzes the current catalog header in common /MAPHDR/ (In INCLUDE DCAT.INC). Details of the call arguments and the pointers etc. set are found at the end of this chapter.

3.7.6 Files common, DFIL.INC

Many tasks open a number of cataloged files and create several scratch files. The status of the cataloged files are marked 'READ' or 'WRIT' in the catalog directory and need to be cleared by the end of the program. Scratch files must be destroyed by the end of the program. Since an error might terminate the program at any stage, the program must be prepared to clear catalog files and destroy scratch files under any circumstances in which it controls its death.

Many tasks accomplish these functions through use of the common obtained from the INCLUDE DFIL.INC and use of the termination routine DIE (which will be discussed in a later section). The contents of DFIL.INC is found at the end of this chapter.

In this common, NSCR is the number of scratch files that have been created, SCRCNO contains the catalog numbers of the scratch files, and SCRVOL contains the disk numbers of the scratch files.

NCFILE tells how many catalog files are marked, FVOL contains the disk numbers of the cataloged files marked, FCNO contains the catalog slot numbers of the marked files, and FRW contains flags for each of the marked catalog files (0 ='READ', 1='WRIT', 2='WRIT' but destroy if the task fails).

IBAD is an array to contain the disk drive numbers on which not to put scratch files; IBAD is used by the scratch file creation routine SCREAT. RQUICK is also carried along in this common so that AIPS can be restarted by the shutdown routines if necessary. If the information in this common is kept current, catalog file status words will be cleared and scratch files deleted by the shutdown routine DIE. If the DFIL.INC common is being used, it should be initialized with the following statements before use:

```
NSCR = 0
NCFILE = 0
```

and by initializing the array IBAD to zeroes or the values of BADDISK sent by AIPS.

3.8 Input and Output File Names

The input and output file name, class, sequence etc. passed to a task are subject to a number of default and wild-card conventions in the case that they are not completely specified. For the most part, these conventions are incorporated into the standard utility routines. For some tasks, there are logical default values which are not the standard defaults and which must be handled by the task. An example of this is the output class for APCLN. If the input class is IMAP and the output class is not specified (all blanks), then APCLN uses ICLN for the output class.

The standard defaults for input names are as follows: If the disk is not specified, all disks are searched in order starting with disk 1. If the name and/or class is not specified, then the catalog (or catalogs) are searched until a file satisfying all specified criteria is found. If the sequence number is not specified, then the file with the *highest* sequence number meeting all specified criteria is picked. In addition to the default conventions, AIPS also supports two types of wild-cards; "*" means any number, including none, of any character will be accepted, "?" means exactly one character of any type will be accepted as a match. The standard defaults and wild-cards are fully supported by the standard catalog routines. The standard default for the output name is the input name; the standard default for the output class is the name of the task, and the standard default for the output sequence is 1 higher than the highest sequence number on any disk for any file with the same name and class; if there are no other matching files, the sequence number is 1. The default output disk is the highest numbered disk on which space is available. Wild-cards are supported in the output name; basically a wild-card in the output name and class means to use the corresponding character (or characters) from the input name or class. Only one "*" is allowed in an output name or class; others are ignored. These defaults and wild-card conventions are implemented in the utility MAKOUT. MAKOUT must be called by all tasks which may create an output file. The details of the call sequence of MAKOUT are given at the end of this chapter.

3.9 Copying Extension Files

Each cataloged file may (and usually does) have auxiliary files containing information related to the cataloged file; these files are called extension files. There are usually several of these extension files that a task must copy if it is creating a new output file. The most important of these is the history file (file type "HI") which should be updated as well as copied. For uv data files, the ANtenna tables (type "AN"), FreQuency tables (type "FQ") and any relevant calibration tables should be copied and for images any CLEAN components tables (type "CC") should be copied. Other extension file types may also have to be copied. The following sections describe how to copy and/or update these files.

3.9.1 History

Information describing the processing history of a data set is kept in an extension file to each main data file. These files consist of 72 character records using the FITS convention for history records. Each task writes into the history file records which begin with the name of the task and contain information about how data was processed by that task. This is usually in the form "adverb name=" followed by the actual value used. These records should be able to be parsed in the same manner as FITS header records. Comments are preceded by a "/".

There are a number of utility routines to simplify handling history files. A short description of each follows and the details of the call sequences can be found at the end of this chapter.

- HIINIT initializes the history common.
- HISCOP creates and catalogs a new history file, opens it, opens an old history file and copies it to the new history file, and leaves the old history file closed and the new file open.
- HIADD adds a history card to an open history file.
- HIADDN adds a history card to number of open history files.
- HIAD80 adds an 80-character card image into an open history file.

- HICLOS closes a history file, flushing the buffer if requested.
- HIMERG creates several history files by merging several old files.
- HIREAD reads the next history card from an open history file.

Once the history file is open, entries can be made in it by first WRITEing the message (up to 72 characters) into a CHARACTER array dimensioned to be at least 72 characters and calling HIADD. We wish to encourage the convention of using the name "HILINE" for this CHARACTER variable. An example:

```
      CHARACTER HILINE*72
      INCLUDE 'INCS:DMSG.INC'
              .
              .
              .
      WRITE (HILINE,2000) TSKNAM, FACTOR
 2000 FORMAT (A6,' FACTOR=',F5.2,' / CORRECTION FACTOR')
      CALL HIADD (NLUH, HILINE, BUFFER, IERR)
```

Once all new entries have been made to the history file, the buffer is flushed and the file closed by a call to HICLOS. (HICLOS should normally be called with UPDATE=.TRUE. for a history file being written)

It should be noted that HISCOP will also work properly if the old and new history files are actually the same file. In this case, it simply opens the new file to add new entries. Several other history utilities, which may occasionally be useful, are HICREA which creates a history file, HIOPEN which opens a history file and HICOPY which copies the contents of one history file onto the end of another history file. The functions of these routines are incorporated into the routines described above so they are normally not of great interest to the programmer.

3.9.2 Extension tables (ALLTAB, TABCOP)

All tables extension files may be copied with a single call to ALLTAB. ALLTAB also accepts a list of table types not to be copied. Certain nontable extension file types are excluded from being copied by ALLTAB, these being history files (type "HI") and plot files (type "PL"). A description of the call sequence to ALLTAB is given at the end of this chapter. Routine TABCOP can be used to copy tables of a given type.

An older form of extension file was managed by the pair of routines EXTINI and EXTIO. Files of this type can be copied by the routine EXTCOP.

3.10 Communication with the user

3.10.1 Writing messages

Most of the important communications between a user and AIPS and its tasks are sent to both a monitor terminal, which may be the users own terminal, and to a disk log file. This logged information is primarily of use to the user, but is frequently of great use in debugging a program. The basic way a task communicates to the user is through the utility routine MSGWRT. A message of up to 80 characters ($\leq$ 64 is best) is first written into array MSGTXT in the message common, which is invoked by the include DMSG.INC. By convention, error messages should be all in upper case and warning or informative messages should be mixed case.

A call is made to the routine MSGWRT with a single INTEGER argument which is the priority level to write the message. The meaning of the priority is as follows:

Priority	Use
0	Write to log file only
1	Write to monitor terminal only
2	Low interest normal messages

```
3-4          Normal message
 5           High interest normal message.
6-8          Error message
9-10         Severe error messages
```

An example of the use of MSGWRT follows:

```
      INCLUDE 'INCS:DMSG.INC'
        .
        .
      WRITE (MSGTXT,1000) IERR
      CALL MSGWRT (6)
        .
        .
 1000 FORMAT ('ENCOUNTERED ERROR ',I3)
```

3.10.2 Turning off system messages

Many of the AIPS utility routines give messages which may or may not indicate a problem such as the "FILE ALREADY EXISTS" message from ZCREAT. Most of the messages are written at priority level 6 or 7 and may be turned off by setting the variable MSGSUP in INCLUDE DMSG.INC (the same one MSGTXT lives in) to 32000. This variable should be restored as soon as possible to a value of 0 to enable level 6 and 7 messages.

3.10.3 Writing to the line printer

The standard Fortran logical unit number for the line printer in the AIPS system is unit 1. Writing to the line printer can be done with normal formatted Fortran writes. Before writing to the line printer it should be opened with a call to ZOPEN and a header page prepared for batch jobs with a call to BATPRT. When the task is finished writing to the printer, a second call to BATPRT will write a trailer page, a call to ZENDPG will eject a page (very important on electrostatic printers), and a call to ZCLOSE will close the file and send it to the printer spooler. An example follows:

```
      INTEGER   LPLUN, LPIND, BUFFER(256), IPCNT
      LOGICAL   T,F
      REAL      VALUE1, VALUE2
      CHARACTER LPNAME*48
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
      PARAMETER (LPLUN = 1)
      PARAMETER (LPNAME = '    ')
      INCLUDE 'INCS:DDCH.INC'
          .
          .
C                                       Open the printer.
      CALL ZOPEN (LPLUN, LPFIND, 1, LPNAME, F, T, T, IERR)
        (handle error condition if detected)
C                                       Header page if batch
      CALL BATPRT (1, BUFFER)
      IPCNT = 0
          .
```

```
C                                       Increment line count
      IPCNT = IPCNT + 1
C                                       Check if page full.
      IF (IPCNT .LT. PRTMAX) GO TO 100
C                                       Write new page header
          .
          .
         ICPNT = 0
C                                       Write to printer
 100  WRITE (LPLUN,1000) VALUE1, VALUE2
          .
          .
C                                       Trailer page if batch
      CALL BATPRT (2, BUFFER)
C                                       Eject a page
      CALL ZENDPG (IPCNT)
C                                       Close printer and send to
C                                       spooler.
      CALL ZCLOSE (LPLUN, LPIND, IERR)
          .
          .
 1000 FORMAT (' VALUE1 =',F10.5, ' VALUE2 =',1PE12.6)
```

The number of lines per page on the line printer is obtained, as shown in the example, by the variable PRTMAX in the device characteristics common (DDCH.INC). In the example above, ZOPEN recognized the unit number (LPLUN) value of 1 as meaning the line printer, so most of the arguments to ZOPEN are dummy in this case.

In the real world, the use of line printers is more complicated than this. For example, line printers have not only a variable number of lines per page, but also a variable number of characters across a page (NCHPRT in DDCH.INC). Line printers are often located at some distance from the user's terminal. As a result, all AIPS printing tasks allow the user the DOCRT option, which specifies that the terminal, rather than the printer, is to be used. DOCRT may also be used to specify the width of the terminal (see PRTIM help file earlier in the chapter). Thus, standard AIPS print programs must handle variable width formats, pagination, alternate output devices, pausing on page full for terminal output, etc. The subroutine PRTLIN will provide many of these services. A description of the call sequence of PRTLIN is given at the end of this chapter. Read the code of the task PRTUV to see a good example of the full AIPS handling of a print job.

3.10.4 Writing to the Terminal (ZTTYIO)

Many mainframe computers are batch oriented and discourage programs from talking directly to a terminal. To get around this problem, AIPS has a "Z" routine for this purpose. ZTTYIO, rather than Fortran reads and writes to units 5 and 6, is used to communicate with the terminal.

If a task is going to talk to the user terminal, it should not call RELPOP until after communication with the user terminal is complete. If AIPS is restarted too soon, both AIPS and the task will be trying to talk to the terminal at the same time; this will probably confuse the user.

Before calling ZTTYIO, the device must be opened by a call to ZOPEN, and after the task is through talking to the terminal, it should be closed with a call to ZCLOSE. Use a value of 5 for the LUN. In the call to ZOPEN, the file name and disk number are dummy parameters since ZOPEN recognizes LUN=5 as a Fortran device. Write messages to be sent into an array and send the array to ZTTYIO. Lines read from the terminal will be returned by ZTTYIO as a CHARACTER string. An example of the use of ZTTYIO is the following:

```
      INTEGER   TTYLUN, TTYIND, IRET
      LOGICAL   T, F
      PARAMETER (TTYLUN = 5)
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
      CHARACTER LINE*72
                  .
                  .
C                                     Open the terminal
      CALL ZOPEN (TTYLUN, TTYIND, 1, LINE, F, T, T, IERR)
C                                     Error if IERR .NE. 0
                  .
                  .
C                                     Write message for terminal
      WRITE (LINE,1000)
C                                     Send to terminal
C                                     Set here to read and write
C                                     up to 72 characters per
C                                     transmission.
      CALL ZTTYIO ('WRIT', TTYLUN, TTYIND, 72, LINE, IERR)
C                                     Error if IERR .NE. 0
                  .
                  .
C                                     Read from terminal.
C                                     Up to 72 characters.
      CALL ZTTYIO ('READ', TTYLUN, TTYIND, 72, LINE, IERR)
C                                     Error if IERR .NE. 0
                  .
                  .
C                                     Close terminal
      CALL ZCLOSE (TTYLUN, TTYIND, IERR)
                  .
                  .
 1000 FORMAT (' Hi there')
```

3.11 Scratch Files

Many tasks require the use of scratch files which must be created at the beginning of the task and destroyed at the end of the task. Since the task may detect an error condition and decide to quit at an arbitrary place in the program, some provision must be made to destroy the scratch files under all conditions for which the task controls its death. Scratch files are cataloged as type 'SC' so that the user can directly delete them. The DFIL.INC commons described in a previous section are designed for this purpose.

A simple way to create scratch files is to use the common /CFILES/ and the routine SCREAT. SCREAT will try to scatter the scratch files among as many disk drives as possible, will try all of the disks if necessary to find space for a scratch file, and can be prohibited from putting scratch files on certain disks by use of the array IBAD (adverb array BADDISK in AIPS). Details of the call sequence for SCREAT can be found at the end of this chapter.

An example of the use of SCREAT is the following:

```
      INTEGER   IRET, NX, NY, NP(2), BUFF(512), SIZE
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DDCH.INC'
```

```
C                                       NX, NY are the size of an
C                                       image.  Make a scratch file
C                                       big enough for a copy of the
C                                       image.
C
C                                       Compute the size.
      NP(1) = NX
      NP(2) = NY
C                                       Compute size needed
      CALL MAPSIZ (2, NP, SIZE)
C                                       Create scratch file.
      CALL SCREAT (SIZE, BUFF, IRET)
C                                       Test for errors...
```

In the above example, the scratch file created will be entered in the DFIL.INC common as number NSCR (which was incremented). The disk and catalog slot numbers are thus SCRVOL(NSCR) and SCRCNO(NSCR). This scratch file can be opened as follows:

```
      INTEGER   LUN, IND
      CHARACTER FILE*48
      INCLUDE 'INCS:DFIL.INC'
              ...
C                                       ISCR = DFIL.INC slot number.
      CALL ZPHFIL ('SC', SCRVOL(ISCR), SCRCNO(ISCR), 1, FILE, IRET)
      CALL ZOPEN (LUN, IND, SCRVOL(ISCR), FILE, .TRUE., .TRUE.,
     *   .TRUE., IRET)
```

Once opened, these files can be initialized and read or written in the same way as permanent cataloged data files.

Since scratch files are cataloged, they have an associated catalog header record. SCREAT fills in nominal values, but, if the scratch file contains data in the same form as an image or uv data, the appropriate information can be placed in the header to describe the data. This allows using the header record to specify the contents of a file in utility routines, simplifies the interface to the routine, and allows the routine to work equally well on permanent or scratch files. This technique is used in a number of utility routines such as VISDFT.

3.12 Terminating the Program

Most tasks create scratch files or open cataloged files which have status words marked in the catalog directory. These scratch files should always be destroyed by the end of the program, and the catalog files should be unmarked. Also AIPS may have to be restarted at the end of the program. For these and other reasons, we strongly advise that when error conditions are detected that the routine finding the error set the appropriate error code and return; all the way back to the main routine. Then a call to one of the shutdown routines can be followed by a Fortran STOP statement. *There should be no other STOP statements in the program.*

In the section describing initialization of the DFIL.INC common, there is a discussion of using it to carry information about scratch and cataloged files. If this common is used, the shutdown routine DIE will take care of deleting all scratch files, unmarking catalog files, and restarting AIPS if necessary. If the DFIL.INC common is not used, the routine DIETSK will restart AIPS and take care of the other shutdown functions. (DIE calls DIETSK). Both of these routines accept a return code which is sent to AIPS if it is restarted at that time; a nonzero value of the return code indicates that the program failed. Descriptions of DIE and DIETSK can be found at the end of this chapter.

3.13 Batch Jobs

AIPS has a capability to run tasks in the batch mode. It usually makes little difference to a task if it is being run in batch or interactive mode, but use of some devices is forbidden to batch tasks. These devices are the tape drive, the graphics device, and the television. After the calls to GTPARM and ZDCHIN, a task can determine if it is running as a batch task by comparing the value of NINTRN (number of interactive AIPS allowed) from the device characteristics common (DDCH.INC) with NPOPS (the AIPS number of the initiating task) from the message common (DMSG.INC). If NPOPS is greater than NINTRN, then the task is running as a batch task and use of the devices mentioned above is disallowed. A new, better way to make this determination is to test the value of ISBTCH in the device characteristics common. If ISBTCH = 32000, the task is to act as a batch job, no matter what its value of NPOPS. The user can set this condition into an apparently interactive AIPS session with the pseudoverb statement ISBATCH TRUE. Batch jobs always run with RQUICK (DOWAIT in AIPS) true and thus do not restart AIPS until they are done. GTPARM enforces this on the RQUICK parameter.

3.14 Installing a New Task

The procedure to install a task depends a great deal on the host computer and operating system. Appendix A at the end of this volume describes how to test and install new software and describes the directory structure.

3.15 INCLUDEs

There are several types of INCLUDE files which are distinguished by the first character of their name. Different INCLUDE file types contain different types of Fortran declaration statements as described in the following list.

- Pxxx.INC. These INCLUDE files contain declarations for parameters and the PARAMETER statements.
- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations, COMMON and EQUIVALENCE statements.
- Vxxx.INC. These contain Fortran DATA statements.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

3.15.1 DDCH.INC

```
C                                                               Include DDCH.
C                                        AIPS system parameters
      CHARACTER SYSNAM*20, VERNAM*4, RLSNAM*8, DEVNAM(10)*48,
     *   NONNAM(8)*48, MAPNAM(12)*48, SYSTYP*4, SYSVER*8
      HOLLERITH HBLANK
      DOUBLE PRECISION DBLANK
      REAL   XPRDMM, XTKDMM, TIMEDA(15), TIMESG, TIMEMS, TIMESC, TIMECA,
     *   TIMEBA(4), TIMEAP(3), FBLANK, RFILIT(14)
      INTEGER   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3,
     *   NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2), CSIZPR(2),
     *   NINTRN, KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK(2), CSIZTK(2), DASSGN(8,15), SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR, DEVTAB(50), FTAB(1024)
      COMMON /DCHCHM/ SYSNAM, VERNAM, SYSTYP, SYSVER, RLSNAM,
```

```
     *   DEVNAM, NONNAM, MAPNAM
      COMMON /DCHCOM/ DBLANK, XPRDMM, XTKDMM, TIMEDA, TIMESG, TIMEMS,
     *   TIMESC, TIMECA, TIMEBA, TIMEAP, FBLANK, RFILIT, HBLANK,
     *   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3, NTAB3,
     *   NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR, CSIZPR, NINTRN,
     *   KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK, CSIZTK, DASSGN, DEVTAB, SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR
      COMMON /FTABCM/ FTAB
C                                                             End DDCH.
```

3.15.2 DFIL.INC

```
C                                                             Include DFIL.
C                                        AIPS system catalog and scratch
      INTEGER   NSCR, SCRVOL(128), SCRCNO(128), IBAD(10), LUNS(10),
     *   NCFILE, FVOL(128), FCNO(128), FRW(128), CCNO
      LOGICAL   RQUICK
      COMMON /CFILES/ RQUICK, NSCR, SCRVOL, SCRCNO, NCFILE, FVOL, FCNO,
     *   FRW, CCNO, IBAD, LUNS
C                                                             End DFIL.
```

3.15.3 DMSG.INC

```
C                                                             Include DMSG.
C                                        AIPS system message common
      INTEGER   MSGCNT, NPOPS, NLUSER, NACOUN, MSGSUP, MSGREC,
     *   MSGKIL, ISBTCH, DBGAIP, MSGDM1, MSGDM2, MSGDM3
      CHARACTER MSGTXT*80, TSKNAM*6
      COMMON /MSGCOM/ MSGCNT, NPOPS, NLUSER, NACOUN, MSGSUP, MSGREC,
     *   MSGKIL, ISBTCH, DBGAIP, MSGDM1, MSGDM2, MSGDM3
      COMMON /MSGCHR/ MSGTXT, TSKNAM
C                                                             End DMSG.
```

3.15.4 DUVH.INC

```
C                                                             Include DUVH.
C                                        If you change this include you
C                                        must also change common
C                                        /CATHDR/ in DBCON
C                                        Include for uv header info
      INTEGER   NVIS
      INTEGER   ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, ILOCFQ,
     *   JLOCC, JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC,
     *   NCOR, INCS, INCF, INCIF, ICORO, TYPUVD
      CHARACTER   SOURCE*8, ISORT*2
      DOUBLE PRECISION FREQ, RA, DEC
      COMMON /UVHDR/ FREQ, RA, DEC, NVIS, ILOCU, ILOCV, ILOCW, ILOCT,
     *   ILOCB, ILOCSU, ILOCFQ, JLOCC, JLOCS, JLOCF, JLOCR, JLOCD,
     *   JLOCIF, INCS, INCF, INCIF, ICORO, NRPARM, LREC, NCOR, TYPUVD
```

```
      COMMON /UVHCHR/ SOURCE, ISORT
C                                                        End DUVH.
```

3.15.5 PUVD.INC

```
C                                                        Include PUVD
C                                       Parameters for uv data
      INTEGER   MAXANT,  MXBASE,  MAXIF, MAXFLG, MAXFLD, MAXCHA
C                                       MAXANT = Max. no. antennas.
      PARAMETER (MAXANT=45)
C                                       MXBASE = max. no. baselines
      PARAMETER (MXBASE= ((MAXANT*(MAXANT+1))/2))
C                                       MAXIF=max. no. IFs.
      PARAMETER (MAXIF=15)
C                                       MAXFLG= max. no. flags active
      PARAMETER (MAXFLG=1000)
C                                       MAXFLD=max. no fields
      PARAMETER (MAXFLD=16)
C                                       MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                       Parameters for tables
      INTEGER MAXCLC, MAXSNC, MAXANC, MAXFGC, MAXNXC, MAXSUC,
     *   MAXBPC, MAXBLC, MAXFQC
C                                       MAXCLC=max no. cols in CL table
      PARAMETER (MAXCLC=41)
C                                       MAXSNC=max no. cols in SN table
      PARAMETER (MAXSNC=20)
C                                       MAXANC=max no. cols in AN table
      PARAMETER (MAXANC=12)
C                                       MAXFGC=max no. cols in FG table
      PARAMETER (MAXFGC=8)
C                                       MAXNXC=max no. cols in NX table
      PARAMETER (MAXNXC=7)
C                                       MAXSUC=max no. cols in SU table
      PARAMETER (MAXSUC=21)
C                                       MAXBPC=max no. cols in BP table
      PARAMETER (MAXBPC=14)
C                                       MAXBLC=max no. cols in BL table
      PARAMETER (MAXBLC=14)
C                                       MAXFQC=max no. cols in FQ table
      PARAMETER (MAXFQC=5)
C                                                        End PUVD.
```

3.16 Routines

3.16.1 ALLTAB

ALLTAB copies all Table extension file(s). The output files must be new - old ones cannot be rewritten. The output file must be opened WRIT in the catalog and will have its CATBLK updated on disk.

```
ALLTAB (NONOT, NOTTYP, LUNOLD, LUNNEW, VOLOLD, VOLNEW,
  *   CNOOLD, CNONEW, CATNEW, BUFF1, BUFF2, IRET)
```

```
Inputs:
 NONOT       I     Number of "Forbidden" types to copy.
 NOTTYP(*)   C*2   Table types to ignore (2 char meaningful, blank
                   filled)
 LUNOLD      I     LUN for old file
 LUNNEW      I     LUN for new file
 VOLOLD      I     Disk number for old file.
 VOLNEW      I     Disk number for new file.
 CNOOLD      I     Catalog slot number for old file
 CNONEW      I     Catalog slot number for new file
In/out:
 CATNEW(256)I      Catalog header for new file.
Output:
 BUFF1(1024) I      Work buffer
 BUFF2(1024) I      Work buffer
 IRET        I      Return error code  0 => ok, otherwise TABCOP
                                        or 10*CATIO error.
```

3.16.2 CHCOPY

CHCOPY moves characters from one HOLLERITH string to another

```
CHCOPY (NCHAR, NP1, STR1, NP2, STR2)
Inputs:
   NCHAR   I       Number of characters to move
   NP1     I       Start char position in input string
   STR1    H(*)    Input string
   NP2     I       Start char position in output string
Output:
   STR2    H(*)    Output string
```

3.16.3 CHCOMP

CHCOMP compares two HOLLERITH strings

```
CHCOMP (NCHAR, KP1, STR1, KP2, STR2, EQUAL)
Inputs:
   NCHAR  I       # characters to compare
   KP1    I       starting character in string 1
   STR1   H(*)    string 1
   KP2    I       starting character in string 2
   STR2   H(*)    string 2
Output:
   EQUAL  L       T => strings are same
```

3.16.4 CHFILL

CHFILL fills a HOLLERITH string with a character

```
CHFILL (NCHAR, CHAR, NBP, STRING)
Inputs:
   NCHAR   I       Number of char positions to fill
   CHAR    H       Char in char position 1
   NBP     I       Start char position to fill
Output:
   STRING  H(*)    Filled string
```

3.16.5 CHLTOU

CHLTOU converts any lower case characters in a CHARACTER string to upper case.

```
CHLTOU (N, STRING)
Inputs:
   N       I       Number of characters
In/out:
   STRING  C*(*)   String to be converted.
```

3.16.6 CHMATC

searches one HOLLERITH string for the occurrence of another string.

```
CHMATC (NA, JA, CA, NB, JB, CB, NP)
Inputs:
   NA   I        Number of characters in CA (start at JA)
   JA   I        Start at char position JA in CA
   CA   H(*)     Packed substring to be found in CB
   NB   I        Number of characters in CB (n.b. TOTAL)
   JB   I        Start search at offset in CB
   CB   H(*)     Packed string.
Output:
   NP   I        start position in CB of CA, 0 if none.
                 w.r.t. start of string
```

3.16.7 CHR2H

Convert a Fortran CHARACTER variable to an AIPS HOLLERITH string. IF NCH > LEN (ISTR) then blank fill the rest.

```
CHR2H (NCH, ISTR, OUTPNT, OSTR)
Inputs:
   NCH     I      Number of characters
   ISTR    C*(*)  Input CHARACTER string
   OUTPNT  I      Start position in output string
Output:
   OSTR    H(*)   Output AIPS string
```

3.16.8 CHWMAT

CHWMAT matches a pattern string containing "wild-card" characters with a test string. The wild cards '*' for any number and '?' for exactly 1 of any character are supported.

```
CHWMAT (NPM, PS, IPT, NTS, TS, EQUAL)
Inputs:
   NPM     I         Length of test string (not incl NTS-1
                     characters)
   PS      C*(*)     Pattern string
   IPT     I(NPM)    Pattern array prepared by PSFORM
   NTS     I         Start char position in TS for testing
   TS      C*(*)     Test string
Output:
   EQUAL   L         T => they match
```

3.16.9 DIE

DIE does the housekeeping necessary for an orderly death of the task. Primarily clearing catalog flags and destroying scratch files. It also calls RELPOP if RQUICK is false.

```
DIE (ICODE, BUFF)
Inputs:
   ICODE    I        Return code: 0 => good, other => bad end
   BUFF     I(256)   Work buffer
Locations in catalog are communicated by COMMON /CFILES/:
   NCFILE     I    Number of files marked in catalog.
   FVOL       I    Volume numbers of the maps.
   FCNO       I    Slot numbers of the maps.
   FRW        I    A 0 if READ , 1 if WRITE clear desired,
                   a 2 if a new file with Write, destroy on ICODE
                   bad;  other values => file already closed.
   NSCR       I    Number of scratch files to be destroyed
   SCRVOL     I    Scratch file volume numbers
   SCRCNO     I    Scratch file catalog numbers.
```

3.16.10 DIETSK

DIETSK must be called at the end of each task as the last real statement before the final RETURNs and STOP statement. It issues a closing message, terminates the accounting, and, if RQUICK is false, restarts the initiating AIPS program.

```
DIETSK (IRET, RQUICK, IBUF)
Inputs:
   IRET     I         0 => ok, else bad end
   RQUICK   L         T => initiator already resumed
Output:
   IBUF     I(256)    Scratch buffer
```

3.16.11 EXTCOP

EXTCOP copies an extension file(s) of the EXTINI-EXTIO variety.

```
EXTCOP (TYPE, INVER, OUTVER, LUNOLD, LUNNEW, VOLOLD,
  *   VOLNEW, CNOOLD, CNONEW, CATNEW, BUFF1, BUFF2, BUFF3, IRET)
Inputs:
   TYPE    C*2      Extension file type eg 'CC','AN'
   INVER   I        Version number to copy, 0=>copy all.
   OUTVER  I        Version number on output file, if more than one
                    copied (INVER=0) this will be the # of the first
                    file.  If OUTVER=0 the EXTINI defaults are used.
   LUNOLD  I        LUN for old file
   LUNNEW  I        LUN for new file
   VOLOLD  I        Disk number for old file.
   VOLNEW  I        Disk number for new file.
   CNOOLD  I        Catalog slot number for old file
   CNONEW  I        Catalog slot number for new file
   CATNEW  I(256)   Catalog header for new file.
In/out:
   BUFF1   I(>512) Work buffer: 256 words + n * 256 words (enough
                   to hold at least one logical record)
   BUFF2   I(>512) Work buffer: as BUFF1
```

```
   BUFF3   I(*)    Buffer large enough to hold one logical record
Output:
   IRET    I     Return error code  0 => ok
                                    1 => files the same, no copy
                                    2 => no input files exist
                                    3 => failed
                                    4 => no output files created
```

3.16.12 GTPARM

GTPARM obtains the activator task number, obtains the transmitted parameters, initializes the message common, and outputs the message 'task NAME begins'. It also handles startup accounting.

```
GTPARM (NAME, NPARMS, RQUICK, RPARM, SCRTCH, IERR)
Inputs:
   NAME    C*6          Task name
   NPARMS  I            Number of real variables wanted
Outputs:
   RQUICK  L            T => release POPs as soon as possible
                        F => wait until you have finished
   RPARM   R(NPARMS)    Parameters received
   SCRTCH  I(256)       Scratch buffer
   IERR    I            Error code: 0 -> ok
                                    1 -> initiator (AIPS) not found
                                    2 -> disk troubles
                                    3 -> initiator zeroed
```

3.16.13 GTTELL

GTTELL gets any parameters sent to the current task by AIPS verb TELL. All entries for the task in the TC file are cleared and the most recent is returned to the calling routine.

```
GTTELL (NPARMS, OPTELL, PARMS, SCRTCH, IERR)
Inputs:
   NPARMS   I          Number REAL parameters
Output:
   OPTELL   C*4        Opcode from TELL
   PARMS    R(*)       Returned parameters
   SCRTCH   I(256)     Scratch buffer
   IERR     I          0 => okay (no parms)
                       1 => okay (got parms)
                       2 => TELL orders quit
                       3 => TELL orders abort
Note: if GTTELL encounters an internal error, i.e. file open, read,
etc. failure, it returns IERR = 0 after emitting a message.
```

3.16.14 HIADD

HIADD adds a history card to a history file. I/O takes place only if necessary. Thus UPDATE = .TRUE. on HICLOS is required.

```
HIADD (HLUN, CARD, BUFFER, IERR)
Inputs:
   HLUN    I         LUN of HI file (must be open!!)
   CARD    C*72      new card
In/out:
```

```
   BUFFER  I(256)    HI work buffer
Output:
   IERR    I         Error return: 0 => ok, other set by HIIO
```

3.16.15 HIADDN

HIADDN is used by HIMERG for output to avoid large numbers of loops.

```
HIADDN (LUN, N, HILINE, BUF, CHK, IERR)
Inputs:
   LUN  I(N)      Input LUNs.
   CHK  I(N)      Only write if CHK(I) = 0.
   N              Number of files (including "dummies").
   HILINE         String to add.
In/Out:
   BUF  I(256,N) Working buffers
   IERR I         Error code = max error code generated by HIADD
```

3.16.16 HIAD80

HIAD80 puts an 80-character card image into a history file. It actually puts 0 (CARD all), 1 ($\leq$ 72 chars), or 2 cards in the file.

```
HIAD80 (HLUN, IST, CARD, HBLK, IERR)
Inputs:
   HLUN   I          LUN of open history file
   IST    I          Start character position in card
   CARD   C*80       80-character "card"
In/out:
   HBLK   I(256)     HI I/O buffer
Output:
   IERR   I          Error code of HIADD
```

3.16.17 HICLOS

HICLOS closes a history file updating it if requested.

```
HICLOS (HLUN, UPDATE, BUFFER, IERR)
Inputs:
   HLUN    I          file LUN (already open!!)
   UPDATE  L          T => write last record & update pointers
In/out:
   BUFFER  I(266)     HI work buffer
Output:
   IERR    I          error code : 0 - ok
                                   1 - LUN not open
                                 2-6 - ZFIO errors
```

3.16.18 HIINIT

HIINIT initializes the history common area /HICOM /

```
HIINIT (NFILES)
Inputs:
   NFILES   I      number of HI files open at once (max)
                   at least 20 are available via DHIS.INC
```

3.16.19 HIMERG

HIMERG merges NOLD history files an copies them to NNEW new history files. The merged history file consists of the whole of the "master" history file, followed by the other history files starting from the first line at which each differs from the master. The history files are separated by comment lines noting the number of lines omitted. If any of the input files is unreadable, it is omitted from the input list and a comment is inserted in the merged history. If there is trouble writing to any of the output files, copying to that file is stopped. If one of the output files is also one of the (readable) input files, that file is designated the master. If more that one of the output files is in the input list, the last such duplicate is the master; no history copy is attempted for the earlier duplicate files and the program returns an error code of 2. If none of the output files is in the input list, the master file is the first (readable) input file. If a read error is encountered while copying, the output HI files are reset to their pristine state, i.e. empty for new files and with the original contents for old files. The task name, date, and time are entered on the new files. This is a generalised version of HISCOP/HICOPY.

```
HIMERG (LUNOLD, LUNNEW, VOLOLD, VOLNEW, CNOOLD,
  *   CNONEW, NOLD, NNEW, CATBLK, BUFER1, BUFER2, IERR)
Inputs:
   LUNOLD   I(NOLD)  LUNs for old history file.
   LUNNEW   I(NNEW)  LUN for new history file.
   VOLOLD   I(NOLD)  Vol. number for old history file.
   VOLNEW   I(NNEW)  Vol. number for new history file.
   CNOOLD   I(NOLD)  Catalog slot number of old history file.
   CNONEW   I(NNEW)  Catalog slot number of new history file.
   NOLD     I        Number of old history files.
   NNEW     I        Number of new history files.
In/Out:
   CATBLK   I(256,NNEW)   Catalog header of map for new file.
   BUFER1   I(256,NOLD)   Work buffer, used for old files.
   BUFER2   I(256,NNEW)   Work buffer, new file; must be used
                          in further HIADD calls until file
                          is closed.
Output:
   IERR     I        Return error code: 0 => OK.
                        1 => could not open old history file.
                        2 => could not copy old history file.
                        3 => could not write time on new file
                        4 => could not create/open new HI file.
                        5 => Two or more output files the same.
                        6 => Wrong number of input files.
NOTE: IERR < 3 is a warning only, = 3 serious, = 4 a real problem.
Calling programs should ignore IERR < 3, branch to HICLOS of the
new HI file on IERR = 3, and skip over all HI stuff on IERR = 4.
Errors 5 and 6 should not occur in working programs.
```

3.16.20 HIREAD

HIADD reads next history card from a history file. IO takes place only if necessary.

```
HIREAD (HLUN, HIREC, CARD, BUFFER, IERR)
Inputs: HLUN    I         lun of HI file (must be open!!)
        HIREC   I         logical rec no to read
IN/out: BUFFER  I(256)    HI work buffer
Output: IERR    I         0 => ok, other set by HIIO
        HIREC+1 I         lrecno incremented for next read
        CARD    I(*)      card
```

3.16.21 HISCOP

HISCOP copies one history file to another. If the new history file already exists the only action is to open it. At finish the old history file is closed; the new history file is open. The task name, date, and time are entered on the new file.

```
HISCOP (LUNOLD, LUNNEW, VOLOLD, VOLNEW, CNOOLD,
  *   CNONEW, CATBLK, BUFER1, BUFER2, IERR)
Inputs:
   LUNOLD   I        LUN for old history file.
   LUNNEW   I        LUN for new history file.
   VOLOLD   I        Vol. number for old history file.
   VOLNEW   I        Vol. number for new history file.
   CNOOLD   I        Catalog slot number of old history file.
   CNONEW   I        Catalog slot number of new history file.
In/Out:
   CATBLK   I(256)   Catalog header of map for new file.
   BUFER1   I(256)   Work buffer, used for old file.
   BUFER2   I(256)   Work buffer, new file; must be used in
                     further HIADD calls until file is closed.
Output:
   IERR     I        Return error code: 0 => OK.
                        1 => could not open old history file.
                        2 => could not copy old history file.
                        3 => could not write time on new file
                        4 => could not create/open new HI file.
NOTE: IERR < 3 is a warning only, = 3 serious, = 4 a real problem.
Calling programs should ignore IERR < 3, branch to HICLOS of the
new HI file on IERR = 3, and skip over all HI stuff on IERR = 4.
```

3.16.22 H2CHR

Convert an AIPS HOLLERITH string to a Fortran CHARACTER variable. Blank fills the full OSTR variable.

```
H2CHR (NCH, INPNT, ISTR, OSTR)
Inputs:
   NCH    I       Number of characters
   INPNT  I       Start position in input string
   ISTR   H(*)    Input AIPS string
Output:
   OSTR   C*(*)   Output CHARACTER string
```

3.16.23 MAKOUT

MAKOUT applies the wild card standards to complete the preparation of the output file name parameters. Namely:

```
OUTS  <=  -1          becomes  OUTS = INSEQ
OUTN  =   ' '         becomes  OUTN = INN
          'yy*zz '    becomes  OUTN = INN   with first n characters
                      replaced by yy and last m chars with zz - if
                      yy or zz contain ?'s don't replace those char
                      positions
OUTCL =   '   '       becomes  OUTCL= DEFCLS
          'yy*zz '    becomes  OUTCL= DEFCLS with same as OUTN
```

```
        If the 1st character of OUTCL is a '_' then the default
        is replaced with INCL and the remaining 5 characters of
        OUTCL are used as normal.

MAKOUT (INN, INCL, INS, DEFCLS, OUTN, OUTCL, OUTS)
Inputs:
   INN     C*12     Input file name
   INCL    C*6      Input file class
   INS     I        Input file sequence number
   DEFCLS  C*6      Default output file class 6 packed chars
                    if = ' ', use task name
In/Out:
   OUTN    C*12     User-supplied OUTNAME adverb
   OUTCL   C*6      User-supplied OUTCLASS adverb
   OUTS    I        User-supplied OUTSEQ adverb in integer
NOTE: the actual Input file name parameters must be supplied, not
the user adverbs (which can themselves contain wild cards, pure
blank fields, zeros, and the like.
```

3.16.24 PRTLIN

PRTLIN handles actual printing on the line printer or CRT for tasks. For the CRT, it also handles page-full user communication.

```
PRTLIN (OUTLUN, OUTIND, DOCRT, NC, T1, T2, LINE, NLINE,
  *   IPAGE, SCRTCH, IERR)
Inputs:
   OUTLUN  I       LUN for print device (open)
   OUTIND  I       FTAB pointer for print device
   DOCRT   R       > 0. => use CRT, else line printer
   NC      I       Number characters in line
   T1      C*132   Page title line 1
   T2      C*132   Page title line 2
   LINE    C*132   Text line
In/out:
   NLINE   I       Number lines so far on page
                      > 1000 => just ask about continuing
                      = 999 => just start new page
   IPAGE   I       Current page number
                      = 0 => just start new page
Output:
   SCRTCH  C*(*)   Scratch core > 132
   IERR    I       Error code: 0 => OK, -1 user asks to quit
```

3.16.25 PSFORM

PSFORM prepares a string pattern array for use by CHWMAT (the wild card matching subroutine).

```
PSFORM (NC, PS, IPT)
Inputs:
   NC   I       Number characters in pattern possible
   PS   C*(*)   Pattern string
Output:
   IPT  I(NC)   Coded array:
                    value = -2 => position is *
```

```
                value = -1 => position is ?
                value = 0  => position is a blank
                value > 0  => there are IPT(i) real chars
                              including the present following
```

3.16.26 RELPOP

RELPOP places the specified return code in the appropriate location of the first record of the Task Data (TD) file. This will allow the calling program (AIPS, AIPSC, AIPSB, BATER) to resume normal operations.

```
RELPOP (RETCOD, SCRTCH, IERR)
Inputs:
   RETCOD  I       return code number
Outputs:
   SCRTCH  I(256)  scratch buffer
   IERR    I       error number: 0 -> ok
                             1,2 -> task not resumed
                               3 -> NPOPS out of range
                               4 -> parameter not passed
```

3.16.27 SCREAT

SCREAT is intended to replace all previous scratch file creation routines in AIPS (beginning on February 11, 1985). It uses the Common included via DFIL.INC and returns the scratch file disk and catalog number in variables SCRVOL(NSCR) and SCRCNO(NSCR), where NSCR is updated on successful creation. It attempts to avoid the disk used for the previously created scratch file.

```
SCREAT (SIZE, WBUFF, IERR)
Input:
   SIZE    I          Desired size in AIPS bytes
Output:
   WBUFF   I(512)     Scratch buffer    (NOTE 512 integers)
   IERR    I          error: 0 => ok
                         1 => catalog error in setting name
                         2 => catalog error on open
                         3 => CATIO error writing header to catlg
                         4 => No allowed disk with room
Commons:
   /MAPHDR/   in      scratch file image header - contents mostly
                      ignored
   /CFILES/   in/out  file info
Note: this common uses IBAD to specify BADDISKs which are avoided.
```

3.16.28 TABCOP

TABCOP copies Table extension file(s). The output file must be a new extension - old ones cannot be rewritten. The output file must be opened WRIT in the catalog and will have its CATBLK updated on disk.

```
TABCOP (TYPE, INVER, OUTVER, LUNOLD, LUNNEW, VOLOLD,
  *   VOLNEW, CNOOLD, CNONEW, CATNEW, BUFF1, BUFF2, IRET)
Inputs:
   TYPE     C*2     Extension file type (e.g. 'CC','AN')
   INVER    I       Version number to copy, 0 => copy all.
   OUTVER   I       Version number on output file, if more than one
                    copied (INVER=0) this will be the number of the
                    first file.  If OUTVER = 0, it will be taken as
```

```
                  1 higher than the previous highest version.
   LUNOLD   I     LUN for old file
   LUNNEW   I     LUN for new file
   VOLOLD   I     Disk number for old file.
   VOLNEW   I     Disk number for new file.
   CNOOLD   I     Catalog slot number for old file
   CNONEW   I     Catalog slot number for new file
In/out:
   CATNEW   I(256) Catalog header for new file.
Output:
   BUFF1    I(256) Work buffer
   BUFF2    I(256) Work buffer
   IRET     I      Return error code  0 => ok
                     1 => files the same, no copy.
                     2 => no input files exist
                     3 => failed
                     4 => no output files created.
                     5 => failed to update CATNEW
                     6 => output file exists
```

3.16.29 UVPGET

UVPGET determines pointers and other information from a UV CATBLK. The address relative to the start of a vis record for the real part for a given spectral channel (CHAN) and stokes parameter (ICOR) is given by NRPARM+(CHAN-1)*INCF+ABS(ICOR-ICOR0)*INCS+(IF-1)*INCIF

Single dish data, i.e. randomly sampled data in the image plane, is also recognized and ILOCU and ILOCV point to the longitude like and latitude like random parameters. Also a "BEAM" random parameter may be substitued for the "BASELINE" random parameter. The data type present may be determined from the common variable TYPUVD.

```
UVPGET (IERR)
Inputs: From common /MAPHDR/ (DCAT.INC)
   CATBLK   I(256)   Catalog block
   CATH     H(256)   same as CATBLK
   CATR     R(256)   same as CATBLK
   CATD     D(128)   same as CATBLK
Output: In common /UVHDR/ (DUVH.INC)
   SOURCE   C*8      Source name.
   ILOCU    I        Offset from beginning of vis record of U
                     or longitude for single dish format data.
   ILOCV    I        Offset from beginning of vis record of V
                     or longitude for single dish format data.
   ILOCW    I        Offset from beginning of vis record of W.
   ILOCT    I                    "                    Time
   ILOCB    I                    "                 Baseline
                                                   (or beam)
   ILOCSU   I                    "                Source id.
   ILOCFQ   I                    "                Freq id.
   JLOCC    I        0-rel. order in data of complex values
   JLOCS    I        Order in data of Stokes' parameters.
   JLOCF    I        Order in data of Frequency.
   JLOCR    I        Order in data of RA
   JLOCD    I        Order in data of dec.
   JLOCIF   I        Order in data of IF.
```

```
   INCS     I        Increment in data for stokes (see above)
   INCF     I        Increment in data for freq. (see above)
   INCIF    I        Increment in data for IF.
   ICORO    I        Stokes value of first value.
   NRPARM   I        Number of random parameters
   LREC     I        Length in values of a vis record.
   NVIS     I        Number of visibilities
   FREQ     D        Frequency (Hz)
   RA       D        Right ascension (1950) deg.
   DEC      D        Declination (1950) deg.
   NCOR     I        Number of correlators (Stokes' parm.)
   ISORT    C*2      Sort order 1st 2 char meaningful.
   TYPUVD   I        UV data type, 0=interferometer,
                       1=single dish unprojected,
                       2=single dish projected RA and Dec.
   IERR     I        Return error code: 0=>OK,
                       1, 2, 5, 7 : not all normal rand parms
                       2, 3, 6, 7 : not all normal axes
                       4, 5, 6, 7 : wrong bytes/value
```

3.16.30 ZDCHIN

Initialize the device characteristics common and the FCB's (file control blocks) in FTAB(*) for the maximum number of different file types that can be open at the same time. Initialize also other machine-dependent commons and the message common. Note that the task name is not set here.

ZDCHIN starts with hard-coded values. Then, if DODISK is true, resets those contained in the system parameter file. The utility program SETPAR is used to alter the system parameter file values.

Critical system constants (all "words" are local integers, all "bytes" are AIPS-bytes, i.e., 1/2 a local integer and on 64 bit architectures, double precision contructs should be preprocessed into their single precision counterparts):

```
ZDCHIN (DODISK, JOBLK)
Inputs:
   DODISK   L        Get SETPAR-controlled parameters from disk
Inputs from common: DMSG.INC
   TSKNAM   C*6      Task name if known - else ' ' (used in ABORT
                     handler mostly to separate standalones and
                     tasks)
Output:
   JOBLK    I(256)   I/O block - no longer used
Output in commons: DDCH.INC DMSG.INC
   all      ...      All values set to init except TSKNAM
```

3.16.31 ZTTYIO

Perform I/O to a terminal.

```
ZTTYIO (OPER, LUN, FIND, NCHARS, BUFF, IERR)
Inputs:
   OPER     C*4   Operation code 'READ' or 'WRIT'
   LUN      I     Logical unit number
   FIND     I     Index in FTAB to file control block for LUN
   NCHARS   I     # characters to transfer (<= 132)
In/out:
```

```
    BUFF     C*(*) I/O buffer containing characters (1-256)
  Output:
    IERR     I     Error return code: 0 => no error
                      1 => file not open
                      2 => input error
                      3 => I/O error
                      4 => end of file
```

Chapter 4

The AIPS Program

4.1 Overview

The AIPS program is the portion of the AIPS system with which the user normally interacts. The major functions of the AIPS program are: (1) prepare the parameters for and initiate the tasks which do most of the computations, (2) allow interactive use of TV and graphics devices, (3) provide limited direct analysis capability and (4) provide a high level of control logic to allow simple functions to be grouped into more complex functions (i.e., a programming language).

The basis of the AIPS program is the POPS (People Oriented Parsing Service) language processor. POPS is an interpretive language processor which can either accept statements for immediate execution or in the form of programs, called procedures, which are compiled and stored for later execution. Operations on data, images etc. are performed by means of "verbs" and "tasks". Verbs are operations which are done directly by the AIPS program and tasks are programs which are run asynchronously from AIPS. Both verbs and tasks are controlled by a set of global parameters called "adverbs". Verbs may change the values of adverbs whereas tasks cannot.

This chapter will attempt to describe the basic methods of the POPS processor and explain how to add new verbs and adverbs. The AIPS program does not know directly about tasks, so adding tasks requires no modifications to the AIPS program.

Other documentation about POPS processors may be found in a report by Jerome A. Hudson entitled "POPS People-Oriented Parsing Service Language Description and Program Documentation" and *POPS An Interactive Terminal Language with Applications in Radio Astronomy* by A. Sume, 1978, Internal Report no. 115, Research Laboratory of Electronics and Onsala Space Observatory, Chalmers University of Technology, Gothenburg, Sweden.

4.2 Structure of the AIPS Program

The basis of the AIPS program is a POPS processor which interprets user instructions and calls the relevant applications routines and spawns the desired tasks. Input to the POPS processor is in the form of statements which may do one of the following:

1. Modify an adverb value. This may be either by specifying a literal constant or an arithmetic, logical or character string expression.

2. Invoke an applications verb. These are the verbs which are specific to a given data analysis problem, such as displaying an image on the TV, rather than general control verbs such as loop control or sine functions etc.

3. Logic flow control. These statements control the execution of other statements, e.g., loop control, IF, THEN, ELSE etc.

4. Spawn tasks. Tasks are programs which take relatively long times to run and are executed asynchronously from AIPS. Communication between AIPS and tasks is primarily by disk files.

5. Prepare and edit procedures. POPS programs called procedures may be entered and compiled for later execution. These procedures may later be edited.

6. Prepare batch files. AIPS can run in a batch mode. To do this, the user enters and/or edits a list of commands in a batch file for later execution. This can be done either in the normal AIPS or a special batch version of AIPS named BATER.

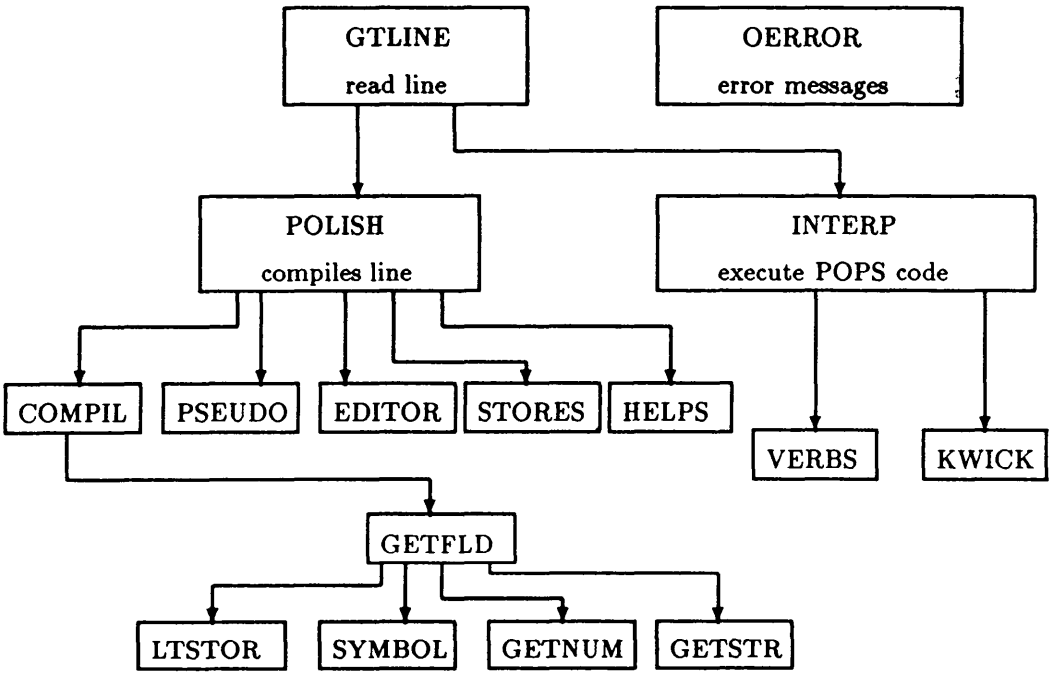
7. Store the current environment including all current procedures via the STORE or SAVE command. This environment is restored via the RESTORE or GET command.

4.2.1 The POPS processor

POPS uses an "inverse POLISH" stack to store operands and operation codes. Symbolics such as verb, adverb or procedure names are stored in a symbol table and each is identified by a type (TYPE) and a number (TAG). The initial entries in the symbol table and initial values of the adverbs are read from an external disk file which is prepared by the stand alone utility routine POPSGN. The various tables and stack pointers etc. are carried in common and the tables are equivalenced into an array known as the "K array".

Multiple statements, separated by semicolons, may be entered in a single line. There are a number of special verbs known as "pseudo" verbs which are executed as soon as they are encountered, causing any other instructions on the same line to be parsed in special fashions, ignored, or handled normally depending on the pseudoverb.

The basic structure of the AIPS program is very hierarchal. The main routine calls a startup routine, AIPBEG, a shutdown and error routine, AIPERR and a single routine GTLINE which controls the bulk of the processing. The structure of the basic routines in the POPS processor is shown in the following figure:



Structure of POPS

More details of each of these routines is given in the following:

- *GTLINE* is the main POPS routine. It causes lines to be read by PREAD, parsed and compiled or executed (in the case of pseudo verbs) by POLISH, and finally executed by INTERP. GTLINE returns only on error or requested termination of the program.
- *OERROR* displays an error message on the user terminal and resets POPS.
- *INTERP* causes POPS code to be executed by placing operands on the V and STACK stacks and calling VERBS and KWICK for verbs.
- *VERBS* calls the relevant applications verb routines based on the verb number. Functions are grouped together in routines named AUn. The appropriate routine is called with a branch code as an argument. This branch code in the verb number minus the first verb number in that AU routine plus one. The verb numbers are defined in an external file but VERBS must also know which verb numbers correspond to which AU routine.
- *KWICK* executes the basic POPS control verbs. These are the verbs which don't depend particularly on a given application but are frequently encountered.
- *POLISH* parses the character string entered by the user and translates it to Polish postfix notation. The result is a string of integers representing code for the POPS interpreter. Negative tokens are operand pointers while positive tokens are operator codes. The array A, which is equivalenced to STACK, holds the list of tokens; AP points to the most recent entry and SP points to the next entry. The operand pointers are to the location of the adverb or temporary variable in the K array.
- *COMPIL* does the actual interpretation of instructions and adds them to the stacks. COMPIL exits when a pseudo-verb or end-of-line is encountered.
- *PSEUDO* handles procedure and adverb declarations, sets up for the runtime operators IF, THEN, ELSE, WHILE (which require forward references and an additional cleanup pass) and the FINISH operator.
- *EDITOR* performs the operations required to begin and stop editing an existing procedure.
- *STORES* stores either the procedure source code, procedure object code, or handles the procedure source code.
- *HELPS* handles the user assistance facilities HELP, INPUT, EXPLAIN and RUN and other functions which require access to external text files. HELP lists symbols by type or lists a text file whose member name matches a user name. RUN sets the input to a specified member of a text file. This allows users to have personal strings of commands (e.g., procs, verbs, adverb settings). INPUTS lists the adverbs and their current values and brief descriptions on the terminal. Subroutine HELPS simply parses the user input in a more friendly fashion and places appropriate verb numbers and strings on the stacks.
- *GETFLD* finds the next non-blank character in the input buffer, KARBUF, and determines whether the token begun with that character is symbolic (1st char is A–Z), numeric (1st char is 0–9 or .), or hollerith (1st char is '). After the field length is found, appropriate calls are made to the symbol processing routine, number scanning routine, etc. Communication back to POLISH is via TYPE and TAG parameters determined by the processors SYMBOL, GETNUM, LTSTOR...
- *LTSTOR* searches the list of literals in the K array. If a matching literal is found, the TAG is returned. If not, a new one is generated and linked to the literal list. Note: a "literal" is a constant having either a numeric, character, or logical value.
- *SYMBOL* finds a symbol in the symbol list. The result is returned as TYPE and TAG through a common. If the routine is in the variable declaration mode, a new entry will be made in the symbol table if it does not already exist.
- *GETNUM* converts a character string into a DOUBLE PRECISION value.
- *GETSTR* obtains a character string from a buffer.

4.2.2 POPS commons

Most of the communication between POPS subroutines is by means of commons. As with most commons in the AIPS system, these commons are obtained by use of include files. The contents and uses of these commons are described in the following. The text of the include files is given at the end of this chapter.

DCON.INC

This common contains the basic POPS "memory" or K array, i.e., the symbol tables, adverb values, procedures etc. This common consists of equivalenced, INTEGER (K), REAL (C) and HOLLERITH (CH) arrays. Included in the latter part of this array are the adverb values. The variables used for the installed (predefined) adverbs are declared in the include DAPL.INC and follow a shortened declaration of the K array in DCON.INC. They specify the adverbs as equivalences to the K array beginning at K(KXORG+10).

User defined adverbs as well as as procedures and temporary literal values are stored beginning at K(301). The names of all symbolics (adverbs, verbs and procedures) are kept in a symbol table which is a linked list of symbol names containing the symbol type (TYPE), location in the K array (TAG) and the location of the array or string descriptor entries if appropriate. The first entry in the symbol table is pointed to by K(1) and a zero link indicates the last entry in the table. More details are given in later sections.

Literals (constants) are kept in a literal table which is also a linked list in the K array. The first entry is pointed to by K(4) and the last entry is pointed to by K(10). The literal table entry contains the type, length, and value of the literal.

The current compiled version of procedures is also kept in the K array. Each procedure may be divided into several blocks in the K array; the blocks are connected by forward links. A pointer is kept to the first location of the source version of the procedure in the LISTF array kept in the working memory file (kept on disk). The first block of a procedure is pointed to by the symbol table.

The different portions of the K array are used as follows:

```
K(1)      Symbol table link, points to first entry in the symbol table.
K(2)      Program link, points to first program (Procedure)
K(3)      Next free cell in K array to be allocated.
K(4)      Constants (literal) link, points to first entry in the
          literal table.
K(5)      Number of cells allocatable. Currently 14760.
K(6)      KTEMP, pointer to KKT (temporary value) area.
K(7)      Symbol protect limit.  Names with TAGs greater than this
          value may be changed.  This is used to protect
          procedures compiled by POPSGN.
K(8)      KXORG, pointer to KX array (data area). Currently 14761.

K(9)      Last symbol pointer.
K(10)     Last literal pointer
K(11-50)  Not used

                    KKT area, temporary storage for MODE=0

K(51)     Not used
K(52)     Program link
K(53)     Next free cell
K(54)     Constants link
K(55)     Number of cells allocatable
K(56-59)  not used
K(60)     Last constant pointer.

K(301...) Used for program storage, constants, symbols etc. for the
           remainder of the program position of the K array.
```

```
                    KX area, data storage

K(KXORG+0)    Not used
K(KXORG+1)    not used
K(KXORG+2)    Next free cell
K(KXORG+4)    Number of cells allocatable
K(KXORG+5)    not used
K(KXORG+6)    Highest adverb address in K not changeable by user.
K(KXORG+7->+9) not used
K(KXORG+10...) data storage.

                    Symbol table entries.

Word 1:   Link to next symbol table entry. Zero if end of list.
     2:   bits 2**0 to 2**3 = type.
          bits 2**4 to 2**15 = number of words in symbol
     3:   TAG (location in core where the data is kept)
     4:   Array data block counter if symbol is an array name,
          string, or procedure.
     5:   Bytes 1 and 2 of the name.
     6:   Bytes 3 and 4 of symbol name.
     7:   etc.

                    Array data blocks, define arrays.
                      (pointed to by symbol table)

Word 1:   Total array size
     2:   Number of dimensions
     3:   Initial index for first index
     4:   first dimension
     5:   Initial index for second dimension
     6:   etc.

                      Strings and string arrays
                     (pointed to by symbol table)

Word 1:  Total array size
     2:  Number of dimensions
     3:  1
     4:  no. floating point words in each element.
     5:  initial index for first subscript, if any
     6:  first subscript range, if any
     7:  etc.

                    Literal table entries

Word 1:   Pointer to next literal table entry, zero if last entry.
     2:   Bits 2**0 to 2**3 = type.  the types are 11=>floating point
               real (2 integer words), 14=>character string, 15=>
               logical constant (TRUE or FALSE)
          Bits 2**4 to 2**15 length of literal in integers.
     3:   First integer word in literal.
     4:   etc.
```

```
                Procedure storage (compiled code)
                  (pointed to by symbol table)

Word 1:  Link to next program block, zero if last.
     2:  Pointer to text array for purposes of listing.
     3:  first interpreter instruction.
     4:  etc.
         ...
     N:   1  An opcode of 1 terminates a block.  If the link to the
         next block is zero the procedure terminates.
```

/POPS/

This common carries the various stacks, stack pointers and other values. This common is obtained from include DPOP.INC. The contents of this common are described in the following:

```
 V(60)       R  Operand stack for REAL variables.
 XX          R  Intermediate REAL value
 KT          I  Starting location in K array of KKT (temporary) area.
 LPGM        I  Start address of an entry in the K array.  Used while
                allocating storage.
 LLIT        I  Not used
 LAST        I  Last token (opcode); if zero, finished with line.
                Used by COMPIL.
 IDEBUG      I  A debug flag used in various places.  If true
                (.GE.0) then debug info about POPS is given.
 MODE        I  The current mode of the POPS processor.
                   0 => immediate execution of an input line
                   1 => compile a procedure
                   2 => finishing a procedure
                   3 => editing a procedure
                   69 => adding a new symbol to symbol table
 IFFLAG      I  = 1 if an operator has been found in the current
                instruction; 0 otherwise.
 LINK        I  A link (pointer in K array)
 L           I  Another link (pointer in K array)
 NAMEP       I  Pointer in K array to a name in the symbol table.
 IP          I  Pointer in K array
 LP          I  Pointer in K array
 SLIM        I  Maximum allowed index in the stacks (currently 60)
 AP          I  Pointer to last entry in STACK
 BP          I  Pointer to last entry in CSTACK
 ONE         I  Pointer in C to value of 1.0
 ZERO        I  Pointer in C to value of 0.0
 TRUE        I  Pointer in C to value .TRUE.
 FALSE       I  Pointer in C to value .FALSE.
 STACK(60)   I  Instruction stack
 CSTACK(60)  I  Second (temporary) instruction stack
 SP          I  Pointer in STACK
 CP          I  Pointer in CSTACK
 SPO         I  Another pointer in STACK
 MPAGE       I  Number of pages (512 bytes) in the Memory file.
```

```
                (LISTF + K array)
LPAGE       I   Number of pages (512 bytes) of the memory file
                which contain LISTF (procedure source code)
```

DSMS.INC

This common contain various important values passed between routines. The contents of this common are described in the following.

```
KPAK(5)     H   Temporary array for storing a symbol name.
NKAR        I   The number of characters in KPAK
KBPTR       I   A character pointer in KARBUF, the input line buffer.
NEWCOD      I   Tag given by SYMBOL when allocating space for a new
                adverb.
TYPE        I   Symbol type.  See section on TAG and TYPE.
SKEL        R   Not used.
TAG         I   Symbol number.  See section on TAG and TYPE.
LEVEL       I   Precedence level bias.
LX          I   Number of integer words in character string X.
NEXTP       I   Precedence level of next item on A-stack.
X(25)       R   Temporary storage for character strings.
LOCSYM      I   Location in symbol or literal table of current
                symbol.
```

DIO.INC

This common contains short I/O buffers and related information. The contents of this common are described in the following.

```
IPT     C*1     Prompt character
NBYTES  I       Number of valid characters in KARBUF, number of last
                non-blank character.
KARBUF  C*80    A buffer containing the current input line.
JBUFF   C*80    Buffer used to read user input.
KARLIM  I       Number of characters in KARBUF
IUNIT   I       Input unit number for PREAD; 1=> user terminal, 2=>
                text editor, 3=>batch input file 4=>text entered
                during screen hold.
HOLDUF  C*80    Buffer for storing text entered during screen hold by
                SCHOLD.
```

4.2.3 TAG and TYPE

Adverbs, verbs, procedures etc. are all represented by symbolic names to the user. Internally, POPS identifies symbolics by TYPE and TAG. TYPE determines the type of symbolic (e.g., scalar, character string, verb etc.) and TAG is a label for the particular symbolic (e.g., a verb number). The TYPE of all symbols and the TAG of verbs are specified to POPSGN in the POPSDAT.HLP file. The TAG of an adverb is computed by POPS and is the start address of the value field.

The current list of symbolic types is given in the following list.

```
TYPE = 1    REAL scalar.
       2    REAL array.
```

```
 3   Procedure name.
 4   Verb name
 5   Pseudo verb name.
 6   Quit (used by POPSGN)
 7   Character string
 8   Element of character string
 9   substring of a character string
10   not used
11   Numeric constant
14   Character constant
15   Logical constant.
```

4.2.4 Error Handling

If a subroutine determines that an error condition exists, it sets the variable ERRNUM in INCLUDE DERR.INC to an error code known to the routine OERROR, increments ERRLEV in DERR.INC, and, if ERRLEV .LE. 5, copies the name of the subroutine into DERR.INC array PNAME. Following this, the subroutine returns. Thus, after each call to another AIPS subroutine, a subroutine should check ERRNUM and, if it is not zero, then that subroutine should increment ERRLEV, add its name to PNAME and exit. If GTLINE determines that an error has occurred, it returns to to the main AIPS routine which calls AIPERR which calls OERROR. This provides a traceback capability which can be exercised setting the AIPS adverb DEBUG to 1.0.

The messages displayed to the user corresponding to the defined values of ERRNUM are shown in the following:

ERRNUM	Message	ERRNUM	Message
1	'BLEW CORE! '	33	'CTLG PROBLEM'
2	'SYMBOL? '	34	'HISTORY FILE'
3	'BAD (OR) '	35	'FIT FAILS '
4	'LINE SIZE '	36	'NO PROC MODE'
5	'SYMBOL SIZE!'	37	'TEKS IN USE '
6	'ARRAY LIMITS'	38	'VERS TOO NEW'
7	'STACK LIMITS'	39	'NOT YES / NO'
8	'SYNTAX! '	40	'BATCH ERROR '
9	'CHARACTER? '	41	'NO RET CODE '
10	'PRINT '	42	'TASK ACTIVE '
11	'NO PROGRAM '	43	'NOT TASK '
12	'ARG LIST? '	44	'SYNC. FAILS '
13	'STRING SIZE '	45	'FILE MISSING'
14	'ALREADY DF '	46	'NO DESTROY '
15	'CONTROL '	47	'INVALID TAPE'
16	'LOGIC EXP? '	48	'TAPE PROBLEM'
17	'FOR---END? '	49	'TV PROBLEM '
18	'INF LOOP? '	50	'DISK PROBLEM'
19	'NO OPERATOR!'	51	'TV UNAVAILAB'
20	'DIVIDE BY 0 '	52	'OPEN FILE? '
21	'IF OR LOOP! '	53	'NOT IN RUN '
22	'READ '	54	'NOT INPUTS '
23	'DATA TYPE? '	55	'CREATE FILE?'
24	'USING WHAT? '	56	'CLOSE FILE? '
25	'PROTECTED! '	57	'PRINTER ERR.'
26	'SQRT NEGTIVE'	58	'FILE NOT OPN'
27	'NUMBER SIZE '	59	'TEXT READER '
28	'RUN IN A RUN'	60	'NOT IN BATCH'

```
  29       'LOG NEGATIVE'   61       'DISK PROBLEM'
  30       'VERS TOO OLD'   62       'BAD EXPONENT'
  31       'UNAVAILABLE!'   70       'ONLY IN PROC'
  32       'BOUNDARY LIM'   71       'NOT IN PROC '
 100      'ABORT!!!    '
```

4.2.5 Memory Files

The contents of the K array and LISTF, the source code for procedures, are initially obtained by AIPS from a memory file (type "ME"). The user may save the contents of LISTF and the K array by the pseudo verbs STORE or SAVE. The contents of these arrays can be recovered by the pseudo verbs RESTORE and GET. The working version of LISTF is stored at the beginning of the memory file.

The structure of the memory file is illustrated in the following. The size of the LISTF is given in pages (512 bytes) by variable LPAGE in common /POPS/ and the combined number of pages used by the LISTF and the K array are given by MPAGE in the same common. The current values of LPAGE and MPAGE are 16 and 90, respectively.

```
| Lw | K0 | L0 | K1 | L1 | K2 | L2 | ...

          where Lw = working version of LISTF
                K0 = startup version of the K array
                     initialized by POPSGN.
                L0 = startup version of the LISTF
                     initialized by POPSGN.
                K1 = user STORE area 1 for K array.
                L1 = user STORE area 1 for LISTF.
                K2 = user STORE area 2 for K array.
                L2 = user STORE area 2 for LISTF.
                etc.
```

4.2.6 Special modes

In the normal mode in which AIPS operates, the user types in instructions which are executed immediately. There are several alternate modes in which AIPS can operate. These modes are described briefly in the following sections.

RUN files

AIPS can be directed to read input from a disk text file which can be prepared with the local source editor. The instructions in such a file will be treated in the same fashion as if they were typed in through the terminal. RUN files are used mostly for permanent storage of complex procedures or other fixed data processing schemes. In AIPS, if IUNIT=3 in common /IO/, instructions are read from the RUN file until an end-of-file or an error is encountered.

Batch

AIPS can also be made to run in batch mode at a lower priority. To run AIPS batch, the user edits a file of instructions which are the same as would be given to an interactive AIPS. The major difference is that all tasks are run with DOWAIT=TRUE. This causes AIPS to suspend itself until the task is finished. Another difference is that tape drives, TVs, and graphics devices are not allowed for batch jobs.

The batch file can be created either by an interactive AIPS or a special version of AIPS, called BATER, for this purpose. Once the file is created, the SUBMIT verb sends it to AIPSC which checks the syntax.

One of several possible AIPSBs, the batch AIPSs, is scheduled to execute the batch file. Each of the three versions of AIPS (AIPS, the interactive program; AIPSC, the batch checker; and AIPSB, the batch AIPS) has a separate version of the subroutine VERBS called VERBS, VERBSC and VERBSB, respectively.

Procedures

POPS programs, called procedures, can be entered into the K array or edited by the user with the editor in the POPS processor. Alternately, procedures can be entered by POPSGN when creating the POPS memory files. As a procedure is entered, it is compiled line by line and the final compiled code is stored in the K array. Editing or modifying a procedure will cause the procedure to be recompiled and replaced in the K array.

The source version of the procedures is stored in an array called LISTF which is kept on disk in the current working memory file. All access to the source code causes this file to be read and/or written.

When procedures are recompiled and stored in the K array, the space for the old instructions is not recovered. The verb, COMPRESS, which was to recover this unused space, has never been implemented.

4.3 Example of the POPS processor

The following discussion of the POPS compiler and an example of its action is lifted (with some updates) from the 1978 Sume report.

4.3.1 The Compiler

POPS compiles expressions into reverse polish stacks, which can then be executed by the interpreter. Operators are translated into integers 1, 2, 3,... and operands into negative integers. The magnitudes of the negative integers are the addresses within the K array of the operands. Arithmetic operators carry a precedence which is used in converting expressions into polish sequences. Some operators, such as (and ; are used only at compile time to signal the elevation of precedence of operators, the end of a statement, etc.

The following table lists POPS operators and their precedence level.

Symbol	Meaning	Precedence
=	Store	1
!	Or	2
&	And	2
^	Not	2
=	Equal (as opposed to store)	3
>	Greater than	3
<	Less than	3
<=	Greater or equal	3
>=	Less or equal	3
<>	Not equal	3
TO	Loop control	4
:	Loop control	4
BY	Loop control	4
!!	String concatenation	4
+	Add	5
-	Subtract	5
SUBSTR	String extraction, insertion	5
*	Multiply	6
/	Divide	6
**	Exponentiate	7
-	Unary -	8
+	Unary +	0
Verbs ;,FOR,END,READ,TYPE,PRINT, RETURN, AND DUMP		0
All other verbs		9

Translation to polish form takes place in the routines POLISH and COMPIL as follows: Three push-down stacks, A, B, and BPR, hold operands, operators, and operator precedents respectively, while an expression is scanned from left to right. The expression is contained in the array KARBUF and the tokens are obtained from KARBUF by the subroutine GETFLD (in POLISH) called from COMPIL. Operands are placed on the A stack in order of appearance. Operators are placed on the B stack if their precedence (NEXTP) exceeds the precedence of the last operator on the stack, or if the B stack is empty. Using the BCLEAN subroutine, operators are taken off the B stack and pushed onto A if their precedence is equal to or greater than the precedence of the operator currently being scanned. This takes place until the top operator on the B stack has precedence lower than the one being scanned, or the B stack is emptied, whence the new operator is pushed onto the B stack, and its precedence onto the BPR stack at the corresponding position. If the "(" operator is encountered, the precedence of every subsequent operator is raised by an amount MAXLEV (=10) while ")" lowers the level by MAXLEV. The end of a statement "operator", the ";" operator, and others with which arithmetic expressions may be associated, such as TO, BY, THEN, ELSE, etc. are taken to have lowest possible precedence, so that they have the effect of emptying the B stack. We are then left with the polish sequence of operators and operands in the A stack. For example, the expression.

```
Y = A*(B*X + C);
```

would be translated with the following steps:

Step	Token	Prec(token)	A-stack	B-stack	BPR-stack
(1)	Y	...	(empty)	(empty)	(empty)
(2)	=	3	Y	(empty)	(empty)
(3)	A	...	Y	=	3
(4)	*	6	Y A	=	3
(5)	(	raise level	Y A	= *	3 6
(6)	B	...	--------- SAME ----------		
(7)	*	6+MAXLEV	Y A B	= *	3 6
(8)	X	...	Y A B	= * *	3 6 6+MAXLEV
(9)	+	5+MAXLEV	Y A B X	= * *	3 6 6+MAXLEV
(10)	C	...	Y A B X *	= * +	3 6 5+MAXLEV
(11)	)	decrement	Y A B X * C	= * +	3 6 5+MAXLEV
(12)	;	0	------------ SAME ------------		
(13)	Final result		Y A B X * C + * =	(empty)	(empty)

4.3.2 The Interpreter

The POPS interpreter executes polish postfix code left by the POPS compiler. To do so requires 3 run-time stacks: the main stack (STACK), the control stack (CSTACK) and a value stack (V).

The main stack holds operand addresses (tags). Corresponding to each operand, the appropriate position in the value stack is loaded with a floating point number, found in core at the stack address. This number may or may not be meaningful, depending on the type of data kept at that address. Operators will make use of the address or value depending on which is appropriate.

The control stack is used to save the run-time location counter (L) and the program chunk link (LINK), together with saved stack pointers, etc. While the main stack could be so used, it was felt that greater reliability would ensue if the control stack were kept separate, guarding from user-caused stack errors (such as leaving garbage on the main stack). Operations using the control stack require an authentication code to appear on the top of the stack before they are activated.

The interpreter expects all operands to be negative integers; all operators, save 0 to be positive (0 is considered a legitimate operand). Operands will be pushed onto the main stack. The value stack, described above, holds intermediate results of computations, as well as the contents of memory when the stack was loaded.

An example, using the arithmetic expression described in the polish compile segment:

```
Source code:  Y = A * (B * X + C)

          Compiled code
          ---------------
           (1)  -addr. of   Y
           (2)  -addr. of   A
           (3)  -addr. of   B
           (4)  -addr. of   X
           (5)  +TAG of     * operator
           (6)  -addr. of   C
           (7)  +TAG of     + operator
           (8)  +TAG of     * operator
           (9)  +TAG of     = operator

Execution: Suppose A = 1.5, B = 2.5, C = 3.5, X = 10.0

          Token being
 Step     executed          stack          V
-----     -----------      -------       -------

(1)          Y             (empty)       (empty)

(2)          A               Y           ********

(3)          B               Y           ********
                             A             1.5

(4)          X               Y           ********
                             A             1.5
                             B             2.5

(5)          *               Y           ********
```

```
                          A          1.5
                          B          2.5
                          X         10.0

(6)          C            Y      ********
                          A          1.5
                     *********       25.0

(7)          +            Y      ********
                          A          1.5
                     *********       25.0
                          C          3.5

(8)          *            Y      *********
                          A          1.5
                     **********      28.5

(9)          =            Y      *********
                     **********      42.75

(10)       finish       (empty)    (empty)
```

4.4 Installing new VERBS

To install a new verb in AIPS several actions are required.

1. Enter the new verb in POPSDAT.HLP and run POPSGN. The new verb will probably be TYPE 4 and should be assigned a verb number (TAG) greater than 100; making sure the verb number is not already used. It should be noted that contiguous groups of verb numbers will use the same AU routine. If the new verb is similar to existing verbs it should be put in the same AU routine if possible.

2. Create or modify an AU routine to perform the desired function. If there are available verb numbers in the range available to the relevant AU routine, then the function can be added to that AU routine. If not, then a new AU routine is required. Note that the branch code sent to the AU routine is the verb number (one) relative to the first verb number in that AU routine. If the verb requires more than a few lines of Fortran, the AU routine should call a subroutine to do the work.

3. Modify VERBS, if necessary, to call the necessary AU routine when it is given the new verb number (J in VERBS). The range of verb numbers in each routine is defined in the arrays IAB and IAE. If new AU routines are added the dimensions of IAB and IAE should be changed and the upper limit on the DO loop index for the loop terminating at statement label 5 should be changed. The computed GO TO in this loop should be modified to include the new AU routine. New AU routines should be added at the end of the list for simplicity. Note that there are three versions of VERBS (VERBS, VERBSC, and VERBSB) for the interactive AIPS, the batch AIPS checker program, and batch AIPS respectively. All three must have corresponding changes although an error return may be desired for the two batch versions in the implementation of a new verb.

4. Compile the necessary subroutines and add them to the AIPS program subroutine library.

5. Recompile and link edit AIPS.

6. Create a HELP file for the verb in the same manner as for a task. Verbs will work without a HELP file, but it is much friendlier to write one.

As a convenience for developing new verbs, nine temporary verbs are available, T1VERB, T2VERB, ..., and T9VERB (verb numbers 900–908) These are accessible through the routine AUT. To use one of these verbs all that is necessary is to modify AUT, recompile it, replace it in the AIPS program subroutine library (COMRPL), and recompile AIPS and relink it. Once verbs are tested, they should be moved to a more permanent AU routine if they meet AIPS' portability standards.

The branch code sent to the AU routine is (one) relative to the first verb number in that AU routine. If the verb has one or more arguments, they will be found in the value stack V in common /POPS/ in the reverse of the order in which they were specified. Real values can then be obtained as in the following example:

```
      SUBROUTINE TESTXX
C-----------------------------------------------------------------------
C   Routine to average the top two numbers on the V stack.
C   This routine is designed to be run from VERBS rather than KWICK,
C   that is, it should be called from an AU routine.
C-----------------------------------------------------------------------
      REAL      V1, V2, RESULT
      INTEGER   POTERR
      CHARACTER PRGNAM*6
      PARAMETER ( PRGNAM = 'TESTXX')
      INCLUDE 'INCS:DPOP.INC'
      INCLUDE 'INCS:DERR.INC'
C-----------------------------------------------------------------------
C                                     Set potential error number,
C                                     7 = 'STACK LIMIT'
      POTERR = 7
C                                     Check that stack not
C                                     exhausted.
      IF (SP.GE.2) THEN
C                                     Get values from stack.
         V1 = V(SP-1)
         V2 = V(SP)
C                                     Average.
         RESULT = (V1 + V2) / 2.0
C                                     For two operands change SP and,
C                                     STACK, for one don't change
C                                     SP or STACK.
         SP = SP - 1
         STACK(SP) = 0
C                                     If the verb returns a value,
C                                     RESULT, do the following.
         V(SP) = RESULT
C                                     Set error code
      ELSE
         ERRNUM = POTERR
C                                     Fill in /ERRORS/.
         ERRLEV = ERRLEV + 1
         IF (ERRLEV.LE.5) PNAME(ERRLEV) = PRGNAM
         END IF
C                                     Return
 999  RETURN
      END
```

The stack contents are as follows when a verb is called with an immediate argument:

1. For a real scalar including a subscripted real array adverb,

```
SP = 1     STACK(SP) = TAG     V(SP) = C(TAG) (=value)
```

2. For an array adverb,

```
SP = 1     STACK(SP) = TYPE     V(SP) may be ignored
     2                 M
     3                 TAG
     4                 2
where for TYPE = 2,7   M = K array pointer to array
                           descriptor block,
               14      = number of characters,
                9      = 100 * character offset +
                         # characters
```

Adverbs may be accessed by name using the name as defined in the include DAPL.INC. Note that the order of adverbs is really defined in the POPSDAT.HLP file and the order in DAPL.INC must correspond exactly. Also, all adverbs are of Fortran data type REAL, although they may contain character strings. Note that character strings are stored as HOLLERITH which are allocated storage space as (NCHAR+3)/4 REAL locations.

4.5 Installing new ADVERBS

New, temporary, adverbs can be created in an executing AIPS task by SCALAR, ARRAY or STRING statements in a procedure. Permanent installation of an adverb requires entering it in POPSDAT.HLP, running POPSGN to update the memory files, and adding a variable into the declarations in common /CORE/ in the include DAPL.INC. The new adverbs should be entered in the same relative location among the other adverbs in common as in the POPSDAT file. At the end is, of course, best. The adverb value will be kept in this variable and is, therefore, directly available to verbs. A HELP file should be written for any permanent adverb.

4.6 POPSGN

The initial contents of the POPS memory files, and hence the LISTF and K arrays, are set by the stand alone utility program POPSGN. This program takes as input the file POPSDAT.HLP.

4.6.1 Function

The function of POPSGN is to initialize the contents of LISTF (the source code for procedures) and the K array when AIPS starts up by storing the contents in the POPS memory ("ME") files. This program is normally found in the same place as the AIPS program itself and asks for instructions directly from the key board. When the program begins it asks:

```
ENTER NPOPS1,NPOPS2,IDEBUG,MNAME,VERSION
```

The response is in free format (blanks between entries) and should be as follows:

```
NPOPS1  The lowest POPS number for this run of POPSGN, this
        is normally 1.

NPOPS2  The highest POPS number for this run of POPSGN, this
        is normally the highest POPS number run = No.
        interactive POPS + number of batch queues + 1.

IDEBUG  If not 0, POPSGN will give lots of debug messages.
        Use 0.

MNAME   The name of the file in the HELP area that contains
        the input file for POPSGN.  This is normally
        POPSDAT.HLP; type only 'POPSDAT'.

VERSION  This specifies the version of AIPS to have the
         memory files updated.  Normally this is blank
         which will update the 'TST' area; 'NEW' and 'OLD'
         are also understood by POPSGN.
```

After POPSGN has digested POPSDAT.HLP it will return a ">" prompt. Type a blank line to terminate the input and POPSGN will update the memory files.

4.6.2 POPSDAT.HLP

The bulk of the definitions of verbs, adverbs, and standard procedures are defined in the POPSDAT file. A "C-" in columns one and two indicate a comment line. A "/" character conventionally indicates the beginning of an end-of-line comment which must begin after column 44. The names of symbols begin in column 1 with no embedded blanks and may have no more than 8 characters. The POPSDAT file is read with a (A8,1X,I3,1X,I3,1X,I4,1X,I4,2(1X,F7.2)) format.

The first portion of the POPSDAT file defines the POPS verbs. Most of these verbs and pseudo verbs with verb numbers (TAG) less than 100 reside in the AIPS routine KWICK. Verb numbers greater than 100 are all in AU routines called by VERBS. The values following the symbol name are (1) the number of characters in the symbol name, (2) the symbol type (4 or 5 for verbs and pseudo verbs) and (3) the TAG, in this case the verb number. The end-of-line comments for verbs with numbers (TAG) greater than 100 tell the AU routine in which that verb is found.

Following the verbs come the adverb definitions. The values following the symbol name are: (1) the number of characters in the symbol name, (2) the symbol type (see the section of TYPEs and TAGs). For scalar, real adverbs (TYPE 1) the next two integer fields are blank and the following REAL field (F7.0) is taken to be the initial value of that scalar.

For real arrays (TYPE 2), the first value past the TYPE field is the number of dimensions (1 or 2), the next integer field is blank and the following one or two REAL (F7.1) fields give the number of positions in each of the one or two dimensions.

For character string variables (TYPE 7) the first integer field past the TYPE is the number of dimensions; where the characters in the each string constitute the first dimension. Thus there can only be single dimensional character string array adverbs. The next integer field is blank and the next REAL (F7.0) field is the number of characters in the string. For character string arrays the following REAL (F7.0) field is the second dimension, i.e., the number of elements in the array.

An adverb named QUIT with TYPE = 6 tells POPSGN that all verb and adverb definitions have been read. Following this, normal POPS commands may be entered and the definitions of the standard procedures are normally entered here. A "*" in column 1 indicates a POPS comment line. The end of file terminates the input.

The current contents of POPSDAT is shown in the following:

```
; POPSDAT
;---------------------------------------------------------------
;! lists all POPS symbols, used to create them in MEmory files
;# List POPS
;  This software is the subject of a User agreement and is
;  confidential in nature. It shall not be sold or otherwise
;  made available or disclosed to third parties.
;---------------------------------------------------------------
POPSDAT   LLLLLLLLLLLLUUUUUUUUUUUU CCCCCCCCCCCCCCCCCCCCCCCCCCCC
---------------------------------------------------------------
C-                                        This module is POPSDAT.
,           1   4    1                                --|
(           1   4    2                                  |
)           1   4    3                                  |
=           1   4    4                                  |
+           1   4    5                                  |
-           1   4    6                                  | subtract
*           1   4    7                                  |
/           1   4    8                                  |
**          2   4    9                                  |
>           1   4   10                                  |
<           1   4   11                                  |
+           1   4   12                                  |
-           1   4   13                                  | unary
^           1   4   14                                  |
TO          2   4   15                                  |
:           2   4   15                                  |
BY          2   4   16                                  |
=           1   4   17                                  | logical
!           1   4   18                                  |
&           1   4   19                                  |
;           1   4   20                                  |
FOR         3   4   21                                  |
END         3   4   22                                  |
READ        4   4   23                                  |
TYPE        4   4   24
PRINT       5   4   24                                  |
RETURN      6   4   25
LENGTH      6   4   26                                  |
C-              27                              | res array equates
C-RUN       3   4   28                                  |
C-EXIT      4   4   29                                  |
C-RESTART   7   4   30                                  |
LOG         3   4   31
LN          2   4   32
MOD         3   4   33
MODULUS     7   4   34
ATAN2       5   4   35
SIN         3   4   36
COS         3   4   37
TAN         3   4   38
ATAN        4   4   39
SQRT        4   4   40
DUMP        4   4   41
```

```
<=            2   4   42                  |
>=            2   4   43                  |
<>            2   4   44                  |
EXP           3   4   45
SUBSTR        6   4   46
!!            2   4   47
CHAR          4   4   48
VALUE         5   4   49
MSGKILL       7   5   50            --| PSEUDO
PROCEDURE     9   5   51            --|
PROC          4   5   51              |
ARRAY         5   5   52              |
ELSE          4   5   53              |
THEN          4   5   54              |
FINISH        6   5   55              |
DEBUG         5   5   56              |
IF            2   5   57              |
STRING        6   5   58
WHILE         5   5   59
SCALAR        6   5   60

EDIT          4   5   61              | EDITOR
ENDEDIT       7   5   62            --|
MODIFY        6   5   63
C-storecode       5   64            --| reserved
STORE         5   5   65              |
RESTORE       7   5   66              |
SAVE          4   5   67              |
GET           3   5   68              |
LIST          4   5   69              | -- STORES
CORE          4   5   70              |
SCRATCH       7   5   71              |
COMPRESS      8   5   72              |
C-endmodify       5   73            --| reserved
SHOW          4   5   76              | -- HELPS
TELL          4   5   77              | -- HELPS
ISBATCH       7   5   78              | -- PSEUDO
ERASE         5   5   79              | -- EDITOR
RUN           3   5   80              | -- HELPS
HELP          4   5   81            --|
INP           3   5   82            --|
INPUTS        6   5   83
GO            2   5   84
TGET          4   5   85
SGDESTR       7   5   86
ABORTASK      8   5   87
TPUT          4   5   88
WAITTASK      8   5   89
EXPLAIN       7   5   90
CEIL          4   4   91
FLOOR         5   4   92
ABS           3   4   93
MAX           3   4   94
MIN           3   4   95
```

```
C-                          96          --| res: END
C-                          97              res: WHILE
C-                          98              res: SUBS
C-                          99              res: NOP

PRTMSG      6   4  100                     | AU1
EXIT        4   4  101
RESTART     7   4  102
CLRMSG      6   4  103
C-HELP          4  110                     | AU1A
C-INP           4  111
C-INPUTS        4  112
C-EXPLAIN       4  113
C-SHOW          4  114
C-GO        2   4  120                     | AU2
SPY         3   4  121
C-WAITTASK      4  122
C-ABORTASK  8   4  123
C-TPUT      4   4  124
C-TELL      4   4  125
STQUEUE     7   4  126
SETDEBUG    8   4  127
C-TGET          4  130                     | AU2A
C-SGDESTR       4  131
TGINDEX     7   4  132
SGINDEX     7   4  133
CATALOG     7   4  150                     | AU3
MCAT        4   4  151
IMHEADER    8   4  152
ZAP         3   4  153
UCAT        4   4  154
QHEADER     7   4  155
PCAT        4   4  156
FREESPAC    8   4  160                     | AU3A
ALLDEST     7   4  161
TIMDEST     7   4  162
SAVDEST     7   4  163
SCRDEST     7   4  164
RENUMBER    8   4  170                     | AU3B
RECAT       5   4  171
TPHEAD      6   4  180                     | AU4
AVFILE      6   4  181
AVMAP       5   4  182
REWIND      6   4  183
AVEOT       5   4  184
MOUNT       5   4  185
DISMOUNT    8   4  186
TVINIT      6   4  200                     | AU5
TVCLEAR     7   4  201
GRCLEAR     7   4  202
TVON        4   4  203
TVOFF       5   4  204
GRON        4   4  205
GROFF       5   4  206
```

```
TV3COLOR     8   4  207
TVPOS        5   4  208
IMXY         4   4  209
IMPOS        5   4  210
TVNAME       6   4  211
CURBLINK     8   4  212
TVLOD        5   4  220                    | AU5A
TVROAM       6   4  221
SETROAM      7   4  222
REROAM       6   4  223
TVLABEL      7   4  240                    | AU5B
TVWLABEL     8   4  241
TVANOT       6   4  242
TVWEDGE      7   4  250                    | AU5C
IMWEDGE      7   4  251
WEDERASE     8   4  252
IMERASE      7   4  253
TVWINDOW     8   4  254
TVBOX        5   4  255
TVSLICE      7   4  256
REBOX        5   4  257

TVMOVIE      7   4  260                    | AU5D
REMOVIE      7   4  261
TVCUBE       6   4  262
OFFPSEUD     8   4  280                    | AU6
OFFZOOM      7   4  281
OFFSCROL     8   4  282
TVZOOM       6   4  283
TVSCROL      7   4  284
TVPSEUDO     8   4  285
TVHUEINT     8   4  286
OFFTRAN      7   4  290                    | AU6A
TVTRANSF     8   4  291
TVBLINK      7   4  292
TVMBLINK     8   4  293
TVLUT        5   4  294
TVMLUT       6   4  295
TVSPLIT      7   4  296
CURVALUE     8   4  300                    | AU6B
C-TVALL        5   4  305                  | AU6C
TVFIDDLE     8   4  306
TVSTAT       6   4  310                    | AU6D
IMSTAT       6   4  311
PRTHI        5   4  330                    | AU7
RENAME       6   4  331
RESCALE      7   4  332
CLRSTAT      7   4  333
AXDEFINE     8   4  334
ALTDEF       6   4  335
ALTSWTCH     8   4  336
CELGAL       6   4  337
ADDBEAM      7   4  340                    | AU7A
PUTHEAD      7   4  341
```

```
GETHEAD     7   4  342
PUTVALUE    8   4  343
CLRNAME     7   4  360                    | AU8
GETNAME     7   4  361
GET2NAME    8   4  362
GET3NAME    8   4  363
EXTDEST     7   4  364
CLR2NAME    8   4  365
CLR3NAME    8   4  366
EGETNAME    8   4  367
GETONAME    8   4  368
CLRONAME    8   4  369
EXTLIST     7   4  370                    | AU8A
MAXFIT      6   4  390                    | AU9
IMVAL       5   4  391
QIMVAL      6   4  392
TKPOS       5   4  400                    | AU9A
TKVAL       8   4  401
TKXY        4   4  402
TKSLICE     7   4  410                    | AU9B
TKASLICE    8   4  411
TKMODEL     7   4  412
TKAMODEL    8   4  413
TKRESID     7   4  414
TKARESID    8   4  415
TKGUESS     7   4  416
TKAGUESS    8   4  417
TKSET       5   4  420                    | AU9C
TK1SET      6   4  421
SUBMIT      6   4  440                    | AUA
BATCH       5   4  441                    | AUB
BATEDIT     7   4  442
UNQUE       5   4  443
BATCLEAR    8   4  444
BATLIST     7   4  445
QUEUES      6   4  446
JOBLIST     7   4  447
BAMODIFY    8   4  448
GRIPE       5   4  460                    | AUC
GRINDEX     7   4  461
GRLIST      6   4  462
PASSWORD    8   4  463
GRDROP      6   4  464
T1VERB      6   4  900                    | AUT
T2VERB      6   4  901
T3VERB      6   4  902
T4VERB      6   4  903
T5VERB      6   4  904
T6VERB      6   4  905
T7VERB      6   4  906
T8VERB      6   4  907
T9VERB      6   4  908

USERID      6   1          0.00
```

```
INNAME       6   7    1        12.00
INCLASS      7   7    1         6.00
INSEQ        5   1              0.00
INDISK       6   1              0.00
INTYPE       6   7    1         2.00
IN2NAME      7   7    1        12.00
IN2CLASS     8   7    1         6.00
IN2SEQ       6   1              0.00
IN2DISK      7   1              0.00
IN2TYPE      7   7    1         2.00
IN3NAME      7   7    1        12.00
IN3CLASS     8   7    1         6.00
IN3SEQ       6   1              0.00
IN3DISK      7   1              0.00
IN3TYPE      7   7    1         2.00
OUTNAME      7   7    1        12.00
OUTCLASS     8   7    1         6.00
OUTSEQ       6   1              0.00
OUTDISK      7   1              1.00
OUT2NAME     8   7    1        12.00
OUT2CLAS     8   7    1         6.00
OUT2SEQ      7   1              0.00
OUT2DISK     8   1              1.00
INEXT        5   7    1         2.00
IN2EXT       6   7    1         2.00
IN3EXT       6   7    1         2.00
INVERS       6   1              0.00
IN2VERS      7   1              0.00
IN3VERS      7   1              0.00
BADDISK      7   2    1        10.00
INTAPE       6   1              1.00
OUTTAPE      7   1              1.00
NFILES       6   1              0.00
NMAPS        5   1              0.00
TASK         4   7    1         8.00
DOWAIT       6   1             -1.00
PRIORITY     8   1              0.00
BLC          3   2    1         7.00
TRC          3   2    1         7.00
XINC         4   1              1.00
YINC         4   1              1.00
PIXXY        5   2    1         7.00
PIXVAL       6   1              0.00
PIXRANGE     8   2    1         2.00
FACTOR       6   1              0.00
OFFSET       6   1              0.00
TVBUT        5   1              0.00
XTYPE        5   1              5.00

XPARM        5   2    1        10.00
YTYPE        5   1              5.00
YPARM        5   2    1        10.00
OPCODE       6   7    1         4.00
FUNCTYPE     8   7    1         2.00
```

ROTATE	6	1		0.00	
GAIN	4	1		0.10	
NITER	5	1		0.00	
FLUX	4	1		0.00	
OBJECT	6	7	1	8.00	
QUAL	4	1		-1.00	
STOKES	6	7	1	4.00	
BAND	4	7	1	4.00	
TVCHAN	6	1		1.00	
GRCHAN	6	1		0.00	
TVLEVS	6	1		256.00	
TVCORN	6	2	1	2.00	
COLORS	6	1		0.00	
TVXY	4	2	1	2.00	
DOTV	4	1		-1.00	
BATQUE	6	1		2.00	
BATFLINE	8	1		0.00	
BATNLINE	8	1		0.00	
JOBNUM	6	1		0.00	
LTYPE	5	1		3.00	
PLEV	4	1		0.00	
CLEV	4	1		0.00	
LEVS	4	2	1	30.00	
XYRATIO	7	1		0.00	
DOINVERS	8	1		-1.00	
DOCENTER	8	1		1.00	
ZXRATIO	7	1		0.25	
SKEW	4	1		45.00	
DOCONT	6	1		1.00	
DOVECT	6	1		1.00	
ICUT	4	1		0.10	
PCUT	4	1		0.10	
DIST	4	1		3.00	
IMSIZE	6	2	1	2.00	
CELLSIZE	8	2	1	2.00	
SHIFT	5	2	1	2.00	
SORT	4	7	1	2.00	
UVTAPER	7	2	1	2.00	
UVRANGE	7	2	1	2.00	
UVWTFN	6	7	1	2.00	
UVBOX	5	1		0.00	
DOGRIDCR	8	1		1.00	
ZEROSP	6	2	1	5.00	
BITER	5	1		0.00	
BMAJ	4	1		0.00	
BMIN	4	1		0.00	
BPA	3	1		0.00	
NBOXES	6	1		0.00	
BOX	3	2	2	4.00	10.00
DOEOF	5	1		1.00	
NDIG	4	1		0.00	
DOCAT	5	1		1.00	
DOHIST	6	1		-1.00	
BDROP	5	1		0.00	

EDROP	5	1		0.00	
ASPMM	5	1		0.00	
MINPATCH	8	1		51.00	
APARM	5	2	1	10.00	
BPARM	5	2	1	10.00	
GPOS	4	2	2	2.00	4.00
GMAX	4	2	1	4.00	
GWIDTH	6	2	2	3.00	4.00
DOPOS	5	2	2	2.00	4.00
DOMAX	5	2	1	4.00	
DOWIDTH	7	2	2	3.00	4.00
NGAUSS	6	1		0.00	
TRANSCOD	8	7	1	14.00	
AXREF	5	1		1.00	
NAXIS	5	1		3.00	
AXINC	5	1		0.00	
AXVAL	5	2	1	2.00	
AXTYPE	6	7	1	8.00	
DOSLICE	7	1		1.00	
DOMODEL	7	1		-1.00	
DORESID	7	1		-1.00	
ROMODE	6	1		0.00	
DETIME	6	1		0.00	
DOCRT	5	1		-1.00	
CHANNEL	7	1		0.00	
CPARM	5	2	1	10.00	
DPARM	5	2	1	10.00	
DOALIGN	7	1		1.00	
NPOINTS	7	1		1.00	
AX2REF	6	1		0.00	
DOALL	5	1		-1.00	
TXINC	5	1		1.00	
TYINC	5	1		1.00	
TBLC	4	2	1	7.00	
TTRC	4	2	1	7.00	
VERSION	7	7	1	48.00	
DOEOT	5	1		1.00	
DOSTOKES	8	1		-1.00	
PRTLEV	6	1		0.00	
DOARRAY	7	1		-1.00	
ZINC	4	1		1.00	
TZINC	5	1		1.00	
BCHAN	5	1		1.00	
ECHAN	5	1		0.00	
RESTFREQ	8	2	1	2.00	
INFILE	6	7	1	48.00	
IN2FILE	7	7	1	48.00	
OUTFILE	7	7	1	48.00	
DENSITY	7	1		1600.00	
KEYWORD	7	7	1	8.00	
KEYVALUE	8	2	1	2.00	
KEYSTRNG	8	7	1	16.00	
BCOUNT	6	1		1.00	

ECOUNT	6	1		0.00	
NCOUNT	6	1		0.00	
DOTABLE	7	1		1.00	
DOTWO	5	1		-1.00	
COPIES	6	1		1.00	
PRNUMBER	8	1		0.00	
PRTIME	6	1		0.00	
PRTASK	6	7	1	5.00	
CTYPE	5	2	1	4.00	
PIXAVG	6	1		0.00	
PIXSTD	6	1		0.00	
DOCIRCLE	8	1		-1.00	
CHINC	5	1		1.00	
NFIELD	6	1		1.00	
FLDSIZE	7	2	2	2.00	16.00
RASHIFT	7	2	1	16.00	
DECSHIFT	8	2	1	16.00	
PHAT	4	1		0.00	
GAINERR	7	2	1	30.00	
TIMSMO	6	2	1	30.00	
DOOUTPUT	8	1		-1.00	
DOCONCAT	8	1		-1.00	
DONEWTAB	8	1		1.00	
DOCONFRM	8	1		-1.00	
DOALPHA	7	1		-1.00	
ERROR	5	1		-1.00	
GRNAME	6	7	1	20.00	
GRADDRES	8	7	1	48.00	
GRPHONE	7	7	1	16.00	
SLOT	4	1		1.00	
VLAOBS	6	7	1	6.00	
VLAMODE	7	7	1	2.00	
CMETHOD	7	7	1	4.00	
CMODEL	6	7	1	4.00	
BCOMP	5	2	1	16.00	
NCOMP	5	2	1	16.00	
LPEN	4	1		3.00	
PRSTART	7	1		1.00	
OPTYPE	6	7	1	4.00	
DOWEDGE	7	1		1.00	
SOURCES	7	7	2	16.00	30.00
CALSOUR	7	7	2	16.00	30.00
TIMERANG	8	2	1	8.00	
SUBARRAY	8	1		1.00	
BIF	3	1		0.00	
EIF	3	1		0.00	
ANTENNAS	8	2	1	50.00	
BASELINE	8	2	1	50.00	
DOCALIB	7	1		1.00	
INTERPOL	8	7	1	4.00	
SMOTYPE	7	7	1	4.00	
INTPARM	7	2	1	3.00	
FLAGVER	7	1		0.00	
GAINVER	7	1		0.00	

```
GAINUSE      7   1              0.00
REASON       6   7    1        24.00
SAMPTYPE     8   7    1         4.00
CODETYPE     8   7    1         4.00
NOISE        5   2    1        64.00
PBSIZE       6   2    1        64.00
OUTVERS      7   1              0.00
DOCELL       6   1              1.00
PIX2XY       6   2    1         7.00
PIX2VAL      7   1              0.00

STFACTOR     8   1              0.00
CUTOFF       6   1              0.00
OPTELL       6   7    1         4.00
FORMAT       6   1              0.00
BLVER        5   1             -1.00
BPVER        5   1             -1.00
ANTWT        5   2    1        30.00
SOLINT       6   1              0.00
CALCODE      7   7    1         4.00
REFANT       6   1              0.00
SMODEL       6   2    1         7.00
SOLTYPE      7   7    1         4.00
SOLMODE      7   7    1         4.00
SOLCON       6   1              0.00
WTUV         4   1              0.00
DODELAY      7   1             -1.00
SYSVEL       6   1              0.00
VELDEF       6   7    1         8.00
VELTYP       6   7    1         8.00
RESTFREQ     8   2    1         2.00
DOHMS        5   1             -1.00
BLOCKING     8   1              1.00
PMODEL       6   2    1         7.00
DOPOL        5   1             -1.00
DOBAND       6   1             -1.00
SMOOTH       6   2    1         3.00
DOUVCOMP     8   1             -1.00
REWEIGHT     8   2    1         2.00
REFDATE      7   7    1         8.00
SELBAND      7   1             -1.00
SELFREQ      7   1             -1.00
FREQID       6   1             -1.00
CHANSEL      7   2    2         3.00   10.00
FQTOL        5   1             -1.00
C-   Adverbs below are dummys for testing.
STRA1        5   7    1         4.00
STRA2        5   7    1         8.00
STRA3        5   7    1        12.00
STRB1        5   7    1         4.00
STRB2        5   7    1         8.00
STRB3        5   7    1        12.00
STRC1        5   7    1         4.00
STRC2        5   7    1         8.00
```

```
STRC3        5   7    1        12.00
ARRAY1       6   2    1        10.00
ARRAY2       6   2    2        20.00    2.00
ARRAY3       6   2    1         3.00
SCALR1       6   1              1.00
SCALR2       6   1              0.00
SCALR3       6   1              0.00
C-  Quit tells POPSGN 'end of adverbs'.
QUIT         4   6

*
VERSION = ' '; OPTELL = 'CHAN'
DOPOS = 1 ;  DOMAX = 1 ;  DOWIDTH = 1 ;
*
PROC TSTDUM
SCALAR X, Y, I , J , DELTAX , DELTAY
FINISH
*
PROC SETXWIN(DELTAX,DELTAY);IMXY;BLC(1)=PIXXY(1)-DELTAX/2
TRC(1)=BLC(1)+DELTAX;BLC(2)=PIXXY(2)-DELTAY/2;
TRC(2)=BLC(2)+DELTAY;RETURN;FINISH
*
PROC OFFROAM;I=TVCHAN;J=GRCHAN;TVCHAN=1234;GRCHAN=1234;
OFFSCROL;TVOFF;GRCHAN=J;TVCHAN=I;TVON;RETURN;FINISH
*
PROC OFFHUINT; I=ABS(TVCHAN); IF I < 12  THEN I=12; END
J=MOD(I/10,10);I=MOD(I,10);TVOFF(1234);OFFPS;TVCH=I;OFFTR;
TVCH=J;OFFTR;TVON;RETURN
FINISH
*
PROC TKWIN;TKXY;BLC=PIXXY;TKXY;TRC=PIXXY;
RETURN;FINISH
*
PROC TKBOX(I); TKXY;BOX(1,I)=PIXXY(1);BOX(2,I)=PIXXY(2)
TKXY;BOX(3,I)=PIXXY(1);BOX(4,I)=PIXXY(2);RETURN;FINISH
*
PROC TKNBOXS(NBOXES); FOR J=1:NBOXES;
TYPE 'SET BOX NUMBER',J,' :';TKBOX(J);END;RETURN
FINISH
*
PROC TVRESET; COLOR=0;TVOFF(12345);TVON(TVCH); OFFZ; OFFSC;
OFFPS; GRCH=0;GRCLEAR; OFFTR;RETURN; FINISH
*
PROC TVALL; TVOFF(1234);OFFZOOM;TVON(TVCH);GROFF(1234);J=GRCH;
GRCH=24;GRCL;GRCH=J;TVCL;TVLOD;TVWED(16);TVWLAB;TVFID;RETURN
FINISH
*-----------------------------------------------------------------
```

4.7 INCLUDEs

4.7.1 DAPL.INC

```
C                                                    Include DAPL.
C                                  length before adverb values
```

```
      INTEGER   K(14770)
C                                        full length
      REAL      C(18944)
      HOLLERITH CH(18944)
C                                        character strings
      HOLLERITH INNAM(3), INCLS(2), INTYP, IN2NAM(3), IN2CLS(2), IN2TYP,
     *   IN3NAM(3), IN3CLS(2), IN3TYP, OUTNAM(3), OUTCLS(2), OU2NAM(3),
     *   OU2CLS(2), INEXT, IN2EXT, IN3EXT, TASK(2), OPCODE, FUNTYP,
     *   OBJECT(2), STOKES, BAND, SORT, UVWTFN, TRANSC(4), AXTYPE(2),
     *   VERSON(12), INFLL(12), IN2FLL(12), OUTFLL(12), KEYWRD(2),
     *   KEYSTR(4), PRTASK(2), GRNAME(5), GRADDR(12), GRPHON(4),
     *   VLAOBS(2), VLAMOD, CMETHX, CMODXX, OPTYPE, CSOURS(4,30),
     *   CCALS(4,30), CINTPL, CSMOTY, CREASO(6), SMPTYP, CDETYP, OPTELL,
     *   VELDEF(2), VELTYP(2), XREFDA(2),
     *         STRA1, STRA2(2), STRA3(3), STRB1, STRB2(2), STRB3(3),
     *         STRC1, STRC2(2), STRC3(3)
C                                        numeric variables
      REAL   XTRUE, XFALSE, USERID, INSEQ, INDSK, IN2SEQ, IN2DSK,
     *   IN3SEQ, IN3DSK, OUTSEQ, OUTDSK, OU2SEQ, OU2DSK, INVER, IN2VER,
     *   IN3VER, BADDSK(10), INTAPE, OUTTAP, NFILES, NMAPS, DOWAIT,
     *   PRIOTY, BLCORN(7), TRCORN(7), XINC, YINC, PIXXY(7), PIXVAL,
     *   PXRANG(2), FACTOR, OFFSET, TVBUTT, XTYPE, XPARM(10), YTYPE,
     *   YPARM(10), ROTATE, GAIN, NITER, FLUX, QUAL, TVCHAN, GRCHAN,
     *   TVLEVS, TVCORN(2), COLORS, TVXY(2), DOTV, BATQUE, BTFLIN,
     *   BTNLIN, JOBNUM, LTYPE, PLEV, CLEV, LEVS(30), XYRATO, DOINVR,
     *   DOCENT, ZXRATO, SKEW, DOCONT, DOVECT, ICUT, PCUT, DIST,
     *   IMSIZE(2)
      REAL   CELSIZ(2), SHIFT(2), UVTAPR(2), UVRANG(2), UVBOX, DOGRDC,
     *   ZEROSP(5), BITER, CBMAJ, CBMIN, CBPA, NBOXES, BOX(4,10), DOEOF,
     *   NDIG, DOCAT, DOHIST, BDROP, EDROP, ASPMM, MPTCH, APARMS(10),
     *   BPARMS(10), GPOS(2,4), GMAX(4), GWIDTH(3,4), ERRPOS(2,4),
     *   ERRMAX(4), ERRWTH(3,4), NGAUSS, AXREF, NAXIS, RAXINC, AXVAL(2),
     *   DOSLIC, DOMODL, DORESI, ROMODE, DETIME, DOCRT, CHANNL,
     *   CPARM(10), DPARM(10), DOALIN, NPONTS, AX2REF, DOALL, TVXINC,
     *   TVYINC, TVBLCO(7), TVTRCO(7), DOEOT, DOSTOK, LEVPRT, DORRAY,
     *   ZINC, TVZINC, BECHAN, ENCHAN, RESTFR(2), DENSTY, KEYVAL(2)
      REAL   BEGCNT, ENDCNT, NUMCNT, DOTABL, DOTWO, COPIES, PRNUMB,
     *   PRTIME, CTYPES(4), PIXAVG, PIXRMS, DOCIRC, XCHINC, XNFIEL,
     *   XFLDSZ(2,16), XRASHF(16), XDCSHF(16), XPHAT, XGNERR(30),
     *   XTMSMO(30), DOOUTP, DOCNCT, DONEW, DOCONF, DOALPH, ERRORA,
     *   SLOTAD, XBCOMP(16), XNCOMP(16), QMSPEN, PRSTRT, DOWDGE,
     *   XTIMR(8), XSUBAR, XBIF, XEIF, XANTS(50), XBASLN(50), XDOCAL,
     *   XINTPR(3), XFLGVE, XGAVER, XGAUSE, ANOISE(64), PBSIZE(64),
     *   OUTVER, DOCELL, PIX2XY(7), PIX2VL, STFCTR, CUTOFF, TAMROF,
     *   XBLV, XBPV, XANTWT(30), XSOLIN, XCALCO, XREFA, XSMODE(7)
      REAL   XSOLTY, XSOLMO, XSOLCO, XWTUV, XDODEL, XSYSVL, DOHMS,
     *   BLOCKD, XPMDL(7), XDOPOL, XDOBND, XSMOTH(3), XDOUVC, XXREWT(2),
     *   XSELBN, XSELFQ, XFQID, CHNSEL(3,10), XFQTOL,
     *         ARRAY1(10), ARRAY2(20,2), ARRAY3(3), SCALR1, SCALR2,
     *         SCALR3
      COMMON /CORE/ K, XTRUE, XFALSE, USERID, INNAM, INCLS, INSEQ,
     *   INDSK, INTYP, IN2NAM, IN2CLS, IN2SEQ, IN2DSK, IN2TYP, IN3NAM,
     *   IN3CLS, IN3SEQ, IN3DSK, IN3TYP, OUTNAM, OUTCLS, OUTSEQ, OUTDSK,
     *   OU2NAM, OU2CLS, OU2SEQ, OU2DSK, INEXT, IN2EXT, IN3EXT, INVER,
```

```
     *   IN2VER, IN3VER, BADDSK, INTAPE, OUTTAP, NFILES, NMAPS, TASK,
     *   DOWAIT, PRIOTY, BLCORN, TRCORN, XINC, YINC, PIXXY, PIXVAL,
     *   PXRANG, FACTOR, OFFSET, TVBUTT, XTYPE, XPARM, YTYPE, YPARM,
     *   OPCODE, FUNTYP, ROTATE, GAIN, NITER, FLUX, OBJECT, QUAL,
     *   STOKES, BAND, TVCHAN, GRCHAN, TVLEVS, TVCORN, COLORS, TVXY,
     *   DOTV, BATQUE, BTFLIN, BTNLIN, JOBNUM, LTYPE, PLEV, CLEV, LEVS,
     *   XYRATO, DOINVR, DOCENT, ZXRATO, SKEW, DOCONT, DOVECT, ICUT,
     *   PCUT, DIST, IMSIZE
      COMMON /CORE/ CELSIZ, SHIFT, SORT, UVTAPR, UVRANG, UVWTFN, UVBOX,
     *   DOGRDC, ZEROSP, BITER, CBMAJ, CBMIN, CBPA, NBOXES, BOX, DOEOF,
     *   NDIG, DOCAT, DOHIST, BDROP, EDROP, ASPMM, MPTCH, APARMS,
     *   BPARMS, GPOS, GMAX, GWIDTH, ERRPOS, ERRMAX, ERRWTH, NGAUSS,
     *   TRANSC, AXREF, NAXIS, RAXINC, AXVAL, AXTYPE, DOSLIC, DOMODL,
     *   DORESI, ROMODE, DETIME, DOCRT, CHANNL, CPARM, DPARM, DOALIN,
     *   NPONTS, AX2REF, DOALL, TVXINC, TVYINC, TVBLCO, TVTRCO, VERSON,
     *   DOEOT, DOSTOK, LEVPRT, DORRAY, ZINC, TVZINC, BECHAN, ENCHAN,
     *   RESTFR, INFLL, IN2FLL, OUTFLL, DENSTY, KEYWRD, KEYVAL, KEYSTR,
     *   BEGCNT, ENDCNT, NUMCNT, DOTABL, DOTWO, COPIES, PRNUMB, PRTIME,
     *   PRTASK, CTYPES, PIXAVG, PIXRMS, DOCIRC, XCHINC, XNFIEL, XFLDSZ
      COMMON /CORE/ XRASHF, XDCSHF, XPHAT, XGNERR, XTMSMO, DOOUTP,
     *   DOCNCT, DONEW, DOCONF, DOALPH, ERRORA, GRNAME, GRADDR, GRPHON,
     *   SLOTAD, VLAOBS, VLAMOD, CMETHX, CMODXX, XBCOMP, XNCOMP, QMSPEN,
     *   PRSTRT, OPTYPE, DOWDGE, CSOURS, CCALS, XTIMR, XSUBAR, XBIF,
     *   XEIF, XANTS, XBASLN, XDOCAL, CINTPL, CSMOTY, XINTPR, XFLGVE,
     *   XGAVER, XGAUSE, CREASO, SMPTYP, CDETYP, ANOISE, PBSIZE, OUTVER,
     *   DOCELL, PIX2XY, PIX2VL, STFCTR, CUTOFF, OPTELL, TAMROF, XBLV,
     *   XBPV, XANTWT, XSOLIN, XCALCO, XREFA, XSMODE, XSOLTY, XSOLMO,
     *   XSOLCO, XWTUV, XDODEL, XSYSVL, VELDEF, VELTYP, DOHMS, BLOCKD,
     *   XPMDL, XDOPOL, XDOBND, XSMOTH, XDOUVC, XXREWT, XREFDA, XSELBN,
     *   XSELFQ, XFQID, CHNSEL, XFQTOL,
     *        STRA1, STRA2, STRA3, STRB1, STRB2, STRB3, STRC1, STRC2,
     *        STRC3, ARRAY1, ARRAY2, ARRAY3, SCALR1, SCALR2, SCALR3
      EQUIVALENCE (K(1), C(1), CH(1))
C                                                       End DAPL.
```

4.7.2 DBAT.INC

```
C                                                       Include DBAT.
      INTEGER   BATLUN, BATIND, BATREC, BATDUM, BATDAT(256)
      COMMON /BATCH/ BATLUN, BATIND, BATREC, BATDUM, BATDAT
C                                                       End DBAT.
```

4.7.3 DBWT.INC

```
C                                                       Include DBWT.
      INTEGER   BWTNUM, BWTLUN, BWTIND, BWTREC, BWTDAT(256)
      LOGICAL   WASERR
      CHARACTER BWTNAM*48
      COMMON /BWTCHC/ BWTNAM
      COMMON /BWTCH/ BWTDAT, BWTNUM, BWTLUN, BWTIND, BWTREC, WASERR
C                                                       End DBWT.
```

4.7.4 DCON.INC

```
C                                                          Include DCON.
      INTEGER   K(18944), KXORG
      REAL      C(18944)
      HOLLERITH CH(18944)
      COMMON /CORE/ K
      EQUIVALENCE (K(1), C(1), CH(1)),     (K(8), KXORG)
C                                                          End DCON.
```

4.7.5 DERR.INC

```
C                                                          Include DERR.
C                                        POPS error handling for OERROR
      INTEGER   ERRNUM, IERROR(10), ERRLEV
      CHARACTER PNAME(10)*6
      COMMON /ERRORS/ ERRNUM, IERROR, ERRLEV
      COMMON /ERRORC/ PNAME
C                                                          End DERR.
```

4.7.6 DIO.INC

```
C                                                          Include DIO.
      INTEGER   NBYTES, KARLIM, IUNIT
      CHARACTER KARBUF*80, JBUFF*80, HOLDUF*80, IPT*1
      COMMON /POPSIO/ NBYTES, KARLIM, IUNIT
      COMMON /POPCIO/ KARBUF, JBUFF, HOLDUF, IPT
C                                                          End DIO.
```

4.7.7 DPOP.INC

```
C                                                          Include DPOP.
      INTEGER   KT, LPGM, LLIT, LAST, IDEBUG, MODE, IFFLAG, LINK,
     *   L, NAMEP, IP, LP, SLIM, AP, BP, ONE, ZERO, TRUE, FALSE,
     *   STACK(100), CSTACK(100), SP, CP, SP0, MPAGE, LPAGE
      REAL   V(100), XX
      COMMON /POPS/ V, XX, KT, LPGM, LLIT, LAST, IDEBUG, MODE, IFFLAG,
     *   LINK, L, NAMEP, IP, LP, SLIM, AP, BP, ONE, ZERO, TRUE, FALSE,
     *   STACK, CSTACK, SP, CP, SP0, MPAGE, LPAGE
C                                                          End DPOP.
```

4.7.8 DSMS.INC

```
C                                                          Include DSMS.
      INTEGER   NKAR, KBPTR, NEWCOD, TYPE, TAG, LEVEL, LX, NEXTP, LOCSYM
      REAL      SKEL, X(25)
      HOLLERITH KPAK(5)
      COMMON /SMSTUF/ KPAK, X, SKEL, NKAR, KBPTR, NEWCOD, TYPE,
     *   TAG, LEVEL, LX, NEXTP, LOCSYM
C                                                          End DSMS.
```

Chapter 5

Catalogs

5.1 Overview

AIPS keeps a catalog with a directory which contains an entry for each data file and its associated extension files. The catalog header record is used to keep various pieces of information about the data in the main data file and keeps track of the number and types of extension files associated with the main data file. These catalog header records are kept in individual files. The intent of this chapter is to describe the contents of the catalog header and to describe the use of the routines that access the catalog header record.

The information in the catalog header record is patterned after the FITS format tape header, although it is not as flexible. The catalog header describes the order and amount of data and maximum and minimum values, etc.

AIPS data files have a structure very similar to the structure of data of FITS format tapes. An image consists of a rectangular array of up to 7 dimensions. Pixel locations must be evenly spaced along each axis, although a proper redefinition of the axis can usually make this possible. The header record contains the number of pixels along each axis, a label for each axis, the number of the reference pixel (may be a fractional pixel and need not be in the portion of the axis covered), the coordinate at the reference pixel, the coordinate increment between pixels and the coordinate rotation. The axes of images may be in any order.

The AIPS format for uv data is also similar to the FITS convention. Each data point has a number of "random parameters", usually "u", "v", time, baseline number etc., followed by a rectangular array similar to, but usually smaller than, an image data array. Up to 14 random parameters have labels kept in the catalog header. More than 14 random parameters can be used, but the labels for the fifteenth and following are lost.

Most tasks read an old data file, do some operation on the data and write a new data file. In this case, the task simply takes the old catalog header record and modifies it to describe the data in the new file.

AIPS also keeps a catalog of the images displayed on all display devices. This image catalog allows AIPS interactive verbs to use the display devices without having to find and read the original catalog header record.

5.2 Public and Private catalogs

AIPS catalogs may be either public, i.e., all files on a given disk are in the same catalog, or private, i.e., each user has a separate catalog on each disk. The stand-alone utility program, SETPAR, is used to specify which type is currently in use. The distinction is completely transparent to the programmer; all distinctions between the two types are hidden in ZPHFIL and the catalog routines.

5.3 File Names

AIPS data files, especially cataloged files, are referenced in a number of different ways. The following list summarizes the three basic ways of specifying AIPS data files:

1. AIPS logical names. The full AIPS logical file specification is given by disk number, file name, file class, file sequence number, file physical type, user number, and, for extension files, the version number. This is the fundamental way an AIPS user specifies a file; although some of these, such as physical type and user number, may not have to be specified directly. In a task, these values are used by CATDIR (which may be called by a higher level routine such as MAPOPN) to locate the desired file in the AIPS catalog using various default and wild-card conventions.

2. Disk and catalog number. Just as the AIPS user frequently uses the disk and catalog numbers to specify files using the verb GETNAME, programs usually keep track of cataloged files by means of the disk and catalog numbers, file types, and version numbers for extension files.

3. Physical name. The host operating system needs a name for the file for its own catalog. The allowed physical file specifications depends on the host operating system, so AIPS tasks use the Z routine ZPHFIL to create the physical name from the disk and catalog numbers, the file type and version, and the user number for systems with private catalogs. These physical names may be up to 48 characters long.

   An example from a VAX system with private catalogs is "DA0n:ttrcccvv.uuu", where n is the zero-relative disk drive number, DA0n: is a logical variable which is assigned to a directory, tt is a two character file type (e.g., "MA"), r is a data format version number (see Appendix A), ccc is the catalog slot number, vv is the version (01 for "MA", "SC" and "UV" files), and uuu is the user's number. All numbers are expressed in hexadecimal notation.

5.4 Data Catalog

The data catalog actually consists of many separate files. There is one directory file (type "CA") per user for private catalogs per disk drive. Each cataloged file has its own catalog header file ("CB").

5.4.1 Catalog Directory

Each catalog directory contains a one block (256-word) header and a number of catalog directory blocks. The header block contains principally the number of catalog blocks in the file; this is set when the file is initialized or expanded. The directory blocks contain a 32-byte reference to each catalog header record. The directory is used to speed catalog searches and also contains the map status words that register map file activity.

5.4.2 Header block

The format of the Header Block is as follows:

```
OFFSET  TYPE         DESCRIPTION
   0     I   Volume number of disk containing this catalog
   1     I   Unused
   2     I   Number of catalog blocks in this file
```

5.4.3 Directory Section

The Mth directory block contains NLPR entries, each NWPL words, indexing the NLPR*(M-1)+1 to the NLPR*M-1 catalog records. The first directory block is the 2nd block in the file. The parameters are given by NWPL = 11 and NLPR = 256/NWPL.

The description of a directory entry is as follows:

```
OFFSET LENGTH TYPE         DESCRIPTION
   0      1     I    User ID number; or -1 if slot is empty
   1      1     I    Map file activity status
```

```
 2      2    H(2) Date/Time file was cataloged
 4      1    I    User defined sequence number 1 to 9999
 5      3    H(3) User defined map name, 12 characters
 8      2    H(2) User defined map class, 6 characters
10      1    H    Map type, 2 characters
   ...
```

5.4.4 Directory Usage

Map name and class are user defined character strings of 12 and 6 characters that can be used to identify and locate a specific map. The strings are stored as HOLLERITH strings together with the 2-character HOLLERITH string which identifies the "physical" map type, in their slots in the directory. The sequence number is similarly an arbitrary integer reference number.

The Map Status is an integer registering the activity of the map file itself.

```
STATUS = 0    => no programs are accessing the map file
       = n>0  => n programs are reading the map
       = -1   => one program is writing into the file
       = n<0  => 1 + n programs are reading the map, one
                 program is writing into the file.
```

Maintaining the integrity of the catalog entries is essential to ensure reliable access to the cataloged files. Thus certain rules should be followed when using the catalog. These rules are coded in to the utility routines described below; these routines should be used when at all possible to access the catalog.

Rules:

1. Take exclusive use of the catalog whenever you access it. The required operation should be done quickly and then the catalog file should be closed and released.

2. The status word must be monitored to see if an intended catalog or map operation will disturb an (asynchronous) operation already in progress.

Specifically: Do not modify a catalog block, nor write into a map file which is not in a rest state (STATUS = 0).

If you intend to write into a map and STATUS = 0, change the status to "WRITE" (STATUS = -1) before releasing exclusive use of the catalog.

If you intend to read a map file or catalog block, check to see if someone else is writing on it (STATUS < 0). If so, decide whether this is acceptable to your program. If so, modify the status to "READ":

```
STATUS = 1 + STATUS if STATUS > 0
STATUS =-1 + STATUS if STATUS < 0.
```

Clear status when you have finished your operation. If you were reading, reverse the process just described. If you were writing, STATUS = - (1 + STATUS).

5.4.5 Structure of The Catalog Header Record

The catalog header block is a fixed format data structure 512 bytes long (one byte is defined in AIPS as half an integer). The catalog header block contains double and single precision floating point numbers, integers, and hollerith strings. The catalog header record is accessed by equivalencing integer, hollerith, real and double precision arrays, and obtaining the information from the array of the appropriate data type. Since the structure of the catalog header is subject to change we use pointers for the different arrays that are computed by VHDRIN. These pointers are kept in a common invoked with the INCLUDE DHDR.INC.

The uses of the pointers are given in the following table. In this table, the term "random parameters" refers to the portion of a uv data record that contain u, v, w, time, baseline etc.; the term "indeterminate" pixel means a pixel whose value is not given.

TYPE	POINTER	DESCRIPTION
H(2)	KHOBJ	Source name
H(2)	KHTEL	Telescope, i.e., 'VLA'
H(2)	KHINS	e.g., receiver or correlator
H(2)	KHOBS	Observer name
H(2)	KHDOB	Observation date in format 'DD/MM/YY'
H(2)	KHDMP	Date map created in format 'DD/MM/YY'
H(2)	KHBUN	Map units, i.e., 'JY/BEAM '
H(2)(14)	KHPTP	Random Parameter types
	KIPTPN= 14	Max. number of labeled random paramaters
H(2)(7)	KHCTP	Coordinate type, i.e., 'RA---SIN'
	KICTPN= 7	Max. number of axes
D(7)	KDCRV	Coordinate value at reference pixel
R(7)	KRCIC	Coordinate value increment along axis
R(7)	KRCRP	Coordinate Reference Pixel
R(7)	KRCRT	Coordinate Rotation Angles
R	KREPO	Epoch of coordinates (years)
R	KRDMX	Real value of data maximum
R	KRDMN	Real value of data minimum
R	KRBLK	Value of indeterminate pixel (real maps only)
I	KIGCN	Number of random par. groups. This is the number of uv data records.
I	KIPCN	Number of random parameters
I	KIDIM	Number of coordinate axes
I(7)	KINAX	Number of pixels on each axis
I	KIIMS	Image sequence no.
H(3)	KHIMN	Image name (12 characters)
	KHIMNO= 1	Character offset in HOLLERITH string
H(2)	KHIMC	Image class (6 characters)
	KHIMCO= 13	Character offset in HOLLERITH string
H	KHPTY	Map physical type (i.e., 'MA','UV') (2 char)
	KHPTYO= 19	Character offset in HOLLERITH
I	KIIMU	Image user ID number
I	KINIT	# clean iterations
R	KRBMJ	Beam major axis in degrees
R	KRBMN	Beam minor axis in degrees
R	KRBPA	Beam position angle in degrees
I	KITYP	Clean map type: 1-4 => normal, components, residual, points. For uv data this word contains a two character sort order code.
I	KIALT	Velocity reference frame: 1-3 => LSR, Helio, Observer + 256 if radio definition.
D	KDORA	Antenna pointing Right Ascension
D	KDODE	Antenna pointing Declination
D	KDRST	Rest frequency of line (Hz)

```
D       KDARV       Alternate ref pixel value
                    (frequency or velocity)
R       KRARP       Alternate ref pixel location
                    (frequency or velocity)
R       KRXSH       Offset in X (rotated RA) of phase center
R       KRYSH       Offset in Y (rotated Dec) from tangent pt.
H(20)   KHEXT       Names of extension file types (2 char)
        KHEXTN= 20  Max number of extension files
I(20)   KIVER       Number of versions of corresponding
                    extension file.
```

The actual values of the pointers depend on the size of the various data types and are computed in the routine VHDRIN. Note that VHDRIN should be called *after* ZDCHIN is called because it uses values set by ZDCHIN. VHDRIN has no call arguments.

The name of the pointer tells which data type array the data is to be read from: KInnn indicates the integer array, KHnnn indicates the hollerith array, KRnnn indicates the real array, and KDnnn indicates the double precision array. Conversion of HOLLERITH data to and from CHARACTER variables is done using routines H2CHR and CHR2H. The Name, class, and physical type are contained in HOLLERITH strings as are the labels of the regular and random axes. This is best explained by an example:

```
      INTEGER   NDIM1, INDEX
      REAL      CRPIX2
      CHARACTER CLASS*6, ALABE2*8
      DOUBLE PRECISION CRVAL3
C                                     Include for header pointers
      INCLUDE 'INCS:DHDR.INC'
C                                     Include for catalog header
C                                     common
      INCLUDE 'INCS:DCAT.INC'
          .
          .
          .
C                                     Get the dimension of
C                                     the first axis (I)
      NDIM1 = CATBLK(KINAX)
C                                     Get reference pixel
C                                     of second axis (R)
      CRPIX2 = CATR(KRCRP+1)
C                                     Get coordinate at reference
C                                     pixel on third axis. (R)
      CRVAL3 = CATD(KDCRV+2)
C                                     Copy axis label for third
C                                     axis (H array).
      INDEX = KHPTP + 2 * 2
      CALL H2CHR (8, 1, CATH(INDEX), ALABE2)
C                                     Copy image class.
      CALL H2CHR (6, KHIMCO, CATH(KHIMC), CLASS)
```

In the example above the catalog header block is obtained from a common defined in INCLUDE DCAT.INC. Many AIPS utility routines get the catalog header record from this common, so it is a good place to store it.

Keyword-Value Pairs

Arbitrary sets of keyword value pairs can be stored in an extension of the catalog header using routine CATKEY. The values may be of the following types: DOUBLE PRECISION, REAL, HOLLERITH (up to 8 char), INTEGER and LOGICAL. A description of CATKEY is given at the end of this chapter.

Image Files

An image consists of a single multidimensional (up to 7), rectangular array of pixel values. The structure of this array is defined by the catalog header record, which contains the number of dimensions (KIDIM) and the number of pixels on each axis (KINAX). All images are stored as REAL values.

The label for each axis is in a HOLLERITH string array pointed to by KHCTP. The coordinate increment between pixels must be a constant on each axis, and the array of axis increments is obtained using the pointer KRCIC. The array of coordinate reference pixels (the pixel at which the coordinate value is that pointed to by KDCRV) is pointed to by KRCRP; the reference pixel need not be either an integral pixel or in the range covered by the data. The coordinate values at the reference pixels are pointed to by KDCRV.

Each axis also has an associated rotation angle, but the only rotation currently supported is that on the plane of the sky. This rotation value is kept on the declination/Galactic latitude/Ecliptic latitude/Y axis and is the rotation of the coordinate system from north toward east.

Since there is no explicit provision made in the catalog header for such important parameters as position, frequency, and polarization, these are always declared as axes even if that axis contains only one pixel. This allows a place in the header record for these parameters.

Since the Stokes' axis is not inherently an ordered set, we use the following definitions for the values along the stokes' axis.

```
0  => beam                  5  => Percent polarization
1  => I                     6  => Fractional polarization
2  => Q                     7  => Polarization position angle
3  => U                     8  => Spectral index
4  => V                     9  => Optical depth
```

Pixel values may be blanked using "magic value" blanking. The magic (stored) value for images is given by KRBLK (always 'INDE').

Each row of an image (first dimension) starts on a disk sector boundary (as defined on the local system) unless several rows may fit in a sector. In the latter case, as many rows as possible are put in a sector, but a row is not allowed to cross a sector boundary. Each plane in the image (dimension 3 and higher) starts on a sector boundary.

All angles in the header record are in degrees.

Uv Data Files

Uv data files consist of a sequence of interferometer visibility records each of which contains all data measured on a given baseline (pair of antennas) in a given integration period. The number of visibility records is given in the catalog header record by the integer value pointed to by KIGCN. The order of the visibility records are given by the two character code pointed to by KITYP. (More details of the sort order can be found in the chapter on disk I/O). All values are in floating point (except for compressed data).

Each visibility record consists of a number (KIPCN) of "random" parameters, followed by a data array similar to a miniature image. Any number of random parameters are allowed, but only the labels of 14 (KIPTPN) can be kept in the header. These labels are kept in Hollerith strings pointed to by KHPTP. The random parameters are used for values which vary "randomly" from visibility to visibility (i.e., u, v, w, time, baseline). The data array is described by the catalog header record in the same ways as for an image file.

The tangent point of the data (position for which the u, v, and w are computed) is kept as the RA and Dec axis in the data array. The offset in x and y (RA and dec after rotation) are pointed to by KRXSH and KRYSH. All angles in the catalog header record are in degrees.

Uv data may contain correlator based polarization or true Stokes' parameters. In the former case, the following Stokes' values are defined:

```
-1  => RR
-2  => LL
-3  => RL
-4  => LR
-5  => XX  Orientation of X and Y are defined in the
-6  => YY  AN table
-7  => XY
-8  => YX
```

Visibility records are allowed to span disk sector boundaries. More details about the uv data file format are given in the chapter on disk I/O.

Single Dish Data Files

Randomly sampled sky brightness measurments may be stored in data files which are similar to uv data files (the file "type" of the files is "UV"). The random parameters use for this type of data give the celestial position and beam or feed number rather than the location in the uv plane or a baseline. This type of data is described in more detail in next chapter.

5.4.6 Routines to Access the Data Catalog

MAPOPN and MAPCLS

There are a number of utility routines to access the catalog header record. In many cases, most of the catalog operations can be taken care of by the pair of routines MAPOPN and MAPCLS. MAPOPN will locate the correct catalog entry from a given name, class, disk, sequence and physical type following all default and wild-card conventions. MAPOPN then reads the catalog header record, opens the main data file and marks the catalog status word. Following a call to an initialization routine, the file can be read from or written to. After all I/O to the file is complete, MAPCLS will close the file, update the catalog header record if requested and clear the catalog status word for the file. A description of the call sequence of MAPOPN and MAPCLS is given at the end of this chapter.

CATDIR and CATIO

If MAPOPN and MAPCLS are not appropriate, then the use of more specialized routines is necessary. First the desired file must be located in the catalog directory. The routine CATDIR is the basic method of accessing the catalog directory. This routine will find the desired file given the name, class, etc. following the usual default and wild-card conventions. CATDIR returns the disk number and catalog slot number. Given a disk number and catalog slot number, CATIO can read or write a catalog header record and/or change the status word. Detailed descriptions of CATDIR and CATIO can be found at the end of this chapter.

5.4.7 Routines to Interpret the Catalog Header

There are a number of specialized routines which obtain information from the catalog header record. The following list gives a short description of each and detailed descriptions of the call sequence are found at the end of this chapter.

- AXEFND will return the axis number of a given type of random or regular axis.
- ROTFND returns the angle of rotation on the sky of either an image or uv data file.
- UVPGET obtains a number of pointers and other pieces of information which simplify accessing uv data.

5.4.8 Catalog Status

The AIPS catalog directory keeps a status word for each cataloged file. This status word is used to help prevent conflicting use of the file. The status may be marked as either "READ" or "WRIT"; the status of each file can be seen in AIPS by listing the catalog. A file can be marked "READ" multiple times, but a file marked "WRIT" cannot be marked "READ" or "WRIT" again, and a file marked "READ" cannot be marked "WRIT".

The use of the status word can complicate updating of the catalog header with CATIO. If the status of a file has been marked as "WRIT", then the opcode in the call to CATIO must be "UPDT". If the status is not marked, the opcode must be "WRIT" to update the catalog header block.

5.5 Image Catalog

5.5.1 Overview

The image catalog contains data for images stored on the TV device that identify the images, refer them back to their original map files, and specify scaling of the X-Y and intensity coordinates. There is a separate image catalog which performs the same functions for graphics devices (e.g., TEK4012 storage screens).

There is one image catalog file for each television device, whose physical name corresponds to ICv0000n, where v = version code and n = the device number (0 for graphics, 1 to n for TVs). They reside on disk 1 and must be created at AIPS installation, usually by FILAIP.

5.5.2 Data Structures

General: For each gray-scale image plane of the TV device, the IC contains N 1-block (256-word) records for cataloging up to N subimages, plus a (N-1)/51+1 block directory. The directory immediately precedes the catalog blocks for each image plane. For each TV graphics overlay plane there is one catalog block with no directory. These blocks follow immediately after the last gray-scale block.

The IC for pure graphics devices (called TK devices) has one image catalog block for each device in the system including all "local" TK devices followed by all remote-entry devices. Record number n in this file is associated with TK device number n (NTKDEV in /DCHCOM/ from include DDCH.INC).

The image catalog blocks themselves are essentially duplicates of the map catalog blocks except that scaling information replaces the extension file index of the map catalog.

The following is a description of the format of the directory block and the portions of the image catalog block which is different from the normal catalog header block.

```
   Directory Block (Gray-scale image)

OFFSET  TYPE    DESCRIPTION
     0   I     Sequence number of last sub-image cataloged
               on this plane
     1   I     Seq. no. of sub-image in slot 1; 0 if slot empty
     2   I(4)  TV pixel positions of corners of 1st sub-image,
               x1,y1,x2,y2
     6   I     Seq. no. of sub-image in slot 2; 0 if empty
     7   I(4)  TV pixel positions of corners of 2nd sub-image
     .
     .
     .
```

Catalog Block for each image or subimage:

Most of the Image Catalog block is identical to the map CAtalog block of the source of the image. (See section on CB files.) The information on antenna pointing, alternate frequency/velocity axis descriptions, and

extension files (KIALT, KDORA, KDODE, KDRST, KDARV, KRARP, KHEXT and KIVER) is replaced in the IC by:

```
TYPE POINTER DESCRIPTION

 R(2) IRRAN Map values displayed as min & max brightness.  I IIVOL
Disk volume from which map came I IICNO Catalog slot number of orig.
map I(4) IIWIN Map pixel positions of corners of displayed image (rel.
to orig. map) I(5) IIDEP Depth of displayed image in 7 - dimensional
map (axes 3 - 7) I(4) IICOR TV pixel positions of corners of image on
screen I IHTRA 2-char code for transfer function used to compute TV
brightness from map intensity values.  I IIPLT Code for type of plot.
I(31)IIOTH Misc. plot type dependent info.  (at the moment no more
than 20 used)
```

The standard pointer values are computed by VHDRIN and are available through the common /HDRVAL/ via include DHDR.INC. They are machine-dependent and are used in the same way as the normal catalog pointers.

5.5.3 Usage notes

We assume that single images only are stored on graphics planes; there is no directory.

When a gray-image plane is cleared, its directory is zeroed. As images are added to the plane, their coordinates are written into an open directory slot for that plane, along with the current value of the plane sequence number. The sequence number is then incremented. If an old image is completely overwritten by a new one, its directory slot is cleared. For partially overlapping images, the sequence # allows the user to select the one most recently loaded into a given part of the plane.

5.5.4 Subroutines

There are a number of routines to manipulate the image catalog. The following is a short description of each; detailed descriptions of the call sequences is given at the end of this chapter.

- YCINIT clears the Image Catalog for a given plane.
- YCOVER asks if there are any overlapped images in each quadrant visible.
- YCWRIT adds a new block to the catalog.
- YCREAD returns the block corresponding to a given TV pixel.
- TVFIND determines desired image, asks user if > 1 visible.

These routines expect the "plane number" as an argument. TV gray scale planes are numbered 1–NGRAY, TV graphics overlay planes are numbered (NGRAY+1)–(NGRAY+NGRAPH), and TK devices are referenced by any plane number greater than NGRAY+NGRAPH.

5.5.5 Image Catalog Commons

The COMMON /TVCHAR/ referenced by INCLUDE 'DTVC.INC' contains TV device characteristics such as:

```
NGRAY       # of gray-scale planes on this device
NGRAPH      # of graphics planes
MAXXTV(2)   Maximum number of pixels in x,y directions in image
```

The listing of DTVC.INC is given at the end of this chapter.

The common /DCHCOM/ (from DDCH.INC) contains two important parameters in this regard: NTVDEV and NTKDEV. The subroutine ZDCHIN sets these to the actual number of such devices present locally. Then, the routines ZWHOMI (in AIPS only) and GTPARM (in all tasks) reset them to the device number assigned to the current user. ZWHOMI determines these assignments.

5.6 Coordinate Systems

Astronomical images are usually represented as projections onto a plane causing the true position on the sky of a pixel to be a nonlinear function of the pixel location. In a similar fashion, most spectral observations are done with evenly spaced frequency channels which results in a nonlinear relation between the velocity of a channel and the channel number. AIPS Memos Nos. 27 and 46 describe in great detail the approach AIPS uses to these problems. Much of the following sections is taken from these memos.

5.6.1 Velocity and Frequency

The physically meaningful measure in a spectrum is the radial velocity of a feature; unfortunately, observations are normally made using a uniform spacing in frequency (and may contain Doppler tracking to remove the effects of the earth's motion). Thus it is necessary to convert between frequency and velocity. The details of the conversion are in AIPS Memo No. 27 and will not be reproduced here. Conversion can be done using the routines described in the section on celestial positions. The following sections describe the naming conventions and the way in which the necessary information is stored in the catalog header block.

Axis Labels

The AIPS convention is to use the axis label to denote the axis type with the first four characters and the inertial reference system with the last four characters. The axis types currently supported are 'FREQ...' which is regularly gridded in frequency, 'VELO...' which is regularly gridded in velocity, and 'FELO...' which is regularly gridded in frequency, but expressed in velocity units in the optical convention.

The inertial reference systems currently supported are '-LSR', '-HEL', and '-OBS' indicating Local Standard of Rest, heliocentric, and geocentric. Others may be added if necessary.

Catalog Information

In addition to the normal axis coordinate information carried in the catalog header, described previously in this chapter, the catalog header record has provision for storing an alternate frequency axis type. The AIPS verb ALTDEF allows the user to switch the two axis definitions. The pointers for these values are given in the following:

```
KDRST    Rest frequency (Hz)
KRARP    Alternate reference pixel
KDARV    Alternate reference value
KIALT    axis type code. 1=>LSR, 2=>HEL, 3=>OBS
         (plus 256 if radio convention).
         0 implies no alternate axis.
```

5.6.2 Celestial Positions

The following sections will describe the AIPS conventions and routines for determining positions from images with different projections.

Axis Labels

The AIPS convention is to use the first four characters of the axis type and the second four characters to denote the projection. The standard nonlinear axis types are given in the following:

- RA–– denotes Right ascension
- DEC– denotes declination
- GLON denotes galactic longitude
- GLAT denotes galactic latitude
- ELON denotes Ecliptic longitude
- ELAT denotes Ecliptic latitude

The geometry used for the projection is given in the axis label using the codes given in the following list:

- –TAN denotes tangent projection. This projection is commonly used in optical astronomy.
- –SIN denotes sine projection. This projection is commonly used in radio aperture synthesis images.
- –ARC denotes arc projection. In this geometry, angular distances are preserved and it is commonly used for Schmidt telescopes and for single dish radio telescopes.
- –NCP denotes a projection to a plane perpendicular to the North Celestial Pole. This geometry is used by Westerbork.
- –STG denotes stereographic projection. This is the tangent projection from the opposite side of the celestial sphere.
- –AIT denotes Aitoff projection. This is used for very large fields.
- –GLS denotes Global sinusoidal projection. This is also used for very large fields.
- –MER denotes Mercator projection.

Determining Positions

There are a number of AIPS utility routines which help determine the position of a given location in an image. These routines use values in theINCLUDE DLOC.INC. A listing of this include can be found at the end of this chapter.

Position Routines The upper level position determination routines are briefly described in the following; details of the call sequences are given at the end of this chapter.

- SETLOC initializes the DLOC.INC INCLUDE based on the current catalog header block in the DCAT.INC (CATBLK) common.
- XYPIX determines the pixel location corresponding to a specified coordinate value.
- XYVAL determines the coordinate value (X,Y,Z) corresponding to a given pixel location.
- FNDX returns the X axis coordinate value of a point given the Y axis coordinate value and the X axis pixel position of a point. Does rotations and non linear axes.
- FNDY returns the Y axis coordinate value of a point given the X axis coordinate value and the Y axis pixel position of a point. Does rotations and non linear axes.
- COORDT converts between celestial, galactic and ecliptic coordinates.

Include DLOC.INC The commons in INCLUDE DLOC.INC are used by the position routines and the plot labeling routines to keep constants needed for the coordinate transformation. The contents of these commons is described in the following:

```
RPVAL    D(4)     Reference pixel values
COND2R   D        Degrees to radians multiplier = pi/180
AXDENU   D        delta(nu) / nu(x) when a FELO axis is
                  present.
GEOMD1   D        Storage for parameter needed for geometry.
GEOMD2   D                        "
GEOMD3   D                        "
GEOMD4   D                        "
RPLOC    R(4)     Reference pixel locations
AXINC    R(4)     Axis increments
CTYP     C(4)*20  Axis types
CPREF    C(2)*5   x,y axis prefixes for labeling
ROT      R        Rotation angle of position axes
SAXLAB   C(2)*20  Labels for axes 3 and 4 values
ZDEPTH   I(5)     Value of Idepth from SETLOC call
ZAXIS    I        1 relative number of z axis
AXTYP    I        Position axis code
CORTYP   I        Which position is which
LABTYP   I        Special x,y label request
SGNROT   I        Extra sign to apply to rotation
AXFUNC   I(7)     Kind of axis code
KLOCL    I        0-rel axis number-longitude axis
KLOCM    I        0-rel axis number-latitude axis
KLOCF    I        0-rel axis number-frequency axis
KLOCS    I        0-rel axis number-stokes axis
KLOCA    I        0-rel axis number-"primary axis" 3
KLOCB    I        0-rel axis number-"primary axis" 4
NCHLAB   I(2)     Number of characters in SAXLAB
```

Several of the above values need further explanation:

```
AXTYP    value = 0  no position-axis pair
                = 1  x-y are position pair
                = 2  x-z are position pair
                = 3  y-z are position pair
                = 4  2 z axes form a pair
CORTYP   value = 0  linear x,y axes
                = 1  x is longitude, y is latitude
                = 2  y is longitude, x is latitude
                = 3  x is longitude, z is latitude
                = 4  z is longitude, x is latitude
                = 5  y is longitude, z is latitude
                = 6  z is longitude, y is latitude
 LABTYP value = 10 * ycode + xcode
        code  = 0   use CPREF, CTYP
                = 1   use Ecliptic longitude
                = 2   use Ecliptic latitude
                = 3   use Galactic longitude
                = 4   use Galactic latitude
                = 5   use Right Ascension
```

```
                = 6   use declination
      AXFUNC value = -1  no axis
                = 0   linear axis
                = 1   FELO axis
                = 2   SIN projection
                = 3   TAN projection
                = 4   ARC projection
                = 5   NCP projection
                = 6   GLS projection
                = 7   MER projection
                = 8   AIT projection
                = 9   STG projection
```

The KLOCx parameters have a value of -1 if the corresponding axis does not exist. If AXTYP is 2 or 3, the pointer KLOCA will always point at the z axis. In this case, SETLOC does not have enough information to prepare SAXLAB. The string must be computed later when an appropriate x,y position is specified.

5.6.3 Rotations

The use of one rotation angle per axis, as provided in the AIPS catalog header, is obviously not enough to completely describe an arbitrary rotation of the coordinate system. In practice, the only rotation currently used in AIPS is the rotation in the sky plane (projected RA and dec, galactic latitude and longitude, or ecliptic latitude and longitude). The rotation angle in this plane of the actual coordinate system of the image, in the usual astronomical north through east convention, is given on the axis corresponding to the declination, galactic latitude, or ecliptic latitude as appropriate.

Another convention followed in AIPS involving rotations is related to precession. As the earth precesses, the north-south line in a field will rotate; this causes a rotation in an image made of a given field on the sky. This "differential precession" will cause problems determining positions away from the field center and comparing images made at different epochs. To avoid this problem, the coordinate system used for the u-v data is rotated to the orientation as of the mean epoch (1950 or 2000).

5.7 Text of INCLUDE files

5.7.1 DCAT.INC

```
C                                                               Include DCAT.
C                                        catalog header common
      INTEGER   CATBLK(256)
      REAL      CATR(256)
      HOLLERITH CATH(256)
      DOUBLE PRECISION CATD(128)
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE (CATBLK, CATR, CATH, CATD)
C                                                               End DCAT.
```

5.7.2 DHDR.INC

```
C                                                               Include DHDR.
      INTEGER   KHOBJ, KHTEL, KHINS, KHOBS, KHDOB, KHDMP, KHBUN,
     *   KHPTP, KHCTP, KRCIC, KRCRP, KRCRT, KREPO, KRDMX, KRDMN, KRBLK,
     *   KHIMN, KHIMC, KHPTY, KRBMJ, KRBMN, KRBPA, KRARP, KRXSH, KRYSH,
```

```
     *   KHIMNO, KHIMCO, KHPTYO,
     *   KDCRV, KDORA, KDODE, KDRST, KDARV,
     *   KIGCN, KINIT,
     *   KIPTPN, KICTPN, KIEXTN,
     *   KIPCN, KIDIM, KINAX, KIIMS, KIIMU, KITYP, KIALT, KHEXT, KIVER,
     *   IRRAN, IIVOL, IICNO, IIWIN, IIDEP, IICOR, IITRA, IIPLT, IIOTH,
     *   KIRES, KIRESN
      COMMON /HDRVAL/  KHOBJ, KHTEL, KHINS, KHOBS, KHDOB, KHDMP, KHBUN,
     *   KHPTP, KHCTP, KRCIC, KRCRP, KRCRT, KREPO, KRDMX, KRDMN, KRBLK,
     *   KHIMN, KHIMC, KHPTY, KRBMJ, KRBMN, KRBPA, KRARP, KRXSH, KRYSH,
     *   KNIMNO, KHIMCO, KHPTYO,
     *   KDCRV, KDORA, KDODE, KDRST, KDARV,
     *   KIGCN, KINIT,
     *   KIPTPN, KICTPN, KIEXTN,
     *   KIPCN, KIDIM, KINAX, KIIMS, KIIMU, KITYP, KIALT, KHEXT, KIVER,
     *   IRRAN, IIVOL, IICNO, IIWIN, IIDEP, IICOR, IITRA, IIPLT, IIOTH,
     *   KIRES, KIRESN
C                                                              End DHDR.
```

5.7.3 DLOC.INC

```
C                                                              Include DLOC.
C                                        Position labeling common
      DOUBLE PRECISION RPVAL(4), COND2R, AXDENU, GEOMD1, GEOMD2, GEOMD3,
     *   GEOMD4
      CHARACTER CTYP(4)*20, CPREF(2)*5, SAXLAB(2)*20
      REAL      RPLOC(4), AXINC(4), ROT
      INTEGER   ZDEPTH(5), ZAXIS, AXTYP, CORTYP, LABTYP, SGNROT,
     *   AXFUNC(7), KLOCL, KLOCM, KLOCF, KLOCS, KLOCA, KLOCB,
     *   NCHLAB(2)
      COMMON /LOCATC/ CTYP, CPREF, SAXLAB
      COMMON /LOCATI/ RPVAL, COND2R, AXDENU, GEOMD1, GEOMD2, GEOMD3,
     *   GEOMD4, RPLOC, AXINC, ROT, ZDEPTH,
     *   ZAXIS, AXTYP, CORTYP, LABTYP, SGNROT, AXFUNC, KLOCL, KLOCM,
     *   KLOCF, KLOCS, KLOCA, KLOCB, NCHLAB
C                                                              End DLOC.
```

5.7.4 DTVC.INC

```
C                                                              Include DTVC.
      INTEGER   NGRAY, NGRAPH, NIMAGE, MAXXTV(2), MAXINT, LUTOUT,
     *   OFMINP, OFMOUT, SCXINC, SCYINC, MXZOOM, CSIZTV(2), TYPSPL,
     *   TVALUS, TVXMOD, TVYMOD, ISUNUM,
     *   TVDUMS(10),
     *   TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *   TVSPLM, TVSPLC, TYPMOV(16), YBUFF(168)
      COMMON /TVCHAR/ NGRAY, NGRAPH, NIMAGE, MAXXTV, MAXINT, LUTOUT,
     *   OFMINP, OFMOUT, SCXINC, SCYINC, MXZOOM, CSIZTV, TYPSPL,
     *   TVALUS, TVXMOD, TVYMOD, ISUNUM,                 TVDUMS,
     *   TVZOOM, TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC,
     *   TYPMOV, YBUFF
C                                                              End DTVC.
```

5.8 Routines

5.8.1 AXEFND

AXEFND determines the order number of an axis whose name is in the character string TYPE. It will work for either regular or random axes.

```
AXEFND (NCHC, TYPE, NAXIS, AXES, IOFF, IERR)
Inputs:
   NCHC       I    Compare only first NCHC characters of axis type
   TYPE       C*8  Axis type.
   NAXIS      I    the number of axes to search,
                      for uniform axes use: CATBLK(KIDIM) or KICTPN
                      for random axes use:  CATBLK(KIPCN) or KIPTPN
   AXES(*)    N    Catalog axis name list,
                      for uniform axes use: CATH(KHCTP)
                      for random axes use : CATN(KHPTP)
Output:
   IOFF       I    Axis offset ( zero relative axis number)
   IERR       I    Return error code, 0=>OK, 1=>could not find.
```

5.8.2 CATDIR

CATDIR manipulates catalog directory.

```
CATDIR (OP, IVOL, CNO, CNAME, CCLASS, SEQ, PTYPE, USID,
  *   STAT, BUFF, IERR)
Inputs:
   OP     C*4   searches find entry with specified data:
                'SRCH' high seq # (if SEQ 0), return things
                'SRNH' high seq # (if SEQ 0), NOT return things
                'SRCN' next match, return things
                'SRNN' next match, NOT return things
                'OPEN' = create a new slot (and init header file)
                'CLOS' = destroy a slot
                'INFO' = return contents of a slot
                'CSTA' = modify status of a slot
   IVOL   I     Disk volume containing catalog
                0 => all on searches, OPEN
   CNO    I     Slot number to begin: SRCN, SRNN, OPEN
                Ignored if IVOL = 0 : searches, OPEN
                Slot number to examine (solely): CLOS, INFO, CSTA
   CNAME  C*12  Map name: searches, OPEN, CLOS
   CCLASS C*6   Map type: searches, OPEN, CLOS
   SEQ    I     Map sequence number: searches, OPEN, CLOS
   PTYPE  C*2   Map physical type (2 chars): searches, OPEN, CLOS
   USID   I     User identification #: searches, OPEN, CLOS
   STAT   C*4   Status (OP=CSTA): READ, WRIT, CLRD, or CLWR
Outputs:
   CNO    I     Slot number found: searches, OPEN
   IVOL   I     If 0 on input, value actually used: searches, OPEN
   CNAME  C*12  Map name: SRCH, SRCN, INFO
```

```
CCLASS C*6   Map type: SRCH, SRCN, INFO
SEQ    I     Map sequence number: SRCH, SRCN, INFO
PTYPE  C*2   Map physical file type: SRCH, SRCN, INFO
USID   I     User identification #: SRCH, SRCN, INFO
STAT   C*4   Status: INFO
BUFF   I(256)    Working buffer
IERR   I     Error return
               1 =>  can't open cat file or header file
               2 =>  input error
               3 =>  can't read catalog or header file
               4 =>  CLOSE blocked by non-REST status
               5 =>  end of catalog on OPEN or SRCH i.e.
                     no open slots or slot not found
               6 =>  on INFO requested slot not open
               7 =>  can't use WRIT status because now READ
               8 =>  on CLOSE the ID's don't match
               9 =>  Warning: read status added on a file
                     being written
              10 =>  Clear read/write when didn't exist warning
```

5.8.3 CATKEY

Reads or writes KEYWORDs from or to an AIPS image (or uv) header. The order of the keywords is arbitrary. Uses LUN 15, so any CA or CB files must be closed before calling this routine.

```
CATKEY (OPCODE, IVOL, CNO, KEYWRD, NUMKEY, LOCS, VALUES, KEYTYP,
  *   BUFFER, IERR)
Inputs:
   OPCODE   C*4         Operation desired, 'READ', 'WRIT',
                           'ALL ' => Read all.
                           'REED' => no error msg if some missing
   IVOL     I           File disk number
   CNO      I           File catalog block number
In/out:
   KEYWRD   C(*)*8      Keywords to read/write: output on ALL
   NUMKEY   I           Number of keywords to read/write.
                           Input on OPCODE='ALL' = max. to read.
                           Output on OPCODE='ALL' = no. read.
   LOCS     I(NUMKEY)   The word offset of first short integer
                        word of keyword value in array VALUES.
                        Output on READ, input on WRIT.
                        On READ this value will be -1 for keywords
                        not found.
   VALUES   I           The array of keyword values; due to word
                        alignment problems on some machines values
                        longer than a short integer should be copied,
                        eg. if the  5th keyword (XXX) is a R*8:
                             IPOINT = LOCS(5)
                             CALL COPY (NWDPDP, VALUES(IPOINT), XXX)
                        Output on READ, input on WRIT
   KEYTYP   I(NUMKEY)   The type code of the keywords:
                           1 = Double precision floating
                           2 = Single precision floating
                           3 = Character string (8 HOLLERITH chars)
                           4 = integer
```

```
                        5 = Logical
Output:
   BUFFER   I(256)     Scratch buffer
   IERR     I          Return code, 0=>OK,
                          1-10 => ZFIO error
                          19   => unrecognized data type.
                          20   => bad OPCODE
                          20+n => n keywords not found on READ.
                                  This produces messages at level 6
                                  suppress them w MSGSUP if needed
```

5.8.4 CATIO

CATIO reads or writes blocks in the map catalog header file.

```
CATIO (OP, IVOL, CNO, CATBLK, STAT, BUFF, IERR)
Inputs:
   OP       C*4      'READ' => get block into CATBLK
                     'WRIT' => put CATBLK onto disk catalog
                     'UPDT' => as WRIT but for use when the calling
                               program has previously set the
                               status to WRITE
   IVOL     I        Disk volume containing catalog (1 rel)
   CNO      I        Slot number of interest
   CATBLK   I(256)   Array to be written on disk: WRIT, UPDT
   STAT     C*4      Status desired for slot after operation
                        'READ','WRIT','REST' where REST => no
                        change of status is desired
Outputs:
   CATBLK   I(256)   Array read from disk: READ
   BUFF     I(256)   Working buffer
   IERR     I        Error code: 0 => ok
                        1 => cannot open catalog file
                        2 => input parameter error
                        3 => cannot read catalog file
                        4 => cannot WRIT/UPDT: file is busy
                        5 => did READ/UPDT, cannot add STAT = WRIT
                        6 => Warning on READ, file writing
                        7 => As 6, also added STAT=READ
                        8 => As 6, STAT inconsistent or wrong
                        9 => Warning: STAT inconsistent/wrong
The requested OP is performed unless IERR = 1 through 4.  The
final status requested is not set if IERR = 1 - 5, 8 - 9.  The
latter are probably unimportant.
```

5.8.5 COORDT

COORDT translates between types of coordinates:

```
COORDT (ITI, ITO, LONGI, LATI, EPOK, LONGO, LATO, IERR)
Inputs:
   ITI    I     Input type (1 Ra, Dec; 2 gal, 3 ecliptic)
   ITO    I     Output type
   LONGI  D     Input longitude
   LATI   D     Input latitude
```

```
   EPOK   R     Epoch of positions (used very simply with
                ecliptic coords only)
                1950 assumed in Galactic conversions!!!!!!!!!
Output:
   LONGO  D     Output longitude
   LATI   D     Output latitude
   IERR   I     error: 0 ok, 1 input error, 2 conversion fails
```

5.8.6 FNDX

FNDX returns the X axis coordinate value of a point given the Y axis coordinate value and the X axis pixel position of the point. Needed for rotations and non-linear axes (L-M).

```
FNDX (XPIX, YVAL, XVAL, IERR)
Inputs:
   XPIX   R     X pixel position
   YVAL   D     Y coordinate value
Output:
   XVAL   D     X coordinate value
   IERR   I     0 ok, 1 out of range, 2 bad type, 3 undefined
Common:
   Pos. parms in DLOC.INC must have been set up by SETLOC
```

5.8.7 FNDY

FNDY returns the Y axis coordinate value of a point given the X axis coordinate value and the Y axis pixel position of the point. Needed for rotations and non-linear axes (L-M).

```
FNDY (YPIX, XVAL, YVAL, IERR)
Inputs:
   YPIX   R     Y pixel position
   XVAL   D     X coordinate value
Output:
   YVAL   D     Y coordinate value
   IERR   I     0 ok, 1 out of range, 2 bad type, 3 undefined
Common:
   Pos. parms in DLOC.INC must have been set up by SETLOC
```

5.8.8 MAPCLS

closes a cataloged file, updates header on disk, clears catalog status.

```
MAPCLS (OP, IVOL, CNO, LUN, IND, CATBLK, CATUP, WBUFF, IERR)
Inputs:
   OP      C*4     OPcode used by MAPOPN to open this file
   IVOL    I       Disk volume containing map file
   CNO     I       Catalog slot number of file
   LUN     I       Logical unit # used for file
   IND     I       FTAB pointer for LUN
   CATBLK  I(256)  New catalog header which can optionally
                   be written into header if OP=WRIT or INIT
                   Dummy arguement if OP=READ
   CATUP   L       If TRUE, write CATBLK into catalog,
                   ignored if OP = READ
Outputs:
   IERR    I       0 = O.K.
```

```
                    1 = CATDIR couldn't access catalog
                    5 = illegal OP code
```

5.8.9 MAPOPN

MAPOPN opens a map file marking the catalog entry for the desired type of operation.

```
MAPOPN (OP, IVOL, NAMEIN, CLASIN, SEQIN, TYPIN, USID,
  *   LUN, IND, CNO, CATBLK, WBUFF, IERR)
Inputs:
   OP       C*4      Operation: READ, WRIT, or INIT where INIT is for
                     known creation processes (it ignores current file
                     status & leaves it unchanged).  Also: HDWR for
                     use when the header is being changed, but the
                     data are to be read only.
   LUN      I        Logical unit # to use
In/out:
   NAMEIN   C*12     Image name (name)
   CLASIN   C*6      Image name (class)
   SEQIN    I        Image name (seq.#)
   USID     I        User identification #
   IVOL     I        Input disk unit
   TYPIN    C*2      Physical type of file
Outputs:
   IND      I        FTAB pointer
   CNO      I        Catalog slot containing map
   CATBLK   I(256)   Buffer containing current catalog block
   WBUFF    I(256)   Working buffer for CATIO and CATDIR
   IERR     I        Error output: 0 = OK
                        2 = Can't open WRIT because file busy
                            or can't READ because file marked WRITE
                        3 = File not found
                        4 = Catalog i/o error
                        5 = Illegal OP code
                        6 = Can't open file
```

5.8.10 ROTFND

ROTFND finds the map rotation angle from a given catalog block.

```
ROTFND (CATR, ROT, IERR)
Inputs:
    CATR(*)   R    Map catalog header
Outputs:
    ROT       R    Map rotation angle (degrees)
    IERR      I    Error code. 0=>OK, 1=>couldn't find axis.
```

5.8.11 SETLOC

SETLOC uses the catalog header to build the values of the position commons in INCLUDE DLOC.INC for use by position finding and axis labeling routines (at least).

```
SETLOC (DEPTH, SWAPOK)
Inputs:
   DEPTH   I(5)      Position of map plane axes 3 - 7
   SWAPOK  L         T => okay to swap axes if rotation near 90
```

```
Common:
   DCAT.INC catalog block (not modified)
   DLOC.INC position parms - filled in here
```

5.8.12 TVFIND

TVFIND determines which of the visible TV images the user wishes to select. If there is more than one visible image, it requires the user to point at it with the cursor. The TV must already be open.

```
TVFIND (MAXPL, TYPE, IPL, UNIQUE, CATBLK, SCRTCH, IERR)
Inputs:
   MAXPL    I        Highest plane number allowed (i.e. do graphics
                     planes count?)
   TYPE     C*2      2-char image type to restrict search
Output:
   IPL      I        Plane number found
   UNIQUE   L        T => only one image visible now
                     (all types except zeroed ones ('ZZ'))
   CATBLK   I(256)   Image catalog block found
   SCRTCH   I(256)   Scratch buffer
   IERR     I        Error code: 0 => ok
                        1 => no image
                        2 => IO error in image catalog
                        3 => TV error
```

5.8.13 UVPGET

UVPGET determines pointers and other information from a UV CATBLK in the common in INCLUDE DCAT.INC. The address relative to the start of a vis record for the real part for a given spectral channel (CHAN) and stokes parameter (ICOR) is given by NRPARM+(CHAN-1)*INCF+ABS(ICOR-ICOR0)*INCS + (IF-1)*INCIF

Single dish data, i.e. randomly sampled data in the image plane, is also recognized and ILOCU and ILOCV point to the longitude like and latitude like random parameters. Also a "BEAM" random parameter may be substituted for the "BASELINE" random parameter. The data type present may be determined from the common variable TYPUVD. Two types of single dish data are recognized:

TYPUVD=1 = unprojected RA and Dec and
TYPUVD=2 = projected RA and Dec (ready for GRIDR)

```
UVPGET (IERR)
Inputs: From common /MAPHDR/ (DCAT.INC)
   CATBLK   I(256)   Catalog block
   CATH     H(256)   same as CATBLK
   CATR     R(256)   same as CATBLK
   CATD     D(128)   same as CATBLK
Output: In common /UVHDR/ (DUVH.INC)
   SOURCE   C*8      Source name.
   ILOCU    I        Offset from beginning of vis record of U
                     or longitude for single dish format data.
   ILOCV    I        Offset from beginning of vis record of V
                     or longitude for single dish format data.
   ILOCW    I        Offset from beginning of vis record of W.
   ILOCT    I                    "                    Time
   ILOCB    I                    "                 Baseline
                                                   (or beam)
   ILOCSU   I                    "                Source id.
```

```
ILOCFQ   I                    "                    Freq id.
JLOCC    I       0-rel. order in data of complex values
JLOCS    I       Order in data of Stokes' parameters.
JLOCF    I       Order in data of Frequency.
JLOCR    I       Order in data of RA
JLOCD    I       Order in data of dec.
JLOCIF   I       Order in data of IF.
INCS     I       Increment in data for stokes (see above)
INCF     I       Increment in data for freq. (see above)
INCIF    I       Increment in data for IF.
ICORO    I       Stokes value of first value.
NRPARM   I       Number of random parameters
LREC     I       Length in values of a vis record.
NVIS     I       Number of visibilities
FREQ     D       Frequency (Hz)
RA       D       Right ascension (1950) deg.
DEC      D       Declination (1950) deg.
NCOR     I       Number of correlators (Stokes' parm.)
ISORT    C*2     Sort order 1st 2 char meaningful.
TYPUVD   I       UV data type, 0=interferometer,
                    1=single dish unprojected,
                    2=single dish projected RA and Dec.
IERR     I       Return error code: 0=>OK,
                    1, 2, 5, 7 : not all normal rand parms
                    2, 3, 6, 7 : not all normal axes
```

5.8.14 XYPIX

XYPIX determines the pixel location corresponding to a specified coordinate value. The pixel location is not necessarily an integer. The position parms are provided by the commons in DLOC.INC which requires a previous call to SETLOC.

```
XYPIX (X, Y, XPIX, YPIX, IERR)
Inputs:
   X      D     X-coordinate value (header units)
   Y      D     Y-coordinate value (header units)
Output:
   XPIX   R     x-coordinate pixel location
   YPIX   R     y-coordinate pixel location
   IERR   I     0 ok, 1 out of range, 2 bad type, 3 undefined
```

5.8.15 XYVAL

XYVAL determines the coordinate value (X,Y,Z) corresponding to the pixel location (XPIX,YPIX). The pixel values need not be integers. The necessary map header data is passed via commons in DLOC.INC requiring a previous call to SETLOC. This program is the inverse of XYPIX.

```
XYVAL (XPIX, YPIX, X, Y, Z, IERR)
Inputs:
   XPIX    R    Pixel location, x-coordinate
   YPIX    R    Pixel location, y-coordinate
Outputs:
   X       D    X-coordinate value at pixel location
   Y       D    Y-coordinate value at pixel location
   Z       D    Z-coordinate value (if part of a position
```

```
                pair with either X or Y)
   IERR    I    0 ok, 1 out of range, 2 bad type, 3 undefined
Common inputs:
   DLOC.INC position parms deduced from the map header by
            subroutine SETLOC.
Units are as in the mapheader: degrees for position coords
```

5.8.16 YCINIT

Initialize image catalog for plane IPLANE - TK now done with TKCATL

```
YCINIT (IPLANE, BUFF)
Input:
   IPLANE   I        Image plane to initialize
Output:
   BUFF     I(256)   Working buffer
```

5.8.17 YCOVER

YCOVER checks to see if there are partially replaced images in any of the TV planes currently visible by quadrant.

```
YCOVER (OVER, BUF, IERR)
Outputs:
   OVER   L(4)     T => there are in quadr. I
   BUF    I(512)   scratch
   IERR   I        Error code: 0 => ok, other catlg IO error
```

5.8.18 YCWRIT

Write image catalog block in CATBLK into image catalog.

```
YCWRIT (IPLANE, IMAWIN, CATBLK, BUFF, IERR)
Inputs:
   IPLANE   I        image plane involved
   IMAWIN   I(4)     Corners of image on screen
   CATBLK   I(256)   Image catalog block
Outputs:
   BUFF     I(256)   working buffer
   IERR     I        error code: 0 => ok
                        1 => no room in catalog
                        2 => IO problems
```

5.8.19 YCREAD

Read image catalog block into CATBLK - TV only (TK in TKCATL).

```
YCREAD (IPLANE, IX, IY, CATBLK ,IERR)
Inputs:
   IPLANE  I       plane containing image whose block is wanted
   IX      I       X pixel coordinate of a point within image
   IY      I       Y pixel coordinate of point within image
Outputs:
```

```
CATBLK  I(256)  Image catalog block
IERR    I       error codes: 0 => ok
                   1 => IX, IY lies outside image
                   2 => Catalog i/o errors
                   3 => refers to TK device
```

Chapter 6

Disk files

6.1 Overview

Most images, uv data sets, scratch files, and other information in the AIPS system are kept in cataloged disk files. The purpose of this chapter is to describe the general techniques for accessing data in disk files.

Associated with each image or uv data file may be a number of auxiliary files known as "extension" files containing information about the main file. Examples of extension files are the history file, CLEAN components files and antenna files. Details of the structure of the various files used in AIPS programs are described in Appendix C. Except for the image and uv data files, the details of the file structure will not be described here.

The amount of data in the image and uv data files can be rather large, so it is important that the routines accessing them be relatively efficient. This efficiency comes at the cost of increased complexity. There are a number of features of AIPS I/O routines for handling large amounts of data which are designed for efficiency.

1. Fixed record length. All files internal to AIPS have a fixed logical record length. This allows the I/O routines to block disk transfers into a number of logical records.
2. Large double buffered transfers. The upper level I/O routines automatically make data transfers as large as possible and when possible double buffer the transfers.
3. Visible I/O buffers. To avoid an in-core transfer of all data, most AIPS routines work directly from the I/O buffer.

Extension files are handled somewhat differently. Since the amount of data in these files is rather small, friendlier, but less efficient, techniques are used. Logical records have a fixed length, but the basic I/O routine (TABIO) returns the data in an array which allows implementation of data structures.

This chapter discusses the various aspects of disk files — creating, destroying, reading, writing, etc. The cataloging of these files has been covered in a previous chapter. A typical programmer will not need to understand all of the material in this chapter to program effectively in AIPS. The detailed descriptions of the major routines discussed will be given at the end of the chapter.

6.2 Types of Files

AIPS has two logically different types of files, which on some machines are also physically different. The first type, known as regular disk files, is used mainly for extension files. I/O to this type of file is done in 512-AIPS byte blocks. The second type of file, known as "map" files, is used for image and uv data files. I/O to this type of file is usually done in the double buffered mode with large size transfers. (Double buffering is when the program works out of one half of a buffer, while the other half is being read from, or written to, the external device.) Both of these types of files may be expanded or contracted.

The principle distinction between the two types of files are the file creation and opening routines. Many of the higher level creation and file open routines hide this distinction from the programmer. These routines will be discussed later in this chapter.

6.3 File Management

AIPS has a set of utility routines for creating and managing disk files. The four functions covered in this section are file creation, destruction, extension and contraction.

6.3.1 Creating Files

There are several higher level file creation routines, one for each of several applications. These applications are image files, UV data files, scratch files, general extension files and history files. The basic file creation routine is ZCREAT.

- MCREAT creates and catalogs an image file (type "MA") using the description of the file contained in a catalog header record passed to MCREAT via the common /MAPHDR/ (INCLUDE DCAT.INC). All information in the header defining the size and name of the file must be filled in before calling MCREAT. The catalog header record is described in detail in another chapter.
- UVCREA creates and catalogs a uv data file (type "UV") using the description of the file contained in the catalog header record passed to UVCREA in the common /MAPHDR/ (INCLUDE DCAT.INC). The catalog header record must be sufficiently complete to determine the name, class, etc. and size of the required file.
- SCREAT will create scratch files using the /CFILES/ (INCLUDE DFIL.INC) common system; thus the scratch files will be automatically deleted when the task calls the shutdown routine DIE. Scratch files are cataloged as type "SC" files. Use of SCREAT is described in more detail in the chapter describing tasks.
- TABINI. The creation of most tables extension files is hidden from the casual programmer in the create/open/initialize routine TABINI. TABINI will be discussed in more detail in the chapter on tables.
- HICREA. The creation of history files is normally hidden in the upper level routine HISCOP. The use of HISCOP and HICREA are described in more detail in the chapter on writing tasks.
- ZCREAT. The basic file creation routine is ZCREAT. If none of the other file creation routines are applicable, then use ZCREAT. ZCREAT needs the physical name of the file and the size of the file in bytes. ZCREAT does not catalog the file created.

6.3.2 Example Using ZCREAT

The use of ZCREAT is demonstrated in the following:

```
      INTEGER   IRET, NX, NY, NP(2), NBYTE, IVOL, CNO, IVER, IERR
      CHARACTER PHNAME*48
      LOGICAL   MAP
      PARAMETER (MAP = .TRUE.)
      INCLUDE 'INCS:DDCH.INC'
           ...
C                                     NX, NY are the size of an
C                                     image.  Determine file size.
      NP(1) = NX
      NP(2) = NY
      CALL MAPSIZ (2, NP, NBYTE)
C                                     Size now in NBYTE
```

```
C                                       Make physical name.
C                                       IVOL = disk number
C                                       CNO = catalog slot number
C                                       IVER = extension file
C                                           version number.
C                                           1 for main cataloged files.
      CALL ZPHFIL ('MA', IVOL, CNO, IVER, PNNAME, IERR)
C                                       filename now in PHNAME.
C                                       (error if IERR not 0)
C                                       Create file of type 'MA'
      CALL ZCREAT (IVOL, PNNAME, NBYTE, MAP, IERR)
C                                       Test for errors...
```

In the example above, a map file was created large enough to hold a NX by NY image using the routine MAPSIZ to compute the correct size for the file. To catalog this file a catalog header record should be constructed and calls made to CATDIR and CATIO *before* the call to ZPHFIL to get the catalog slot number needed to form the physical name of the file. A detailed description of the calling sequence for ZCREAT can be found at the end of this chapter. (In practice, one would use MCREAT to catalog and create the file shown in the example above.)

6.3.3 Destruction Routines

There are a number of special purpose file destruction routines; the basic file destruction routine is ZDESTR. A brief description is given here of these utility routines; a description of the call sequence is given at the end of this chapter.

- MDESTR will delete a catalog entry for a file, delete all extension files for that file, and then delete the file. The file must be in the REST state. Since catalog files can be marked "WRITE — Destroy if task fails" which will cause the shutdown routine DIE to destroy the file there is seldom a need to call MDESTR directly. MDESTR will destroy either cataloged image or uv data files.
- SNDY will destroy scratch files described in the /CFILES/ common (INCLUDE DFIL.INC). SNDY is called by the shutdown utility DIE so tasks do not have to call it separately.
- ZDESTR is the basic file destruction routine. ZDESTR will not uncatalog the file destroyed. CATDIR should be used to uncatalog a cataloged file destroyed.

6.3.4 Expansion and Contraction of Files

All files can be both expanded and compressed. Since most extension file access is by TABIO, the expansion of extension files is hidden from the programmer. Expansion of files is done with routine ZEXPND and compression is done using routine ZCMPRS. Details of the call sequences of these routines are given at the end of this chapter.

6.4 I/O to Disk Files

There are a number of steps necessary in order to access a disk file. Normal Fortran I/O hides a number of these steps but they are all visible in at least some AIPS applications. This increased complexity of the I/O system gives the programmer a high degree of control over how the I/O is actually done. One or more of the steps in accessing a file may be performed with a single call. In general, access of a disk file is as follows:

1. Forming the physical name of the file. The AIPS utility ZPHFIL is always used for this purpose. The name is derived from file type, the disk number, catalog slot number, version number and user ID number. Also a revision code is usually included in the physical name of the file so that versions of AIPS with incompatable file formats can coexist in the same directories. The file type of image files is "MA", of uv data files is "UV" and of scratch files is "SC". The disk number and catalog slot number for cataloged files may have to be obtained from the AIPS utility routine CATDIR before calling ZPHFIL. This step is incorporated in a number of routines such as SCREAT, TABINI and MAPOPN.

2. Opening the file. This is done with routine ZOPEN for binary files and ZTOPEN or ZTXOPN for text files. In either case, the file must be given a logical unit number (LUN) and the opening routine returns a pointer to the AIPS I/O table (FTAB) which, with the LUN, must be used in all subsequent calls. This step is incorporated in the routines TABINI and MAPOPN.

3. Initializing the transfers. The AIPS higher level I/O routines need to be told a number of parameters about the data transfers, such as whether a read or write is desired, the size and number of logical records, and the location and size of the buffer to be used. In several cases the range of data desired can also be specified. This step is usually done in one of the specialized routines to be described later.

4. Data transfers. This is when the data is transferred from the disk to the specified buffer or vice versa. Actual data transfers are done by Direct Memory Access (DMA) and are usually in large blocks for "map" files and in 512-byte blocks for non-map (extension) files. Since the transfers usually consist of a number of logical records, the programmer is unaware of when transfers actually take place. Because the programs frequently work directly from the I/O buffer, many of the I/O routines return a pointer to the first word in the buffer of the next logical record.

5. Flushing the buffer (writing only). When all calls to disk write routines are complete, there may still be data in the buffer which has not been written. In this case, a call must be made to the appropriate I/O routine telling it to flush the buffer to disk.

6. Closing the file. When all operations on a file are complete the file needs to be closed. This is usually done with an explicit call to the appropriate close routine.

6.4.1 Upper Level I/O Routines

There are a number of AIPS upper level I/O routines which do most of the bookkeeping. The following is a short description of the more commonly used of these; detailed descriptions of the call sequences are found at the end of the chapter. The use of many of these routines is discussed later in this chapter.

- TABINI opens and initializes an table extension file, will create and catalog the extension file if necessary. See the chapter on tables for more details.
- TABIO does random access mixed reads and writes to extension tables. TABIO deals with one logical record at a time in an array which can be used as a data structure. TABIO takes care of file expansion and other bookkeeping chores. Requires initialization by TABINI.
- MAPOPN finds a cataloged image or uv data file in the catalog, opens it and returns the catalog header and marks the catalog status.
- MINIT initializes I/O for image files; can specify a subimage for reads.
- MDISK does double buffered I/O for image files; requires initialization by MINIT.
- UVINIT initializes I/O for uv data files; can specify a starting visibility record number.
- UVDISK does double buffered I/O for uv data files; requires initialization by UVINIT.
- MAPCLS closes a cataloged image or uv data file, updates the catalog header block if requested and clears the catalog status.

6.4.2 Logical unit numbers

Many logical unit numbers in AIPS have special meanings which indicate to the I/O routines what kind of device or file is involved. The information about which LUN corresponds to which device is contained in a table (DEVTAB) in the device characteristics common (INCLUDE DDCH.INC). AIPS has 50 defined LUN values, i.e., DEVTAB has 50 entries, and the type of device or file type for each LUN is given in DEVTAB with the following codes:

```
DEVTAB(LUN) = 0   LUN is for disk file requiring I/O control area in
                  FTAB.  Multi-record I/O is possible.
DEVTAB(LUN) = 1   Device not requiring I/O control area in FTAB.
                  I/O done by Fortran (terminals, printer/plotter).
DEVTAB(LUN) = 2   LUN is for device requiring I/O control area in
                  FTAB.  Multi-record I/O not allowed (e.g., tapes)
DEVTAB(LUN) = 3   Similar to 1.  VAX uses this code to defer opens
                  from ZOPEN to ZTOPEN for text files.
DEVTAB(LUN) = 4   LUN is for TV device requiring special I/O routine
                  and normal I/O control area in FTAB.
```

In addition, many LUNs have predefined values as shown in the following table.

LUN	Use
1	Line printer
2	Plotter
3	Reserved
4	Input to batch processors
5	Input CRT
6	Output CRT
7	Graphics CRT
8	Array Processor (roller)
9	TV device
10	POPS "run" files
11	POPS "help" files
12	Log/error file (used by MSGWRT).
13	Task communication file.
14	POPS "memory" file
15	Catalog files.
16 - 25	Map (image or uv data) files.
26	Graphics files
27 - 30	General (non-map) disk files.
31 - 3?	Magnetic tape drives (31 - 30+NTAPED)

6.4.3 Contents of the Device Characteristics Commons

The device Characteristics commons, obtained from the INCLUDE DDCH.INC contains a number of useful parameters about the host system.

6.4.4 Image Files

A disk image file contains an ordered, binary sequence of pixel values with logical records consisting of single "rows" of the image. The pixel values are arranged in the order defined in the catalog header block, the first axis going the fastest. Blanking of pixels is allowed by use of a special value (magic value blanking) specified by the header. For more information about the catalog header and the typical axes used, see the chapter on the catalog.

Image files are stored on the disk with each row beginning on a block boundary. An exception to this is when multiple rows will fit into a single block, in which case multiple rows can be in a given disk block. In this latter case, rows are not allowed to span block boundaries.

Opening Image Files

The simplest way to find, open and close a cataloged image file is with the routines MAPOPN and MAPCLS. These routines and the alternate ways to find an image in the catalog are discussed in the chapter on the catalog and details of the call sequence are found at the end of this chapter.

If the use of MAPOPN and MAPCLS is not appropriate to open and close the image file, then the routines ZPHFIL, ZOPEN and ZCLOSE are to be used to (1) form the physical name of the file, (2) open the file, both in the AIPS and system tables, and (3) close the file when done. The details of these routines are given at the end of this chapter. These operations are demonstrated in the following example.

```
      INTEGER    IRET, CNO, IVOL, IVER, LUN, IND
      LOGICAL    MAP, EXCL, WAIT
      CHARACTER  PHNAME*48
      PARAMETER (MAP = .TRUE.)
      PARAMETER (EXCL = .TRUE.)
      PARAMETER (WAIT = .TRUE.)
      PARAMETER (LUN = 16)
           .
           .
      DATA IVER /1/
           .
           .
C                                     Make physical name.
C                                     'MA' = file type
C                                     IVOL = disk number
C                                     CNO = catalog slot number
C                                         (arbitrary for
C                                         uncataloged files).
C                                     IVER = extension file
C                                         version number.
C                                         1 for main cataloged
C                                         files. Arbitrary
C                                         otherwise.
      CALL ZPHFIL ('MA', IVOL, CNO, IVER, PHNAME, IRET)
C                                     filename now in PHNAME.
C                                     (error if IRET not 0)
C                                     Open file
      CALL ZOPEN (LUN, IND, IVOL, PHNAME, MAP, EXCL, WAIT, IRET)
C                                     Test for errors (IRET not 0)
           .
           .
    ( I/O to file )
```

```
            .
            .
C                                       Close file.
      CALL ZCLOSE (LUN, IND, IRET)
```

MINIT and MDISK

Once the image file is opened, I/O is normally initialized by a call to MINIT; I/O is done by calls to MDISK, with a final call to MDISK to flush the buffer, if necessary. MINIT sets up the bookkeeping for one plane of an image at a time; if multiple planes are to be read, multiple calls to MINIT must be made. A rectangular window in a given plane can be specified to MINIT, and it can be instructed to read or write the rows in reverse order by reversing the values of WIN(2) and WIN(4). A subimage cannot be specified for write.

Due to the use of buffer pointers, MDISK must be called for WRITE *before* placing data into the buffer. This produces a rather strange logic flow, but is necessary. Details of the call sequences to MINIT and MDISK are given at the end of this chapter.

Multi-plane Images (COMOFF)

If the image has more than two dimensions, planes parallel to the first plane can be accessed using the block offset argument to MINIT. The subroutine COMOFF is to be used to compute the block offset. The block offset is an integer whose value for the first plane is 1. COMOFF returns a value which is to be added to the block offset for the first plane.

An example of the use of COMOFF to compute the block offset:

```
      INTEGER   BLKOF, PLARR(5), IERR
      INCLUDE 'DHDR.INC'
      INCLUDE 'DDCH.INC'
      INCLUDE 'DCAT.INC'
            .
            .
C                                       Get second plane on third
C                                       axis, first pixel on
C                                       the remaining axes.
      PLARR(1) = 2
      PLARR(2) = 1
      PLARR(3) = 1
      PLARR(4) = 1
      PLARR(5) = 1
C                                       PLARR specifies desired plane
C                                       Use header block from DCAT.INC
      CALL COMOFF (CATBLK(K2DIM), CATBLK(K2NAX), PLARR, BLKOF, IERR)
C                                       Add block offset for first
C                                       plane.
      BLKOF = BLKOF + 1
C                                       BLKOF now contains the value
C                                       to send to MINIT to get the
C                                       specified plane.
```

A detailed description of the call sequence for COMOFF is given at the end of this chapter.

Example of MINIT and MDISK

In the following is an example in which two files are read, the pixel values are added and a third file is written.

```
      SUBROUTINE FLADD (NX, NY, ISCR1, ISCR2, ISCR3, IERR)
C-----------------------------------------------------------------------
C   FLADD adds the values in the scratch files in the /CFILES/ common
C   (include DFIL.INC) number ISCR1 and ISCR2 and writes them in the
C   /CFILES/ scratch file number ISCR3
C   Inputs:
C      NX, NY  I  Number of pixels per row and number of rows
C      ISCR1   I  /CFILES/ scratch file number of first input file
C      ISCR2   I  /CFILES/ scratch file number of second input file
C      ISCR3   I  /CFILES/ scratch file number of output file
C   Output:
C      IERR    I  Return code, 0=>OK, otherwise error.
C-----------------------------------------------------------------------
      INTEGER   NX, NY, ISCR1, ISCR2, ISCR3, IERR
C
      INTEGER   FIND1, FIND2, FIND3, BIND1, BIND2, BIND3, BO,
     *   WIN(4), BUFSZ1, BUFSZ2, BUFSZ3, LUN1, LUN2, LUN3
      LOGICAL   T, F
      CHARACTER FILE*48
      REAL      BUFF1(4096), BUFF2(4096), BUFF3(4096)
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DFIL.INC'
      DATA BO, WIN /1, 4*0/
C                                        Use LUNs 16, 17, 18
      DATA LUN1, LUN2, LUN3 /16,17,18/
C-----------------------------------------------------------------------
C                                        Set buffer sizes
      BUFSZ1 = 4096 * 2
      BUFSZ2 = 4096 * 2
      BUFSZ3 = 4096 * 2
C                                         Open and init ISCR1
      CALL ZPHFIL ('SC', SCRVOL(ISCR1), SCRCNO(ISCR1), 1, FILE, IERR)
      CALL ZOPEN (LUN1, FIND1, SCRVOL(ISCR1), FILE, T, F, T, IERR)
C                                         Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'READ', 1
         GO TO 990
         END IF
      CALL MINIT ('READ', LUN1, FIND1, NX, NY, WIN, BUFF1, BUFSZ1, BO,
     *   IERR)
C                                         Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'READ', 1
         GO TO 990
         END IF
C                                         Open and init ISCR2
      CALL ZPHFIL ('SC', SCRVOL(ISCR2), SCRCNO(ISCR2), 1, FILE, IERR)
      CALL ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), FILE, T, F, T, IERR)
```

```
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'READ', 1
         GO TO 990
         END IF
      CALL MINIT ('READ', LUN2, FIND2, NX, NY, WIN, BUFF2, BUFSZ2, BO,
     *   IERR)
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'READ',2
         GO TO 990
         END IF
C                                       Open and init ISCR3
      CALL ZPHFIL ('SC', SCRVOL(ISCR3), SCRCNO(ISCR3), 1, FILE, IERR)
      CALL ZOPEN (LUN3, FIND3, SCRVOL(ISCR3), FILE, T, F, T, IERR)
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'WRIT'
         GO TO 990
         END IF
      CALL MINIT ('WRIT', LUN3, FIND3, NX, NY, WIN, BUFF3, BUFSZ3, BO,
     *   IERR)
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'WRIT'
         GO TO 990
         END IF
C                                       Loop, adding rows.
      DO 110 I = 1,NY
C                                       Read ISCR1
         CALL MDISK ('READ', LUN1, FIND1, BUFF1, BIND1, IERR)
C                                       Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'READ', 1
            GO TO 990
            END IF
C                                       Read ISCR2
         CALL MDISK ('READ', LUN2, FIND2, BUFF2, BIND2, IERR)
C                                       Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'READ',2
            GO TO 990
            END IF
C                                       Write ISCR3
         CALL MDISK ('WRIT', LUN3, FIND3, BUFF3, BIND3, IERR)
C                                       Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'WRIT'
            GO TO 990
            END IF
C                                       Add row.
         DO 100 J = 1,NX
C                                      Note: buffer pointer is to
C                                      first element so need zero
```

```
C                                       relative index for each pixel.
            J1 = J - 1
            BUFF3(BIND3+J1) = BUFF1(BIND1+J1) + BUFF2(BIND2+J1)
 100        CONTINUE
 110     CONTINUE
C                                        Flush buffer.
      CALL MDISK ('FINI', LUN3, FIND3, BUFF3, BIND3, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'FINI'
         GO TO 990
         END IF
C                                        Close files.
      CALL ZCLOSE (LUN1, FIND1, IERR)
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 1
         GO TO 990
         END IF
      CALL ZCLOSE (LUN2, FIND2, IERR)
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 2
         GO TO 990
         END IF
      CALL ZCLOSE (LUN3, FIND3, IERR)
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 3
         GO TO 990
         END IF
C                                       Finished OK.
      GO TO 999
C                                       An error has occurred - send
C                                       message
 990  CALL MSGWRT (8)
C
 999  RETURN
C---------------------------------------------------------------------
 1000 FORMAT ('FLADD: ERROR',I3,' OPEN FOR ',A4,' FILE',I2)
 1010 FORMAT ('FLADD: ERROR',I3,' INIT FOR ',A4,' FILE',I2)
 1060 FORMAT ('FLADD: ERROR',I3,1X,A4,'ING FILE',I2)
      END
```

MINSK and MSKIP

There are some operations, such as transposing images, in which it is convenient to read every n'th row of an image. The pair of routines MINSK and MSKIP will do this operation. Descriptions of these routines can be found at the end of this chapter.

6.4.5 Image File Manipulation Routines

There are a number of AIPS utility routines available to operate on files. Many of these involve copying data from catalog files to scratch files or vice versa. Details of the call sequences to these routines are given at the end of this chapter.

- PLNGET reads a selected portion of a selected plane from a cataloged file and writes it into a specified scratch file. The output file will be zero padded and a shift of the center may be specified.
- PLNPUT writes a subregion of a scratch file image into a cataloged image.

6.4.6 UV Data Files

Interferometers take samples of the visibility (also called coherence) function of a wavefront at random locations so these data must be stored differently from images. Also, this data may be in a variety of forms, calibrated or raw, one source per file or many. The following sections describe these uv data files.

Single-source Files

The simplest form of a uv data file is the single source file which contains data from a single celestial source and is usually assumed to be calibrated and edited. A given visibility record consists of all data taken on a given baseline (with a pair of antennas) at a given time. Thus, this record may contain data for a number of frequencies and/or polarizations. Each measurment consists of a triplet of values giving the real part, the imaginary part and the weight of a given visibility sample. (This may be modified for compressed data; see the section on compressed data). The visibility records in this type of file may be in an arbitrary order depending on the application.

Each uv data file needs one or more antenna (AN) tables to describe the locations and other properties of the antennas used. The visibility data contains, in a coded form, the numbers of the antennas involved in each baseline. The antenna numbers refer to entries in the Antenna table.

There are occasions where data is sampled at a number of relatively arbitrary frequencies. For these cases we have introduced a frequency-like axis called IF. The offset of the frequency of the reference pixel in each IF group of data from the file reference frequency is given in the FQ table. Since these sets of frequencies may change in a given file (e.g. bandwidth synthesis or rotation measure studies) a random parameter in the data file labeled ('FQID') points to a given entry in the FQ table. In general, a single source file will contain only a single FQ id.

Multi-source Files

In order to allow the use of calibration and editing software, multi-source files contain data from more than one source. In addition, the data are in relatively raw form and have associated calibration and editing tables which must be applied before the data are used. This type of file has an index and must be in strict time-baseline order. The structure of multi-source data files is very similar to the single source file. The use of multi-source files is described in detail in the chapter on Calibration and Editing in Volume 2.

The principal difference between the single-source files and the multi-source files is the addition, in the latter, of a source number random parameter and a number of associated tables. Several of these tables are described in the following:

- SU table. This table contains the information specific to a given source (e.g., position)
- NX table. This table contains an index for the file, telling when each source was observed.
- CL tables. These tables contain the information necessary to calibrate the data.
- FG tables. These tables contain the information necessary to flag bad data.

Read access to multi-source files is through the routines UVGET and CALCOP. UVGET selects, reformats, flags and calibrates data as specified and returns one visibility per call after setup. CALCOP will copy all selected records after setup by UVGET. The details of the call sequences of these routines are given at the end of this chapter. These routines handle all of the I/O chores described in this chapter and will also work for single source data files.

Compressed Data

AIPS supports a "Compressed" format for uv data. In this form there is a single "Weight" and scaling random parameter in each visibility record and the real and imaginary parts of the correlation values are packed into a single REAL value with magic value blanking. The details of the packing is machine dependent and is implemented via the "Z" routines ZUVPAK and ZUVXPN. Compressed data can be identified by a dimension of 1 on the "COMPLEX" (first) axis of the data array. The calibration package (UVGET) will automatically unpack visibility data.

Subarrays

Since uv data sets frequently contain data from physically separate arrays, AIPS uv data sets can contain "sub arrays". This is necessary so that the physical identity of each antenna in a visibility record can be uniquely established. Each subarray has its own antenna file, which contains the true frequency and date of observation and the locations and other information about each antenna.

When uv data sets are concatenated, the u, v and w terms of each subsequent data set are converted to wavelengths at the reference frequency defined by the first data set. The subarray number is encoded into the baseline number in each visibility record. The older practice of offsetting times by (subarray-1) * 5 days is being phased out, but still appears in some applications.

Visibility record structure

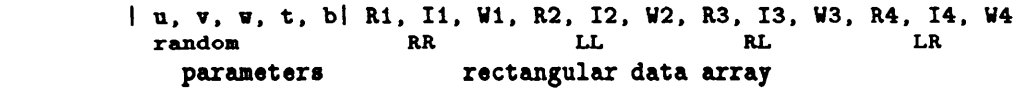
AIPS uv data is organized in the data file in the same way that similar data is organized on a FITS random groups format tape. Each logical record consists of all data on a given baseline for a given integration period; that is, all polarizations, frequencies, and IFs are contained in a given logical record. The first portion of a logical record is a list of the "random" parameters such as u, v, time, etc. Following the random parameters comes a regular array of data, which is very similar to a small image file. The length and structure of the visibility logical record is fixed in a given data base, but may vary from one data base to another. Records may span disk sector boundaries.

The random parameters can be in any order, but the names of only the first 14 are kept in the catalog header record; this list defines the order in which the values occur. The labels for the normal u, v and w random parameters are "UU-L-SIN", "VV-L-SIN", "WW-L-SIN" indicating that the coordinates correspond to the tangent point of the data computed using sine projection and the units are wavelengths at the reference frequency. The label for the time random parameter is "TIME1" for historical reasons and the label for the baseline parameter is "BASELINE". The label for the source number random parameter is "SOURCE"; the source number points to an entry in the source (SU) table. Other "standard" but optional random parameters are "FQID" for the FQ table identifier and "WEIGHT" and "SCALE" for compressed data.

The regular portion of the array is like an image array in that the order of the axes is arbitrary. In practice for uncompressed data, the first axis should be the COMPLEX axis (real, imaginary, weight). As in image files, the RA, Dec and frequency (for continuum data) are dummy axes which provide a place to store the values for these parameters.

A "regular" axis, which is not intrinsically regular, is what will be referred to as IFs. These are the results of separate receivers (either at RF or IF) which are randomly spaced, but have one or more regularly spaced frequency channels. The pixel number of these IFs points to an entry in the FQ table which gives the frequency offset from the reference frequency for that IF. The FQ table is accessed by the routine CHNDAT, whose call sequence is given at the end of this chapter. The values of the frequency offsets are allowed to be variable inside of a given data set and the set of frequencies and bandwidths used in a given visibility record is specified by an optional FQ identifier random parameter labeled "FQID". One entry is made in the FQ table for each set of frequencies and/or bandwidths.

The structure of a typical VLA data record from a single source file with a single IF is shown in the following figure.



The symbols in the above are:

- u = u coordinate in wavelengths at the reference frequency
- v = v coordinate
- w = w coordinate
- t = time in days since reference date given in antenna file for this subarray. (The time may be offset by 5 x (subarray no. - 1))
- b = baseline code; 256 x antenna 1 no. + antenna 2 no. + 0.01 x (subarray no. - 1). (see later section for more details)
- Rn = the real part of a correlator value in Jy.
- In = the imaginary part of a correlator value.
- Wn = the weight assigned to the correlator value. In general, it is arbitrary. Data with Wn $\leq$ 0 are "flagged" (to be ignored).

AIPS uv data sets may contain data in either true Stokes' parameters or correlator based values for circularly polarized IFs. Since Stokes' parameters are not an inherently ordered set, we have adopted the following convention for the values along the Stokes' axis:

Stokes' (or correlator) parameter	Value
I	1
Q	2
U	3
V	4
RR	-1
LL	-2
RL	-3
LR	-4
XX	-5
YY	-6
XY	-7
YX	-8

The orientations of the "X and "Y" linearly polarized feeds are defined in the antenna (AN) table.

The order of the visibility records in a single source file may be changed; this is usually done with the task UVSRT. Sorting is done using a two key sort and the current sort order is described in the catalog header record (CATBLK(KITYP)) as a two-character HOLLERITH string. The codes currently defined for the sort order are given in the following table, the first key in the sort order varies more slowly.

```
B => baseline number
T => time order
U => u spatial freq. coordinate
V => v spatial freq. coordinate
W => w spatial freq. coordinate
R => baseline length.
P => baseline position angle.
```

```
X => descending ABS(u)
Y => descending ABS(v)
Z => ascending ABS(u)
M => ascending ABS(v)
* => not sorted
```

As examples of the use of the sort order, the older mapping routines require "XY" sorted data (actually they are happy as long as the first key is "X"), self calibration tasks require "TB" order, etc.

Data Order, UVPGET

The position in the record of the standard random parameters (u,v,w,t,b) and the order of the regular axes can be obtained using the routine UVPGET. UVPGET determines pointers and other information from a uv data file catalog header record in common /MAPHDR/ from include DCAT.INC. These pointers are placed in commons which are obtained by the DUVH.INC INCLUDE. The address relative to the start of a vis record for the real part for a given spectral channel (CHAN), IF (NIF) and Stokes parameter (ICOR) is given by:

```
NRPARM + (CHAN-1) * INCF + (NIF-1) INCIF + ABS (ICOR-ICOR0) * INCS
```

Antenna and Subarray Numbers

Antenna and subarray numbers are coded into a single floating word. Some care must be used in decoding these values. The following example shows how to extract these values from a buffer BUFF with UVDISK pointer IBIND and baseline offset ILOCB from DUVH.INC.

```
      INTEGER   IBASE, BIND, ANT1, ANT2, SUBAR
      REAL      BUFF(*), BASE
            ...
C                                       Extract from buffer
      BASE = BUFF(BIND+ILOCB)
C                                       First antenna number
      ANT1 = (BASE / 256.0) + 0.1
C                                       Second antenna number
      ANT2 = (BASE - ANT1 * 256.0) + 0.1
      IBASE = BASE + 0.1
C                                       Subarray number
      SUBAR = (BASE - IBASE) * 100.0 + 1.1
```

Data Reformatting Routines

The variety of different uv data formats, especially different polarization types, allowed in AIPS uv data bases complicates handling of uv data. If a routine is to read and write uv data, it must be prepared to handle any allowed data type. If the routine is only reading the data, reformatting the data to a standard form is practical. There are a number of reformatting routines available.

Efficient reformatting requires two routines, one to setup arrays of pointers and factors and the second to reformat each record. The following list describes several such pairs; detailed descriptions of the call sequence to the routines can be found at the end of this chapter.

- SET1VS, GET1VS return a single visibility value in true Stokes' parameter (I, Q, U, V) or circular polarization (RCP, LCP). They may be requested to work on multiple frequency channels. Does not allow specification of IF at present; defaults to the first.
- SETVIS, GETVIS return several visibility values in the form of true Stokes' parameter (I, Q, U, V) or circular polarization (RCP, LCP). They may be requested to work on multiple frequency channels. A single IF may be specified.

- DGINIT, DGGET are the most general data selection/Stokes' translation routines.
- UVGET sets up, selects, reformats, calibrates, edits either single- or multi-source data files. After set up by UVGET, CALCOP can be used to copy the contents of a file to another file.

6.4.7 UV Data Access

The following is a discussion of the routines to access UV data.

UVGET and CALCOP

Routine UVGET allows relatively easy access to all kinds of AIPS interferometer uv data from both single- and multi-source files, in either normal or compressed format and can optionally select, calibrate, edit and convert the stokes parameter of the data selected. After an initialization call UVGET returns one visibility at a time. UVGET can apply SN, BL or BP calibration tables and/or make polarization corrections as specified in the AN table to single source files. Most of the communication with UVGET is through the commons in INCLUDE file DSEL.INC which are described in the description of UVGET at the end of this chapter. These values may be initialized using routine SELINI whose description appears at the end of this chapter.

If it is more convienent to operate on a uv data scratch file than on one visibility at a time (e.g. multiple passes throught the data are required) then CALCOP can be used to produce a file containing the selected data with any calibration etc. operations done on them. CALCOP will optionally create the scratch file. A description of CALCOP is given at the end of this chapter.

UVINIT and UVDISK

UV data files may be located and opened using routine MAPOPN.Data are read or written using UVINIT and UVDISK in much the same manner in which image files are read with MINIT and MDISK. One significant difference between UVDISK and MDISK is that UVDISK can be requested to process multiple logical records (NPIO) in a single call. If NPIO is 0, then the largest value consistent with double buffering will be used; if NPIO is too large for the buffer provided, it will be reduced to the largest value consistent with single buffering. This is useful when large amounts of data are to be sent to a sorting routine or to the array processor or to reduce the overhead of many subroutine calls.

Another difference between MINIT and UVINIT is that, unlike MINIT, UVINIT returns the buffer pointer for the first call so the output buffer can be written into *before* the first call to UVDISK.

UVINIT sets up the bookkeeping for UVDISK which does double buffered (if possible) quick-return I/O. UVDISK will run much more efficiently if, on disk, the requested transfer (logical record length x the number of records per call) is an integral number of disk blocks. Otherwise, partial writes or oversize reads will have to be done. Minimum disk I/O is one block.

The buffer size for UVDISK should include an extra NBPS bytes for each buffer for reads, if NPIO records does not correspond to an integral number of disk sectors (NBPS bytes). 2*NBPS extra bytes required for each (single or double) buffer for writes. More details about the call sequence to UVINIT and the use of the FTAB are given at the end of this chapter.

UVDISK reads and writes records of arbitrary length, especially uv visibility data. There are three operations which can be invoked: READ, WRITE and FLUSH (OPcodes "READ", "WRIT" and "FLSH").

If the requested transfers are too large to double buffer with the given buffer size, then UVDISK will single buffer the I/O. If it is possible to do double buffered physical transfers of some multiple of the requested number of records, then this is done.

OPcode ="READ" reads the next sequential block of data as specified to UVINIT and returns the actual number of visibilities, NIO, and the pointer, BIND, to the first word of this data in the buffer.

OPcode="WRIT" collects data in a buffer half until it is full. Then, as many full blocks as possible are written to the disk with the remainder left for the next disk write. For writes, left-over data is transferred to the beginning of the next buffer half to be filled. The value of NIO in the call is the number of visibility records to be added to the buffer and may be fewer than the number specified to UVINIT. On return, NIO

is the maximum number which may be sent next time. On return, BIND is the pointer in BUFFER to begin filling new data.

OPcode="FLSH" writes integral numbers of blocks and moves any data left over to the beginning of buffer 1. One exception to this is when NIO $\leq$ 0, in which case the entire remaining data in the buffer is written (if NIO $<$ 0 then ABS (NIO) visibilities are to be written). After the call, BIND is the pointer in BUFFER for new data. The principal difference between FLSH and WRIT is that FLSH always forces an I/O transfer. This may cause trouble if a transfer of less than 1 block is requested. A call with a nonpositive value of NIO should be the last call and corresponds to a call to MDISK with opcode "FINI".

The input/output argument to UVDISK, NIO, can be very useful for controlling the loop reading and/or writing uv data. The value of NIO for reads is the number of values in the buffer that are available. When the file has been completely read, the value of NIO returned by UVDISK on the next call is 0; this value can be used to determine when all of the data has been read. The example in the following section uses this feature in UVDISK. More details about the call sequence can be found at the end of this chapter.

Example using UVINIT and UVDISK

```
      SUBROUTINE UVCONJ (ISCR1, ISCR2, LUN1, LUN2, BUFF1, BUFF2,
     *   BUFSZ1, BUFSZ2, IERR)
C-----------------------------------------------------------------
C   UVCONJ takes the complex conjugate of the values in a uv data set
C   in a scratch file in the /CFILES/ common (INCLUDE DFIL.INC) number
C   ISCR1 and writes them in the /CFILES/ scratch  file number ISCR2.
C   The current values in the /UVHDR/ commons (INCLUDE DUVH.INC) are
C   assumed to describe the uv data files.
C   Inputs:
C      ISCR1   I     /CFILES/ scratch file number of input file
C      ISCR2   I     /CFILES/ scratch file number of output file
C      LUN1    I     Logical unit number to use for file 1
C      LUN2    I     Logical unit number to use for file 2
C      BUFF1   R(*)  I/O buffer to use for file 1
C      BUFF2   R(*)  I/O buffer to use for file 2
C      BUFSZ1  I     Size of BUFF1 in AIPS bytes (2*no. words)
C      BUFSZ2  I     Size of BUFF2 in AIPS bytes
C   Inputs from common /UVHDR/ (DUVH.INC)
C      NVIS    I     Number of visibility records
C      LREC    I     logical record length.
C      NRPARM  I     number of random parameters.
C      ICOR0   I     (signed) value of first Stokes' parameter.
C      JLOCF   I     zero relative order of the frequency axis in
C                    the data array.
C      JLOCS   I     relative order of the Stokes' axis.
C      JLOCIF  I     relative order of the IF axis.
C      INCF    I     word increment in the data array between
C                    successive frequencies at the same location on
C                    all other axes.
C      INCS    I     word increment in the data array between
C                    successive Stokes' values.
C      INCS    I     word increment in the data array between
C                    successive IF values.
C   Inputs from common /MAPHDR/ (DCAT.INC)
C      CATBLK  I(256) Catalog header record
C   Output:
C      IERR    I     Return code, 0=>OK, otherwise error.
C-----------------------------------------------------------------
```

```
      INTEGER   ISCR1, ISCR2, LUN1, LUN2, BUFSZ1, BUFSZ2, IERR
      REAL      BUFF1(*), BUFF2(*)
C
      INTEGER   FIND1, FIND2, BIND1, BIND2, NFREQ, NSTOKE, NIF, I, IV,
     *   IFQ, IST, IIF, NIOIN, NIOUT, INDEX, JCORO, NILIM, BO, VO
      LOGICAL   T, F
      CHARACTER FILE*48
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
C                                     Listings of the standard
C                                     INCLUDE files are at the end
C                                     of the chapter on tasks.
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DUVH.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DCAT.INC'
      DATA VO, BO /0,1/
C-----------------------------------------------------------------------
C                                     Take absolute value of first
C                                     Stokes' value.
      JCORO = ABS (ICORO)
C                                     Find dimension of freq
C                                     and Stokes axes.
      NFREQ = CATBLK(KINAX+JLOCF)
      NSTOKE = CATBLK(KINAX+JLOCS)
C                                     May not have IF axis
      NIF = 1
      IF (JLOCIF.GT.0) NIF = CATBLK(KINAX+JLOCIF)
C                                      Open and init ISCR1
      CALL ZPHFIL ('SC', SCRVOL(ISCR1), SCRCNO(ISCR1), 1, FILE, IERR)
      CALL ZOPEN (LUN1, FIND1, SCRVOL(ISCR1), FILE, T, F, T, IERR)
C                                      Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'READ', 1
         GO TO 990
         END IF
C                                     Let UVINIT determine #/call
      NIOIN = 0
      CALL UVINIT ('READ', LUN1, FIND1, NVIS, VO, LREC, NIOIN,
     *   BUFSZ1, BUFF1, BO, BIND1, IERR)
C                                      Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'READ', 1
         GO TO 990
         END IF
C                                      Open and init ISCR2
      CALL ZPHFIL ('SC', SCRVOL(ISCR2), SCRCNO(ISCR2), 1, FILE, IERR)
      CALL ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), FILE, T, F, T, IERR)
C                                      Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'WRIT', 2
         GO TO 990
         END IF
      NIOUT = 0
```

```
      CALL UVINIT ('WRIT', LUN2, FIND2, NVIS, VO, LREC, NIOUT,
     *   BUFSZ2, BUFF2, BO, BIND2, IERR)
      NILIM = NIOUT
      NIOUT = 0
C                                      Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'WRIT', 2
         GO TO 990
         END IF
C                                     Loop through data file.
C                                     Read input file
         CALL UVDISK ('READ', LUN1, FIND1, BUFF1, NIOIN, BIND1, IERR)
C                                      Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'READ', 1
            GO TO 990
            END IF
C                                     Check if data all read.
         IF (NIOIN.LE.0) GO TO 120
C                                     Loop through records
         DO 100 IV = 1,NIOIN
C                                     Loop through IF
            DO 90 IIF = 1,NIF
C                                     Loop through Stokes' axis
               DO 80 IST = 1,NSTOKE
C                                     Loop through frequency axis
                  DO 80 IFQ = 1,NFREQ
C                                     Compute pointer in the
C                                     buffer to imag. part
                     INDEX = NRPARM + (IFQ-1) * INCF + (IIF-1) * INCIF
                        + (IST-JCORO) * INCS + 1 + (BIND1 - 1)
C                                     Conjugate visibility
                     BUFF1(INDEX) = - BUFF1(INDEX)
  80                 CONTINUE
  90            CONTINUE
C                                     Copy record to output buffer
            CALL RCOPY (LREC, BUFF1(BIND1), BUFF2(BIND2))
C                                     Update buffer pointers
            BIND1 = BIND1 + LREC
            BIND2 = BIND2 + LREC
            NIOUT = NIOUT + 1
C                                      Write output when necessary
            IF (NIOUT.GE.NILIM) THEN
               CALL UVDISK ('WRIT', LUN2, FIND2, BUFF2, BIND2,
                  NIOUT, IERR)
               NILIM = NIOUT
               NIOUT = 0
C                                      Check for error
               IF (IERR.NE.0) THEN
                  WRITE (MSGTXT,1060) IERR, 'WRIT', 2
                  GO TO 990
                  END IF
               END IF
 100        CONTINUE
```

```
C                                       Loop back for more data
 110     GO TO 60
C                                       Finished, flush buffer.
C                                       No more output records.
 120  NIOUT = -NIOUT
      CALL UVDISK ('FLSH', LUN2, FIND2, BUFF2, BIND2, NIOUT, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'FLSH', 2
         GO TO 990
         END IF
C                                        Close files.
 130  CALL ZCLOSE (LUN1, FIND1, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 1
         GO TO 990
         END IF
      CALL ZCLOSE (LUN2, FIND2, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 2
         GO TO 990
         END IF
      IERR = 0
      GO TO 999
C                                        Error.
 990  CALL MSGWRT (8)
 999  RETURN
C-------------------------------------------------------------------
 1000 FORMAT ('UVCONJ: ERROR',I3,' OPEN FOR ',A4,' FILE',I2)
 1010 FORMAT ('UVCONJ: ERROR',I3,' INIT FOR ',A4,' FILE',I2)
 1060 FORMAT ('UVCONJ: ERROR',I3,1X,A4,'ING FILE',I2)
      END
```

6.4.8 Single Dish Data

AIPS has a limited capacity to handle single dish data; that is measurments of sky brightness at random locations on the sky. The format of single dish data is much like that of interferometer data so that many of the utility routines will work for both. The "COMPLEX" axis still has 3 values (except for the compressed form) which are: the measured sky brightness, any baseline or other offset removed and a weight. In place of the U and V random parameters are sky positions which may be either absolute or in a specified tangent plane projection. The structure of Single dish data is described in the following.

Single dish random parameters

The single dish random parameter types are described in the following:

1. 'RA' and 'DEC': These random parameters are the Right Ascension and Declination of the observation in degrees. If the coordinates have been projected onto the tangent plane then the RA and Declination types become 'RA-xxxx' and 'DEC-xxxx' where xxxx is the projection code. See the chapter on AIPS catalog headers and/or AIPS memoes 27 and 46 for details of the projection codes. These random parameter these are required but the order is arbitrary.

2. 'TIME1': The time tags for the data are kept in days since the reference day.
3. 'BEAM': This random parameter gives the beam number + 256. The beam offset makes the data look more like uv data and more of the the AIPS uv data tasks will work for this data.
4. 'SCAN': This random parameter gives the scan number. This random parameter is optional.
5. 'SAMPLE': This random parameter gives the sample number in the scan. This random parameter is optional.

Single dish regular axis coordinates

The units of the regular axis coordinates are defined by convention; the conventions used by AIPS for the regular axis types are the following:

1. 'COMPLEX': the "complex" axis consists of the measured brightness, subtracted baseline, and (optional) weight. Magic value blanking is supported. This axis is required.
2. 'STOKES': this axis is used to describe which Stokes' parameters are given; the conventions are the same as used for uv data. This axis is required.
3. 'FREQ': the frequency axis coordinates are in Hz. This axis is required.
4. 'IF': The IF axis is a construct which allows irregularly spaced groups of frequency channels. The IF number specifies an entry in the FQ table which must be present if this axis is present. This axis is optional.
5. 'RA' and 'DEC': the celestial coordinates are given in degrees. The values associated with these axes are irrelevant (although they should be present) for unprojected data. For data with projected coordinates the coordinate values of these axes should be the tangent point, i.e. the position on the sky at which the plane onto which the coordinates are projected is tangent to the celestial sphere and these axes should become 'RA---ccc' and 'DEC--ccc' where ccc is the projection code. These axes are required.

Weights and flagging are handled the same as for visibility data. Sort order is the same as for visibility data except that the sort codes for sorting by u and v become:

```
U => ordered by RA
V => ordered by Declination
X => descending ABS (RA)
Y => descending ABS (Declination)
Z => ascending ABS (RA)
M => ascending ABS (Declination)
```

UVPGET and Single Dish Data

The routine UVPGET that interpretes uv data catalog headers also understands single dish data. The values passed in INCLUDE DUVH.INC which differ for single dish and interferometer data are described in the following:

- TYPUVD. This integer has a value of 1 for unprojected and 2 for projected sky coordinates. (0 indicates interferometer data).
- ILOCU. This integer gives the 0-rel offset from the beginning of the record of the longitude like celestial coordinate random parameter.
- ILOCV. This integer gives the 0-rel offset from the beginning of the record of the latitude like celestial coordinate random parameter.
- ILOCB. This integer gives the 0-rel offset from the beginning of the record of the beam number random parameter if present; -1 if absent.

Access to and Calibration of Single Dish Data

Single dish data may be read with UVINIT/UVDISK in the same manner as interferometer data. There is a calibration system for single dish data that parallels the interferometer system. Access to this system is through the routine SDGET which is the single dish analog to UVGET for interferometer data with most of the control parameters being passsed through the commons in DSEL.INC. SDGET can optionally apply calibration information from the CS table and flagging from the FG table. A detailed description of SDGET is given at the end of this chapter. A more complete description of the calibration system is given in Volume 2.

6.4.9 Extension files

Extension files contain a great variety of different types of data, but usually are small compared to the data files. Thus, for extension file I/O, the routines are friendlier, but less efficient. In many cases, the data stored in extension files consist of logical records which contain different data types and are, in fact, data structures.

One type of extension file is the table. This type of file contains a self-describing header and is useful for most types of data which can be forced into a tabular structure. The principal advantage of tables is that generalized table manipulating routines, including writing to, and reading from, FITS files automatically are available.

TABINI and TABIO

The routines TABINI and TABIO do I/O to extension tables. A single call to TABINI will create an extension table if necessary, catalog it, open the file, and initialize the I/O. TABIO then allows random access, with mixed reads and write allowed, to the extension file. TABINI returns a set of pointers which can be used to access data in a record. In practice, another level of specific routines for each table type is useful to access tables. Use of tables in AIPS is dealt with in more detail in another chapter in this manual.

EXTINI and EXTIO

NOTE: TABINI and TABIO are strongly preferred over EXTINI and EXTIO as EXTINI/EXTIO files are not copied to FITS format files.

The routines EXTINI and EXTIO make I/O to extension files much simpler than the image and uv data routines. A single call to EXTINI will create an extension file if necessary, catalog it, open the file, and initialize the I/O. EXTIO then allows random access, with mixed reads and write allowed, to the extension file. EXTIO copies the data into a specified array so that a data structure can be formed by means of a Fortran equivalence, either an explicit EQUIVALENCE statement or through the use of a common.

The structure of the extension file is a header record of 512 bytes, some of which are used by EXTINI and EXTIO for bookkeeping, but many of which are available for use. Following the header record come the fixed length logical records which are physically blocked in 512 byte blocks. A single logical record may use several physical blocks or several logical records may be in a given 512 byte block. Details of the call sequences for EXTINI and EXTIO and a description of the file header record are given at the end of this chapter.

Simple copies of any and/or all EXTINI-EXTIO files of a given type may be copied with a single call to EXTCOP. A description of the call sequence for EXTCOP is given at the end of the chapter on tasks.

6.4.10 Text files

AIPS uses a number of text files, such as the HELP and RUN files; in addition, there is the capability to read and write arbitrary text files. There are several routines which allow access to text files: ZTXOPN, ZTXIO, ZTXCLS, ZTOPEN, ZTREAD, ZTCLOS, and KEYIN.

- ZTXOPN opens a text file for read or write.
- ZTXIO reads/writes a line from/to a text file opened by ZTXOPN.

- ZTXCLS closes a text file opened via ZTXOPN.
- ZTOPEN opens a text file. It is similar to ZOPEN except that it has an additional input argument (MNAME) which gives the name of the desired file or member.
- ZTREAD returns one 80-character line of text from a file opened by ZTOPEN.
- ZTCLOS closes the text file.
- KEYIN is the AIPS version of the Cal Tech VLBI parsing routine. This a very flexible routine for obtaining values from external text files.

ZTREAD has a number of standard places that it can find text files. These include the RUN file area, the HELP file area, and various source code areas. To access files in the "RUN" area, a file name (PNAME) should be constructed with ZPHFIL with type "RU"; other inputs are dummy. ZTOPEN should then be called with LUN=10 and this value of PNAME.

Arbitrary text files can be read or written using ZTXIO which needs ZTXOPN to open a file and ZTXCLS to close it. Details of the call sequences are given at the end of this chapter and an example of their use follows. In this example a text file whose name is in the CHARACTER variable FILNAM is read. This file contains lines of text no longer than 80 characters.

```
      INTEGER   LUN, FIND, IERR
      LOGICAL   F
      CHARACTER FILNAM*48, LINE*80
      PARAMETER (F = .FALSE.)
      PARAMETER (LUN = 10)
      INCLUDE 'INCS:DMSG.INC'
                     .
                     .
C                                     Open file
      CALL ZTXOPN ('READ', LUN, FIND, FILNAM, F, IERR)
C                                     Check error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'OPENING'
         GO TO 990
         END IF
C                                     Loop over file.
C                                     Read next line
 100     CALL ZTXIO ('READ', LUN, FIND, LINE, IERR)
C                                     EOF
         IF (IERR.EQ.2) GO TO 700
C                                     Check error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1000) IERR, 'READING'
            GO TO 990
            END IF
C                                     Process LINE
                     .
                     .
C                                     Loop for next LINE
         GO TO 100
C                                     Close file
 700  CALL ZTXCLS (LUN, FIND, IERR)
C                                     Check error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'CLOSING'
```

```
         GO TO 990
         END IF
      GO TO 999
C                                        Error
 990  CALL MSGWRT (8)
C
 999  RETURN
C-------------------------------------------------------------------------
 1000 FORMAT ('ERROR ',I3,1X,A,' TEXT FILE')
      END
```

In the example above, calls to KEYIN could have replaced the calls to ZTXIO.

The file name passed to ZTXOPN should contain a logical pointing to the directory containing the file. In VMS this may be a complete specification of the directory but in Unix an environment must be set outside of AIPS.

Examples:

```
FILNAM='DISK$RES:[USERNAME]CAL.DAT'     (VMS)
FILNAM='MYAREA:CAL.DAT'                 (Unix)
   where MYAREA is an environment variable set before
   starting AIPS:
   %setenv MYAREA /mnt/username
```

6.5 Bottom Level I/O Routines

The routines described so far in this chapter have been relatively high level routines which have hidden a great deal of bookkeeping. In addition, the image and uv data I/O routines work basically sequentially with some data selection ability. Beneath the higher level routines there are, of course, lower level routines. These routines have a great deal more flexibility than the higher level routines, but usually at a cost of a great deal of bookkeeping.

The basic AIPS I/O routines are intrinsically random access, although a data transfer must start on a disk block boundary. "Map" type files (image and uv data) are read with a pair of routines ZMIO and ZWAIT. Non-map (extension) files are read with ZFIO.

6.5.1 ZMIO and ZWAIT

ZMIO initiates a data transfer to or from one of two possible buffer halves and returns without waiting for the operation to complete. ZWAIT is a timing routine which suspends the task until the specified I/O operation is complete. In this manner, I/O and computation can be overlapped.

The I/O commons (INCLUDE DDCH.INC) contain an array, FTAB, which contains AIPS and host system I/O tables. ZOPEN returns a pointer in FTAB to the area to use for a given file. The first 16 integer words of this area are available for AIPS program use, the remainder of an FTAB entry is used for the host system I/O tables. These 16 words are normally used for bookkeeping information (the first always contains the value of the LUN). Examples of the use of the FTAB are found in MINIT, MDISK, MINSK, MSKIP, UVINIT and UVDISK which use ZMIO and ZWAIT. Descriptions of the way these routines use the FTAB are to be found at the end of this chapter. A description of the call arguments to ZMIO and ZWAIT are also found at the end of this chapter.

6.5.2 ZFIO

Extension file I/O and single buffer non-disk I/O is usually done with the routine ZFIO. For disk files, ZFIO reads a 512 byte block from a specified offset in the file. This block size is independent of the true physical block size on the disks being used. The I/O transfer is complete when ZFIO returns.

Details of the call sequence for ZFIO are found at the end of this chapter. An example of the use of ZFIO may be found in the source code for TABINI and TABIO.

6.6 Text of INCLUDE files

6.6.1 DDCH.INC

```
C                                                               Include DDCH.
C                                       AIPS system parameters
      CHARACTER SYSNAM*20, VERNAM*4, RLSNAM*8, DEVNAM(10)*48,
     *   MONNAM(8)*48, MAPNAM(12)*48, SYSTYP*4, SYSVER*8
      HOLLERITH HBLANK
      DOUBLE PRECISION DBLANK
      REAL   XPRDMM, XTKDMM, TIMEDA(15), TIMESG, TIMEMS, TIMESC, TIMECA,
     *   TIMEBA(4), TIMEAP(3), FBLANK, RFILIT(14)
      INTEGER   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3,
     *   NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2), CSIZPR(2),
     *   NINTRN, KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK(2), CSIZTK(2), DASSGN(8,15), SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR, DEVTAB(50), FTAB(1024)
      COMMON /DCHCHM/ SYSNAM, VERNAM, SYSTYP, SYSVER, RLSNAM,
     *   DEVNAM, MONNAM, MAPNAM
      COMMON /DCHCOM/ DBLANK, XPRDMM, XTKDMM, TIMEDA, TIMESG, TIMEMS,
     *   TIMESC, TIMECA, TIMEBA, TIMEAP, FBLANK, RFILIT, HBLANK,
     *   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3, NTAB3,
     *   NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR, CSIZPR, NINTRN,
     *   KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK, CSIZTK, DASSGN, DEVTAB, SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR
      COMMON /FTABCM/ FTAB
C                                                               End DDCH.
```

6.6.2 DSEL.INC

```
C                                                               Include DSEL.
C                                       Commons for UVGET use
      INTEGER   XCTBSZ, XBTBSZ, XPTBSZ, XSTBSZ, XTTSZ, XBPSZ,
     *   XBPBUF
C                                       XCTBSZ=internal gain table size
      PARAMETER (XCTBSZ=2500)
C                                       XBTBSZ=baseline table size
      PARAMETER (XBTBSZ=3500)
C                                       XPTBSZ=polar. corr. table size
      PARAMETER (XPTBSZ=16384)
C                                       XSTBSZ=Source no. table size
      PARAMETER (XSTBSZ=500)
C                                       XTTSZ=Pol. trans. table size
      PARAMETER (XTTSZ=MAXIF*MAXCHA*2)
C                                       XBPSZ=max. no. BP time entries
      PARAMETER (XBPSZ=50)
C                                       XBPBUF=internal BP I/O buffer
      PARAMETER (XBPBUF=65536)
```

```
C                                       Data selection and control
      INTEGER   ANTENS(50), NANTSL, NSOUWD, SOUWAN(XSTBSZ), SOUWTN(30),
     *   NCALWD, CALWAN(XSTBSZ), CALWTN(30), SUBARR, SMOTYP, CURSOU,
     *   NXKOLS(MAXNXC), NXNUMV(MAXNXC), MVIS, JADR(2,XTTSZ), PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU, KLOCFQ,
     *   SELQUA, SMDIV, SMOOTH(3), KLOCIF, KLOCFY, KLOCWT, KLOCSC,
     *   NDECMP, DECMP(2,MAXIF*4), BCHANS, ECHANS, FRQSEL, FSTRED,
     *   FQKOLS(MAXFQC), FQNUMV(MAXFQC)
      LOGICAL   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL, DOSMTH, ISCMP,
     *   DOXCOR, DOACOR, DOWTCL, DOFQSL
      INTEGER   INXRNO, NINDEX, FSTVIS, LSTVIS, IFQRNO
      REAL      TIMRNG(8),  UVRNG(2), INTPRM(3), UVRA(2), TSTART, TEND,
     *   SELFAC(2,XTTSZ), SMTAB(2500), SUPRAD, SELBAN
      CHARACTER SOURCS(30)*16, CALSOU(30)*16, STOKES*4, INTFN*4,
     *   SELCOD*4
      DOUBLE PRECISION UVFREQ, SELFRQ
C                                       Flag table info
      REAL      TMFLST, FLGTND(MAXFLG)
      INTEGER   IFGRNO
      LOGICAL   DOFLAG, FLGPOL(4,MAXFLG)
      INTEGER   FGVER, NUMFLG, FGKOLS(MAXFGC), FGNUMV(MAXFGC),
     *   KNCOR, KNCF, KNCIF, KNCS,
     *   FLGSOU(MAXFLG), FLGANT(MAXFLG), FLGBAS(MAXFLG), FLGSUB(MAXFLG),
     *   FLGBIF(MAXFLG), FLGEIF(MAXFLG), FLGBCH(MAXFLG), FLGECH(MAXFLG)
C                                       CAL table info
      REAL      GMMOD, CURCAL(XCTBSZ), LCALTM, CALTAB(XCTBSZ,2),
     *   CALTIM(3), RATFAC(MAXIF), DELFAC(MAXIF), DXTIME, DXFREQ,
     *   LAMSQ(MAXCHA, MAXIF), IFRTAB(MAXANT, 2), IFR(MAXANT)
      INTEGER   ICLRNO, NCLINR, MAXCLR, CNTREC(2,3)
      LOGICAL   DOCAL, DOAPPL
      INTEGER   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CIDSOU(2),
     *   CLKOLS(MAXCLC), CLNUMV(MAXCLC), LCLTAB, LCUCAL, ICALP1, ICALP2,
     *   POLOFF(4,2)
C                                       Baseline table info
      REAL      LBLTM, BLTAB(XBTBSZ,2), BLFAC(XBTBSZ), BLTIM(3)
      INTEGER   IBLRNO, NBLINR
      LOGICAL   DOBL
      INTEGER   BLVER, BLKOLS(MAXBLC), BLNUMV(MAXBLC), IBLP1, IBLP2
C                                       Polarization table.
      REAL      POLCAL(2,XPTBSZ), PARAGL(2,MAXANT), PARTIM
      INTEGER   PARSOU
      LOGICAL   DOPOL
C                                       Bandpass table
      DOUBLE PRECISION BPFREQ(MAXIF)
      REAL      PBUFF(XBPBUF), TIMENT(XBPSZ), BPTIM(3), LBPTIM, CHNBND
      CHARACTER BPNAME*48
      INTEGER   IBPRNO, NBPINR, ANTPNT(2), NVISM, NVISS, NVIST
      INTEGER   BPVER, BPKOLS(MAXBPC), BPNUMV(MAXBPC), NANTBP, NPOLBP,
     *   NIFBP, NCHNBP, BCHNBP, DOBAND, ANTENT(XBPSZ,MAXANT),
     *   BPDSK, BPVOL, BPCNO, USEDAN(MAXANT), BPGOT(2),
     *   KSNCF, KSNCIF, KSNCS, MXANUM
C                                       Channel 0 stuff
      INTEGER   FSTVS3, LREC3, LSTVS3, NREAD3, FSTRD3, KLOCW3,
     *   KLOCS3, NDECM3, DECM3(2,MAXIF*4), BIND3, RECNO3, LENBU3
```

```
      LOGICAL   ISCMP3, DOUVIM
C                                       File specification.
      INTEGER   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN,
     *   IXLUN, IBLUN, IPLUN, IQLUN, LUNSBP, BPFIND, CATUV(256),
     *   CATBLK(256)
      REAL      USEQ, UDISK
      CHARACTER UNAME*12, UCLAS*6, UFILE*48
C                                       I/O buffers
      INTEGER   CLBUFF(1024), FGBUFF(512), NXBUFF(512), BLBUFF(512),
     *   BPBUFF(32767), FQBUFF(512)
      REAL      UBUFF(8192)
C                                       Character common
      COMMON /SELCHR/ SOURCS, CALSOU, STOKES, INTFM, SELCOD, UNAME,
     *   UCLAS, UFILE, BPNAME
C                                       Common for UVGET use
C                                       Data selection and control
      COMMON /SELCAL/ UVFREQ, SELFRQ,
     *   USEQ, UDISK, TIMRNG, UVRNG, INTPRM, UVRA, TSTART, TEND, UBUFF,
     *   SELFAC, SMTAB, SUPRAD, SELBAN,
     *   INXRNO, NINDEX, FSTVIS, LSTVIS, IFQRNO,
     *   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL, DOSMTH, ISCMP, DOXCOR,
     *   DOACOR, DOWTCL, DOFQSL,
     *   CLBUFF, FGBUFF, NXBUFF, BLBUFF, BPBUFF, FQBUFF,
     *   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN, IXLUN,
     *   IBLUN, IPLUN, IQLUN, LUNSBP, BPFIND, CATUV, ANTENS, NANTSL,
     *   NSOUWD, SOUWAN, SOUWTN, NCALWD, CALWAN, CALWTN,
     *   SUBARR, SMOTYP, CURSOU, NXKOLS, NXNUMV, FQKOLS, FQNUMV,
     *   NVIS, JADR, PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU,
     *   KLOCFQ, SELQUA, SMDIV, SMOOTH, KLOCIF, KLOCFY, KLOCWT,
     *   KLOCSC, NDECMP, DECMP, BCHANS, ECHANS, FRQSEL, FSTRED
C                                       FLAG table info
      COMMON /CFMINF/ TMFLST, FLGTND, IFGRNO, DOFLAG, FLGPOL,
     *   FGVER, NUMFLG, FGKOLS, FGNUMV, KNCOR, KNCF, KNCIF, KNCS,
     *   FLGSOU, FLGANT, FLGBAS, FLGSUB, FLGBIF, FLGEIF, FLGBCH, FLGECH
C                                       CAL table info
      COMMON /CGNINF/ GMMOD, CURCAL, LCALTM, CALTAB, CALTIM, RATFAC,
     *   DELFAC, DXTIME, DXFREQ,
     *   ICLRNO, NCLINR, MAXCLR, CNTREC,
     *   DOCAL, DOAPPL,
     *   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CIDSOU, CLKOLS, CLNUMV,
     *   LCLTAB, LCUCAL, ICALP1, ICALP2, POLOFF,
     *   LAMSQ, IFRTAB, IFR
C                                       BL table info
      COMMON /CBLINF/ LBLTM, BLTAB, BLTIM, BLFAC,
     *   IBLRNO, NBLINR,
     *   DOBL,
     *   BLVER, BLKOLS, BLNUMV, IBLP1, IBLP2
C                                       Pol. table
      COMMON /CPLINF/ POLCAL, PARAGL, PARTIM, PARSOU, DOPOL
C                                       BP table
      COMMON /CBPINF/ BPFREQ,
     *   PBUFF, TIMENT, BPTIM, LBPTIM, CHNBND,
     *   IBPRNO, NBPINR, ANTPNT, NVISM, NVISS, NVIST,
```

```
     *   BPVER, BPKOLS, BPNUMV, NANTBP, NPOLBP, NIFBP, NCHNBP, BCHHBP,
     *   DOBAND, ANTENT, BPDSK, BPVOL, BPCNO, USEDAN, BPGOT,
     *   KSNCF, KSNCIF, KSNCS, MXANUM
C                                        Channel 0 common
      COMMON /CHNZ/ FSTVS3, LREC3, LSTVS3, NREAD3, FSTRD3, KLOCW3,
     *   KLOCS3, NDECM3, DECM3, BIND3, RECNO3, LENBU3,
     *   ISCMP3, DOUVIN
C
      COMMON /MAPHDR/ CATBLK
C                                                       End DSEL.
```

6.6.3 DUVH.INC

```
C                                                       Include DUVH.
C                                       If you change this include you
C                                       must also change common
C                                       /CATHDR/ in DBCON
C                                       Include for uv header info
      INTEGER   NVIS
      INTEGER   ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, ILOCFQ,
     *   JLOCC, JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC,
     *   NCOR, INCS, INCF, INCIF, ICORO, TYPUVD
      CHARACTER   SOURCE*8, ISORT*2
      DOUBLE PRECISION FREQ, RA, DEC
      COMMON /UVHDR/ FREQ, RA, DEC, NVIS, ILOCU, ILOCV, ILOCW, ILOCT,
     *   ILOCB, ILOCSU, ILOCFQ, JLOCC, JLOCS, JLOCF, JLOCR, JLOCD,
     *   JLOCIF, INCS, INCF, INCIF, ICORO, NRPARM, LREC, NCOR, TYPUVD
      COMMON /UVHCHR/ SOURCE, ISORT
C                                                       End DUVH.
```

6.7 Routines

6.7.1 CALCOP

Routine to copy selected data from one data file to another optionally applying calibration and editing information. The input file should have been opened with UVGET. Both files will be closed on return from CALCOP. Note: UVGET returns the information necessary to catalog the output file. The output file will be compressed if necessary at completion of CALCOP.

```
   CALCOP (DISK, CNOSCR, BUFFER, BUFSZ, IRET)
   Inputs:
      DISK     I       Disk number for catalogd output file.
                       If .LE. 0 then the output file is a /CFILES/
                       scratch file.
      BUFFER   R(*)    Work buffer for writing.
      BUFSZ    I       Size of BUFFER in bytes.
   Input via common:
      LREC     I       (/UVHDR/) length of vis. record in R words.
      NRPARM   I       (/UVHDR/) number of R random parameters.
   In/out:
      CNOSCR   I       Catalog slot number for if cataloged file;
                       /CFILES/ scratch file number if a scratch file,
                       IF DISK=CNOSCR=0 then the scratch is created.
                       On output = Scratch file number if created.
```

```
In/out via common:
   CATBLK   I(256)  Catalog header block from UVGET
                    on output with actual no. records
   NVIS     I       (/UVHDR/) Number of vis. records.
Output:
   IRET     I       Error code: 0 => OK,
                       > 0 => failed, abort process.
Usage notes:
(1) UVGET with OPCODE='INIT' MUST be called before CALCOP to setup
    for calibration, editing and data translation.  If an output
    cataloged file is to be created this should be done after the
    call to UVGET.
(2) Uses AIPS LUN 24
```

6.7.2 CHNDAT

Routine to create/fill/read CH/FQ extension tables. We are phasing out CH tables, so this routine will read them, but will only write FQ tables.

```
CHNDAT (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   NIF, FOFF, ISBAND, FREQID, IERR)
Inputs:
   OPCODE   C*4      Operation code:
                     'WRIT' = create/init for write or read
                     'READ' = open for read only
   BUFFER   I(512)   I/O buffer and related storage, also defines
                     file if open.
   DISK     I        Disk to use.
   CNO      I        Catalog slot number
   CATBLK   I(256)   Catalog header block.
   LUN      I        Logical unit number to use
   FREQID   I        Frequnecy ID #, if FQ tables exists
Input/Output:
   VER      I        CH file version
   NIF      I        Number of IFs.
   FOFF     D(*)     Frequency offset in Hz from ref. freq.
                        True = reference + offset.
   ISBAND   I(*)     Sideband of each IF.
                     -1 => 0 video freq. is high freq. end
                      1 => 0 video freq. is low freq. end
Output:
   IERR     I        Return error code, 0=>OK, else TABINI or TABIO
                     error, -1 => tried to create/write an FQ table
```

6.7.3 COMOFF

Compute the block offset of a 2-D map plane in a NDIM-dimensional map from the beginning of the map.

```
COMOFF (NDIM, MAX, DEPTH, BLKOF, IERR)
Inputs:
   NDIM      I   Number of axes in map
   MAX(7)    I   Number of pixels on each axis
   DEPTH(5 ) I   Depth of required plane along other axes
Outputs:
   BLKOF     I   Block offset
```

```
   IERR      I   Error return 0 = OK, 1= error in NDIM
```

6.7.4 DGGET

Gets requested data from visibility record, reformatting if needed. REQUIRES setup by DGINIT to set values of MVIS, JADR, SELFAC and ALLWT.

```
DGGET (VISIN, IND, MVIS, JADR, SELFAC, ALLWT, VISOUT, DROP)
Inputs:
   VISIN   R(IND,*)   Input visibility array
   IND     I          First dimension of VISIN (CATBLK(KINAX))
   MVIS    I          Number of visibilities in requested output
                      format.
   JADR    I(2,*)     Pointers to the first and second visibility
                      input records to be used in the output record.
                      If JADR(1,n) is negative use IABS (JADR(1,n))
                      and multiply the visibility by i (=SQRT(-1))
   SELFAC  R(2,*)     Factors to be multiplied by the first and
                      second input vis's to make the output vis.
   ALLWT   L          Flag, = .TRUE. if all visibilities must have
                      positive weight.
  Output:
   VISOUT  R(3,*)     Output visibility record
   DROP    L         .TRUE. if all data in record flaged.
```

6.7.5 DGINIT

Sets up tables for selecting data from vis. record. Checks if requested data in data base. Requires catalog header record from include DCAT.INC and setup of commons in INCLUDE DUVH.INC by UVPGET before call.

Note: STOKES='HALF' will work if only partial information (i.e. 1 polarization) is available in the data.

```
DGINIT (STOKES, BCHAN, ECHAN, BIF, EIF, MVIS, JADR, SELFAC, ALLWT,
  *   PMODE, IERR)
Inputs:
   STOKES   C*4       Desired output data format: 'I','V','Q','U',
                      'IQU','IQUV','IV','RR','LL','RL','LR'
                      'HALF' (=parallel pol.), 'FULL' (=RR,LL,RL,LR)
   BCHAN    I         First channel desired.
   ECHAN    I         Last channel desired.
   BIF      I         First IF desired.
   EIF      I         Last IF desired.
Input from common /MAPHDR/
   CATBLK   I(256)    Catalog header record.
Output:
   MVIS     I         Number of visibilities in requested output
                      format.
   JADR     I(2,*)    Pointers to the first and second visibility
                      input records to be used in the output record.
                      If JADR(1,n) is negative use IABS (JADR(1,n))
                      and multiply the visibility by i (=SQRT(-1))
   SELFAC   R(2,*)    Factors to be multiplied by the first and
                      second input vis's to make the output vis.
   ALLWT    L         Flag, = .TRUE. if all visibilities must have
```

```
                    positive weight.
   PMODE     I      Polarization mode:
                      1 = I,    2 = V,   3 = Q
                      4 = U,    5 = IQU, 6 = IQUV
                      7 = IV,   8 = RR,  9 = LL
                     10 = RL,  11 = LR, 12 = parallel (RR,LL)
                     13 = (RR,LL,RL,LR)
   IERR      I      Error flag. 0 => ok, 1 = unrecognized stokes,
                     2 = data unavailable.
```

6.7.6 EXTINI

EXTINI creates/opens an extension file. If a file is created it is cataloged by a call to CATIO which saves the updated CATBLK.

```
   EXTINI (OPCODE, PTYP, VOL, CNO, VER, CATBLK, LUN, IND,
     *   LREC, NREC, BUFFER, IERR)
   Inputs:
      OPCODE   C*4      Operation code, 'READ' => read only,
                                        'WRIT' => read/write
      PTYP     C*2      Physical extension type (eg. 'CC')
      VOL      I        Volume number
      CNO      I        Catalog slot number
      LUN      I        Logical unit number to use.
      NREC     I        Number of logical rec. for create/extend
   In/out:
      VER      I        in: Version number: (<= 0 => write a new one,
                            read the latest one)
                        out: Version number used.
      CATBLK   I(256)   Catalog block of cataloged file, ext info is
                        updated if necessary.
      LREC     I        in: Record length in units of REALs (write new)
                        out: Logical record length (in units of REALs)
                             for read/write old files
      BUFFER   I(*)     Work buffer, at least 1024 bytes in size,
                        more if logical record longer than 512 bytes
                        out: Header info. for EXTIO
   Output:
      IND      I        FTAB pointer.
      IERR     I        Return error code. 0 => OK
                                           1 => bad input.
                                           2 => could not find or open
                                           3 => create/I/O problem.
   Useage notes:
   For sequential access, EXTINI leaves pointers for EXTIO such that
   if IRNO .le. 0 reads will begin at the start of the file and writes
   will begin after the last previous record.
   File should be marked 'WRIT' if the file is to be created.

   Header record:
   Each extension file using this system must have the first physical
   (512 bytes) record containing necessary information.  In addition
   space in this first record not reserved can be used for other
   purposes.  The header record contains the following:
```

```
 I   word(s)                  description
    1          # 512-byte records in the existing file
    2          # logical records to extend the file when req.
    3          max. # of logical records
    4          current number of logical records
    5          # bytes per value
    6          # values per logical record.
    7          # of logical records per physical record, if neg then
               the # of physical records per logical record.
 8 - 10        Creation task name (6 Hollerith characters)
11 - 16        Creation date, time
17 - 28        File name (48 Hollerith characters)
   29          Volume number on which file resides.
30 - 32        Last write-access task (6 Hollerith characters)
33 - 38        Last write-access time,date
39 - 56        reserved. (53-56 used by EXTIO:
                          53 = # I   words per logical record.
                          54 = IOP sent to EXTINI
                          55 = current physical record no.
                               (doesn't include header rec.)
                          56 = current logical rec. no.
57 -256        Available for use.
```

6.7.7 EXTIO

EXTIO does random access I/O to an extension files. Mixed reads and writes are allowed if EXTINI was called with 'WRIT'.

```
EXTIO (OPCODE, LUN, IND, IRNO, RECORD, BUFFER, IERR)
Inputs:
   OPCODE         C*4  Opcode 'READ','WRIT','CLOS'
   LUN            I    Logical unit number
   IND            I    FTAB pointer
   IRNO           I    Logical record no. 0=> next.
   RECORD(*)      I    Array containing record to be written
   BUFFER(*)      I    Work buffer = 512 bytes + enough 512 byte
                       blocks for at least one full logical record.
Output:
   RECORD(*)      I    Array containing record read.
   BUFFER(*)      I    buffer.
   IERR           I    Return error code 0 => OK
                              1 => file not open
                              2 => input error
                              3 => I/O error
                              4 => attempt to read past end of data
                                   or write past log. or phys.
                                   record 32766.
IMPORTANT NOTE:  the contents of BUFFER should not be changed
except by EXTIO between the time EXTINI is called until the file
is closed.  The exception is that the user portion of the header
record is available.
```

6.7.8 GETVIS

GETVIS gets and reformats uv data. Requires setup by SETVIS.

```
GETVIS (MODE, MVIS, JADR, SFACT, ALLWT, DATA, WT,
  *   VIS, IERR)
Inputs:
   MODE          I    Operation number (see SETVIS).
                      When MODE = 2 or 3 and RL and LR are given
                      the U visibility is multiplied by i.
   MVIS          I    Number of visibilities wanted.
   JADR(2,MVIS)  I    Pointers set by SETVIS.
   SFACT(2,MVIS) R    Factors set by SETVIS.
   ALLWT         L    Flag set by SETVIS, if .TRUE. all relevant
                      weights must be positive.
   DATA(3,*)     R    Visibility portion of input data.
Outputs:
   WT            R    Average weight.
   VIS(MVIS)     CMPX Visibilities.
   IERR          I    Error code, 0=>OK,
                              1 => bad weights.(data flagged).
                              2 = bad input.
```

6.7.9 GET1VS

GET1VS gets and reformats uv data. Returns one Stokes' type per frequency channel. Requires setup by SET1VS.

```
GET1VS (MODE, MVIS, JADR, JINC, SFACT, ALLWT, STOKES,
  *   DATA, WT, VIS, IRET)
Inputs:
   MODE          I    Operation number (see SET1VS).
                      When MODE = 3 and RL and LR are given,
                      the U visibility is multiplied by i.
   MVIS          I    Number of visibilities wanted.
   JADR(2)       I    Pointers set by SET1VS.
   JINC          I    Increment between vis.
   SFACT(2)      R    Factors set by SET1VS.
   ALLWT         L    If true all vis are required.
   STOKES        L    True if input data true Stokes'.
                      Used for UPOL only.
   DATA(3,*)     R    Visibility portion of input data.
Outputs:
   WT            R    Average weight.
   VIS(MVIS)     CMPX Visibilities.
   IRET          I    Error code, 0=>OK,
                              1 => bad weights.(data flagged).
                              2 = bad input.
```

6.7.10 KEYIN

Standard Fortran version of the CIT VLBI Keyin subroutines. These subroutines read keyed parameters on cards images. The text file should be opened via a call to ZTXOPN before the first call to KEYIN and closed via a call to ZTXCLS after the last call. (HINT: use LUN = 10) Note: in this version time like entries in the form hh:mm:ss will be returned in hours.

```
KEYIN (KEYS, VALUES, VALCHR, N, ENDMRK, MODE, LUN,
  *   FIND, IERR)
Inputs:
```

```
   KEYS(N)     C*8  Array of parameter names .
                    Array names should have the last characters
                    indicate the element number.  Should all be
                    in upper case characters.
   N           I    number of parameters (dimension of keys, values)
   ENDMRK      C*8  special keyword to indicate end of input
   MODE        I    1 = turn on reflection, 0 = turn off
                    2 = interactive mode (prompts for input,
                        no reflection, no limit on errors)
                    3 = Pass values until ENDMRK, File should not
                        contain keywords.
                    Note: currently only reads from a file.
   LUN         I    LUN to read from (used in call to ZTXOPN)
   FIND        I    FTAB pointer for input. (from ZTXOPN)
Input/Output:
   VALUES(N)   D    array to receive numeric values or defaults,
                    each value corresponds to a KEY.
   VALCHR(N)   C*8  array to receive character values or defaults,
                    each value corresponds to a KEY.
Outputs:
   N           I    (MODE=3 only) number of values found
   IERR        I    error code, 0=>OK, 1=>EOF found, 2=>Error
```

6.7.11 MAPSIZ

MAPSIZ computes the correct number of bytes to request from ZCREAT for a file using map I/O methods.

```
MAPSIZ (NAX, NP, ISIZE)
Inputs:
   NAX     I        # axes
   NP      I(NAX)   Number of points on each axis
Output:
   ISIZE   I        File size in AIPS bytes
```

6.7.12 MAPCLS

closes a cataloged file, updates header on disk, clears catalog status.

```
MAPCLS (OP, IVOL, CNO, LUN, IND, CATBLK, CATUP, WBUFF, IERR)
Inputs:
   OP      C*4      OPcode used by MAPOPN to open this file
   IVOL    I        Disk volume containing map file
   CNO     I        Catalog slot number of file
   LUN     I        Logical unit # used for file
   IND     I        FTAB pointer for LUN
   CATBLK  I(256)   New catalog header which can optionally
                    be written into header if OP=WRIT or INIT
                    Dummy arguement if OP=READ
   CATUP   L        If TRUE, write CATBLK into catalog,
                    ignored if OP = READ
Outputs:
   IERR    I        0 = O.K.
                    1 = CATDIR couldn't access catalog
                    5 = illegal OP code
```

6.7.13 MAPOPN

MAPOPN opens a map file marking the catalog entry for the desired type of operation.

```
MAPOPN (OP, IVOL, NAMEIN, CLASIN, SEQIN, TYPIN, USID,
  *   LUN, IND, CNO, CATBLK, WBUFF, IERR)
Inputs:
   OP       C*4      Operation: READ, WRIT, or INIT where INIT is for
                     known creation processes (it ignores current file
                     status & leaves it unchanged).  Also: HDWR for
                     use when the header is being changed, but the
                     data are to be read only.
   LUN      I        Logical unit # to use
In/out:
   NAMEIN   C*12     Image name (name)
   CLASIN   C*6      Image name (class)
   SEQIN    I        Image name (seq.#)
   USID     I        User identification #
   IVOL     I        Input disk unit
   TYPIN    C*2      Physical type of file
Outputs:
   IND      I        FTAB pointer
   CNO      I        Catalog slot containing map
   CATBLK   I(256)   Buffer containing current catalog block
   WBUFF    I(256)   Working buffer for CATIO and CATDIR
   IERR     I        Error output: 0 = OK
                        2 = Can't open WRIT because file busy
                            or can't READ because file marked WRITE
                        3 = File not found
                        4 = Catalog i/o error
                        5 = Illegal OP code
                        6 = Can't open file
```

6.7.14 MCREAT

Subroutine to create a map file using the parameters in a CATBLK. The file will be cataloged and marked with WRITE status. The image name parameters incl. physical type must be filled in. A blank physical type is converted to 'MA'. The OUTSEQ default is applied (0 = highest matching+1). The name must be unique ignoring the physical type. The extension file areas of the CATBLK are cleared and the "DATE-MAP" string is filled in.

```
MCREAT (IVOL, CNO, WBUFF, IERR)
In/Outs:
   IVOL      I        Volume # on which to put file: 0 => ALL
                      on output has volume used
   WBUFF     I(256)   Working buffer
Outputs:
   CNO       I        Catalog slot number
   IERR      I        Error code; 0 => o.k.
                                  1 => couldnt create,no room
                                  2 => no create, duplicate name
                                  3 => no room in catalog
                                  4 => i/o problem on catalog
                                  5 => Other Create errors
                                  6 => no catalog file
```

```
Common: (in/out)
   CATBLK    I(256)   Catalog block (via common MAPHDR)
   CATB4     R(256)   Catalog block (equivalenced to CATBLK)
```

6.7.15 MDESTR

MDESTR will delete a catalog entry for a file, delete all extension files for that file, and then delete the file. The file must be in the REST state.

```
MDESTR (IVOL, ISLOT, CATBLK, IWBLK, INDEST, IERR)
Inputs:
   IVOL    I       disk volume number of the file.
   ISLOT   I       catalog slot number.
In/out:
   INDEST  I       number of extension files destroyed.
                   (if = -32000 on in, suppress normal msg)
Output:
   CATBLK  I(256)  the header block for this file.
   IWBLK   I(256)  work buffer.
   IERR    I       error code: 0 no error
                        1 = disk error
                        2 = map too busy
                        3 = destroy failed somehow
```

6.7.16 MDISK

MDISK reads or writes a row of an image. MDISK is called only after a call to MINIT and you should read the precursor remarks of MINIT. MDISK actually sets an array index (BIND) to the start of the next line wanted. Actual IO is done only when needed and a row is written not on "its" call to MDISK but on some subsequent call (or the FINI call).

```
MDISK (OP, LUN, FIND, BUFF, BIND, IERR)
Inputs:
   OP     C*4   Op code 'WRIT', 'READ', 'FINI' (flush write buffers)
   LUN    I     logical unit number
   FIND   I     Pointer to FTAB returned by ZOPEN
Input and output:
   BUFF   R(*)  Buffer holding data
Output:
   BIND   I     Pointer to position in buffer of first pixel in
                window in the present line
   IERR   I     Error return: 0 => ok
                      1 => file not open
                      2 => input error
                      3 => I/O error
                      4 => end of file
                      5 => beginning of medium
                      6 => end of medium
```

6.7.17 MINIT

MINIT sets up a special section of FTAB for quick-return, double buffered I/O. N.B. This routine is designed to read/write images one plane at a time.

```
MINIT (OP, LUN, IND, LX, LY, WIN, BUFF, BFSZ, BLKOF, IERR)
Inputs:
```

```
   OP    C*4   Operation code character string: 'READ','WRIT','UPDT'
   LUN   I     logical unit number
   IND   I     pointer to FTAB, returned by ZOPEN
   LX    I     Number of pixels per line in X-direction for whole
               map
   LY    I     Number of lines in whole plane
   WIN   I(4)  Xmin,Ymin,Xmax,Ymax defining desired subrectangle in
               the current plane
   BFSZ  I     Size of total available buffer in AIPS bytes
   BLKOF I     block number, 1 relative, of first map pixel in this
               plane of the image
Outputs:
   IERR  I     Error return: 0 => ok
                             1 -> file not open
                             2 => input error
                             7 => Buffer too small
                             3 => i/o error on initialize
                             4 => end of file
                             5 => beginning of medium
                             6 => end of medium
Usage notes: For map i/o the first 16 words in each FTAB entry
contain a user table to handle double buffer i/o,  the rest
contain system-dependent IO tables.  A "major line" is 1 row or
1 sector if more than 1 line fits in a sector.  FTAB user table
entries, with offsets from the FIND pointer are:
   FTAB + 0 =>  LUN using this entry
          1 =>  No. of major lines transfered per i/o op
          2 =>  No. of major times a buffer has been acessed
          3 =>  No. of major lines remaining on disk
          4 =>  Output index for first pixel in window
          5 =>  No. pixels to increment for next major line
          6 =>  Which buffer to use for i/o; -1 => single buffer
          7 =>  Block offset in file for next operation
          8 =>
          9 =>  Block increment in file for each operation
         10 =>  No. of bytes transferred
         11 =>  I/O op code
         12 =>  sum of any buffer numbers needing to be waited upon
         13 =>  # rows / major line (>= 1)
         14 =>  # times this major line has been accessed
         15 =>  # pixels to increment for next row (= LX)
```

6.7.18 MINSK

MINSK initializes the use of MSKIP to read noncontigious but evenly spaced rows in a map. Read is double buffered if possible; in which case MINSK initiates the first read. Single buffering is used if the desired data cannot be double buffered. If more data is required than will fit in the buffer, multiple (NBUF) equally filled buffers are obtained by NBUF calls to MSKIP.

```
MINSK (LUN, FIND, LROW, NROW, ISTRT, NSKIP, BUFF, BUFSZ, BO, NBUF,
  *   IERR)
Inputs:
   LUN    I     Logical unit number.
   FIND   I     pointer to FTAB returned by ZOPEN.
   LROW   I     Length of a row in pixels.
```

```
   NROW   I     Total number of rows this plane.
   ISTRT  I     First row for read.
   NSKIP  I     Number of rows to skip.
   BUFF   R(*)  Output buffer.
   BUFSZ  I     Buffer size in AIPS bytes.
   BO     I     Block offset
   NBUF   I     factor times which LROW is multiplied normally = 1.
OUTPUT:
  NBUF    I     number of buffer fulls to complete read of row.
                MSKIP must be called this number of times to
                complete the read.
  IERR    I     Error code: 0 = OK
                      1 = file not open
                      2 = input error
                      4 = tried to read past end of map.
                    10+ = 10 + ZMIO or ZWAIT error.
FTAB assigments:
           0 = LUN
           1 =
           2 = BO block offset
           3 =
           4 = length of row / [5] in bytes
           5 = multiplier of [4]
           6 = next record number.
           7 = record increment+1 (total increment)
           8 = # calls per record.
           9 = record call # (when MSKIP is called)
          10 = bytes / call
          11 = buffer flag, -1= single, 1=>current buffer is 1
               2=>current buffer=2 (buffer already read)
          12 = buffer size in pixels (1/2 for double buffering)
          13 = NROW (the number of rows to read)
          14 = BTYOFF the byte offset when double buffering.
```

6.7.19 MSKIP

MSKIP reads rows in a map file which are evenly spaced. The reads are double, single buffered or partial buffers if the row size 1) is $\leq$ BUFSZ/2, 2) between BUFSZ/2 and BUFSZ or 3).GT.BUFSZ. For case 3) multiple calls (NBUF from MINSK) are required to read each row. Each call returns LROW*2/NBUF bytes and I/O is single buffered. IFIN = 0 indicates a row is completed. See MINSK for more details.

```
MSKIP (LUN, FIND, BUFF, BIND, IFIN, IERR)
Input:
   LUN    I      Logical unit number.
   FIND   I      pointer for FTAB
   BUFF   R(*)   Buffer
Output:
   BIND   I      Pointer for BUFF
   IFIN   I      0 if row complete, 1 otherwise.
   IERR   I      error code: 0 = OK
                             1 = file not open
                             2 = attempt to read past end of map.
                            10+= I/O error = 10 +  ZWAIT error.
```

6.7.20 PLNGET

PLNGET reads a selected portion of a selected plane parallel to the front and writes it into a specified scratch file. The output file will be zero padded and a shift of the center may be specified. If the input window is unspecified (0's) and the output file is smaller than the input file, the NX x NY region about position (MX/2+1-OFFX, MY/2+1-OFFY) in the input map will be used where MX,MY is the size of the input map. NOTE: If both XOFF and/or YOFF and a window (JWIN) which does not contain the whole map, XOFF and YOFF will still be used to end-around rotate the region inside the window. The image header is taken from the disk catalog AND explicitly will not handle blanked images.

```
PLNGET (IDISK, ICNO, CORN, JWIN, XOFF, YOFF, NOSCR,
  *   NX, NY, BUFF1, BUFF2, BUFSZ1, BUFSZ2, LUN1, LUN2, IRET)
Inputs:
   IDISK    I       Input image disk number.
   ICNO     I       Input image catalog slot number.
   CORN     I(7)    BLC in input image (1 & 2 ignored)
   JWIN     I(4)    Window in plane.
   XOFF     I       offset in cells in first dimension of the center
                    from MX/2+1 (MX 1st dim. of input win.)
   YOFF     I       offset in cells in second dimension of the center
                    from MY/2+1 (MY 2nd dim. of input win.)
   NOSCR    I       Scratch file number in common /CFILES/ for outpu.
   NX       I       Dimension of output file in X
   NY       I       Dimension of output file in Y
   BUFF1    R(*)    Work buffer
   BUFF2    R(*)    Work buffer.
   BUFSZ1   I       Size in AIPS bytes of BUFF1
   BUFSZ2   I       Size in AIPS bytes of BUFF2
   LUN1     I       Logical unit number for input file
   LUN2     I       Logical unit number to use for output
Output:
   IRET     I       Return error code, 0 => OK,
                     1 = couldn't copy input CATBLK
                     2 = wrong number of bits/pixel in input map.
                     3 = input map has inhibit bits.
                     4 = couldn't open output map file.
                     5 = couldn't init input map.
                     6 = couldn't init output map.
                     7 = read error input map.
                     8 = write error output map.
                     9 = error computing block offset
                     10 = output file too small.
Common:
   DCAT.INC CATBLK  is set to the input file CATBLK.
```

6.7.21 PLNPUT

PLNPUT writes a subregion of a scratch file image into a cataloged image.

```
PLNPUT (IDISK, ICNO, CORN, JWIN, NOSCR, NX, NY, BUFF1,
  *   BUFF2, BUFSZ1, BUFSZ2, LUN1, LUN2, IRET)
Input:
   IDISK    I       Output image disk number.
   ICNO     I       Output image catalog slot number.
   CORN     I(7)    BLC in Output image (1 & 2 ignored)
```

```
   JWIN     I(4)   Window in plane in input image.
   NOSCR    I      Scratch file number in common /CFILES/ for
                   input scratch file.
   NX       I      X-dimension of input file.
   NY       I      Y-dimension of input file.
   BUFF1    R(*)   Work buffer
   BUFF2    R(*)   Work buffer.
   BUFSZ1   I      Size in bytes of BUFF1.
   BUFSZ2   I      Size in bytes of BUFF2
   LUN1     I      Logical unit number to use.
   LUN2     I      Second loical unit number to use.
Output:
   IRET     I      Return error code: 0 => OK
                      1 = couldn't read output CATBLK.
                      2 = Output bits/pixel not allowed.
                      3 = Output and input windows not same.
                      4 = couldn't open input map file.
                      5 = couldn't init output map.
                      6 = couldn't init input map.
                      7 = read error input map.
                      8 = write error output map.
                      9 = error writing header to catalog
                     10 = error computing block offset.
Commons:
   CATBLK in /MAPHDR/ is used as the map header.
          Of particular importance is the data max/min values
          which must apply to the map.  As this is read from the
          catalog it must be updated by a call to CATIO etc.
          before calling this routine.
```

6.7.22 SCREAT

SCREAT creates scratch files. It uses the Common included via the DFIL.INC INCLUDE and returns the scratch file disk and catalog number in variables SCRVOL(NSCR) and SCRCNO(NSCR), where NSCR is updated on successful creation. It attempts to avoid the disk used for the previously created scratch file. All files have physical name SCvccc01 where v is the revision code and ccc is the catalog slot number. Their logical names are determined from the routine BLDSNM.

```
SCREAT (SIZE, WBUFF, IERR)
Input:
   SIZE     I          Desired size in AIPS bytes
Output:
   WBUFF    I(512)     Scratch buffer    (NOTE 512 integers)
   IERR     I          error: 0 => ok
                          1 => catalog error in setting name
                          2 => catalog error on open
                          3 => CATIO error writing header to catlg
                          4 => No allowed disk with room
Commons:
   /MAPHDR/   in       scratch file image header - contents mostly
                       ignored
   /CFILES/   in/out   file info
Note: this common uses IBAD to specify BADDISKs which are avoided.
```

6.7.23 SDGET

Subroutine to obtain data from a single dish data base with optional application of flaging and/or calibration and/or pointing information. Reads data with a large variety of selection criteria and will reformat the data as necessary. Does many of the startup operations, finds Single dish uv like data file etc., reads CATBLK and updates the /UVHDR/ commons (INCLUDE DUVH.INC) to reflect the output rather than input data.

```
SDGET (OPCODE, RPARM, VIS, IERR)
Input:
   OPCODE   C*4       Opcode -
                      'INIT' => Open files Initialize I/O.
                      'READ' => Read next specified record.
                      'CLOS' => Close files.
Inputs via common /SELCAL/  (Include DSEL.INC)
   UNAME    C*12      AIPS name of input file.
   UCLAS    C*6       AIPS class of input file.
   UDISK    R         AIPS disk of input file.
   USEQ     R         AIPS sequence of input file.
   SOURCS   C(30)*16  Names of up to 30 sources, '*' => all
                      First character of name '-' => all except those
                      specified.
   TIMRNG   R(8)      Start day, hour, min, sec, end day, hour,
                      min, sec.   0's => all.
   UVRA     R(2)      Range of RA (1) and dec (2) in degrees about
                      the value in CATBLK at time of READ call to
                      SDGET.  0=>all.
   STOKES   C*4       Stokes types wanted.
                      'I','Q','U','V','R','L','IQU','IQUV'
                      '    '=> Leave data in same form as in input.
   BCHAN    I         First channel number selected, 1 rel. to first
                      channel in data base.    0 => all
   ECHAN    I         Last channel selected.   0 => all
   BIF      I         First IF number selected, 1 rel. to first
                      IF in data base.    0 => all
   EIF      I         Last IF selected.   0 => all
   DOCAL    L         If true apply calibration, else not.
   SUBARR   I         Subarray desired, 0 => all
   FGVER    I         FLAG file version number, if < 0 then
                      NO flagging is applied. 0 => use highest
                      numbered table.
   CLUSE    I         Cal (CS) file version number to apply.
Output:
   RPARM    R(*)      Random parameter array of datum.
   VIS      R(3,*)    Regular portion of data array.
   IERR     I         Error code: 0 => OK,
                         -1 => end of data
                         >0 => failed, abort process.
Output in common /SELCAL/: The default values will be filled in
if null values were specified.
   CATBLK   I(256)    Catalog header block, describes the output
                      data rather than input.
   NPRMIN   I         Number or random parameters in the input data.
   TRANSL   L         If true translate data to requested Stokes'
   CNTREC   I(2,3)    Record counts:
                      (1&2,1) Previously flagged (partly, fully)
```

```
                    (1&2,2) Flagged due to gains (part, full)
                    (1&2,3) Good selected (part, full)
Usage notes:
 1) Include DSEL.INC should be declared in the main
    program or at a level that they will not be overlaid while
    SDGET is in use (ie. between the 'INIT' and 'CLOS' calls)
 2) If no sorting is done SDGET uses AIPS luns 25, 28, 29 and 30
   (1 map, 3 non map files).  If sorting is done (usually possible)
   then 8 map and 3 non map files are used (mostly on OPCODE='INIT')
   and LUNs 16,17,18,19,20,21,22,23,24,25, 28,29,30.
 3) OPCODE = 'INIT' does the following:
   - The cataloged data file is located and the catalog header
     record is read.
   - The index file (if any) is initialized.
   - The flag file (if any) is initialized and sorted if necessary
     (Must be in time order).
   - The CS table (if any) is initialized.
   - I/O to the input file is initialized.
        The following LUNs may be used but will be closed on
     return: 16, 17, 18, 19, 20, 21, 22, 23, 24
        The following LUNs may be used but will be open on
     return: 25 (uv data), 28 (NX table), 29 (CS table),
             30 (FG table).
        NO data are returned from this call.
 4) OPCODE = 'READ' reads one  record properly selected,
    transformed (e.g. I pol.), calibrated and edited as requested
    in the call with OPCODE = 'INIT'
 5) OPCODE = 'CLOS' closes all files used by SDGET which are still
    open.  No data are returned.
 6) If DOCAL is true then the common array CNTREC will contain the
    counts of records which are good or fully or partly flagged
    both previously and due to flagged gain solutions.
```

6.7.24 SELINI

Subroutine to initialize the control values for UVGET in commons in DSEL.INC.

```
      SELINI
  Outputs via common /SELCAL/  (Include DSEL.INC)
     UNAME    C*12       AIPS name of input file. (blank)
     UCLAS    C*6        AIPS class of input file. (blank)
     UDISK    R          AIPS disk of input file. (0.0)
     USEQ     R          AIPS sequence of input file. (0.0)
     SOURCS   C(30)*16   Names of up to 30 sources. (blank)
     SELQUA   I          Qualifier wanted (-1 => all)
     SELCOD   C*4        Cal code ('    ')
     TIMRNG   R(8)       Timerange (0s => all)
     UVRNG    R(2)       Baseline range (0s => all)
     STOKES   C*4        Stokes types wanted. (blank)
     BCHAN    I          First channel number selected, (1)
     ECHAN    I          Last channel selected. (0=>all)
     BIF      I          First IF number selected. (1)
     EIF      I          Last IF selected. (0=>all)
     DOCAL    L          If true apply calibration. (false)
     DOPOL    L          If true then correct polarization (false)
```

```
   DOACOR   L         True if autocorrelations wanted (false)
   DOXCOR   L         True if cross-correlations wanted (true)
   DOWTCL   L         True if weight calibration wanted. (false)
   DOFQSL   L         True if FREQSEL random parm present (false)
   FRQSEL   I         Default FQ table entry to select (-1)
   SELBAN   R         Bandwidth (Hz) to select (-1.0)
   SELFRQ   D         Frequency (Hz) to select (-1.0)
   DOBAND   I         >0 if bandpass calibration. (-1)
   BPNAME   C*48      Name of scratch file set up for BP's.
   DOSMTH   L         True if smoothing requested. (false)
   SMOOTH   R(3)      Smoothing parameters (0.0s)
   DXTIME   R         Integration time (days). (1 sec)
   ANTENS   I(50)     List of antennas selected. (0=>all)
   SUBARR   I         Subarray desired. (0=>all)
   FGVER    I         FLAG file version number. (0)
   CLUSE    I         Cal (CL or SN) file version number (0)
   BLVER    I         BL Table to apply (-1)
   BPVER    I         BP table to apply (-1)
```

6.7.25 SETVIS

SETVIS setup the arrays JADR, SFACT and the flag ALLWT for reformatting uv data as specified by MODE. There is also a check to make sure the desired data is available. Calls to GETVIS will reformat the data. Needs values set by UVPGET and VHDRIN. Only 1 IF will be processed.

```
SETVIS (MODE, NCH, IFNUM, MVIS, JADR, SFACT, ALLWT, IERR)
Inputs:
   MODE     I         Desired output data format:
                       1 => I
                       2 => IQU
                       3 => IQUV
                       4 => IV
                       5 => R (right hand circular)
                       6 => L
                       7 => RL
                       8 => straight correlators (used in UVFND)
                       10+n => n I pol. line maps. (n .le. 8)
                       20+n => n R pol. line maps.
                       30+n => n L pol. line maps.
   NCH      I         First line channel desired.
   IFNUM    I         IF number wanted.
Output:
   MVIS     I         Number of visibilities in requested output
                      format.
   JADR     I(2,*)    Pointers to the first and second visibility
                      input records to be used in the output record.
   SFACT    R(2,*)    Factors to be multiplied by the first and
                      second input vis's to make the output vis.
   ALLWT    L         Flag, = .TRUE. if all visibilities must have
                      positive weight.
   IERR     I         Error flag. 0 =>OK, otherwise data unavailable.
Common (input):
   DCAT.INC  must have uv header
   DUVH.INC  must be initialized by UVPGET
```

6.7.26 SET1VS

SET1VS setup the arrays JADR, SFACT and the flag ALLWT for reformatting uv data as specified by MODE. One visibility per frequency channel will be returned by GET1VS. There is also a check to make sure the desired data is available. Calls to GET1VS will reformat the data. Needs values set by UVPGET.

```
SET1VS (MODE, NCH, JADR, SFACT, ALLWT, JINC, IRET)
Inputs:
   MODE          I    Desired output data format:
                      1 => I
                      2 => Q
                      3 => U
                      4 => V
                      5 => RCP
                      6 => LCP
   NCH           I    First line channel desired.
Output:
   JADR(2)     I    Pointers to the first and second visibility
                      input records to be used in the output record.
   SFACT(2)      R    Factors to be multiplied by the first and
                      second input vis's to make the output vis.
   ALLWT         L    If true no flagged data is allowed.
   JINC          I    Visibility increment.
   IRET          I    Error flag. 0 =>OK, otherwise data unavailable.
```

6.7.27 TABINI

TABINI creates/opens a table extension file. If a file is created, it is cataloged by a call to CATIO which saves the updated CATBLK.

```
TABINI (OPCODE, PTYP, VOL, CNO, VER, CATBLK, LUN, NKEY,
  *   NREC, NCOL, DATP, NBUF, BUFFER, IERR)
Input:
   OPCODE   C*4        Operation code, 'READ' => read only,
                                       'WRIT' => read/write
   PTYP     IC*2       Physical extension type (eg. 'CC')
   VOL      I          Disk volume number
   CNO      I          Catalog slot number
   CATBLK   I(256)     Catalog block of cataloged file.
   LUN      I          Logical unit number to use.
   NREC     I          Number of logical rec. for create/extend
   NBUF     I          Number I   words in BUFFER
In/out:
   VER      I          Version number: (<= 0 => write a new one,
                       read the latest one), returns one used.
   NKEY     I          Maximum number of keyword/value pairs
                       input: used in create, checked on write old
                       (0 => any); output: actual
   NCOL     I          Number of logical columns (does not include
                       selection column).  Input: used in create,
                       checked on write old (0=>any); output: actual
   DATP     I(128,2)   DATP(*,1) address pointers (output only)
                       DATP(*,2) column data type codes. Input:
                       used in create only; output: actual.
   BUFFER   I(*)       Work buffer, at least 1024 bytes in size,
```

```
                    more if logical record longer than 512 bytes
                    Output: control info, lookup table, ...
Output:
   IERR     I       Return error code. 0 => OK
                                      -1 => OK, created new file
                                       1 => bad input.
                                       2 => could not find or open
                                       3 => I/O problem.
                                       4 => create problem.
                                       5 => not a table file
Usage notes:
For sequential access, TABINI leaves pointers for TABIO such that,
if IRNO <= 0, reads will begin at the start of the file and writes
will begin after the last previous record.  Cataloged file should
be marked 'WRIT' if the file is to be created.

Header record:
Each extension file using this system must have the first physical
(512 bytes) record containing necessary information. The full table
file format is described in Going AIPS.  The user must read this
section to understand fully how to use such files.  The header
record contains the following:

 I   word(s)          Description
 1              Number 512-byte records now in file
 2
 3              Max number rows allowed in current file
 4
 5              Number rows (logical records) now in file
 6
 7              Number of bytes/value (2 for TA files)
 8              Number values / logical (# Is / row for TA)
 9              > 0 => number rows / physical record
                < 0 => number physical records / row
10              Number logical columns / row
11 - 16         Creation date: ZDATE(11), ZTIME(14)
17 - 28     H   Physical file name (set on each TABINI call)
29 - 30     H   Creation task name
31
32              Disk number
33 - 38         Last access date: ZDATE(33), ZTIME(36)
39 - 40     H   Last access task name
42              Number logical records to extend file if needed
43              Sort order: logical column # of primary sorting
44              Sort order: logical column # of secondary sorting
                     0 => unknown, < 0 => descending order
45              Disk record number for column data pointers (2)
46              Disk record number for row selection strings (3)
47              Disk record number for 1st record of titles (5)
48              Disk record number for 1st record of units
49              Disk record number for 1st record of keywords
50              Disk record number for 1st record of table data
51              DATPTR (row selection column)
52              Maximum number of keyword/value pairs allowed
```

```
 53              Current number of keyword/value pairs in file
 54 - 56         "*AIPS TABLE*" packed string to verify that table.
 57 - 59
 60              If 1 then then table cannot be written as FITS ASCII
 61              Number of selection strings now in file
 62              Next available R   address for a selection string
 63              First R   address of selection string 1
 64              First R   address of selection string 2
 65              First R   address of selection string 3
 66              First R   address of selection string 4
 67              First R   address of selection string 5
 68              First R   address of selection string 6
 69              First R   address of selection string 7
 70              First R   address of selection string 8
********** for TABIO / TABINI use only **********
 71              IOP : 1 => read, 2 => writ
 72              Number I   words per logical record
 73              Current table row physical record in BUFFER
 74
 75              Current table row logical record in BUFFER
 76
 77              Type of current record in BUFFER
 78              Current control physical record number in BUFFER
 79              Current control logical record number in BUFFER
 80              Type of current control record in BUFFER
 81              LUN
 82              FTAB pointer of open file
***********
 83 -100         Reserved
***********
101 -128     M   Table title
129 -256         lookup table as COLPTR(logical column) = phys column
```

6.7.28 TABIO

TABIO does random access I/O to Tables extension files. Mixed reads and writes are allowed if TABINI was called 'WRIT'. Files opened for WRITe are updated and compressed on CLOS.

```
TABIO (OPCODE, IRCODE, IRNO, RECORD, BUFFER, IERR)
Inputs:
   OPCODE   C*4      Opcode 'READ','CLOS'
                        'WRIT' : write data as selected
                        'FLAG' : write data as de-selected
   IRCODE   I        Type of information
                        0 => Table row
                        1 => DATPTR/DATYPE record
                        2 => data selection string
                        3 => title
                        4 => units
                        5 => keyword/value pair
   IRNO     I        Logical record number. 0 => next (can work
                     with row data and latest IRCODE > 0 only)
                        IRNO is row number (IRCODE = 0)
                        IRNO is ignored (IRCODE = 1)
                        IRNO is string number (IRCODE = 2)
```

```
                        IRNO is column number (IRCODE = 3)
                        IRNO is column number (IRCODE = 4)
                        IRNO is keyword number (IRCODE = 5)
   RECORD   I(*)      Array containing record to be written
   BUFFER   I(*)      Work buffer = 512 bytes + enough 512 byte
                      blocks for at least one full logical record.
                      Must be the same one given TABINI.
Output:
   RECORD   I(*)      Array containing record read.
   BUFFER   I(*)      buffer.
   IERR     I         Return error code 0 => OK
                          -1 => on READ: row read is flagged
                           1 => file not open
                           2 => input error
                           3 => I/O error
                           4 => attempt to read past end of data
                                or write past end of data + 1
                           5 => error on expanding the file
IMPORTANT NOTE:  the contents of BUFFER should not be changed
except by TABIO between the time TABINI is called until the file
is closed.  The exception is that the user portion of the header
record is available.
```

6.7.29 UVCREA

Subroutine to create a uv file using the parameters in a CATBLK. The file will be cataloged and marked with WRITE status. The image name parameters must be filled in except that the physical type is converted to 'UV'. The OUTSEQ default is applied (0 = highest matching+1). The name must be unique ignoring the physical type. The extension file areas of the CATBLK are cleared and the "DATE-MAP" string is filled in.

```
UVCREA (IVOL, CNO, WBUFF, IERR)
In/Outs:
   IVOL       I         Volume # on which to put file.  0 => any
                        on output is volume used (IERR = 0)
Outputs:
   WBUFF      I(256)    Working buffer
   CNO        I         Catalog slot number
   IERR       I         Error code; 0 => o.k.
                                    1 => couldnt create,no room
                                    2 => no create, duplicate name
                                    3 => no room in catalog
                                    4 => i/o problem on catalog
                                    5 => Other Create errors
                                    6 => No catalog file on disk
COMMON: /MAPHDR/ catalog block used a lot, final seq # on output
```

6.7.30 UVDISK

UVDISK reads and writes records of arbitrary length especially UV visibility data. Operation is faster if blocks of data are integral numbers of disk blocks. There are three operations which can be invoked: READ, WRITE and FLUSH (OPcodes 'READ', 'WRIT' and 'FLSH').

'READ' reads the next sequential block of data as specified to UVINIT and returns the number of visibilities in NIO and sets the pointer in BUFFER to the first word of this data.

'WRIT' arranges data in a buffer until it is full. Then as many full blocks as possible are written to the disk with the remainder left for the next disk write. For writes, left-over data is transfered to the beginning

of buffer 1 if that is the next buffer to be filled. The value of NIO in the call is the number of vis. rec. to be added to the buffer and may be fewer than the number specified to UVINIT. On return NIO is the maximum number which may be sent next time. On return BIND is the pointer in BUFFER to begin filling new data.

'FLSH' writes integral numbers of blocks and moves any data left over to the beginning of buffer 1. One exception to this is when NIO $\leq$ 0, in which case the entire remaining data in the buffer is written (if NIO $>$ 0 then ABS (NIO) visibilities are to be written). After the call BIND is the pointer in BUFFER for new data. The principal difference between 'FLSH' and 'WRIT' is that FLSH always forces an I/O transfer. This may cause trouble if a transfer of less than 1 block is requested. A call with a nonpositive value of NIO should be the last call and corresponds to a call to MDISK with opcode 'FINI'.

NOTE: A call to UVINIT is REQUIRED prior to calling UVDISK.

```
UVDISK (OP, LUN, FIND, BUFFER, NIO, BIND, IERR)
Inputs:
   OP       C*4   Opcode 'READ','WRIT','FLSH' are legal
   LUN      I     Logical unit number
   FIND     I     FTAB pointer returned by ZOPEN
   BUFFER   I(*)  Buffer for I/O
   NIO      I     No. additional visibilities to write.
Output:
   NIO      I     No. visibilities read.
                  Max. no. vis. for next write.
   BIND     I     Pointer to start of data in buffer
   IERR     I     Return error code: 0 => OK
                     1 => file not open in FTAB
                     2 => input error
                     3 => I/O error
                     4 => end of file
                     7 => attempt to write more vis than specified
                          to UVINIT or will fit in buffer.
```

6.7.31 UVGET

Subroutine to obtain data from a data base with optional application of flaging and/or calibration information. Reads data with a large variety of selection criteria and will reformat the data as necessary. Does many of the startup operations, finds uv data file etc, reads CATBLK and updates the DUVH.INC commons to reflect the output rather than input data. Most of the input to UVGET is through the commons in DSEL.INC; the initial (default) values of these may be set using routine SELINI.

```
UVGET (OPCODE, RPARM, VIS, IERR)
Input:
   OPCODE   C*4        Opcode:
                       'INIT' => Open files Initialize I/O.
                       'READ' => Read next specified record.
                       'CLOS' => Close files.
Inputs via common (Include DSEL.INC)
   UNAME    C*12       AIPS name of input file.
   UCLAS    C*6        AIPS class of input file.
   UDISK    R          AIPS disk of input file.
   USEQ     R          AIPS sequence of input file.
   SOURCS   C(30)*16   Names of up to 30 sources, *=>all
                       First character of name '-' => all except
                       those specified.
   TIMRNG   R(8)       Start day, hour, min, sec, end day, hour,
                       min, sec. 0's => all
```

```
   UVRNG    R(2)      Minimum and maximum baseline lengths in
                      1000's wavelengths. 0's => all
   STOKES   C*4       Stokes types wanted.
                      'I','Q','U','V','R','L','IQU','IQUV'
                      '    '=> Leave data in same form as in input.
   BCHAN    I         First channel number selected, 1 rel. to first
                      channel in data base. 0 => all
   ECHAN    I         Last channel selected. 0=>all
   BIF      I         First IF number selected, 1 rel. to first
                      IF in data base. 0 => all
   EIF      I         Last IF selected. 0=>all
   DOCAL    L         If true apply calibration, else not.
   DOPOL    L         If true then correct for feed polarization
                      based on antenna file info.
   DOSMTH   L         True if smoothing requested.
   DOACOR   L         True if autocorrelations are requested.
   DOWTCL   L         True if weight calibration wanted.
   DOFQSL   L         True if FREQSEL random parm present (false)
   FRQSEL   I         Default FQ table entry to select (-1)
   SELBAN   R         Bandwidth (Hz) to select (-1.0)
   SELFRQ   D         Frequency (Hz) to select (-1.0)
   DOBAND   I         >0 if bandpass calibration. (-1)
   BPNAME   C*48      Name of scratch file set up for BP's.
   DOSMTH   L         True if smoothing requested. (false)
   SMOOTH   R(3)      Smoothing parameters (0.0s)
   DXTIME   R         Integration time (days). Used when applying
                      delay corrections to correct for delay error.
   ANTENS   I(50)     List of antennas selected, 0=>all,
                      any negative => all except those specified
   SUBARR   I         Subarray desired, 0=>all
   FGVER    I         FLAG file version number, if < 0 then
                      NO flagging is applied. 0 => use highest
                      numbered table.
   CLUSE    I         Cal (CL or SN) file version number to apply.
   BLVER    I         BL Table to apply .le. 0 => none
   BPVER    I         BP table to apply .le. 0 => none
Output:
   RPARM    R(*)      Random parameter array of datum.
   VIS      R(3,*)    Regular portion of visibility data.
   IERR     I         Error code: 0 => OK,
                          -1 => end of data
                          >0 => failed, abort process.
Output in commons in DSEL.INC: The default values will be filled in
if null values were specified.
   UVFREQ   D         Frequency corresponding to u,v,w
   CATBLK   I(256)    Catalog header block, describes the output
                      data rather than input.
   NPRMIN   I         Number or random parameters in the input data.
   TRANSL   L         If true translate data to requested Stokes'
   CNTREC   I(2,3)    Record counts:
                      (1:2,1) Previously flagged (partly, fully)
                      (1:2,2) Flagged due to gains (part, full)
                      (1:2,3) Good selected (part, full)
   ISCMP    L         True if input data is compressed.
```

```
   KLOCSU   I        0-rel random parm. pointer for source in input
                     file.
   KLOCFQ   I        0-rel random parm. pointer for FQ id in input
                     file.
   KLOCIF   I        0-rel random parm. pointer for IF in input
                     file.
   KLOCFY   I        0-rel random parm. pointer for freq. in input
                     file.
   KLOCWT   I        0-rel random parm. pointer for weight in
                     input file.
   KLOCSC   I        0-rel random parm. pointer for scale in
                     input file.
Usage notes:
 1) Include DSEL.INC should be declared in the main program or at a
    level that they will not be overlaid while UVGET is in use (ie.
    between the 'INIT' and 'CLOS' calls). SELINI can be used to
    initialize the control variables in these commons.
 2) If no sorting is done UVGET uses AIPS luns 25, 28, 29 and 30
   (1 map, 3 non map files). If sorting is done (usually possible)
   then 8 map and 3 non map files are used (mostly on OPCODE='INIT')
   and LUNs 16,17,18,19,20,21,22,23,24,25, 28,29,30,40,42,43,44,45.
 3) OPCODE = 'INIT' does the following:
   - The catalgue data file is located and the catalog header
     record is read.
   - The source file (if any) is read.
   - The index file (if any) is initialized.
   - The flag file (if any) is initialized and sorted if necessary
     (Must be in time order).
   - The gain table (if any) is initialized.
   - The bandpass table (if any) is initialized
   - The smoothing convolution table (if any) is initialized
   - I/O to the input file is initialized.
        The following LUNs may be used but will be closed on
     return: 16, 17, 18, 19, 20, 21, 22, 23, 24
        The following LUNs may be used but will be open on
     return: 25 (uv data), 28 (NX table), 29 (CL or SN table),
             30 (FG table), 40 (BL table), 41 (BP table).
        NO data are returned from this call.
 4) OPCODE = 'READ' reads one visibility record properly selected,
    transformed (e.g. I pol.), calibrated and edited as requested
    in the call with OPCODE = 'INIT'
 5) OPCODE = 'CLOS' closes all files used by UVGET which are still
    open. No data are returned.
 6) If DOCAL is true then the common array CNTREC will contain the
    counts of records which are good or fully or partly flagged
    both previously and due to flagged gain solutions.
 7) Only one subarray can be calibrated at a time if DOPOL is true.
    This is because the polarization information for only one
    subarray is kept at a time.
```

6.7.32 UVINIT

UVINIT sets up bookkeeping for the UV data I/O routine UVDISK. I/O for these routines is double buffered (if possible) quick return I/O. UVDISK will run much more efficiently if on disk LREC*NPIO is an integral

number of blocks. Otherwise, partial writes or oversize reads will have to be done. Minimum disk I/O is one block. Smaller calls to UVINIT may be made as long as the buffer is large enough. The buffer size should include an extra NBPS bytes for each buffer for read if NPIO records does not correspond to an integral number of disk sectors (NBPS bytes). 2*NBPS extra bytes required for each buffer for write. NPIO will be adjusted to the maximum allowed for double buffering if the input value is .LE. 0, or the maximum allowed single buffering value if NPIO is too large. If it is positive and useable it is used.

```
UVINIT (OP, LUN, FIND, NVIS, VISOFF, LREC, NPIO, BUFSZ,
  *   BUFFER, BO, BIND, IERR)
Inputs:
   OP       C*4   OP code, 'READ' or 'WRIT' for desired operation.
   LUN      I     Logical unit number of file.
   FIND     I     FTAB pointer for file returned by ZOPEN.
   NVIS     I     Number of visibilities to be transfered.
   VISOFF   I     Offset in vis. rec. of first vis. rec. from BO.
   LREC     I     Number of values in a visibility record.
   NPIO     I     Number of visibilities per call to UVDISK.
                  Determines block size for tape I/O
                  0 => decide (see note above)
   BUFSZ    I     Size in bytes of the buffer.
   BUFFER   R(*)  Buffer
   BO       I     Block offset to begin transfer from (1-relative)
Output:
   NPIO     I     The max. number of visibilities which can be
                  be written or will be read per call.
   BIND     I     Pointer in BUFFER for WRITE operations.
   IERR     I     Return error code:
                  0 => OK
                  1 => file not open in FTAB
                  2 => invalid input parameter.
                  3 => I/O error
                  4 => End of file.
                  7 => buffer too small
Note: VISOFF and BO are additive.
UVINIT sets and UVDISK uses values in the FTAB:
   FTAB(FIND+0) = LUN
             1  = # Bytes per I/O
             2  = # vis. records left to transfer. For double buffer
                  read, 1 more I/O will have been done than shown
             3  =
             4  = Block offset for next I/O.
             5  =
             6  = byte offset of next I/O
             7  =
             8  = Current buffer #, -1 => single buffering
             9  = OPcode 1 = read, 2 = write.
            10  = Values per visibility record.
            11  = # vis. records per UVDISK call
            12  = max. # vis. per buffer.
            13  = # vis. processed in this buffer.
            14  = Buffer pointer for start of current buffer
                  (in values).  Used for WRIT only; includes any
                  data carried over from the last write.
            15  = Buffer pointer for call (values)
```

6.7.33 UVPGET

UVPGET determines pointers and other information from a UV CATBLK. The address relative to the start of a vis record for the real part for a given spectral channel (CHAN) and stokes parameter (ICOR) is given by NRPARM+(CHAN-1)*INCF+ABS(ICOR-ICOR0)*INCS+(IF-1)*INCIF Single dish data, i.e. randomly sampled data in the image plane, is also recognized and ILOCU and ILOCV point to the longitude like and latitude like random parameters. Also a "BEAM" random parameter may be substitued for the "BASELINE" random parameter. The data type present may be determined from the common variable TYPUVD. Two types of single dish data are recognized:

TYPUVD=1 => unprojected RA and Dec and
TYPUVD=2 => projected RA and Dec (ready for GRIDR)

```
UVPGET (IERR)
Inputs: From common /MAPHDR/ (DCAT.INC or DSEL.INC)
   CATBLK   I(256)   Catalog block
   CATH     H(256)   same as CATBLK
   CATR     R(256)   same as CATBLK
   CATD     D(128)   same as CATBLK
Output: In common /UVHDR/ (DUVH.INC)
   SOURCE   C*8      Source name.
   ILOCU    I        Offset from beginning of vis record of U
                     or longitude for single dish format data.
   ILOCV    I        Offset from beginning of vis record of V
                     or longitude for single dish format data.
   ILOCW    I        Offset from beginning of vis record of W.
   ILOCT    I                    "                      Time
   ILOCB    I                    "                   Baseline
                                                     (or beam)
   ILOCSU   I                    "                  Source id.
   ILOCFQ   I                    "                  Freq id.
   JLOCC    I        0-rel. order in data of complex values
   JLOCS    I        Order in data of Stokes' parameters.
   JLOCF    I        Order in data of Frequency.
   JLOCR    I        Order in data of RA
   JLOCD    I        Order in data of dec.
   JLOCIF   I        Order in data of IF.
   INCS     I        Increment in data for stokes (see above)
   INCF     I        Increment in data for freq. (see above)
   INCIF    I        Increment in data for IF.
   ICOR0    I        Stokes value of first value.
   NRPARM   I        Number of random parameters
   LREC     I        Length in values of a vis record.
   NVIS     I        Number of visibilities
   FREQ     D        Frequency (Hz)
   RA       D        Right ascension (1950) deg.
   DEC      D        Declination (1950) deg.
   NCOR     I        Number of correlators (Stokes' parm.)
   ISORT    C*2      Sort order 1st 2 char meaningful.
   TYPUVD   I        UV data type, 0=interferometer,
                        1=single dish unprojected,
                        2=single dish projected RA and Dec.
   IERR     I        Return error code: 0=>OK,
                        1, 2, 5, 7 : not all normal rand parms
                        2, 3, 6, 7 : not all normal axes
                        4, 5, 6, 7 : wrong bytes/value
```

6.7.34 ZCLOSE

Close the file associated with LUN removing any exclusive use state and clear the FTAB entry for the LUN.

```
ZCLOSE (LUN, FIND, IERR)
Inputs:
   LUN      I          Logical unit number
   FIND     I          Index in FTAB to file control block for LUN
Output:
   IERR     I          Error return code: 0 => no error
                          1 => close error
                          2 => file already closed in FTAB
                          3 => both errors
                          4 => erroneous LUN
```

6.7.35 ZCMPRS

ZCMPRS releases unused disk space from the end of an open disk file. AIPS "Byte" is defined as 1/2 of a integer.

```
ZCMPRS (IVOL, PNAME, LUN, LSIZE, SCRTCH, IERR)
Inputs:
   IVOL    I        volume number
   PNAME   C*48     physical file name
   LUN     I        logical unit number under which file is open.
In/Out:
   LSIZE   I        (In) desired final size in AIPS bytes
                    (Out) actual final size in AIPS bytes
Outputs:
   SCRTCH  I(256)   scratch buffer (not used under UNIX).
   IERR    I        error code: 0 => ok
                                1 => input data error
                                2 => compress error
```

6.7.36 ZCREAT

Create a disk file of a specified name and size reserving the disk space.

```
ZCREAT (IVOL, PNAME, RSIZE, MAP, ASIZE, SCRTCH, IERR)
Inputs:
   IVOL     I        Disk volume containing file
   PNAME    C*48     Physical file name
   RSIZE    I        Requested size of the file in AIPS-bytes (1/2
                     of a local integer)
   MAP      L        Is this a "map" file?
Output:
   ASIZE    I        Actual size of file in AIPS-bytes
   SCRTCH   I(256)   Scratch buffer
   IERR     I        Error return code: 0 => no error
                        1 => file already exists
                        2 => volume not found
                        3 => insufficient space
                        4 => other
                        5 => forbidden (reserved)
```

6.7.37 ZDESTR

Destroy (i.e., delete) a file. The file should already be closed.

```
ZDESTR (IVOL, PNAME, IERR)
Inputs:
   IVOL    I       Disk volume containing file, 1,2,3,...
   PNAME   C*48    Physical file name (left justified)
Output:
   IERR    I       Error return code: 0 => no error
                      1 => file not found (no message)
                      2 => device not found
                      3 => file in use
                      4 => other
```

6.7.38 ZEXPND

Increase the size of a disk file — it must be open.

```
ZEXPND (LUN, IVOL, PNAME, NREC, IERR)
Inputs:
   LUN     I       LUN of file open file
   IVOL    I       Disk volume containing file, 1,2,3,...
   PNAME   C*48    Physical file name
In/Out:
   NREC    I       # 256-integer records requested/received
Output:
   IERR    I       Error return code: 0 => no error
                      1 => input error
                      2 => expansion error
                      3 => ZEXIST error
```

6.7.39 ZFIO

Transfer one logical record between an I/O buffer and device LUN. For disk devices, the record length is always 256 local small integers and NREC is the random access record number. For non-disk devices, NREC is the number of 8-bit bytes.

```
ZFIO (OPER, LUN, FIND, NREC, BUFF, IERR)
Inputs:
   OPER    C*4       Operation code 'READ' or 'WRIT'
   LUN     I         Logical unit number
   FIND    I         Index in FTAB to file control block for LUN
   NREC    I         Random access record number (1-relative) for
                     disk transfers or number of 8-bit bytes for
                     sequential device transfers (e.g., Tektronix
                     terminals)
   BUFF    I(256)    I/O buffer
Output:
   IERR    I         Error return code: 0 => no error
                        1 => file not open
                        2 => input error
                        3 => I/O error
                        4 => end of file
```

6.7.40 ZMIO

Low level random access, large block, double buffered device I/O.

```
ZMIO (OPER, LUN, FIND, BLKNO, NBYTES, BUFF, IBUFF,
  *   IERR)
Inputs:
   OPER     C*4   Operation code 'READ' or 'WRIT'
   LUN      I     Logical unit number
   FIND     I     Index in FTAB to file control bIock for LUN
   BLKNO    I     Beginning virtual block number (1-relative).
                  Block size is given by NBPS in /DCHCOM/.
   NBYTES   I     Number of AIPS-bytes to transfer (an AIPS-byte is
                  1/2 a local integer).
   IBUFF    I     Buffer number to use (1 or 2)
In/out:
   BUFF     I(*)  I/O buffer
Output:
   IERR     I     Error return code: 0 => no error
                     1 => file not open
                     2 => input error
                     3 => I/O error
                     4 => end of file
```

6.7.41 ZOPEN

Open a binary disk file, line printer or tty. Message files, text files, tape devices, Tektronix devices and TV devices are NOT opened using this routine (see ZMSGOP for message files, ZTOPEN for text files, ZTPOPN for tape devices, ZTKOPN for Tektronix devices and the device specific routine for TV devices, e.g., ZM70OP).

```
ZOPEN (LUN, FIND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)
Inputs:
   LUN      I     Logical unit number
   IVOL     I     Disk volume containing file, 1,2,3,...
   PNAME    C*48  Physical file name (from ZPHFIL)
   MAP      L     Is this a "map" file?
   EXCL     L     Exclusive use requested?
   WAIT     L     Wait for exclusive use?
Output:
   FIND     I     Index in FTAB to file control block for LUN
   IERR     I     Error return code: 0 => no error
                     1 => LUN already in use
                     2 => file not found
                     3 => volume/logical not found
                     4 => exclusive use denied
                     5 => no room for LUN in FTAB
                     6 => other open errors
```

6.7.42 ZPHFIL

Construct a physical file name in PNAM from TYPE, IVOL, NSEQ, and IVER - either for public data files or user-specific files.

```
ZPHFIL (TYPE, IVOL, NSEQ, IVER, PNAM, IERR)
Inputs:
```

```
   TYPE    C*2   Type of file: e.g. 'MA' for map file
   IVOL    I     Number of the disk volume to be used (1-15)
   NSEQ    I     Sequence number (000-4095)
   IVER    I     Version number (00-255)
Outputs:
   PNAM    C*48  physical file name, left justified
   IERR    I     Error return code: 0 = good return.   1 = error.

Example: If TYPE='MA', IVOL=7, AIPSVER=C, NSEQ=321, IVER=99,
         NLUSER=762 then
            PNAME='DA07:MAC14163;1'      for public data or
            PNAME='DA07:MAC14163.2FA;1'  for private data
         where 321 = 141 base 16, 99 = 63 base 16, 762 = 2FA base 16

TYPE = 'MT' leads to special name for tapes
TYPE = 'TK' leads to special name for TEK4012 plotter CRT
TYPE = 'TV' leads to special name for TV device
TYPE = 'ME' leads to special logical for POPS memory files
```

6.7.43 ZTCLOS

Close the text file and clear the FTAB entry associated with LUN.

```
ZTCLOS (LUN, FIND, IERR)
Inputs:
   LUN      I   Logical unit number
   FIND     I   Index in FTAB for LUN
Output:
   IERR     I   Error return code: 0 => no error
                   1 => close error
                   2 => file already closed in FTAB
                   3 => both errors
                   4 => erroneous LUN
```

6.7.44 ZTOPEN

Open a text file - logical area, version, member name as arguments

```
ZTOPEN (LUN, FIND, IVOL, PNAME, MNAME, VERSON, WAIT, IERR)
Inputs:
   LUN      I      Logical unit number
   IVOL     I      Disk volume containing file, (not used)
   PNAME    C*48   Physical file name, only used to determine file
                   type or logical area
   MNAME    C*8    Text file name
   VERSON   C*48   Logical name for directory or version of
                   directory to search (for file-specific
                   directories)
   WAIT     L      T => wait until file is available (not used)
Outputs:
   FIND     I      Index in FTAB for LUN
   IERR     I      Error return code: 0 => no error
                      1 => LUN already in use
                      2 => file not found
```

```
                    3 => volume not found
                    4 => file locked
                    5 => no room for LUN in FTAB
                    6 => other open errors
```

6.7.45 ZTREAD

Read the next sequential 80-character card image from a text file.

```
ZTREAD (LUN, FIND, RBUFF, IERR)
Inputs:
   LUN     I      Logical unit number
   FIND    I      Index in FTAB for LUN
Output:
   RBUFF   C*80   I/O buffer for card image
   IERR    I      Error return code: 0 => no error
                     1 => file not open
                     2 => end of file
                     4 => other I/O error
```

6.7.46 ZUVPAK

Routine to pack uv data with magic value blanking. One AIPS logical uv data record is processed at a time.

```
ZUVPAK (NCORR, VISIN, WTSCL, VISOUT)
Inputs:
   NCORR  I       Number of correlator values in data
   VISIN  R(3,*)  Unpacked uv data as real, imag and weight per
                  correlator.
Output:
   WTSCL  R(2)    "Weight" and "scale" random parameters for the
                  packed record.
   VISOUT R(*)    Packed visibility data with local magic value
                  blanking.
```

6.7.47 ZUVXPN

Routine to expand packed uv data to unpacked form. One AIPS logical uv data record is processed at a time.

```
ZUVXPN (NCORR, VISIN, WTSCL, VISOUT)
 Inputs:
  NCORR  I        Number of correlator values in data
  VISIN  R(*)     Packed visibility data with local magic value
                  blanking.
  WTSCL  R(*)    "Weight" and "scale" random parameters for the
                  packed record.
 Output:
  VISOUT  R(3,*)  Unpacked uv data as real, imag and weight per
                  correlator.
```

6.7.48 ZTXCLS

Close the text file and clear the FTAB entry associated with LUN.

```
ZTXCLS (LUN, FIND, IERR)
Inputs:
   LUN    I   Logical unit number
   FIND   I   Index in FTAB for LUN
Output:
   IERR   I   Error return code: 0 => no error
                 1 => close error
                 2 => file already closed in FTAB
                 3 => both errors
                 4 => inputs error
```

6.7.49 ZTXIO

Read/write the next sequential line from/to a text file.

```
ZTXIO (OPER, LUN, FIND, LINE, IERR)
Inputs:
   OPER   C*4     Operation code ('READ' or 'WRIT')
   LUN    I       Logical unit number
   FIND   I       Index in FTAB for LUN
Input/output:
   LINE   C*(*)   Line of text.  For WRIT, ZTXIO writes the full
                  string including any trailing blanks.  Use ITRIM
                  and substring notation in the call if you desire
                  only up to the last non-blank (which is usually
                  preferable!).  On READ, adequate size must be
                  declared in calling routine.
Output:
   IERR   I       Error return code: 0 => no error
                     1 => file not open
                     2 => end of file
                     3 => input error
                     4 => other I/O error
```

6.7.50 ZTXOPN

Open a text file.

```
ZTXOPN (OPCODE, LUN, FIND, OUTFIL, APPEND, IERR)
Inputs:
   OPCODE   C*4    Open for 'READ' or 'WRIT'
   LUN      I      Logical unit number
   OUTFIL   C*48   Physical file name
   APPEND   L      If true append new text to end of old file.
                      (OPCODE='WRIT' only).
Outputs:
   FIND     I      Index in FTAB for LUN
   IERR     I      Error return code: 0 => no error
                      1 => error in inputs
                      2 => LUN already in use
                      3 => no room for LUN in FTAB
                      4 => trouble translating logical
                      5 => file already exists
                      6 => open error
```

6.7.51 ZUVPAK

Routine to pack uv data with magic value blanking. One AIPS logical uv data record is processed at a time.

```
ZUVPAK (NCORR, VISIN, WTSCL, VISOUT)
Inputs:
   NCORR  I       Number of correlator values in data
   VISIN  R(3,*)  Unpacked uv data as real, imag and weight per
                  correlator.
Output:
   WTSCL  R(2)    "Weight" and "scale" random parameters for the
                  packed record.
   VISOUT R(*)    Packed visibility data with local magic value
                  blanking.
```

6.7.52 ZUVXPN

Routine to expand packed uv data to unpacked form. One AIPS logical uv data record is processed at a time.

```
ZUVXPN (NCORR, VISIN, WTSCL, VISOUT)
Inputs:
   NCORR  I        Number of correlator values in data
   VISIN  R(*)     Packed visibility data with local magic value
                   blanking.
   WTSCL  R(*)     "Weight" and "scale" random parameters for the
                   packed record.
Output:
   VISOUT  R(3,*)  Unpacked uv data as real, imag and weight per
                   correlator.
```

6.7.53 ZWAIT

Wait until an asynchronous I/O operation completes.

```
ZWAIT (LUN, FIND, IBUFF, IERR)
Inputs:
   LUN     I   Logical unit number
   FIND    I   Index in FTAB to file control block for LUN
   IBUFF   I   Buffer # to wait for (1 or 2)
Output:
   IERR    I   Error return code: 0 => no error
                  1 => LUN not open in FTAB
                  2 => error in inputs
                  3 => I/O error
                  4 => end of file
                  7 => wait service error
```

Chapter 7

High Level Utility Routines

7.1 Overview

There are a number of high level AIPS utility routines which merit special attention. Many of these routines do complex, but common, operations on data or image files, such as gridding uv data or doing 2-D FFTs. Since many of the routines do a great deal of computation, most use the array processor.

Many of these routines make heavy use of commons or the values in catalog header records for control and internal communication. A number of these routines will create scratch and/or output files if necessary. Several general and somewhat overlapping categories of routines are discussed below.

7.2 File Specification

The routines described in this chapter use several methods to specify the input, output, and scratch files. For cataloged files the file is usually specified by a disk number and a catalog slot number. For scratch files an index in arrays SCRVOL and SCRCNO in the common from include DFIL.INC is passed. The indicated values from SCRVOL and SCRCNO are the disk and catalog slot numbers of the scratch files. These values are filled in by SCREAT when the files are created.

A common convention for the routines described in this chapter is that a disk and "catalog slot" number are passed as call arguments and if the disk number is zero and the "catalog slot" number is positive then the file is a scratch file and the "catalog slot" number is the index in SCRVOL and SCRCNO. Several of the routines in this chapter also allow optional creation of output and/or scratch files.

7.3 Data Calibration and Reformatting Routines

The variety of different uv data formats, especially different polarization types, allowed in AIPS uv data bases complicates handling of uv data. In addition, uncalibrated multi-source uv data files need to have calibration, editing and selection criteria applied. A pair of routines allows simplified read access to either single- or multi-source uv data files. A short description is given here and the details of the subroutine calls are given at the end of this chapter. These routines do not use the array processor.

- UVGET sets up, selects, reformats, calibrates, edits either single- or multi-source data files.
- CALCOP. After set up by UVGET, CALCOP can be used to process the entire selected contents of a file to another file.

7.4 Operations on Images

These operations are those performed on entire image files. A short description is given here and the details of the subroutine calls and interface COMMONs are given at the end of this chapter.

- DSKFFT is a disk-based, two dimensional FFT.
- GRDCOR normalizes and corrects an image for the gridding convolution used to grid the image. Used in conjunction with UVGRID and DSKFFT.
- APCONV convolves two images.

7.5 UV Model Calculations

A system of routines is available to compute the Fourier transform of a model, given as either CLEAN or Gaussian components or an image, at the u,v and w locations of the data in a uv data file and to either subtract the model values from the observed values or divide the model values into the observed values. These routines make heavy use of COMMONs. A short description is given here and the details of the subroutine calls and interface COMMONs are given at the end of this chapter.

- UVMDIV divides model visibilities derived from CLEAN or Gaussian components or images into a uv data set.
- UVMSUB subtracts model visibilities derived from CLEAN or Gaussian components or images from a uv data set.

7.6 Image Formation

Routine MAKMAP makes an image or a dirty beam given a uv data set. The data may be either calibrated or uncalibrated (raw) data and calibration and various selection criteria may be (optionally) applied. MAKMAP makes heavy use of COMMONs and the array processor. The details of the call sequence and interface COMMONs are given at the end of this chapter.

7.7 INCLUDEs

There are several types of INCLUDE file which are distinguished by the first character of their name. Different INCLUDE file types contain different types of Fortran declaration statements as described in the following list.

- Pxxx.INC. These INCLUDE files contain declarations for parameters and the PARAMETER statements.
- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations, COMMON and EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statements.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

7.7.1 PUVD.INC

```
C                                                       Include PUVD
C                                     Parameters for uv data
      INTEGER   MAXANT,  MXBASE,  MAXIF, MAXFLG, MAXFLD, MAXCHA
C                                     MAXANT = Max. no. antennas.
      PARAMETER (MAXANT=45)
C                                     MXBASE = max. no. baselines
      PARAMETER (MXBASE= ((MAXANT*(MAXANT+1))/2))
C                                     MAXIF=max. no. IFs.
```

```
      PARAMETER (MAXIF=15)
C                                       MAXFLG= max. no. flags active
      PARAMETER (MAXFLG=1000)
C                                       MAXFLD=max. no fields
      PARAMETER (MAXFLD=16)
C                                       MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                       Parameters for tables
      INTEGER MAXCLC, MAXSNC, MAXANC, MAXFGC, MAXNXC, MAXSUC,
     *   MAXBPC, MAXBLC, MAXFQC
C                                       MAXCLC=max no. cols in CL table
      PARAMETER (MAXCLC=41)
C                                       MAXSNC=max no. cols in SN table
      PARAMETER (MAXSNC=20)
C                                       MAXANC=max no. cols in AN table
      PARAMETER (MAXANC=12)
C                                       MAXFGC=max no. cols in FG table
      PARAMETER (MAXFGC=8)
C                                       MAXNXC=max no. cols in NX table
      PARAMETER (MAXNXC=7)
C                                       MAXSUC=max no. cols in SU table
      PARAMETER (MAXSUC=21)
C                                       MAXBPC=max no. cols in BP table
      PARAMETER (MAXBPC=14)
C                                       MAXBLC=max no. cols in BL table
      PARAMETER (MAXBLC=14)
C                                       MAXFQC=max no. cols in FQ table
      PARAMETER (MAXFQC=5)
C                                                       End PUVD.
```

7.7.2 DFIL.INC

```
C                                                       Include DFIL.
C                                       AIPS system catalog and scratch
      INTEGER   NSCR, SCRVOL(128), SCRCNO(128), IBAD(10), LUNS(10),
     *   NCFILE, FVOL(128), FCNO(128), FRW(128), CCNO
      LOGICAL   RQUICK
      COMMON /CFILES/ RQUICK, NSCR, SCRVOL, SCRCNO, NCFILE, FVOL, FCNO,
     *   FRW, CCNO, IBAD, LUNS
C                                                       End DFIL.
```

7.7.3 DGDS.INC

```
C                                                       Include DGDS.
C                                       include for uv modeling
      INTEGER   SCRBLK(256), KLNBLK(256), MFIELD, FLDSZ(2,MAXFLD),
     *   CCDISK(MAXFLD), CCCNO(MAXFLD), CCVER(MAXFLD), CNOBEM,
     *   BEMVOL, KSTOK, SCTYPE, VOFF, NSTOK, NCHANG
      LOGICAL   DOFFT, NONEG, DOPTMD, NGRDAT
      INTEGER   NSUBG(MAXFLD), NCLNG(MAXFLD)
      REAL      CELLSG(2), FLUXG(MAXFLD), TFLUXG, SSROT, CCROT,
```

```
     *   XPOFF(MAXFLD), YPOFF(MAXFLD), SCLUG(MAXFLD), SCLVG(MAXFLD),
     *   SCLWG(MAXFLD), SCLUM, SCLVM, FACGRD, DXCG(MAXFLD),
     *   DYCG(MAXFLD), DZCG(MAXFLD), OSFX, OSFY,
     *   PTFLX, PTRAOF, PTDCOF, PARMOD(6)
      DOUBLE PRECISION FREQG(MAXCHA)
      COMMON /MAPDES/  FREQG, SCRBLK, KLNBLK,
     *   CELLSG, SCLUG, SCLVG, SCLWG, SCLUM, SCLVM,
     *   DXCG, DYCG, DZCG, FLUXG, TFLUXG, XPOFF, YPOFF, SSROT, CCROT,
     *   FACGRD, OSFX, OSFY, PTFLX, PTRAOF, PTDCOF, PARMOD,
     *   NCLNG, NSUBG, DOFFT, NONEG, DOPTMD, NGRDAT,
     *   MFIELD, FLDSZ, CCDISK, CCCNO, CCVER, CNOBEM, BEMVOL,
     *   KSTOK, SCTYPE, VOFF, NSTOK, NCHANG
C                                                          End DGDS.
```

7.7.4 DMPR.INC

```
C                                                          Include DMPR.
C                                       include for gridding
C                                       and correction routines.
C                                       NOTE: uses PARAMETER in DGDS.INC
      INTEGER   NXBEM, NYBEM, NXUNF, NYUNF, NXMAX, NYMAX,
     *   ICNTRX(MAXFLD), ICNTRY(MAXFLD), CTYPX, CTYPY, NUVCH, CHUV1,
     *   NCHAVG, UNFBOX, TVFLD, BORES(MAXFLD), BOBEM, MDISKN, MSEQ
      LOGICAL   DOZERO, DOTAPE, DOUNIF
      REAL      XFLD(MAXFLD), YFLD(MAXFLD), XPARM(10), YPARM(10),
     *   TAPERU, TAPERV, ZEROSP(5), BMMAX, BMMIN,
     *   FLDMAX(MAXFLD), FLDMIN(MAXFLD), BEMMAX,
     *   XSHIFT(MAXFLD), YSHIFT(MAXFLD), BLMAX, BLMIN
      CHARACTER MNAME*12, MCLASS*6
      DOUBLE PRECISION FREQUV
      COMMON /GRDCOM/ FREQUV,
     *   XFLD, YFLD, XPARM, YPARM, TAPERU, TAPERV, ZEROSP,
     *   BMMAX, BMMIN, FLDMAX, FLDMIN,
     *   BEMMAX, XSHIFT, YSHIFT, BLMAX, BLMIN,
     *   DOZERO, DOTAPE, DOUNIF,
     *   NXBEM, NYBEM, NXUNF, NYUNF, NXMAX, NYMAX, ICNTRX, ICNTRY,
     *   CTYPX, CTYPY, NUVCH, CHUV1, NCHAVG, UNFBOX,
     *   TVFLD, BORES, BOBEM, MDISKN, MSEQ
      COMMON /GRDCHR/ MNAME, MCLASS
C                                                          End DMPR.
```

7.7.5 DSEL.INC

```
C                                                          Include DSEL.
C                                       Commons for UVGET use
      INTEGER   XCTBSZ, XBTBSZ, XPTBSZ, XSTBSZ, XTTSZ, XBPSZ,
     *   XBPBUF
C                                       XCTBSZ=internal gain table size
      PARAMETER (XCTBSZ=2500)
C                                       XBTBSZ=baseline table size
```

```
      PARAMETER (XBTBSZ=3500)
C                                     XPTBSZ=polar. corr. table size
      PARAMETER (XPTBSZ=16384)
C                                     XSTBSZ=Source no. table size
      PARAMETER (XSTBSZ=500)
C                                     XTTSZ=Pol. trans. table size
      PARAMETER (XTTSZ=MAXIF*MAXCHA*2)
C                                     XBPSZ=max. no. BP time entries
      PARAMETER (XBPSZ=50)
C                                     XBPBUF=internal BP I/O buffer
      PARAMETER (XBPBUF=65536)
C                                     Data selection and control
      INTEGER   ANTENS(50), NANTSL, NSOUWD, SOUWAN(XSTBSZ), SOUWTN(30),
     *   NCALWD, CALWAN(XSTBSZ), CALWTN(30), SUBARR, SMOTYP, CURSOU,
     *   NXKOLS(MAXNXC), NXNUMV(MAXNXC), MVIS, JADR(2,XTTSZ), PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU, KLOCFQ,
     *   SELQUA, SMDIV, SMOOTH(3), KLOCIF, KLOCFY, KLOCWT, KLOCSC,
     *   NDECMP, DECMP(2,MAXIF*4), BCHANS, ECHANS, FRQSEL, FSTRED,
     *   FQKOLS(MAXFQC), FQNUMV(MAXFQC)
      LOGICAL   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL, DOSMTH, ISCMP,
     *   DOXCOR, DOACOR, DOWTCL, DOFQSL
      INTEGER   INXRNO, NINDEX, FSTVIS, LSTVIS, IFQRNO
      REAL      TIMRNG(8),  UVRNG(2), INTPRM(3), UVRA(2), TSTART, TEND,
     *   SELFAC(2,XTTSZ), SMTAB(2500), SUPRAD, SELBAN
      CHARACTER SOURCS(30)*16, CALSOU(30)*16, STOKES*4, INTFN*4,
     *   SELCOD*4
      DOUBLE PRECISION UVFREQ, SELFRQ
C                                     Flag table info
      REAL      TMFLST, FLGTND(MAXFLG)
      INTEGER   IFGRNO
      LOGICAL   DOFLAG, FLGPOL(4,MAXFLG)
      INTEGER   FGVER, NUMFLG, FGKOLS(MAXFGC), FGNUMV(MAXFGC),
     *   KNCOR, KNCF, KNCIF, KNCS,
     *   FLGSOU(MAXFLG), FLGANT(MAXFLG), FLGBAS(MAXFLG), FLGSUB(MAXFLG),
     *   FLGBIF(MAXFLG), FLGEIF(MAXFLG), FLGBCH(MAXFLG), FLGECH(MAXFLG)
C                                     CAL table info
      REAL      GMMOD, CURCAL(XCTBSZ), LCALTM, CALTAB(XCTBSZ,2),
     *   CALTIM(3), RATFAC(MAXIF), DELFAC(MAXIF), DXTIME, DXFREQ,
     *   LAMSQ(MAXCHA, MAXIF), IFRTAB(MAXANT, 2), IFR(MAXANT)
      INTEGER   ICLRNO, NCLINR, MAXCLR, CNTREC(2,3)
      LOGICAL   DOCAL, DOAPPL
      INTEGER   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CIDSOU(2),
     *   CLKOLS(MAXCLC), CLNUMV(MAXCLC), LCLTAB, LCUCAL, ICALP1, ICALP2,
     *   POLOFF(4,2)
C                                     Baseline table info
      REAL      LBLTM, BLTAB(XBTBSZ,2), BLFAC(XBTBSZ), BLTIM(3)
      INTEGER   IBLRNO, NBLINR
      LOGICAL   DOBL
      INTEGER   BLVER, BLKOLS(MAXBLC), BLNUMV(MAXBLC), IBLP1, IBLP2
C                                     Polarization table.
      REAL      POLCAL(2,XPTBSZ), PARAGL(2,MAXANT), PARTIM
      INTEGER   PARSOU
      LOGICAL   DOPOL
C                                     Bandpass table
```

```
      DOUBLE PRECISION BPFREQ(MAXIF)
      REAL      PBUFF(XBPBUF), TIMENT(XBPSZ), BPTIM(3), LBPTIM, CHNBND
      CHARACTER BPNAME*48
      INTEGER   IBPRNO, NBPINR, ANTPNT(2), NVISM, NVISS, HVIST
      INTEGER   BPVER, BPKOLS(MAXBPC), BPNUMV(MAXBPC), NANTBP, NPOLBP,
     *   NIFBP, NCHNBP, BCHNBP, DOBAND, ANTENT(XBPSZ,MAXANT),
     *   BPDSK, BPVOL, BPCNO, USEDAN(MAXANT), BPGOT(2),
     *   KSNCF, KSNCIF, KSNCS, MXANUM
C                                       Channel 0 stuff
      INTEGER   FSTVS3, LREC3, LSTVS3, NREAD3, FSTRD3, KLOCW3,
     *   KLOCS3, NDECM3, DECM3(2,MAXIF*4), BIND3, RECNO3, LENBU3
      LOGICAL   ISCMP3, DOUVIN
C                                       File specification.
      INTEGER   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN,
     *   IXLUN, IBLUN, IPLUN, IQLUN, LUNSBP, BPFIND, CATUV(256),
     *   CATBLK(256)
      REAL      USEQ, UDISK
      CHARACTER UNAME*12, UCLAS*6, UFILE*48
C                                       I/O buffers
      INTEGER   CLBUFF(1024), FGBUFF(512), NXBUFF(512), BLBUFF(512),
     *   BPBUFF(32767), FQBUFF(512)
      REAL      UBUFF(8192)
C                                       Character common
      COMMON /SELCHR/ SOURCS, CALSOU, STOKES, INTFN, SELCOD, UNAME,
     *   UCLAS, UFILE, BPNAME
C                                       Common for UVGET use
C                                       Data selection and control
      COMMON /SELCAL/ UVFREQ, SELFRQ,
     *   USEQ, UDISK, TIMRNG, UVRNG, INTPRM, UVRA, TSTART, TEND, UBUFF,
     *   SELFAC, SMTAB, SUPRAD, SELBAN,
     *   INXRNO, NINDEX, FSTVIS, LSTVIS, IFQRNO,
     *   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL, DOSMTH, ISCMP, DOXCOR,
     *   DOACOR, DOWTCL, DOFQSL,
     *   CLBUFF, FGBUFF, NXBUFF, BLBUFF, BPBUFF, FQBUFF,
     *   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN, IXLUN,
     *   IBLUN, IPLUN, IQLUN, LUNSBP, BPFIND, CATUV, ANTENS, NANTSL,
     *   NSOUWD, SOUWAN, SOUWTN, NCALWD, CALWAN, CALWTN,
     *   SUBARR, SMOTYP, CURSOU, NXKOLS, NXNUMV, FQKOLS, FQNUMV,
     *   NVIS, JADR, PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU,
     *   KLOCFQ, SELQUA, SMDIV, SMOOTN, KLOCIF, KLOCFY, KLOCWT,
     *   KLOCSC, NDECMP, DECMP, BCHANS, ECHANS, FRQSEL, FSTRED
C                                       FLAG table info
      COMMON /CFMINF/ TMFLST, FLGTND, IFGRNO, DOFLAG, FLGPOL,
     *   FGVER, NUMFLG, FGKOLS, FGNUMV, KNCOR, KNCF, KNCIF, KNCS,
     *   FLGSOU, FLGANT, FLGBAS, FLGSUB, FLGBIF, FLGEIF, FLGBCH, FLGECH
C                                       CAL table info
      COMMON /CGNINF/ GMMOD, CURCAL, LCALTM, CALTAB, CALTIM, RATFAC,
     *   DELFAC, DXTIME, DXFREQ,
     *   ICLRNO, NCLINR, MAXCLR, CNTREC,
     *   DOCAL, DOAPPL,
     *   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CIDSOU, CLKOLS, CLNUMV,
     *   LCLTAB, LCUCAL, ICALP1, ICALP2, POLOFF,
     *   LAMSQ, IFRTAB, IFR
```

```
C                                       BL table info
      COMMON /CBLINF/ LBLTM, BLTAB, BLTIM, BLFAC,
     *   IBLRNO, NBLINR,
     *   DOBL,
     *   BLVER, BLKOLS, BLNUMV, IBLP1, IBLP2
C                                       Pol. table
      COMMON /CPLINF/ POLCAL, PARAGL, PARTIM, PARSOU, DOPOL
C                                       BP table
      COMMON /CBPINF/ BPFREQ,
     *   PBUFF, TIMENT, BPTIM, LBPTIM, CHNBND,
     *   IBPRNO, NBPINR, ANTPNT, NVISM, NVISS, NVIST,
     *   BPVER, BPKOLS, BPNUMV, NANTBP, NPOLBP, NIFBP, NCHNBP, BCHNBP,
     *   DOBAND, ANTENT, BPDSK, BPVOL, BPCNO, USEDAN, BPGOT,
     *   KSNCF, KSNCIF, KSNCS, MXANUM
C                                       Channel 0 common
      COMMON /CHNZ/ FSTVS3, LREC3, LSTVS3, NREAD3, FSTRD3, KLOCW3,
     *   KLOCS3, NDECM3, DECM3, BIND3, RECNO3, LENBU3,
     *   ISCMP3, DOUVIN
C
      COMMON /MAPHDR/ CATBLK
C                                                       End DSEL.
```

7.7.6 DUVH.INC

```
C                                                       Include DUVH.
C                                       If you change this include you
C                                       must also change common
C                                       /CATHDR/ in DBCON
C                                       Include for uv header info
      INTEGER   NVIS
      INTEGER   ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, ILOCFQ,
     *   JLOCC, JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC,
     *   NCOR, INCS, INCF, INCIF, ICORO, TYPUVD
      CHARACTER   SOURCE*8, ISORT*2
      DOUBLE PRECISION FREQ, RA, DEC
      COMMON /UVHDR/ FREQ, RA, DEC, NVIS, ILOCU, ILOCV, ILOCW, ILOCT,
     *   ILOCB, ILOCSU, ILOCFQ, JLOCC, JLOCS, JLOCF, JLOCR, JLOCD,
     *   JLOCIF, INCS, INCF, INCIF, ICORO, NRPARM, LREC, NCOR, TYPUVD
      COMMON /UVHCHR/ SOURCE, ISORT
C                                                       End DUVH.
```

7.8 Routines

7.8.1 APCONV

APCONV is a disk based, two dimensional convolution routine. The image to be convolved and the FFT of the convolving function are passed to APCONV along with two scratch files. All are specified as pointers to the arrays in the common (/CFILES/) from INCLUDE DFIL.INC. NOTE: Uses AIPS LUNs 18, 23, 24, 25.

```
APCONV (NX, NY, LI, LW1, LW2, LO, LC, FACTOR, JBUFSZ, BUFF1, BUFF2,
```

```
  *   BUFF3, SMAX, SMIN, IERR)
Inputs:
   NX        I     The number of columns in the input image (must be
                   a power of 2).
   NY        I     The number of rows in the input image.
   LI        I     File number in /CFILES/ of input.
   LW1       I     File number in /CFILES/ of work file no. 1
                      size = (4*NX x NY+2).
   LW2       I     File number in /CFILES/ of work file no. 1
                      size = (4*NX x NY+2).
   LO        I     File number in /CFILES/ of output.
   LC        I     File number in /CFILES/ of FFT of convolving fn.
                      size = (4*NX x NY+2).
   FACTOR    R     Normalization factor for convolving function; i.e.
                   is multiplied by the transform of the convolving
                   function
   JBUFSZ    I     Size of BUFF1,2,3 in AIPS bytes.  Should be
                   large, at least 8192 words.
Output:
   BUFF1     R(*)  Working buffer
   BUFF2     R(*)  Working buffer
   BUFF3     R(*)  Working buffer
   SMAX      R     Maximum value in the output file.
   SMIN      R     Minimum value in the output file.
   IERR      I     Return error code, 0 => OK, otherwise error.
```

7.8.2 CALCOP

Routine to copy selected data from one data file to another optionally applying calibration and editing information. The input file should have been opened with UVGET. Both files will be closed on return from CALCOP. Note: UVGET returns the information necessary to catalog the output file. The output file will be reduced in size if necessary at completion of CALCOP. Makes heavy use of common /CFILES/ from INCLUDE DFIL.INC.

```
CALCOP (DISK, CNOSCR, BUFFER, BUFSZ, IRET)
Inputs:
   DISK      I       Disk number for cataloged output file.
                     If .LE. 0 then the output file is a /CFILES/
                     scratch file.
   BUFFER    R(*)    Work buffer for writing.
   BUFSZ     I       Size of BUFFER in bytes.
Input via common: (DUVH.INC)
   LREC      I       length of vis. record in R words.
   NRPARM    I       number of R random parameters.
In/out:
   CNOSCR    I       Catalog slot number for if cataloged file;
                     (DFIL.INC) scratch file number if a scratch
                     file,
                     IF DISK=CNOSCR=0 then the scratch is created.
                     On output = Scratch file number if created.
In/out via common:
   CATBLK    I(256)  Catalog header block from UVGET
                     on output with actual no. records
   NVIS      I       (DUVH.INC) Number of vis. records.
Output:
```

```
   IRET     I       Error code: 0 => OK,
                       > 0 => failed, abort process.
Usage notes:
(1) UVGET with OPCODE='INIT' MUST be called before CALCOP to setup
    for calibration, editing and data translation.  If an output
    cataloged file is to be created this should be done after the
    call to UVGET.
(2) Uses AIPS LUN 24
```

7.8.3 DSKFFT

DSKFFT is a disk based, two dimensional FFT. If the FFT all fits in AP memory then the intermediate result is not written to disk. Input or output images in the sky plane are in the usual form (i.e. center at the center, X the first axis). Input or output images in the uv plane are transposed (v the first axis) and the center-at-the-edges convention with the first element of the array the center pixel. NOTE: Uses AIPS LUNs 23, 24, 25. Makes use of commons in INCLUDE DFIL.INC.

```
DSKFFT (NR, NC, IDIR, HERM, LI, LW, LO, JBUFSZ, BUFF1,
  *   BUFF2, SMAX, SMIN, IERR)
Inputs:
   NR       I     The number of rows in input array (# columns in
                  output).  When HERM is TRUE and IDIR=-1, NR is
                  twice the number of complex rows in the input file
   NC       I     The number of columns in input array (# rows in
                  output).
   IDIR     I     1 for forward (+i) transform, -1 for inverse (-i)
                  transform.
                  If HERM = .TRUE. the follwing are recognized:
                     IDIR=1 keep real part only.
                     IDIR=2 keep amplitudes only.
                     IDIR=3 keep full complex (half plane)
   HERM     L     When HERM = .FALSE., this routine does a complex to
                  complex transform.
                  When HERM = .TRUE. and IDIR = -1, it does a
                  complex to real transform.  When HERM = .TRUE. and
                  IDIR = 1, it does real to complex.
   LI       I     File number in (DFIL.INC) of input.
   LW       I     File number in (DFIL.INC) of work file (may equal LI)
   LO       I     File number in (DFIL.INC) of output.
   JBUFSZ   I     Size of BUFF1, BUFF2 in bytes.  Should be large
                  at least 4096 R   words.
Output:
   BUFF1    R(*)  Working buffer
   BUFF2    R(*)  Working buffer
   SMAX     R     For HERM=.TRUE. the maximum value in output file.
   SMIN     R     For HERM=.TRUE. the minimum value in output file.
   IERR     I     Return error code, 0 => okay, otherwise error.
```

7.8.4 GRDCOR

GRDCOR normalizes and corrects for the gridding convolution function used in gridding uv data to make the image. Uses AIPS LUNs 18 and 19

```
GRDCOR (IFIELD, DOGCOR, DISKI, CNOSCI, DISKO, CNOSCO,
  *   MAPMAX, MAPMIN, JBUFSZ, BUFF1, BUFF2, BUFF3, IRET)
```

```
Input:
   IFIELD      I     The subfield number, if = 1 the histogram is
                     zero filled first.
                     If IFIELD = 0 the input is assumed to be
                     a beam.
   DOGCOR      L     If TRUE, do gridding convolution correction.
   DISKI       I     Input file disk number for catalogd files,
                     .LE. 0 => /CFILES/ scratch file.
   CNOSCI      I     Input file catalog slot number or /CFILES/
                     scratch file number.
   DISKO       I     Output file disk number for catalogd files,
                     .LE. 0 => /CFILES/ scratch file.
   CNOSCO      I     Output file catalog slot number or /CFILES/
                     scratch file number.
   JBUFSZ      I     Size in bytes of buffers. Dimension of
                     BUFF1,2,3  must be at least 4096 words.
From commons: (Includes DGDS, DMPR, DUVH)
   BEMMAX       R    Sum of the weights used in gridding, used
                     to normalize images.
   CTYPX,CTYPY  I    Convolving function types for RA and Dec
   XPARM(10)    R    Convolving function parameters for RA
                     XPARM(1) = support half width.
   YPARM(10)    R    Convolving function parameters for Dec.
   BORES(16)    I    Block offset desired in output file for
                     an image, 1 per field. (1 rel.)
   BOBEM        I    Block offset desired in output file for
                     an beam. (1 rel.)
   NGRDAT       L    If FALSE get map size, scaling etc. parms
                     from the model map cat. header. If TRUE
                     then the values filled in by GRDAT must
                     already be filled into the common.
The following must be provided if NGRDAT is .TRUE.
   FLDSZ(2,*)       I   Dimension of map in RA, Dec (cells)
   ICNTRX,ICNTRY(*) I   The center pixel in X and Y for each
                        field.
Output:
   MAPMAX      R     The maximum value in the resultant image.
   MAPMIN      R     The minimum value in the resultant image.
   BUFF1       R     Working buffer
   BUFF2       R     Working buffer
   BUFF3       R     Working buffer
   IRET        I     Return error code. 0=>OK, error otherwise.
```

7.8.5 MAKMAP

MAKMAP makes a image or a dirty beam given a uv data set. The data may either calibrated or uncalibrated (raw) data and calibration and various selection criteria may be (optionally) applied. Data in an arbitrary sort order can be processed although only "TB" ordered data can be calibrated or edited.

The weights of the data may (optionally) have the uniform weighting correction made.

The visibilities are convolved onto the grid using the convolving function specified by CTYPX, CTYPY, XPARM, YPARM. The defaults for these values are filled in by a call to GRDFLT. The gridded data is phase rotated so that the map center comes out at location ICNTRX, ICNTRY. If requested, a uv taper is applied to the visibility weights before gridding. If necessary, a three dimension phase reference position shift is done.

Multiple channels may be gridded onto the same grid; a technique calles bandwidth synthesis. This bandwidth synthesis (BS) process may use the SCRWRK file. For bandwidth synthesis both the CNOSCO and SCRWRK files should be big enough for an extra m rows, where m is the half width of the X convolving function. Zero spacing flux densities are gridded if provided.

The final image will be normalized and (optionally) corrected for the effects of the gridding convolution function.

The input and output files are specified by either disk number and catalog number or as pointers in the /CFILES/ common from INCLUDE DFIL.INC. Input uv data file in UV file CNOSCI, DISKI. Output image file in image file CNOSCO, DISKO and may optionally be created as a scratch file.

Communication is through commons in INCLUDES DSEL.INC, DGDS.INC and DMPR.INC.

Uses buffer UBUFF from the UVGET commons (include DSEL.INC)

```
MAKMAP (IFIELD, DISKI, CNOSCI, DISKO, CNOSCO, SCRGRD, SCRWRK,
  *   CHANUV, CHANIM, DOCREA, DOINIT, DOBEAM, DOSEL, DOGCOR,
  *   JBUFSZ, BUFFER, IRET)
Inputs:
   IFIELD      I     Field number to map, if 0 then make a beam.
   DISKI       I     Input file disk number for cataloged files,
                     .LE. 0 => /CFILES/ scratch file.
   CNOSCI      I     Input file catalog slot number or /CFILES/
                     scratch file number.
   DISKO       I     Output file disk number for cataloged files,
                     .LE. 0 => /CFILES/ scratch file.
   CNOSCO      I     Output file catalog slot number or /CFILES/
                     scratch file number.  If DOCREA is FALSE and
                     DISKO=0 and CNOSCO=0 a scratch file is created.
   SCRGRD      I     Grid scratch file number, will be set if the
                     file is created, (DOINIT=TRUE)
   SCRWRK      I     Work scratch file number, will be set if the
                     file is created, (DOINIT=TRUE)
   CHANUV      I     Channel number to grid.  If DOSEL=TRUE
                     then this is 1-rel wrt the selected data.
   CHANIM      I     Channel number of output image.
   DOCREA      L     If TRUE, Create/catalog output image file.
   DOINIT      L     If TRUE, initialize scratch files, set defaults
                     for convolving functions.  Should
                     be TRUE on first call, and FALSE there after.
   DOBEAM      L     If TRUE a grid the beam before gridding the
                     field.  See useage notes.
   DOSEL       L     If true, data need to be reformatted to a
                     single Stokes' type.  If TRUE, the cataloged
                     file NAME, CLASS etc should be filled into
                     UNAME, UCLAS, UDISK, USEQ in common /SELCAL/
   DOGCOR      L     If TRUE, correct image for gridding
                     convolution correction function.
                     (Normally .TRUE.)
   JBUFSZ      I     Size in bytes of buffers. Dimension of
                     BUFFER  must be at least 4096 R.
From commons: (Includes DGDS and DMPR)
   MFIELD       I    The number of fields which are going to
                     to be imaged (excluding any beam).
                     MUST be filled in.
   FLDSZ(2,*)   I    Dimension of map in RA, Dec (cells) of each
                     field.  MUST be completely filled in before the
                     DOINIT=TRUE call if the output file (either
```

```
                    image or scratch) is to be created or zeroed
                    if the files already exist.
   DOUNIF       L   If TRUE, apply Uniform weighting. Should be
                    TRUE on only the first call, otherwise it will
                    be applied again.
   NCHAVG       I   Number of channels to grid together for
                    bandwidth synthesis.
   UNFBOX       I   Half width of unif. wt. counting box size.
   CTYPX,CTYPY  I   Convolving function types for RA and Dec
   XPARM(10)    R   Convolving function parameters for RA
                    XPARM(1) = support half width.
   YPARM(10)    R   Convolving function parameters for Dec.
   UVRNG(2)     R   Minimum and maximum baseline lengths in
                    1000's wavelengths. 0's => all
   XSHIFT(16)   R   Shift in X (after rotation) in asec.
                    in projected coordinates. 1 per field.
   YSHIFT(16)   R   Shift in Y (after rotation) in asec.
                    in projected coordinates. 1 per field.
   STOKES       C*4 Stokes types wanted.
                    'I','Q','U','V','R','L'
   DOZERO       L   If true then do zero spacing flux.
   ZEROSP(5)    R   Zero spacing flux, 1=>flux density (Jy)
                    5 => weight to use.
                    polarization.
   TFLUXG       R   The total flux density removed from the data,
                    this will be subtracted from the zero spacing
                    flux before gridding.
   DOTAPE       L   True if taper requested.
   TAPERU,TAPERV R  TAPER ( to 30%) in u and v (kilolamda)
   NXUNF,NYUNF  I   Dimension (cells) of the map in RA and Dec
                    to be used to set uniform weighting.
                    (should be min. of FLDSZ)
The following must be provided if DOSEL is .FALSE.:
  CATBLK(256)   I   Catalog header for uv data input file.
                    (only used on DOINIT=TRUE call)
The following must be provided if DOCREA is .TRUE. (includes DMPR,
 DGDS)
   MNAME            C*12 Output image name.
   MCLASS           C*6  Output image class.
                         (If more than 1 field the last 2 char
                         are used to encode the field number)
   MDISK            I    Desired image file output disk
   MSEQ             I    Desired image file output sequence no.
The following must be provided if the output file is to be created;
either by setting DOCREA=TRUE or DISKO=CNOSCO=0.
   FLDSZ(2,*)       I    Dimension of map in RA, Dec (cells)
   NXBEM,NYBEM      I    Dimension (cells) of beam.
   CELLSG(2)        R    The cell spacing in X and Y in arcseconds.
   XSHIFT(16)       R    Shift in X (after rotation) in asec.
                         in projected coordinates. 1 per field.
   YSHIFT(16)       R    Shift in Y (after rotation) in asec.
                         in projected coordinates. 1 per field.
   ICNTRX,ICNTRY(*) I    The center pixel in X and Y for each
                         field. 0 values cause the default.
```

```
The following must be provided if DOCREA is FALSE and output
files already exist. (Includes DGDS).
   CCDISK(16)   I    Disk numbers of the output images.
                     (Must be zeroed if not filled in.)
   CCCNO(16)    I    Catalog slot numbers of output images.
                     (Must be zeroed if not filled in.)
The following must be provided if DOSEL is .TRUE.
(Includes DSEL.INC)
   UNAME        C*12 AIPS name of input file.
   UCLAS        C*6  AIPS class of input file.
   UDISK        R    AIPS disk of input file.
   USEQ         R    AIPS sequence of input file.
   FGVER        I    FLAG file version number, if .le. 0 then
                     NO flagging is applied.
   SOURCS(1)    C*16 Name of desired source.
   TIMRNG(8)    R    Start day, hour, min, sec, end day, hour,
                     min,sec. 0's => all
   STOKES       C*4  Stokes types wanted.
                     'I','Q','U','V','R','L'
   BCHAN        I    First channel number selected, 1 rel. to first
                     channel in data base. 0 => all
   ECHAN        I    Last channel selected. 0=>all
   BIF          I    First IF number selected, 1 rel. to first
                     IF in data base. 0 => all
   EIF          I    Last IF selected. 0=>all
   DOCAL        L    If true apply calibration, else not.
The following must be provided if DOCAL is TRUE.
   ANTENS(50)   I    List of antennas selected, 0=>all,
                     any negative => all except those specified
   GAUSE        I    GAIN (CL or SN) file version number to use.
Output:
   DISKI        I    UV data file disk if data reformatted.
   CHOSCI       I    Reformatted uv data scratch file number
                     to be used in subsequent calls.
   DISKO        I    Output image file disk number if output file.
                     created and/or cataloged (DOCREA=TRUE
                     or input DISKO=0 and CHOSCO=0).
   CHOSCO       I    Output image file catalog slot number
                     or scratch file number if output file created.
   SCRGRD       I    Grid scratch file number, will be set if the
                     file is created, (DOINIT=TRUE)
   SCRWRK       I    Work scratch file number, will be set if the
                     file is created, (DOINIT=TRUE)
   DOSEL        L    Set to FALSE if data reformatted.
   DOBEAM       L    Set to FALSE.
   DOINIT       L    Set to FALSE.
   BUFFER(*)    R    Working buffer
   IRET         I    Return error code. 0=>OK, error otherwise.
Output in Common:
   DOUNIF       L    Set to FALSE if uniform weighting applied.
   UBUFSZ       I    Buffer size for UBUFF (UVGET buffer)
   MNAME        C*12 Output image name. (defaults applied)
   MCLASS       C*6  Output image class (defaults applied)
   MDISK        I    Desired image file output disk
```

```
                    (defaults applied)
   MSEQ        I    Desired image file output sequence no.
                    (defaults applied)
   FLDMAX(*)   R    Maximum pixel value in field.
   FLDMIN(*)   R    Minimum pixel value in field.
The following are filled in if a output file is created:
   CCDISK(16)  I    Disk numbers of the output images.
   CCCNO(16)   I    Catalog slot numbers of output images.
Useage Notes:
 1) The input uvdata file is, with one exception, assumed to be
  accurately described by the contents of CATR and the common
  /UVHDR/ (include DUVH).  The exception is that the u, v and
  w may refer to a different frequency.  The reference frequency for
  the u,v and w terms is taken from the input CATBLK in the DOINIT
  TRUE call unless the data is reformatted (DOSEL=TRUE).
  In this latter case this frequency is obtained from UVGET call.
  If DOSEL = TRUE the input value of CATBLK is ignored.
 2) Information about the output image is obtained from the
  catalog header for the relevant file.  If MAKMAP makes the
  output file this information is filled in.  If MAKMAP does not
  make the output image file then this information must be filled
  in before hand.  Routine IMCREA will help do this.  Note: even
  scratch files are cataloged and thus have a catalog header.
  If MAKMAP does not create the output files, CCDISK(IFIELD) and
  CCCNO(IFIELD) should give their disk and catalog slot number
  before the call to MAKMAP.
 3) only one polarization can be processed and the input data
  to the gridding routine is assumed to be in the desired Stokes'
  type (i.e. I, Q, U, V etc.).
     If DOSEL = TRUE the input data will be selected, calibrated
  and reformatted as specified in common (include DSEL).
  Only Stokes' types I,Q,U,V,R,L should be used.
     Multiple channels may be gridded together a la bandwidth
  synthesis by specifying NCHAVG > 1. One channel of several
  channels may be gridded specified by CHANUV.
 4) If DOSEL=FALSE on the first call (i.e. the data is not
  reformatted),  the random parameters in the data should include,
  in order, u, v, w, weight (optional), time (optional) and baseline
  (optional).  While the last are optional and not used, the last
  words of random parameters are used as work space and, if they
  are missing, u, v, and w may be clobbered.  The weights are
  required but may be passed either as random parameters or as
  part of the regular data array, CATR should tell which.
  If DOSEL=TRUE is used these conditions will be satisfied.
 5) The necessary image normalization constant for proper
  normalization of the FFTed image is produced only by gridding the
  beam.  If a beam is to be made, it should be done first; in this
  case DOBEAM should be FALSE in all calls.  If a beam is not
  desired then the first call to MAKMAP should have DOBEAM TRUE and
  FALSE on subsequent calls.  Note MAKMAP sets DOBEAM to FALSE.
 6) Much of the control information used by MAKMAP is passed to and
  stored in commons.  The calling routine should have the following
  includes:
  DHDR.INC, DUVH.INC, DFIL.INC, DMPR.INC, DGDS.INC, DSEL.INC
```

```
NOTE: care should be taken that the contents of these commons
not be clobbered by overlaying.
7) If calibration is applied then up to 8 map and 3 non map files
will be open at once; this should be reflected in the call to
ZDCHIN and the dimension of FTAB in the main routine of the
calling program.  MAKMAP may use AIPS LUNs 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 28, 29, 30.
```

7.8.6 UVGET

Subroutine to obtain data from a data base with optional application of flaging and/or calibration information. Reads data with a large variety of selection criteria and will reformat the data as necessary. Does many of the startup operations, finds uv data file etc, reads CATBLK and updates the DUVH.INC commons to reflect the output rather than input data. Most of the input to UVGET is through the commons in DSEL.INC; the initial (default) values of these may be set using routine SELINI.

```
UVGET (OPCODE, RPARM, VIS, IERR)
Input:
   OPCODE   C*4        Opcode:
                       'INIT' => Open files Initialize I/O.
                       'READ' => Read next specified record.
                       'CLOS' => Close files.
Inputs via common /SELCAL/  (Include DSEL.INC)
   UNAME    C*12       AIPS name of input file.
   UCLAS    C*6        AIPS class of input file.
   UDISK    R          AIPS disk of input file.
   USEQ     R          AIPS sequence of input file.
   SOURCS   C(30)*16   Names of up to 30 sources, *=>all
                       First character of name '-' => all except
                       those specified.
   TIMRNG   R(8)       Start day, hour, min, sec, end day, hour,
                       min, sec. 0's => all
   UVRNG    R(2)       Minimum and maximum baseline lengths in
                       1000's wavelengths. 0's => all
   STOKES   C*4        Stokes types wanted.
                       'I','Q','U','V','R','L','IQU','IQUV'
                       '    '=> Leave data in same form as in input.
   BCHAN    I          First channel number selected, 1 rel. to first
                       channel in data base. 0 => all
   ECHAN    I          Last channel selected. 0=>all
   BIF      I          First IF number selected, 1 rel. to first
                       IF in data base. 0 => all
   EIF      I          Last IF selected. 0=>all
   DOCAL    L          If true apply calibration, else not.
   DOPOL    L          If true then correct for feed polarization
                       based on antenna file info.
   DOSMTH   L          True if smoothing requested.
   DOACOR   L          True if autocorrelations are requested.
   DOWTCL   L          True if weight calibration wanted.
   DOFQSL   L          True if FREQSEL random parm present (false)
   FRQSEL   I          Default FQ table entry to select (-1)
   SELBAN   R          Bandwidth (Hz) to select (-1.0)
   SELFRQ   D          Frequency (Hz) to select (-1.0)
   DOBAND   I          >0 if bandpass calibration. (-1)
   BPNAME   C*48       Name of scratch file set up for BP's.
```

```
   DOSMTH   L         True if smoothing requested. (false)
   SMOOTH   R(3)      Smoothing parameters (0.0s)
   DXTIME   R         Integration time (days). Used when applying
                      delay corrections to correct for delay error.
   ANTENS   I(50)     List of antennas selected, 0=>all,
                      any negative => all except those specified
   SUBARR   I         Subarray desired, 0=>all
   FGVER    I         FLAG file version number, if < 0 then
                      NO flagging is applied. 0 => use highest
                      numbered table.
   CLUSE    I         Cal (CL or SN) file version number to apply.
   BLVER    I         BL Table to apply .le. 0 => none
   BPVER    I         BP table to apply .le. 0 => none
Output:
   RPARM    R(*)      Random parameter array of datum.
   VIS      R(3,*)    Regular portion of visibility data.
   IERR     I         Error code: 0 => OK,
                          -1 => end of data
                          >0 => failed, abort process.
Output in commons in DSEL.INC: The default values will be filled in
if null values were specified.
   UVFREQ   D         Frequency corresponding to u,v,w
   CATBLK   I(256)    Catalog header block, describes the output
                      data rather than input.
   NPRMIN   I         Number or random parameters in the input data.
   TRANSL   L         If true translate data to requested Stokes'
   CNTREC   I(2,3)    Record counts:
                      (1&2,1) Previously flagged (partly, fully)
                      (1&2,2) Flagged due to gains (part, full)
                      (1&2,3) Good selected (part, full)
   ISCMP    L         True if input data is compressed.
   KLOCSU   I         0-rel random parm. pointer for source in input
                      file.
   KLOCFQ   I         0-rel random parm. pointer for FQ id in input
                      file.
   KLOCIF   I         0-rel random parm. pointer for IF in input
                      file.
   KLOCFY   I         0-rel random parm. pointer for freq. in input
                      file.
   KLOCWT   I         0-rel random parm. pointer for weight in
                      input file.
   KLOCSC   I         0-rel random parm. pointer for scale in
                      input file.
Usage notes:
 1) Include DSEL.INC should be declared in the main program or at a
    level that they will not be overlaid while UVGET is in use (ie.
    between the 'INIT' and 'CLOS' calls). SELINI can be used to
    initialize the control variables in these commons.
 2) If no sorting is done UVGET uses AIPS luns 25, 28, 29 and 30
   (1 map, 3 non map files).  If sorting is done (usually possible)
   then 8 map and 3 non map files are used (mostly on OPCODE='INIT')
   and LUNs 16,17,18,19,20,21,22,23,24,25, 28,29,30,40,42,43,44,45.
 3) OPCODE = 'INIT' does the following:
   - The catalgue data file is located and the catalog header
```

```
      record is read.
    - The source file (if any) is read.
    - The index file (if any) is initialized.
    - The flag file (if any) is initialized and sorted if necessary
      (Must be in time order).
    - The gain table (if any) is initialized.
    - The bandpass table (if any) is initialized
    - The smoothing convolution table (if any) is initialized
    - I/O to the input file is initialized.
        The following LUNs may be used but will be closed on
      return: 16, 17, 18, 19, 20, 21, 22, 23, 24
        The following LUNs may be used but will be open on
      return: 25 (uv data), 28 (NX table), 29 (CL or SN table),
              30 (FG table), 40 (BL table), 41 (BP table).
        NO data are returned from this call.
  4) OPCODE = 'READ' reads one visibility record properly selected,
     transformed (e.g. I pol.), calibrated and edited as requested
     in the call with OPCODE = 'INIT'
  5) OPCODE = 'CLOS' closes all files used by UVGET which are still
     open.  No data are returned.
  6) If DOCAL is true then the common array CNTREC will contain the
     counts of records which are good or fully or partly flagged
     both previously and due to flagged gain solutions.
  7) Only one subarray can be calibrated at a time if DOPOL is true.
     This is because the polarization information for only one
     subarray is kept at a time.
```

7.8.7 UVMDIV

UVMDIV divides model visibilities derived from CLEAN or Gaussian components or images into a uv data set. The weights of the data returned will be the input values multiplied by the model amplitude.

A variety of model computation methods are available; if a single pass through VISDFT, the DFT routine, is not sufficient then the data is copied to a scratch file which has space for a second copy of the data, the model values are computed and summed in these locations and finally then model is divided into the data and written to the output file.

Extensive use is made of commons to communicate with UVMDIV, in particular /MAPDES/ (include DGDS.INC) contains most of the critical information about the model components files or images to be used. Common /UVHDR/ (DUVH.INC filled in by UVPGET) is presumed to describe the uv data files.

If the data is not sorted 'X*' and MODEL=1 then UVMSUB will use the DFT irregardless of the value of METHOD.

Also fills in frequency table (NCHANG, FREQG) in INCLUDE DGDS.INC

```
UVMDIV (DISKI, CNOSCI, DISKO, CNOSCO, MODEL, METHOD, DOMSG, CHANEL,
  *   NCHAN, CATBLK, JBUFSZ, FREQID, BUFF1, BUFF2, BUFF3, IRET)
 Inputs:
   DISKI      I   Input disk number. if .LE. 0 then input is a
                  scratch file.
   CNOSCI     I   Input file catalog slot number or /CFILES/
                  scratch file number.
   DISKO      I   Output disk number. if .LE. 0 then output is a
                  scratch file.
   CNOSCO     I   Output file catalog slot number or /CFILES/
                  scratch file number.  If .LE. 0 then one of the
                  internal scratch files will be used.
   MODEL      I   1=> clean components, 2=>image.
```

```
   METHOD      I   1=>gridded, -1=>DFT, 0=>chose.
   DOMSG       L   If true give percent done messages for DFT.
   CHANEL      I   First uv data channel to subtract.
   NCHAN       I   Number of frequency channels to subtract.
   CATBLK(256)I    Uv data catalog header record.
   JBUFSZ      I   Size of BUFF1,2,3 in bytes, must be at least 4096
                   words.
   FREQID      I   Freq ID number, if it exists.
   BUFF1,2,3   R   Work buffers.
Inputs from COMMON /MAPDES/:
   MFIELD      I     Number of fields
   NSUBG(*)    I     Number of components already sub.
   NCLHG(*)    I     Number of components per field.
   CCDISK(*)   I     Disk numbers for CC files
   CCCNO(*)    I     Catalog slot numbers for CC files.
   CCVER(*)    I     CC file version number for each field.
   FACGRD      R     Value to multiply clean component fluxes
                     by before subtraction (negative for sum).
   SCTYPE      C*2   Scratch file type to create. (eg. 'SC')
   NONEG       L    Stop reading comps. from a file past the first
                    negative component. (DFT modeling ONLY)
   DOPTMD      L    Use the point model specified by PTFLX, PTRAOF,
                    PTDCOF (DFT modeling ONLY)
   PTFLX       R    Point model flux density (Jy) (I pol. only)
   PTRAOF      R    Point model RA offset from uv phase center
                    (asec)
   PTDCOF      R    Point model Dec. offset from uv phase center
Input from COMMON /UVHDR/:
   LREC        I     Length of visibility record.
   NVIS        I     Number of visibility records.
   NRPARM      I     "Random" parameters before data, can be used
                     to skip observed values when computing model.
Output:
   CNOSCO      I   Output file catalog slot number or /CFILES/
                   scratch file number.  Value returned if not
                   specified in call.
   IRET        I   Return error code. 0=>OK, otherwise failed.
```

7.8.8 UVMSUB

UVMSUB subtracts a CLEAN or Gaussian model or an image from a set of uv data. Extensive use is made of commons to communicate with UVMSUB, in particular /MAPDES/ (include DGDS.INC) contains most of the critical information about the model components files or images to be subtracted. Common /UVHDR/ (filled in by UVPGET) is presumed to describe the uv data files.

If the data is not sorted 'X*' and MODEL=1 then UVMSUB will use the DFT irregardless of the value of METHOD.

Also fills in frequency table (NCHANG, FREQG) in INCLUDE DGDS.INC

```
UVMSUB (DISKI, CNOSCI, DISKO, CNOSCO, MODEL, METHOD, CHANEL, NCHAN,
  *   DOSUM, DOMSG, CATBLK, JBUFSZ, FREQID, BUFF1, BUFF2, BUFF3,
  *   IRET)
 Inputs:
   DISKI       I   Input disk number. if .LE. 0 then input is a
                   scratch file.
   CHOSCI      I   Input file catalog slot number or /CFILES/
```

```
                scratch file number.
   DISKO      I   Output disk number. if .LE. 0 then output is a
                  scratch file.
   CNOSCO     I   Output file catalog slot number or /CFILES/
                  scratch file number.
   MODEL      I   1=> clean components, 2=>image.
   METHOD     I   1=>gridded, -1=>DFT, 0=>chose.
   CHANEL     I   First uv data channel to subtract.
   NCHAN      I   Number of frequency channels to subtract.
   DOSUM      L   If true then sum component fluxes in FLUXG,
                  TFLUXG.
   DOMSG      L    If true give percent done messages for DFT.
   CATBLK(256)I   Uv data catalog header record.
   JBUFSZ     I   Size of BUFF1,2,3 in bytes, must be at least 4096
                  words.
   FREQID     I   Freq ID number, if it exists.
Inputs from COMMON /MAPDES/:
   MFIELD      I     Number of fields
   NSUBG(*)    I     Number of components already sub.
   NCLNG(*)    I     Number of components per field.
   CCDISK(*)   I     Disk numbers for CC files
   CCCNO(*)    I     Catalog slot numbers for CC files.
   CCVER(*)    I     CC file version number for each field.
   FACGRD      R     Value to multiply clean component fluxes
                     by before subtraction (negative for sum).
   NONEG       L    Stop reading comps. from a file past the first
                    negative component. (DFT modeling ONLY)
   DOPTMD      L    Use the point model specified by PTFLX, PTRAOF,
     .              PTDCOF (DFT modeling ONLY)
   PTFLX       R    Point model flux density (Jy) (I pol. only)
   PTRAOF      R    Point model RA offset from uv phase center
                    (asec)
   PTDCOF      R    Point model Dec. offset from uv phase center
Input from COMMON /UVHDR/ (DUVH.INC):
   LREC        I     Length of visibility record.
   NVIS        I     Number of visibility records.
   NRPARM      I     "Random" parameters before data, can be used
                     to skip observed values when computing model.
   BUFF1,2,3   R   Work buffers.
Output:
   IRET        I   Return error code. 0=>OK, otherwise failed.
```

Chapter 8

WaWa ("Easy") I/O

8.1 Overview

There is a fairrly coherent set of routines which attempt to hide many of the nasty details mentioned in the previous chapters. They perform most catalog file operations for the programmer and hide the details of calls to COMOFF, MINIT, MDISK, ZCREAT, et al. In many cases these cost memory and/or speed, but for computation-bound algorithms these are probably not important.

Any task which uses the WaWa package and creates scratch files should include the /CFILES/ common given in the INCLUDE DFIL.INC. The values of IBAD should be filled in using the contents of AIPS adverb BADDISK. This allows the scratch file creation routine to avoid putting files on user selectable disks.

8.2 Salient Features of the WaWa I/O package

1. Each main task calls a single setup routine; a maximum of 5 simultaneously open image files is allowed.
2. All the parameters needed to specify a cataloged file are gathered into a single array, called a namestring.
3. The WaWa package hides the interface between the parameter passing subroutines (e.g., GTPARM) and the I/O routines.
4. Many subroutine calls are combined so that e.g., ZPHFIL, CATDIR, CATIO, and MINIT, more or less disappear from sight.
5. A general clean-up subroutine for closing files and destroying scratch files is provided.
6. "Hidden" buffers large enough to hold a 2048-point image row are provided. These make double buffered I/O look more like FORTRAN I/O on the large mainframes.

8.3 Namestrings

In order to reduce the many arguments required for the fundamental AIPS I/O routines needed to specify the desired file the WaWa package uses a namestring. With a namestring it is possible to refer to any cataloged file by a character string. The name string used for WAWA I/O is a CHARACTER string of length 36 of the following form:

```
 1:12 C*12 Name
13:18 C*6 Class
19:20 C*2 Physical type
21:27 I7 Sequence number
28:29 I2 Disk number
30:36 I7 User id number
```

- MAPMAX - Find MAX & MIN of a map and enter into catalog.
- FILNUM - Find WaWa pointers to open file (used for history).
- GETHDR - Retrieve catalog header for an open cataloged file.
- SAVHDR - Save header in catalog for an open cataloged file.
- HDRINF - Retrieve specified items from map header.
- TSKBEG - Combination of IOSET and some task startup chores.
- TSKEND - Some task cleanup chores.

8.5 Things WaWa Can't Do Well or At All

There are several applications for which the WaWa routines are inadequate. The non-map I/O routines are much inferior to the other AIPS non-map I/O routines. Other applications, such as uv data handling and plotting, are not provided for at all. History files may be written in tasks using WaWa I/O, but it requires digging in the the WaWa commons. The following sections suggest possible courses of action.

8.5.1 Non-map files

The WaWa package is not overly useful for non-map I/O at the moment. The user will want to consult the chapter on disk I/O and the routines TABINI and TABIO for more useful software.

8.5.2 UV data files

No help here. See the chapter on disk I/O.

8.5.3 Plotting

The WaWa package has no plotting capability. See the chapter in this manual on plotting.

8.5.4 History

The WaWa package has no capacity to copy or write into history files. See the chapter on tasks and in particular the routines HISCOP and HIADD. In addition, you will need to determine the catalog slot numbers of the relevant files from the /WAWAIO/ common variable FILTAB(POCAT,*) (file must be open to do so). Use FILNUM. The task HGEOM provides a useful example of history writing within the WaWA I/O system.

8.5.5 More than 5 I/O Streams at a Time

If a task may need to have more than 5 map or non-map I/O streams open at the same time, then serious restructuring of the WaWa commons is needed. You are better off ignoring WaWa I/O and using the standard I/O described in the chapter on disk I/O.

8.5.6 I/O to Tapes

No help here. See the chapter on device I/O.

8.6 Additional goodies and "helpful" hints

A number of features have been added to the WaWa package to increase it usefulness. These will be discussed in the following sections. Also on occasion the programmer will have to find some of the things the WaWa package has hidden; a discussion of where WaWa hides useful information is also given in the following sections.

8.6.1 Use of LUNs

The LUN used does convey meaning. Legal values range from 9 through 30. However, values 16 through 25 convey an implication that the file is a map file, value 9 is reserved for the TV, and values 10 through 15 may get you into trouble. Use 26–30 for non-maps.

8.6.2 WaWa commons

The WaWa package hides many things in several commons. Frequently the programmer needs to know the contents of these commons. The following sections describe the contents of the commons.

Information common

The primary common in the WaWa package is obtained by the INCLUDE DITB.INC.. The text of this and other relevant includes are shown at the end of this chapter. The name of the primary WaWa I/O common is /WAWAIO/ and its contents are as follows:

```
WRIT        C*4    'WRIT'      I/O control strings
REED        C*4    'READ'
CLWR        C*4    'CLWR'      Catalog control strings
CLRD        C*4    'CLRD'
REST        C*4    'REST'
OPEN        C*4    'OPEN'
CLOS        C*4    'CLOS'
SRCH        C*4    'SRCH'
INFO        C*4    'INFO'
UPDT        C*4    'UPDT'
FINI        C*4    'FINI'      I/O control string
CSTA        C*4    'CSTA'      Catalog control string
INDEF       R      'INDE'      Blanked floating point pixel

SUBNAM      C*6(8) Subroutine names: CATDIR, CATIO, MINIT,
                   MDISK, ZCLOSE, ZCREAT, ZDESTR, ZOPEN

LINT        I      Number integer values in one IO buffer
LREAL       I      Number real values in one IO buffer
NFIL        I      Number simultaneous open map files
EFIL        I      Size of FILTAB ( 5 + NFIL) - number of
                   simultaneous files of all types
QUACK       I      0 => restart AIPS at end, 1 => already done

POLUN       I      FILTAB pointer for LUN value (1)
POFIN       I      FILTAB pointer for I/O table pointer value
                   (2)
POVOL       I      FILTAB pointer for disk number value (3)
POCAT       I      FILTAB pointer for cat location value (4)
POIOP       I      FILTAB pointer for opcode number (5):
                   values 1 => write, 2=> read, <0 => new win
```

```
POASS          I     FILTAB pointer for is it associated file
                     (6): 1 => assoc, 0 => main file
POBPX          I     FILTAB pointer for bytes/pixel code (7)
PODIM          I     FILTAB pointer for # axes (8)
POMAX          I     FILTAB pointer for # points on each of 7
                     axes (9)
POBLC          I     FILTAB pointer for Bottom left corner (16)
POTRC          I     FILTAB pointer for Top right corner (23)
PODEP          I     FILTAB pointer for current depth in I/O on
                     axes 2 - 7 (30), Area (36) used for integer
                     map (input) blanking code.
POBL           I     FILTAB pointer for block offset start I/O
                     in the current plane (37)

FILTAB(38,EFIL)  I       Table to hold all the values pointed
                         at by the PO... pointers above: (e.g.,
                         the cat number is = FILTAB (POCAT, n)
                         where n is found by finding that
                         FILTAB (POLUN, n) which = desired LUN
                         (Only for open files!!)
```

Catalog and Buffer Commons

There are 2 other commons which are used heavily. They are /MAPHDR/ which is a work area for map headers containing the equivalenced arrays CATBLK, CATH, CATR, and CATD. The contents of this common are changed frequently by the basic WaWa I/O routines, but it can be used, for example, to get the catalog header record after a call to FILOPN or OPENCF. This common may be obtained by the include DCAT.INC. The other common, called /WAWABU/ from INCLUDE DBUF.INC, contains:

```
RMAX       R(10)     1-5 used by MAPIO for scale factor
RMIN       R(10)     1-5 used by MAPIO for offset
WBUFF      I(256)    scratch buffer for catalog access
RBUF       R(*)      I/O buffers for map I/O.
```

The areas RMAX and RMIN for subscripts 6 through 10 could be used by a programmer, for example, to keep track of max/min. If no map file is currently open, RBUF is a large and useful scratch area of core.

Declaration of Commons

If a WaWa I/O task (or any other task for that matter) is to be overlaid on some computers, then all commons must be declared in the main program. For the WaWa system, this may be done by the following list of includes:

```
INCLUDE 'INCS:DBUF.INC'     WaWa buffer/table sizes
INCLUDE 'INCS:DITB.INC'     WaWa I/O common
INCLUDE 'INCS:DDCH.INC'     System parms
INCLUDE 'INCS:DHDR.INC'     Header pointers
INCLUDE 'INCS:DMSG.INC'     Messages, POPS #, ...
INCLUDE 'INCS:DCAT.INC'     Catalog header
INCLUDE 'INCS:DFIL.INC'     Gives BADDISK
```

8.6.3 Error return codes

A uniform system of error code numbers has been adopted in the WaWa I/O package. These codes are consistent with the error codes used by many I/O routines, but not with the other error codes in the multitudinous collection of AIPS routines. They are:

```
 1 => File not open
 2 => Input parameter error
 3 => I/O error ("other")
 4 => End of file (hardware generated, see 9)
 5 => Beginning of medium
 6 => End of medium
 7 => buffer too small
 8 => Illegal data type
 9 => Logical end of file (software generated, not hardware)
10 => Catalog operation error
11 => Catalog status error
12 => Map not in catalog
13 => EXT file not in catalog
14 => No room in header/catalog
16 => Illegal window specification
17 => Illegal window specification for writing a file
21 => Create: file already exists
22 => Create: volume unavailable
23 => Create: space unavailable
24 => Create: "other"
25 => Destroy: "other"
26 => Open: "other"
```

8.7 INCLUDEs

There are several types of INCLUDE file which are distinguished by the first character of their name. Different INCLUDE file types contain different types of Fortran declaration statements as described in the following list.

- Pxxx.INC. These INCLUDE files contain declarations for parameters and the PARAMETER statements.
- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations, COMMON and EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statements.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

8.7.1 DBUF.INC

```
C                                                       Include DBUF.
      REAL      RBUF(20480), RMAX(10), RMIN(10)
      INTEGER   WBUFF(256), IBUF(1)
      COMMON /WAWABU/ RMAX, RMIN, WBUFF, RBUF
      EQUIVALENCE (RBUF(1), IBUF(1))
C                                                       End DBUF.
```

8.7.2 DCAT.INC

```
C                                                          Include DCAT.
C                                        catalog header common
      INTEGER   CATBLK(256)
      REAL      CATR(256)
      HOLLERITH CATH(256)
      DOUBLE PRECISION CATD(128)
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE (CATBLK, CATR, CATH, CATD)
C                                                          End DCAT.
```

8.7.3 DFIL.INC

```
C                                                          Include DFIL.
C                                        AIPS system catalog and scratch
      INTEGER   NSCR, SCRVOL(128), SCRCNO(128), IBAD(10), LUNS(10),
     *   NCFILE, FVOL(128), FCNO(128), FRW(128), CCNO
      LOGICAL   RQUICK
      COMMON /CFILES/ RQUICK, NSCR, SCRVOL, SCRCNO, NCFILE, FVOL, FCNO,
     *   FRW, CCNO, IBAD, LUNS
C                                                          End DFIL.
```

8.7.4 DITB.INC

```
C                                                          Include DITB.
C                                        Wawa I/O common
      REAL    INDEF
      CHARACTER WRIT*4, REED*4, CLWR*4, CLRD*4, REST*4, OPEN*4, CLOS*4,
     *   SRCH*4, INFO*4, UPDT*4, FINI*4, CSTA*4, SUBNAM(8)*6
      INTEGER    LINT, LREAL, NFIL, EFIL, QUACK,
     *   POLUN, POFIN, POVOL, POCAT, POIOP, POASS, POBPX,
     *   PODIM, PONAX, POBLC, POTRC, PODEP, POBL, FILTAB(38,10)
      COMMON /WAWCHR/  WRIT, REED, CLWR, CLRD, REST, OPEN, CLOS,
     *   SRCH, INFO, UPDT, FINI, CSTA, SUBNAM
      COMMON /WAWAIO/ INDEF,  LINT, LREAL, NFIL, EFIL, QUACK,
     *   POLUN, POFIN, POVOL, POCAT, POIOP, POASS, POBPX,
     *   PODIM, PONAX, POBLC, POTRC, PODEP, POBL, FILTAB
C                                                          End DITB.
```

8.8 Detailed Descriptions of the Subroutines

8.8.1 A2WAWA

WaWa IO system: Packs Wawa-IO Namestring having format A12, A6, A2, I7, I2, I7 for NAME, CLASS, PTYPE, SEQ, VOL, USID from its component parts

```
A2WAWA (NAME, CLASS, SEQ, PTYPE, VOL, USID, NAMEST)
Inputs:
```

```
   NAME    C*12   file name
   CLASS   C*6    file class (6 chars)
   SEQ     I      file sequence number
   PTYPE   C*2    file physical type (2 chars)
   VOL     I      file disk number
   USID    I      user number
Outut:
   NAMEST  C*36   WaWa Namestring
```

8.8.2 CLENUP

WaWa IO system: Close all files opened with FILOPN. Destroy scratch files.

```
   CLENUP
      no arguments
```

8.8.3 FILCLS

WaWa IO system: Close a file opened by FILOPN, taking care of catalog bookkeeping and flush last write buffers if any.

```
FILCLS (LUN)
Inputs:
   LUN       I      Logical unit no. of file to close
```

8.8.4 FILCR

WaWa IO system: Create an associated or scratch non-map file

```
FILCR (NAMS, TYPE, NBLOCK, VER, ERROR)
Inputs:
   NAMS      C*36  NAMESTRING specifying catalog block to which
                   file is associated: NAME,CLASS,CATTYPE,SEQ,VOL,
                   USID.  NAME,CLASS,USID ignored for scratch files.
   TYPE      C*2   Associated file type for non-scratch files
                   Ignored for scratch files
In/out:
   NBLOCK    I      Number of 512-type blocks in file: in requested,
                    out actual
Outputs:
   VER       I      Version number of file created
   ERROR     I      Error code: 0 => ok
                           10 => catalog error
                           12 => map not in catalog
                           14 => no room for another ext. type
                           21 => ZCREAT: file already exists
                           22 => ZCREAT: volume unavailable
                           23 => Disk space unavailable
                           24 => Other create errors
Common: /MAPHDR/ modfified extensively for scratch file create
   a little for associated file
```

8.8.5 FILDES

WaWa IO system: Destroy the file specified by NAMS, TYPE, VER

```
FILDES (NAMS, ASSOC, TYPE, VER, ERROR)
Inputs:
   NAMS      C*36  NAMESTRING specifying catalog block to which
                   file is associated: NAME,CLASS,CATTYPE,SEQ,VOL,
                   USID.  NAME,CLASS,USID ignored for scratch files.
   ASSOC     L     File is an associated file,i.e. not cataloged
                   ASSOC will be taken as FALSE if NAMS(8)='SCxx'
   TYPE      C*2   Associated file type; ignored if ASSOC is false
   VER       I     Associated file version; ignored if ASSOC is fal
Outputs:
   ERROR     I      Error code:  0 => o.k.
                       10 => catalog error
                       11 => map too busy to destroy
                       12 => map not found in catalog
                       13 => extension file not in catalog
                       25 => other destroy errors
```

8.8.6 FILIO

WaWa IO system: Read or Write a single record from/to a non-map file which has been opened with FILOPN (256 integers). Adds a 'READ' status to catlg on first call.

```
FILIO (OP, LUN, REC, DATA, ERROR)
Inputs:
   OP          C*4      READ or WRIT
   LUN         I        File Logical Unit Number
   REC         I        Which record out of file (1-relative)
In/Out:
   DATA(256)   I        Data record to input or output
Output:
   ERROR       I        Error return from ZFI3
                        0 => o.k.
                        1 => file not open
                        2 => input error e.g. file not opened for
                             desired operation
                        3 => i/o error
                        4 => end of file
                        5 => beginning of medium
                        6 => end of medium (from IO system)
                       10 => catalog error
```

8.8.7 FILNUM

WaWa IO system: find the FILTAB entry for a file

```
FILNUM (LUN, IFIL, ERROR)
Inputs:
   LUN     I       Logical unit number of file
Outputs:
   IFIL    I       Entry number (2nd subscript to FILTAB)
```

```
   ERROR   I       Error code: 0 => ok, 1 => file not open
```

8.8.8 FILOPN

WaWa IO system: Open the file specified by NAMS and associate it with Logical Unit number LUN.

```
FILOPN (LUN, NAMS, ASSOC, TYPE, VER, ERROR)
Inputs:
   LUN       I     Logical Unit Number
   ASSOC     L     Fils is an associated file,i.e. not cataloged
                   ASSOC will be taken as FALSE if NAMS(8)='SCxx'
   TYPE      C*2   Associated file type; ignored if ASSOC is false
   VER       I     Associated file version; ignored if ASSOC is fal
In/Out:
   NAMS      C*36  NAMESTRING specifying catalog block to which
                   file is associated: NAME,CLASS,CATTYPE,SEQ,VOL,
                   USID.  NAME,CLASS,USID ignored for scratch files.
Outputs:
   ERROR     I     Error code:  0 => o.k.
                        2 => input error: bad or in use LUN
                       10 => catalog error
                       12 => map not found
                       13 => extension file not in catalog
                       14 => no room in FILTAB
                       22 => volume not available
                       26 => open error
```

8.8.9 GETHDR

WaWa IO system: Retrieve the catalog header block for a file that is already open (via FILOPN or OPENCF)

```
GETHDR (LUN, CAT, ERROR)
Inputs:
   LUN            I      Logical Unit No. of file
Outputs:
   CAT(256)       I      Returned Header block
   ERROR          I      Error code: 0 => ok
                                     1 => file not open
                                    10 => catlg error
```

8.8.10 HDRINF

WaWa IO system: Return a number of items from the header block of an open, cataloged file.

```
HDRINF (LUN, WTYPE, SITEM, NITEM, OUTPUT, ERROR)
Inputs:
   LUN       I       Logical Unit No. of file
   WTYPE     I       Data type: 1 = I,   2 = R   3 = D   6 = C*8
   SITEM     I       Index # of 1st item wanted, indexed in a
                     system appropriate to WTYPE (R   for C*8)
   NITEM     I       Number of items requested
Outputs:
   OUTPUT(*) ???     Array into which items go
   ERROR     I       Error code: 0 => ok
```

```
                                    1 => file not open
                                    2 => nonsense input parms
                                   10 => catlg read error
Common /MAPHDR/ receives the header read from catlg file
```

8.8.11 H2WAWA

WaWa IO system: packs AIPS adverb values (Holleriths, floating points) into a WaWa IO Namestring having format A12, A6, A2, I7, I2, I7 for NAME, CLASS, PTYPE, SEQ, VOL, USID

```
H2WAWA (NAME, CLASS, SEQ, PTYPE, VOL, USID, NAMEST)
Inputs:
   NAME    H(3)   file name
   CLASS   H(2)   file class (6 chars)
   SEQ     R      file sequence number
   PTYPE   H      file physical type (2 chars)
   VOL     R      file disk number
   USID    R      user number
Output:
   NAMEST  C*36   WaWa Namestring
```

8.8.12 IOSET

This routine initializes the I/O tables; calls ZDCHIN; allocates buffer space for map I/O to 5 files adequate for 2048 real or 1024 complex pixels per line.

```
  IOSET

  no calling arguments
```

8.8.13 MAPCR

WaWa IO system: Create and catalog a map whose catalog description is defined by the namestring NAMS, and whose size is specified by the KIDIM and KINAX parameters in the Header.

```
MAPCR (ONAMS, NAMS, HDR, ERROR)
Inputs:
   ONAMS  C*36    Namestring of related "input" file - must be
                  complete and correct; used to complete defaults
                  in NAMS (typically the input file namestring).
In/Out:
   NAMS   C*36    Namestring NAME:CLASS:TYPE:SEQ:VOL:USID of map to
                  be created; can contain blanks, wildcards...
   HDR    I(256)  Catalog header for map, containing enough info to
                  define size.  The updated header is returned for
                  real images, not SC files
Outputs:
   ERROR  I       Error code: 0 => ok
                             10 => catalog error
                             14 => no room in catalog
                             21 => file already exists
                             23 => create error
```

8.8.14 MAPIO

WaWa IO system: Do I/O from a file opened using FILOPN to area DATA

```
MAPIO (OP, LUN, DATA, ERROR)
Inputs:
   OP            C*4     'READ' or 'WRIT'
   LUN           I       File logical unit no.
Input/output:
   DATA(*)       R       Data in or out
Output:
   ERROR         I       Error code: 0 => ok
                             1 => file not open
                             2 => bad input parms
                           3-6 => IO errors
                             8 => Bad data type (ie write integers)
                             9 => IO is complete (software
                                  generated EOF)
                            10 => catalog read/write error
                            11 => Catalog status error
```

8.8.15 MAPMAX

WaWa IO system: Determine max and min of a map opened by FILOPN and update CAT block accordingly

```
MAPMAX (LUN, XMAX, XMIN, ERROR)
Inputs:
   LUN     I   Logical Unit No. of map
Outputs:
   XMAX    R   Maximum in map
   XMIN    R   Minimum
   ERROR   I   Error codes: 0 => ok
                            1 => file not open
                            2 => input parms error
                          3-6 => IO errors
                           10 => catalog read/... error
```

8.8.16 MAPWIN

WaWa IO system: Set or reset parameters for a window on MAP I/O File must be opened first with FILOPN.

```
MAPWIN (LUN, BLC, TRC, ERROR)
Inputs:
   LUN     I       Logical Unit No. of file (must be open)
   BLC     R(7)    Lower bounds of map subrectangle
   TRC     R(7)    Upper bounds of map subrectangle
Outputs:
   ERROR   I       Error codes: 0 => ok
                        1 => file not open
                       10 => catalog error
                       16 => bad window specification

                       17 => partial row specified on write.
```

8.8.17 MAPXY

WaWa IO system: Set windows so that MAPIO returns a subrectangle of the top plane of a map

```
MAPXY (LUN, WIN, ERROR)
Inputs:
   LUN     I       Logical Unit No. of an open map
   WIN     R(4)    Corners of rectangle.  If WIN(1)=0.0, whole top
                   plane is taken.
Output:
   ERROR   I       As returned by MAPWIN
```

8.8.18 OPENCF

WaWa IO system: Open a MAIN (i.e. Cataloged) file and associate it with Logical Unit Number LUN.

```
OPENCF (LUN, NAMS, ERROR)
Inputs:
   LUN         I       Logical Unit No.
In/out:
   NAMS        C*36    Catalog identification NAMESTRING:
                       NAME:CLASS:PTYPE:SEQN:VOL:USID
                       NAME, CLASS, & USID ignored if PTYPE = 'SC'.
OUTPUTS:
   ERROR       I       Error codes: 0 => o.k.
                       Otherwise as returned by FILOPN
```

8.8.19 SAVHDR

WaWa IO system: Save the catalog header block for a file that is already open (via FILOPN or OPENCF)

```
SAVHDR (LUN, CATBLK, ERROR)
Inputs:
   LUN       I         Logical Unit No. of file
   CATBLK    I(256)    Saved Header block
Outputs:
   ERROR     I         Error code: 0 => ok
                          1 => file not open
                          10 => catlg error
```

8.8.20 TSKBEG

WaWa IO system: Do most of the operations necessary to begin a task: Calls IOSET, calls GTPARM to get parameters, and, if appropriate, calls RELPOP. For ≤ 5 simultaneously open map files. You should end with TSKEND.

```
TSKBEG (PRGNAM, NPARM, RPARM, ERROR)
Inputs:
   PRGNAM    C*6     Task name
   NPARM     I       No. of real parameters passed by AIPS
OUTPUTS:
   RPARM     R(*)    Array to receive passed parameters
   ERROR     I       Error return: 0 => Okay
                        0 < ERROR <10  => Error return from GTPARM
```

8.8.21 TSKEND

WaWa IO system: Terminate a task, including calls to CLENUP and RELPOP if appropriate. Also close down messages.

```
TSKEND (IRET)
Inputs:
   IRET      I      A return code passed to AIPS if task was run
                    in wait mode: 0 => ok, else => failure.
```

8.8.22 UNSCR

WaWa IO system: Destroy all scratch files created by this task.

```
UNSCR
   no arguments
```

8.8.23 WAWA2A

WaWa IO system: unpacks Wawa-IO Namestring having format A12, A6, A2, I7, I2, I7 for NAME, CLASS, PTYPE, SEQ, VOL, USID into component parts

```
WAWA2A (NAMEST, NAME, CLASS, SEQ, PTYPE, VOL, USID)
Input:
   NAMEST  C*36   WaWa Namestring
Outputs:
   NAME    C*12   file name
   CLASS   C*6    file class
   SEQ     I      file sequence number
   PTYPE   C*2    file physical type
   VOL     I      file disk number
   USID    I      user number
```

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.

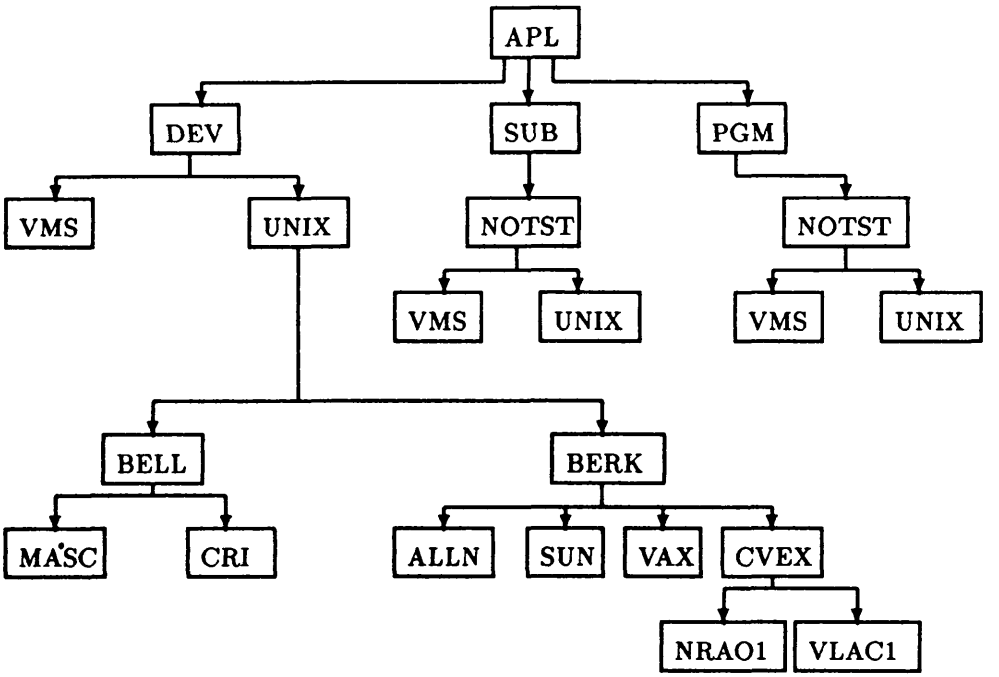
2. Utility routines — library subroutines that may call device-specific routines, but are themselves device independent. A non-standard area is also provided.

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.

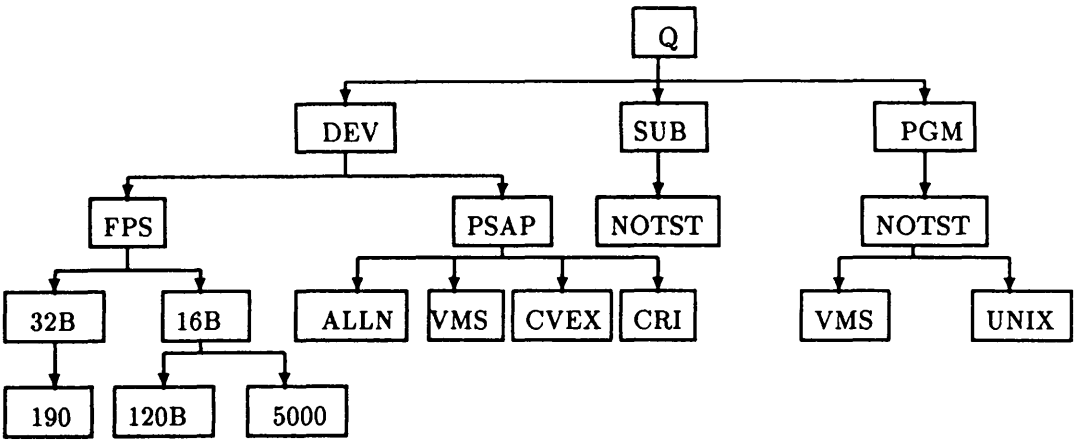


APL Directories

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.

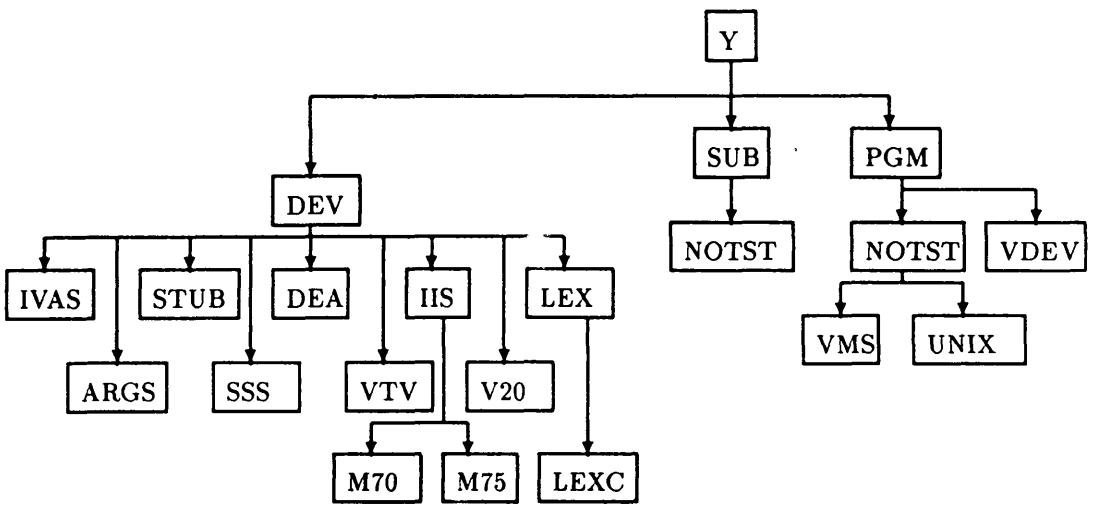


Q Directories

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.

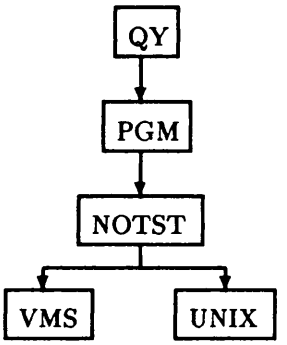


Y Directories

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

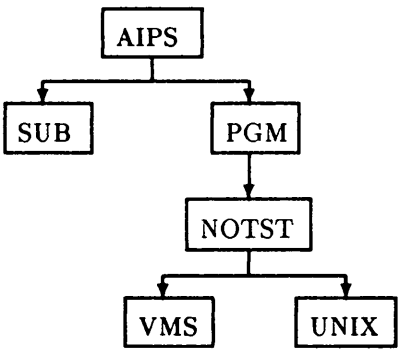
This area is for routines and programs that make use of the both the AP and TV. At present, this only occurs at the program level, so this tree is very simple.



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.

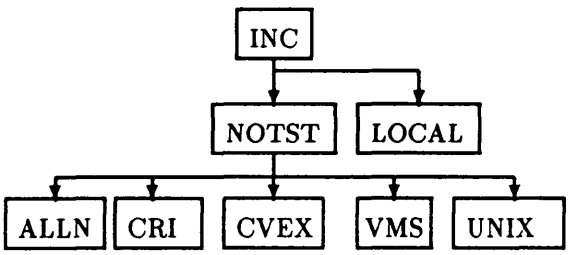


AIPS Directories

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

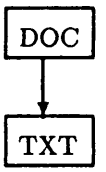
This area is for load modules, i.e., fully linked programs in a form ready to be run. This is split into a standard LOAD area and a few alternative areas immediately below (e.g., LOAD.ALT1). These alternate areas could, for example, be used to keep pseudo-AP versions of programs or versions linked for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

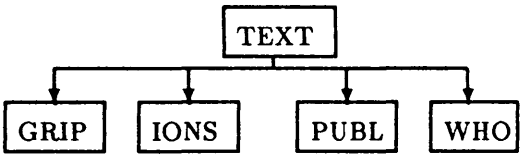
The DOC area is used to store documentation files (this manual, other coding descriptions) in DOCTXT. The directory structure is simple:



Documentation Directories

Text files

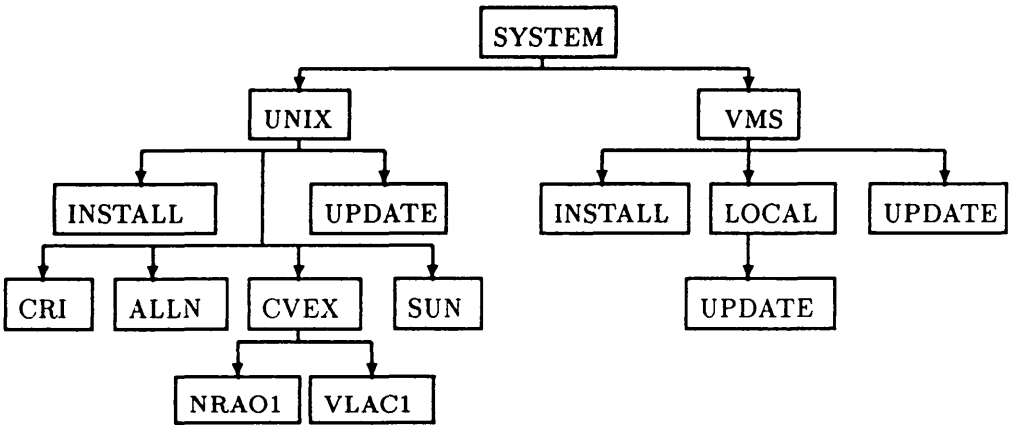
There are a number of directories which reside above the version specific portion of the AIPS directory tree. These include AIPSPUBL containing the *Cookbook* text, AIPSGRIP containing the AIPS gripe system, AIPSIONS containing ionospheric monotoring data and AIPSWHO containing mailing lists etc. The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

Appendix B

Shopping lists

B.1 Introduction

This appendix contains the one line descriptions of each of the AIPS system subroutines that may be called from applications software arrainged by category (a given routine may have several entries). Not all of the subroutines described in the following lists may be called for all applications software. In particular, routines in directory AIPSUB may only be called from program AIPS or other tasks in the AIPPGM directory. Z2 and Y3 routines may only be called from other "Z" or "Y" routines.

This list should simplify finding the appropriate routine inside the AIPS system. Each routine name is prefixed with the logical name of the directory in which it resides. A summary of the categories is given below.

- AP-APPL These are routines that use "Array Processor" routines for a particular operation.
- AP-FFT These are routines that use "Array Processor" routines for FFT (Fast Fourier Transform) operation.
- AP-UTIL These are utility routines that use "Array Processor" routines.
- BATCH These routines are related to AIPS batch functions.
- BINARY These routines process external binary format data.
- CALIBRATION These routines are related to the calibration package of routines.
- CATALOG These routines are related to the AIPS catalog.
- CHARACTER There routines are AIPS character manipulating functions.
- COORDINATES These routines manipulate astronomical coordinate systems.
- EXT-APPL These are applications routines for extension files; generally tables.
- EXT-UTIL These are utility routines for extension files.
- FITS These routine are for processing data in FITS files.
- GRAPHICS These are the AIPS graphics routines.
- HEADER These routines process AIPS catalog header records.
- HISTORY These routines process AIPS history files or records.
- IO-APPL These are applications routines for the AIPS I/O system.
- IO-BASIC These are the basic routines for the AIPS I/O system.

- IO-TV These are the routines that communicate with the image display.
- IO-UTIL These are utility routines for the AIPS I/O system.
- IO-WAWA These are the "WAWA" or "Easy IO" package of routines.
- MAP These routines deal with images.
- MAP-UTIL These are utility routines dealing with images.
- MATH These are basic mathematical routines.
- MESSAGES These routines deal with sending messages to the user.
- MODELING These routine involve model fitting or calculation.
- PARSING These routine involve parsing information from character strings.
- PLOT-APPL These are applications plotting routines.
- PLOT-UTIL These are utility plotting routines.
- POPS-APPL These are POPS applications routines (verbs).
- POPS-LANG These are parts of the POPS language processor.
- POPS-UTIL These are POPS utility routines.
- PRINTER These are routines related to printers.
- SDISH These are routines for processing single dish data.
- SERVICE These are various service routines.
- SLICE These are routines that deal with slices through images.
- SORT These are sorting routines.
- SPECTRAL These are routine related to spectroscopy.
- SYSTEM These are AIPS system functions.
- TAPE These are routines related to reading tape or other external binary files.
- TERMINAL These are routines for I/O to user terminals
- TEXT These are routines related to text files.
- TV These are routines related to the image display.
- TV-APPL These are applications routines related to the image display.
- TV-BASIC These are basic image display routines.
- TV-IO These are I/O applications routines related to the image display.
- TV-UTIL These are utility routines related to the image display.
- UTILITY These are general utility routines.
- UV These routines deal with uv (interferometer) data.
- UV-UTIL These utility routines deal with uv (interferometer) data.
- VLA These are routines that are specific to the VLA (NRAO Very Large Array)

- Y0 These are the main top level TV (image display) routines.
- Y1 These are the second level TV routines.
- Y2 These are the IIS specific TV routines; these are unlikely to be supported on othe displays.
- Y3 These are "Y" routines that can only be called from other "Y" routines.
- Z These are routines which may contain system dependent functions.
- Z-2 These are routines which may contain system dependent functions but may only be called from other "Z" routines.

B.1.1 AP-APPL

```
QNOT:ALGSUB.FOR      Interpolates model visibility grom a grid and subtracts from uv data.
QNOT:APCONV.FOR      Disk based 2-D convolution using FFTs.
QNOT:CONV.FOR        *TESS routine: Convolve a map with a beam.
QNOT:DISPTV.FOR      *TESS routine: Display an image on a TV
QPSAP:Q1FIN.FOR      Finish gridding a row of uv data.
QPSAP:Q1GRD.FOR      Grid a uv data.
QPSAP:QBAKSU.FOR     Back substitution.
QPSAP:QBOXSU.FOR     Boxcar sum of a vector.
QPSAP:QCLNSU.FOR     Low level Clark CLEAN routine.
QPSAP:QCRVMU.FOR     Complex-real vector multiply.
QPSAP:QCSQTR.FOR     inplace transpose of square, complex matrix.
QPSAP:QCTLUT.FOR     Initialize cosine lookup table etc.
QPSAP:QCVCMU.FOR     Scalar complex times conjugate of vector to real.
QPSAP:QCVCON.FOR     Complex conjugate of a vector.
QPSAP:QCVEXP.FOR     Vector complex exponential.
QPSAP:QCVJAD.FOR     Complex vector conjugate of vector add.
QPSAP:QCVMAG.FOR     Complex vector magnitude squared.
QPSAP:QCVMMA.FOR     Max. square of modulus of complex vector.
QPSAP:QCVMOV.FOR     Complex vector move.
QPSAP:QCVMUL.FOR     Complex vector multiply.
QPSAP:QCVSDI.FOR     Divide weighted complex vector by complex scalar.
QPSAP:QCVSMS.FOR     Subtract real vector*complex scalar from vector.
QPSAP:QDIRAD.FOR     Directed vector add.
QPSAP:QFINGR.FOR     Finish gridding row of uv data.
QPSAP:QGADIV.FOR     Divide Gaus. model vis. into uv data.
QPSAP:QGASUB.FOR     Subtract Gaus. model vis. from uv data.
QPSAP:QGET.FOR       Move data from pseudo-AP memory to "host".
QPSAP:QGRD1.FOR      Convolves visibility data onto a grid.
QPSAP:QGRD2.FOR      Convolves linear polarization data onto a grid.
QPSAP:QGRD3.FOR      Convolve visibility data onto a grid.
QPSAP:QGRD4.FOR      Convolves visibility data onto a grid.
QPSAP:QGRDCC.FOR     Grid and FT Clean components.
QPSAP:QGRDFI.FOR     Finish griding a row of uv data.
QPSAP:QGRDMI.FOR     Combined complex vector in gridding uv data.
QPSAP:QGRID.FOR      Grid uv data into row.
QPSAP:QGRIDA.FOR     Grid visibility data.
QPSAP:QHIST.FOR      Make histogram of a vector.
QPSAP:QINT.FOR       Interpolates model visibilities from a grid.
QPSAP:QINTP.FOR      Interpolates model visibilities from a grid.
QPSAP:QLVGT.FOR      Vector logical greater than.
QPSAP:QMAKMS.FOR     Make mask depending on vector, scalar comparison.
```

```
QPSAP:QMAXMI.FOR      Find maximum and minimum of a vector.
QPSAP:QMAXV.FOR       Find maximum value element of a vector.
QPSAP:QMCALC.FOR      Compute model visibility from point model.
QPSAP:QMENT.FOR       MEM routine
QPSAP:QMINV.FOR       Find minimum value element of a vector
QPSAP:QMTRAN.FOR      matrix transpose.
QPSAP:QMTYP.FOR       Chose DFT or gridded interpolation method.
QPSAP:QMULCL.FOR      High level Clark CLEAN routine
QPSAP:QPHSRO.FOR      Add phase gradient to a complex array.
QPSAP:QPOLAR.FOR      Vector rectangular-to-polar conversion.
QPSAP:QPTDIV.FOR      Divide point model visibility into uv data.
QPSAP:QPTFAZ.FOR      Compute phase in model visibilities.
QPSAP:QPTSUB.FOR      Subtract point model visibility from uv data.
QPSAP:QRECT.FOR       Vector polar-to-rectangular conversion.
QPSAP:QRFT.FOR        Does real, inverse FT with arbitrary spacing.
QPSAP:QSEARC.FOR      VLBI fringe search with FFT.
QPSAP:QSPDIV.FOR      Divide Gaussian model  visibility into uv data.
QPSAP:QSPSUB.FOR      Subtract Gaussian model visibility from uv data.
QPSAP:QSVE.FOR        Sum the elements of a vector.
QPSAP:QSVESQ.FOR      Sum the squares of the elements of a vector.
QPSAP:QUVIN.FOR       Interpolate visibility model from a grid.
QPSAP:QUVINT.FOR      Interpolate model visibility from grid.
QPSAP:QVABS.FOR       Vector absolute value.
QPSAP:QVADD.FOR       Vector add.
QPSAP:QVCLIP.FOR      Vector clip.
QPSAP:QVCLR.FOR       Vector zero.
QPSAP:QVCOS.FOR       Vector cosine.
QPSAP:QVDIV.FOR       Vector divide.
QPSAP:QVEXP.FOR       Vector exponentiate.
QPSAP:QVFILL.FOR      Vector fill.
QPSAP:QVFIX.FOR       Vector fix.
QPSAP:QVFLT.FOR       Vector float.
QPSAP:QVIDIV.FOR      Divide a vector by the product of two integers.
QPSAP:QVINDE.FOR      Vector index (gather)
QPSAP:QVLN.FOR        Vector natural logrithm
QPSAP:QVMA.FOR        Vector multiply and vector add.
QPSAP:QVMOV.FOR       Vector move.
QPSAP:QVMUL.FOR       Vector multiply.
QPSAP:QVNEG.FOR       Negate the elements of a vector.
QPSAP:QVRVRS.FOR      Reverse the elements of a vector.
QPSAP:QVSADD.FOR      Vector scalar add.
QPSAP:QVSIN.FOR       Vector sine.
QPSAP:QVSMA.FOR       Vector scalar multiply and vector add.
QPSAP:QVSMAF.FOR      Scalar multiply and and round.
QPSAP:QVSMSA.FOR      Vector scalar multiply and scalar add.
QPSAP:QVSMUL.FOR      Vector scalar multiply.
QPSAP:QVSQ.FOR        Square vector.
QPSAP:QVSQRT.FOR      Vector square root.
QPSAP:QVSUB.FOR       Vector subtract.
QPSAP:QVSWAP.FOR      Vector swap.
QPSAP:QVTRAN.FOR      Inplace transpose of a matrix of vectors.
QPSAP:QVTSMU.FOR      Vector table scalar multiply.
QPSAP:QXXPTS.FOR      Subtract point model visibility from uv data.
QMOT:VISDFT.FOR       Compute DFT of model and subtract/divide from/into uv data.
```

B.1.2 AP-FFT

APLSUB:AP2SIZ.FOR	returns largest power of 2 not exceeding 1024 times first argument
QSUB:APXPOS.FOR	In place transpose of complex array.
QNOT:CONV1.FOR	First of four routines to convolve two real images.
QNOT:CONV2.FOR	Second of four routines to convolve two real images.
QNOT:CONV3.FOR	Third of four routines to convolve two real images.
QNOT:CONV4.FOR	Fourth of four routines to convolve two real images.
APLNOT:DSKFFT.FOR	2-D disk based FFT using AP.
APLNOT:EMPTY1.FOR	DSKFFT utility routine
APLNOT:EMPTY2.FOR	DSKFFT utility routine
QNOT:FFTIM.FOR	FFTs an image for uv interpolation.
APLNOT:FILL1.FOR	DSKFFT utility routine
APLNOT:FILL2.FOR	DSKFFT utility routine.
QNOT:MAKMAP.FOR	Makes image or beam from uv data set.
APLSUB:MINSK.FOR	Inits use of MSKIP to read noncontiguous, evenly spaced rows in a map
APLSUB:MSKIP.FOR	Reads noncontiguous, but evenly spaced rows in a map (see also MINSK)
QSUB:PASS1.FOR	First of two routines to FFT an image file.
QSUB:PASS2.FOR	Second of two routines to FFT an image file.
QPSAP:QCFFT.FOR	Complex 1-D FFT.
QPSAP:QRFFT.FOR	Real-half plane complex FFT

B.1.3 AP-UTIL

QNOT:APIO.FOR	Copies image-like data between disk and "AP memory".
QSUB:APROLL.FOR	Copies AP "memory" to disk, gives up AP then reloads AP
QNOT:CCSGRD.FOR	Transforms CLEAN components to a grid.
QNOT:CONVFN.FOR	Computes convolving fn. kernels and stores them in "AP memory"
QNOT:GRDCOR.FOR	Normalizes and corrects image for gridding convolution fn.
QNOT:GRDCRM.FOR	Loads CLEAN components into AP for uv model computation.
QNOT:GRDSUB.FOR	Subtracts transform of CLEAN components from uv data.
QNOT:GRDTAB.FOR	Computes Fourier transform of gridding convolution function.
QNOT:INTPFN.FOR	Computes interpolation kernals and put them into "AP memory".
QNOT:MAKMAP.FOR	Makes image or beam from uv data set.
QPSAP:QGSP.FOR	Read "S-pad" register
QPSAP:QINIT.FOR	Initialize "AP".
QPSAP:QPUT.FOR	Move data from "host" to "AP" memory.
QPSAP:QRLSE.FOR	Release "AP".
QSUB:QROLL.FOR	Determines if time to roll AP, if so calls APROLL.
QPSAP:QWAIT.FOR	Suspend host until AP done.
QPSAP:QWD.FOR	Suspend host until AP data transfer done.
QPSAP:QWR.FOR	Suspend host until AP computations complete.
QNOT:UVGRID.FOR	Grids uv data to be FFTed.
QNOT:UVMDIV.FOR	Divides a uv data set by the Fourier transform of a model.
QNOT:UVMSUB.FOR	Subtracts the Fourier transform of a model from a uv data set.
QNOT:UVMTYP.FOR	Determines relative CPU times for DFT or gridded interpolation.
QNOT:UVTBGD.FOR	Grids uv data in arbitrary sort order to be FFTed.
QNOT:UVTBUN.FOR	Determines and applies uniform weighting to uv data in arb. order.
QNOT:UVUNIF.FOR	Determines and applies uniform weighting to a uv data set.

B.1.4 BATCH

AIPSUB:AUA.FOR	verb to submit batch jobs to AIPSC and the QMNGR queues
AIPSUB:AUB.FOR	verbs to prepare, edit, and review batch jobs and queues
APLSUB:BATPRT.FOR	prints header/trailer messages for printer tasks when run in batch

```
APLSUB:BATQ.FOR       performs operations on batch queue control file such as OPEN RUN CLOS
AIPSUB:BBUILD.FOR     reads input lines and adds them to the text file for a batch job
```

B.1.5 BINARY

```
APLGEN:ZBYMOV.FOR     move 8-bit bytes from in-buffer to out-buffer
APLGEN:ZBYTFL.FOR     interchange bytes in buffer if needed to go between local & standard
APLGEN:ZC8CL.FOR      convert packed ASCII buffer to local character string
APLGEN:ZCLC8.FOR      convert local character string to packed ASCII buffer
APLGEN:ZDHPRL.FOR     convert 64-bit HP floating buffer to local DOUBLE PRECISION values
APLGEN:ZGETCH.FOR     get a character from a REAL word
APLGEN:ZI16IL.FOR     convert FITS-standard 16-bit integers to local integers
APLGEN:ZI32IL.FOR     convert FITS-standard 32-bit integers from buffer into local integers
APLGEN:ZI8IL.FOR      convert 8-bit unsigned integers in buffer to local integers
APLGEN:ZILI16.FOR     convert local integers to 16-bit FITS integers in a buffer
APLGEN:ZILI32.FOR     convert local integer into FITS-standard 32-bit integers
APLGEN:ZPUTCH.FOR     inserts 8-bit "character" into a word
APLGEN:ZR32RL.FOR     convert 32-bit IEEE floating buffer to local REAL values
APLGEN:ZR64RL.FOR     convert 64-bit IEEE floating-point buffer to local "DOUBLE PRECISION"
APLGEN:ZR8P4.FOR      converts pseudo I*4 to double precision - for tape handling only
APLGEN:ZRDMF.FOR      convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers
APLGEN:ZRHPRL.FOR     convert 32-bit HP floating buffer to local REAL values
APLGEN:ZRLR32.FOR     converts buffer of local REAL values to IEEE 32-bit floating-point
APLGEN:ZRLR64.FOR     convert buffer of local double precision values to IEEE 64-bit float.
APLGEN:ZRM2RL.FOR     convert Modcomp to local single precision floating point
APLGEN:ZUVPAK.FOR     Pack visibility data, 1 correlator per real with magic value blank.
APLGEN:ZUVXPN.FOR     Expands packed visibility data and adds weight
```

B.1.6 CALIBRATION

```
APLNOT:BLGET.FOR      Sets up for interpolation in baseline (BL) table
APLNOT:BLINI.FOR      Create/open/init I/O to BL table
APLNOT:BLREFM.FOR     Checks existence of BL table, changes format if necessary
APLNOT:BLSET.FOR      Fills current baseline calibration table
APLNOT:BPASET.FOR     Sets up the bandpass table array for use by DATBND.
APLNOT:BPGET.FOR      Sets bandpass correction arrays in common
APLNOT:BPINI.FOR      Create/open/initialize bandpass (BP) table
APLNOT:BPREFM.FOR     Checks existence of BP table, changes format if necessary
APLNOT:CALADJ.FOR     Adjusts solution (SN) table phases to a common reference antenna.
APLNOT:CALCOP.FOR     Copies selected uv data with calibration and editing
APLNOT:CALINI.FOR     Creates/opens/initializes calibration (CL) table
APLNOT:CALREF.FOR     Adjusts the reference antenna in an SN table.
APLNOT:CGASET.FOR     Maintains calibration values in an array in common
APLNOT:CHNCOP.FOR     Copies selected portions of the IF table
APLNOT:CHNDAT.FOR     Creates/Opens/Reads/Writes/Closes an IF table.
APLNOT:CLREFM.FOR     Checks existence of CL table, changes format if necessary
APLNOT:CLUPDA.FOR     Concatenates, rereferences, smooths SN tables and applies it to CL.
APLNOT:CMPARM.FOR     Determines blocks of data in a vis. record to decompress
APLNOT:CSINI.FOR      Create/Open/Init Single dish calibration (CS) table
APLNOT:CSLGET.FOR     Reads CL (or SN) table and sets up for interpolation.
APLNOT:DATBND.FOR     Applies the bandpass correction to data.
APLNOT:DATCAL.FOR     Applies calibration to data
APLNOT:DATFLG.FOR     Flags data specified in flagging table
APLNOT:DATGET.FOR     Reads, selects, calibrates and edits data.
```

```
APLNOT:DATPOL.FOR     Apply polarization corrections to data.
APLNOT:DCALSD.FOR     Apply Single dish calibration to data.
APLNOT:DGETSD.FOR     Reads, selects single dish data, calibrates and edits.
APLNOT:DGGET.FOR      Selects uv data and changes Stokes
APLNOT:DGHEAD.FOR     Fills output CATBLK for UVGET
APLNOT:DGINIT.FOR     Sets arrays for selecting data and changing Stokes
APLNOT:FLAGUP.FOR     Updates the Flag (FG) table.
APLNOT:FLGINI.FOR     Create/Open/Init Flag (FG) table.
APLNOT:FLGSTK.FOR     Set Stokes flag for uv flagging.
APLNOT:FNDSOU.FOR     Find source numbers for a list of sources.
APLNOT:FQINI.FOR      Create/open/initialize frequency (FQ) table
APLNOT:FQMATC.FOR     Check if selection criteria match FQ table entries.
APLNOT:GACSIN.FOR     Initializes CS file, and prepares table to be applied.
APLNOT:GAINI.FOR      Creates and initializes gain (GA) extension tables.
APLNOT:GAININ.FOR     Initializes calibration table for application.
APLNOT:GETFQ.FOR      Find info on a given frequency id.
APLNOT:GETSOU.FOR     Find info on a given source id.
APLNOT:INDXIN.FOR     Initializes index (NX) file, finds first scan selected.
APLNOT:IOBSRC.FOR     Search for antennas in the current bandpass buffer.
APLNOT:LXYPOL.FOR     Fills polarization correction table for AT like linear polarization.
APLNOT:MULSDB.FOR     Determines if a uv file is multi- or single- source.
APLNOT:NDXINI.FOR     Create/open/init index (NX) table
APLNOT:NXTFLG.FOR     Manages flagging info in tables in common.
APLNOT:PARANG.FOR     Computes antenna parallactic angles
APLNOT:POLSET.FOR     Fills polarization correction table from info in AN table.
APLNOT:SCINTP.FOR     Interpolates bandpass tables in time.
APLNOT:SCLOAD.FOR     Copies part of one bandpass scratch file to another for efficiency.
APLNOT:SDCGET.FOR     Sets up to interpolate in Single dish calibration (CS) table.
APLNOT:SDCSET.FOR     Interpolates single dish calibration data for current time.
APLNOT:SDGET.FOR      Reads single dish data with optional calibration and flagging
APLNOT:SELINI.FOR     Initialize data selection and control in commons in DSEL.INC
APLNOT:SELSMG.FOR     Selects calibrator data, smooths solutions.
APLNOT:SET1VS.FOR     Sets up pointer and weights arrays for selecting uv data.
APLNOT:SETSM.FOR      Determines type of spectral smoothing and sets up look up table.
APLNOT:SETSTK.FOR     Sets STOKES parameters correctly for plotting routines
APLNOT:SMOSP.FOR      Convolves a spectrum with a tabulated function.
APLNOT:SN2CL.FOR      Apply an SN to a CL table.
APLNOT:SNAPP.FOR      Append SN tables and keep track of reference antennas.
APLNOT:SNINI.FOR      Create/open/initialize solution (SN) tables.
APLNOT:SNREFM.FOR     Checks existence of SN table, changes format if necessary
APLNOT:SNSMO.FOR      Smooths solution (SN) tables
APLNOT:SOUELV.FOR     Computes source hour angles and elevations
APLNOT:SOUFIL.FOR     Fills in arrays of source numbers to be included or excluded.
APLNOT:SOURNU.FOR     Look up source numbers for a list of names.
APLNOT:TABBL.FOR      Do IO to Baseline (BL) table after setup by BLINI.
APLNOT:TABBP.FOR      Does I/O to bandpass (BP) table opened by BPINI
APLNOT:TABCAL.FOR     Does I/O to Calibration (CL) table opened by CALINI
APLNOT:TABCS.FOR      Does I/O to single dish calibration (CS) table opened by CSINI
APLNOT:TABFLG.FOR     Does I/O to Flag (FG) table opened by FLGINI
APLNOT:TABFQ.FOR      Does I/O to frequency (FQ) table opened by FQINI
APLNOT:TABGA.FOR      Does I/O to GAIN (GA) table opened by GAINI
APLNOT:TABNDX.FOR     Does I/O to Index (NX) table opened by NDXINI
APLNOT:TABSN.FOR      Does I/O to Solution (SN) table opened by SNINI
APLNOT:TABSOU.FOR     Does I/O to Source (SU) table opened by SOUINI
```

APLNOT:TABTY.FOR	Does I/O to Tsys (TY) table opened by TYINI
APLNOT:TYINI.FOR	Create/open/initialize Tsys (TY) table
APLNOT:UVGET.FOR	Read UV data with optional calibration, editing, selection, etc.
APLNOT:VISCNT.FOR	Determines number of visibility records requested of UVGET

B.1.7 CATALOG

AIPSUB:AU3.FOR	Verbs to display contents of catalogs and headers: CATA, IMHE ...
AIPSUB:AU7.FOR	Verbs to print history, rescale image, alter axis descriptions
AIPSUB:AU8.FOR	Verbs to get or clear name adverbs, destroy extension files
AIPSUB:CATCR.FOR	Create and initialize catalog (CA) files
APLSUB:CATDIR.FOR	Manipulates the catalog directory: OPEN, CLOS, various SRCHs, ...
APLSUB:CATIME.FOR	Stores current, or recovers previous, date and time in packed format
APLSUB:CATIO.FOR	Reads/writes header blocks in the catalog file
APLSUB:CATKEY.FOR	Reads/writes the Keyword section of an AIPS header file
AIPSUB:CATLST.FOR	List the contents of the catalog directory file
APLSUB:CATOPN.FOR	Opens the catalog directory file and returns its size
APLSUB:CHSTAT.FOR	Changes numeric code used to record the status of the catalog entry
APLSUB:CHWMAT.FOR	Matches a pattern string having wild-card chars with a test string
AIPSUB:DESCR.FOR	Destroys all scratch files for tasks which are no longer active
APLSUB:HDRBUF.FOR	Translates AIPS header to/from FITS-standard integer form
APLSUB:ICOPEN.FOR	Opens image catalog for the specified image plane (call from Y only)
APLSUB:IMA2MP.FOR	Converts pixel numbers in a TV-image into real image pixels
APLSUB:MADDEX.FOR	Adds extension file to catalog header
APLSUB:MAKOUT.FOR	Convert input and output names to actual output names in standard way
APLSUB:MAPCLR.FOR	Clears status flags in catalog and deletes lists of files
APLSUB:MAPCLS.FOR	Closes cataloged file, updating header and catalog status if needed
APLSUB:MAPOPN.FOR	Open file pointed to by catalog entry and mark the entry busy
APLSUB:MCREAT.FOR	Create and catalog a map file
APLSUB:MDESTR.FOR	Deletes a catalog entry and all files assocated with it
APLSUB:MP2IMA.FOR	Convert image pixel positions to TV pixel positions
APLSUB:NXTMAP.FOR	Opens next catalog entry matching the input parameters
APLSUB:PSFORM.FOR	Analyses a wild-card string, preparing an array for pattern matching
AIPSUB:RENUMB.FOR	Renumbers an entry in the catalog (CA) file
APLSUB:STXT.FOR	Translates catalog status code into a character string
APLSUB:TKCATL.FOR	Performs operations on the Graphics image catalog
APLSUB:UVCREA.FOR	Create and catalog a uv data base file

B.1.8 CHARACTER

APLSUB:CH2NUM.FOR	converts string containing an integer in ASCII form into the integer
APLSUB:CNBLNK.FOR	returns position of first non-blank character in portion of string
APLSUB:CNCOMP.FOR	compares two HOLLERITH strings
APLSUB:CHCOPY.FOR	moves characters from one HELLERITH string to another
APLSUB:CHFILL.FOR	fills portion of HOLLERITH string with a specified character
APLSUB:CHLTOU.FOR	converts a CHARACTER string to all upper case letters
APLSUB:CHMATC.FOR	searches one HOLLERITH string for the occurrence of another
APLSUB:CHR2H.FOR	converts a Fortran CHARACTER variable to an AIPS HOLLERITH string
APLSUB:CHWMAT.FOR	matches a pattern string having wild-card chars with a test string
APLSUB:FILZCH.FOR	replaces blank characters with
APLSUB:H2CHR.FOR	convert AIPS Hollerith string to Fortran CHARACTER variable
APLSUB:IFPC.FOR	returns the number of HOLLERITH locations needed to hold N characters
APLSUB:ITRIM.FOR	returns length of CHARACTER variable to last non-blank
APLSUB:JTRIM.FOR	clears nulls, returns length of CHARACTER variable to last non-blank

APLSUB:NAMEST.FOR	packs image name in string with leading and trailing blanks removed
APLSUB:PSFORM.FOR	analyses a wild-card string, preparing an array for pattern matching
APLSUB:SPFIL.FOR	fills HOLLERITH string with blanks beginning at first null
APLSUB:STLTOU.FOR	converts any characters beween single quotes to upper case
APLSUB:TRIM.FOR	removes leading and trailing blanks, returns actual length of string
APLSUB:UNPACK.FOR	converts a packed character buffer into one with 1 character/integer

B.1.9 COORDINATES

APLNOT:ATFPNT.FOR	Routine to calculate X-Y coords from galactic coords
AIPSUB:AU7.FOR	verbs to print history, rescale image, alter axis descriptions
APLSUB:AXSTRN.FOR	encodes axis type and value in a string
APLNOT:BDN.FOR	Computes Besselian day numbers of Julian date.
APLSUB:COORDD.FOR	converts angles between degrees and sexagesimal format
APLSUB:COORDT.FOR	translates between celestial, galactic, and eccliptic coordinates
APLNOT:DA13.FOR	Computes arguments A1, A2 and A3 of the mean motion of the sun.
APLNOT:DA46.FOR	Computes arguments A4, A5 and A6 of the mean motion of the moon
APLNOT:DAPM.FOR	Converts apparent to mean positions.
APLSUB:DIRCOS.FOR	determines direction cosines between ref position and test position
APLSUB:DIRDEC.FOR	finds longitude pixel and latitude given latitude pixel and longitude
APLSUB:DIRRA.FOR	finds latitude pixel and longitude given longitude pixel and latitude
APLNOT:DMAP.FOR	Compute apparent position from mean position
APLSUB:FNDX.FOR	returns X-axis coordinate value given X pixel and Y coordinate value
APLSUB:FNDY.FOR	returns Y-axis coordinate value given Y pixel and X coordinate value
APLNOT:GRD.FOR	Compute the general relativity displacements in RA and DEC.
APLSUB:JABER.FOR	Compute vectors needed for J2000 aberation and GR light bending.
APLSUB:JNUT.FOR	Computes nutation from IAU 1980 series
APLSUB:JPOLAR.FOR	Correct rectangular position for polar motion.
APLSUB:JPRECS.FOR	Precess between apparent and J2000 epoch positions.
APLSUB:JPRENU.FOR	Compute rotation matrix for precession and nutation IAU 1980 series.
APLSUB:LABINI.FOR	initializes commons for labeling of plots (calls SETLOC)
APLSUB:LMPIX.FOR	returns pixel location corresponding to specified coordinates
APLSUB:METSCA.FOR	scale a value to the range 1-999 and provide a metric prefix to match
APLSUB:MP2SKY.FOR	calls SETLOC, XYVAL to convert image pixel to physical coordinates
APLSUB:NEWPOS.FOR	returns astronomical coordinates given direction cosines, projection
APLNOT:NUT2.FOR	Computes nutation in longitude and obliquity for a Julian date.
APLNOT:NUT4.FOR	Computes nutation using a non ridgid earth model
APLNOT:PARANG.FOR	Computes antenna parallactic angles
APLNOT:PRECES.FOR	Convert between mean and apparent positions (B1950 only)
APLSUB:SETLOC.FOR	sets location common for coordinate computations and display
APLSUB:SKY2MP.FOR	calls SETLOC, XYPIX to convert sky coordinates to map pixel locations
APLSUB:SKYFRM.FOR	returns string with character representation of a corrdinate
APLSUB:SLAEVP.FOR	Earth position and motion ephemeris (J2000)
APLSUB:SLBINI.FOR	initializes labeling for slice plots
APLNOT:SOUELV.FOR	Computes source hour angles and elevations
APLSUB:XYPIX.FOR	returns pixel position corresponding to given coordinates
APLSUB:XYVAL.FOR	returns coordinate values corresponding to specified pixel position

B.1.10 EXT-APPL

APLSUB:ANTDAT.FOR	Returns the reference date and frequency for each array in uv dataset
APLSUB:ANTINI.FOR	creates and intializes antenna tables
APLNOT:BLINI.FOR	Create/open/init I/O to BL table
APLNOT:BLREFM.FOR	Checks existence of BL table, changes format if necessary

```
APLNOT:BLSET.FOR      Fills current baseline calibration table
APLNOT:BPASET.FOR     Sets up the bandpass table array for use by DATBND.
APLNOT:BPREFM.FOR     Checks existence of BP table, changes format if necessary
APLNOT:CALADJ.FOR     Adjusts solution (SN) table phases to a common reference antenna.
APLNOT:CALINI.FOR     Creates/opens/initializes calibration (CL) table
APLSUB:CCINI.FOR      creates and/or opens a CC (components) extension table
APLNOT:CCMERG.FOR     Compresses a CLEAN component (CC) table
APLNOT:CLREFM.FOR     Checks existence of CL table, changes format if necessary
APLNOT:CLUPDA.FOR     Concatenates, rereferences, smooths SN tables and applies it to CL.
APLNOT:CSLGET.FOR     Reads CL (or SN) table and sets up for interpolation.
APLSUB:EXTHIS.FOR     adds to history file for contents of FITS extension file being read
APLSUB:EXTREQ.FOR     parse FITS tape record for required extension file FITS keywords
APLNOT:FLAGUP.FOR     Updates the Flag (FG) table.
APLNOT:FNDSOU.FOR     Find source numbers for a list of sources.
APLNOT:GACSIN.FOR     Initializes CS file, and prepares table to be applied.
APLNOT:GAININ.FOR     Initializes calibration table for application.
APLNOT:GETANT.FOR     Reads AN table and stores the info in common.
APLNOT:GETFQ.FOR      Find info on a given frequency id.
APLNOT:GETSOU.FOR     Find info on a given source id.
APLNOT:GNFSMO.FOR     Boxcar smooths and ASCAL solution (GA) file.
APLNOT:GNSMO.FOR      Optimized spline smoothing of amplitudes in ASCAL (GN) file.
APLNOT:GRDAT.FOR      Getn info about CLEAN components for GRDSUB.
QNOT:GRDCRM.FOR       Loads CLEAN components into AP for uv model computation.
APLNOT:INDXIN.FOR     Initializes index (NX) file, finds first scan selected.
APLNOT:ITBSRT.FOR     Read a table and write a scratch file to be sorted.
APLNOT:LXYPOL.FOR     Fills polarization correction table for AT like linear polarization.
APLNOT:MULSDB.FOR     Determines if a uv file is multi- or single- source.
APLNOT:NXTFLG.FOR     Manages flagging info in tables in common.
APLNOT:OTBSRT.FOR     Copies sorted table from scratch file to table form
APLNOT:POLSET.FOR     Fills polarization correction table from info in AN table.
APLNOT:SDCGET.FOR     Sets up to interpolate in Single dish calibration (CS) table.
APLNOT:SELSMG.FOR     Selects calibrator data, smooths solutions.
APLNOT:SETSTK.FOR     Sets STOKES parameters correctly for plotting routines
APLNOT:SN2CL.FOR      Apply an SN to a CL table.
APLNOT:SNAPP.FOR      Append SN tables and keep track of reference antennas.
APLNOT:SNREFM.FOR     Checks existence of SN table, changes format if necessary
APLNOT:SNSMO.FOR      Smooths solution (SN) tables
APLNOT:SOUFIL.FOR     Fills in arrays of source numbers to be included or excluded.
APLNOT:SOURNU.FOR     Look up source numbers for a list of names.
APLNOT:SUMARY.FOR     Accumulates and lists CLEAN components
APLSUB:TABAN.FOR      I/O to antenna tables (following initialization by ANTINI)
APLNOT:TABAXI.FOR     parse FITS tape record for required extension file FITS keywords
APLSUB:TABLIN.FOR     reads a line from the data portion of a FITS extension of type TABLE
APLNOT:TYINI.FOR      Create/open/initialize Tsys (TY) table
APLNOT:VISCNT.FOR     Determines number of visibility records requested of UVGET
```

B.1.11 EXT-UTIL

```
APLSUB:ALLTAB.FOR     Copies all table extension files from one catalog slot to another
AIPSUB:AU8.FOR        verbs to get or clear name adverbs, destroy extension files
APLNOT:BPINI.FOR      Create/open/initialize bandpass (BP) table
APLNOT:CHNCOP.FOR     Copies selected portions of the IF table
APLNOT:CHNDAT.FOR     Creates/Opens/Reads/Writes/Closes an IF table.
APLNOT:CSINI.FOR      Create/Open/Init Single dish calibration (CS) table
```

```
APLSUB:DELEXT.FOR     removes an extension file from the header in the catalog file
APLSUB:EXTCOP.FOR     copies extension file of the EXTINI/EXTIO variety
APLSUB:EXTINI.FOR     creates and/or opens an extension file of the EXTINI/EXTIO type
APLSUB:EXTIO.FOR      does random access IO to extension files of the EXTINI/EXTIO type
APLNOT:FLGINI.FOR     Create/Open/Init Flag (FG) table.
APLSUB:FNDCOL.FOR     locvates logical column numbers for given titles in a Table
APLSUB:FNDEXT.FOR     returns latest version number of specified extension file type
APLNOT:FQINI.FOR      Create/open/initialize frequency (FQ) table
APLNOT:GAINI.FOR      Creates and initializes gain (GA) extension tables.
APLSUB:GETCOL.FOR     returns value and type found at specified column and row in a table
APLSUB:GETHUT.FOR     returns column titles, units, types, lengths in logical column order
APLNOT:GETNAN.FOR     Find number of antennas and subarrays from AN tables.
APLNOT:GTPAIR.FOR     Returns specified Keyword-value pair from an open AIPS table
APLSUB:ISTAB.FOR      finds if an extension file exists and whether it is a standard table
APLSUB:MADDEX.FOR     adds extension file to catalog header
APLNOT:MAKTAB.FOR     Create and initialize table from data in common /TABHDR/ (FITS)
APLNOT:NDXINI.FOR     Create/open/init index (NX) table
APLSUB:OPEXT.FOR      opens a specified extension file
APLSUB:PUTCOL.FOR     returns value and type found at specified column and row in a table
APLNOT:R3DTAB.FOR     Read data from FITS 3-D table and write AIPS table.
APLSUB:RESCSL.FOR     Rescale flux-like data in any SLice files.
APLNOT:RWTAB.FOR      Read FITS ASCII table data and write AIPS table file.
APLNOT:SDTCRD.FOR     Parse "SINGLDSH" FITS table headers, get some keywords.
APLSUB:SELSTR.FOR     builds string displaying the functions applied to columns of table
APLNOT:SNINI.FOR      Create/open/initialize solution (SN) tables.
APLNOT:SOUINI.FOR     Create/initialize/open source (SU) table
APLNOT:TABAPP.FOR     Appends one table to the end of a similar table.
APLNOT:TABBL.FOR      Do IO to Baseline (BL) table after setup by BLINI.
APLNOT:TABBP.FOR      Does I/O to bandpass (BP) table opened by BPINI
APLNOT:TABCAL.FOR     Does I/O to Calibration (CL) table opened by CALINI
APLSUB:TABCOP.FOR     copies one or all tables extension files of specified type
APLNOT:TABCS.FOR      Does I/O to single dish calibration (CS) table opened by CSINI
APLNOT:TABF3D.FOR     Determines repeat count and data type for FITS 3-D tables entries.
APLNOT:TABFLG.FOR     Does I/O to Flag (FG) table opened by FLGINI
APLNOT:TABFQ.FOR      Does I/O to frequency (FQ) table opened by FQINI
APLNOT:TABFRM.FOR     Parses format for FITS ASCI table entries.
APLNOT:TABGA.FOR      Does I/O to GAIN (GA) table opened by GAINI
APLNOT:TABHDK.FOR     Reads a FITS table header.
APLNOT:TABHDR.FOR     Reads a FITS table header.
APLSUB:TABINI.FOR     create/open a table extension file
APLSUB:TABIO.FOR      reads/writes tables extension files
APLSUB:TABKEY.FOR     reads/writes the Keyword section of an AIPS table file
APLSUB:TABMRG.FOR     merges rows of an an input table file
APLNOT:TABNDX.FOR     Does I/O to Index (NX) table opened by NDXINI
APLNOT:TABSN.FOR      Does I/O to Solution (SN) table opened by SNINI
APLNOT:TABSOU.FOR     Does I/O to Source (SU) table opened by SOUINI
APLNOT:TABSPC.FOR     Determines repeat count and data type for FITS 3-D tables entries.
APLNOT:TABSRT.FOR     Sorts the entries in an AIPS table.
APLNOT:TABTY.FOR      Does I/O to Tsys (TY) table opened by TYINI
```

B.1.12 FITS

```
APLNOT:ATCONV.FOR     Fix AIPS FITS tables
APLSUB:CHAVRT.FOR     converts between local NOLL and local INT binary forms for transport
```

```
APLNOT:CHKTAB.FOR     Check fields of known FITS table types.
APLSUB:EXTHIS.FOR     adds to history file for contents of FITS extension file being read
APLSUB:EXTREQ.FOR     parse FITS tape record for required extension file FITS keywords
APLSUB:FPARSE.FOR     interprets card image from FITS header into AIPS header format
AIPSUB:FWRITE.FOR     converts FITS header to AIPS header and displays it with MSGWRT
APLSUB:GETCRD.FOR     parses card image from FITS header, returns recognized keyword
APLSUB:GETLOG.FOR     returns value of logical variable from character buffer
APLSUB:GETNUM.FOR     returns numeric field from character buffer
APLSUB:GETSTR.FOR     returns a string value (was enclosed by quotes) from character buffer
APLSUB:GETSYM.FOR     returns next symbol in character-form card image
APLSUB:GTWCRD.FOR     returns allowed keyword from FITS header card image
APLSUB:IDWCRD.FOR     returns allowed keyword from FITS header card image
APLSUB:JULDAY.FOR     converts a character-encoded calendar date to Julian day number
APLNOT:MAKTAB.FOR     Create and initialize table from data in common /TABHDR/ (FITS)
AIPSUB:MSGHDR.FOR     lists header contents for standard header plus random parameters
APLNOT:PTF3D.FOR      Copies 8-bit bytes to tape.
APLNOT:R3DTAB.FOR     Read data from FITS 3-D table and write AIPS table.
APLSUB:REAVRT.FOR     converts between local REAL and local INT binary forms for transport
APLNOT:RWTAB.FOR      Read FITS ASCII table data and write AIPS table file.
APLNOT:SDTCRD.FOR     Parse "SINGLDSH" FITS table headers, get some keywords.
APLSUB:SETBSC.FOR     determines scaling/offset parameters to convert image to integer
APLSUB:SETDEF.FOR     fills FITS reader area for table-file extensions with defaults
APLSUB:SKPBLK.FOR     find next non-blank card image in a FITS header, read tape if needed
APLSUB:SKPEXT.FOR     finishes reading FITS extension header, skips the extension data
APLNOT:TABAXI.FOR     parse FITS tape record for required extension file FITS keywords
APLNOT:TABF3D.FOR     Determines repeat count and data type for FITS 3-D tables entries.
APLNOT:TABFRM.FOR     Parses format for FITS ASCI table entries.
APLNOT:TABHDK.FOR     Reads a FITS table header.
APLNOT:TABHDR.FOR     Reads a FITS table header.
APLSUB:TABLIN.FOR     reads a line from the data portion of a FITS extension of type TABLE
APLNOT:TABSPC.FOR     Determines repeat count and data type for FITS 3-D tables entries.
APLNOT:TPIOHD.FOR     Reads tape header and tests if FITS, tape labels etc.
APLGEN:ZBYTF2.FOR     interchange bytes in buffer if needed to go between local & standard
APLGEN:ZBYTFL.FOR     interchange bytes in buffer if needed to go between local & standard
APLGEN:ZTPMID.FOR     pseudo-tape disk read/write for 2880-bytes records
APLGEN:ZTPOPD.FOR     open a pseudo-tape, sequential disk file for FITS
APLGEN:ZTPWAD.FOR     "wait" for IO operation to complete on pseudo-tape disk file (ZTPMID)
APLGEN:ZX8XL.FOR      convert FITS table bit array to AIPS bit array
APLGEN:ZXLX8.FOR      convert AIPS bit array to FITS binary table bit array
```

B.1.13 GRAPHICS

```
AIPSUB:AU9A.FOR       verbs to read TEK cursor and display pixel, sky, image values
AIPSUB:AU9B.FOR       verbs to plot slices and models on graphics
AIPSUB:AU9C.FOR       verbs to set initial guesses for slice model fits using TEK graphics
AIPSUB:SET1DG.FOR     sets initial guess parameters with the TEK for fitting slices
APLNOT:SETSTK.FOR     Sets STOKES parameters correctly for plotting routines
AIPSUB:SLOCIN.FOR     initialize location common for slice (on the TEK) model fitting
APLSUB:TEKFLS.FOR     writes any remaining buffer to the TK graphics device, zeros buffer
APLSUB:TEKVEC.FOR     write bright or dark, scaled or unscaled vector to TK graphics device
APLSUB:TKCATL.FOR     performs operations on the Graphics image catalog
APLSUB:TKCHAR.FOR     writes characters to a TK graphics device
APLSUB:TKCLR.FOR      clears the TK graphics screen
APLSUB:TKCURS.FOR     turns on, reads, turns off the TK graphics cursor
```

APLSUB:TKDVEC.FOR	converts vector command to TK graphics commands
AIPSUB:TKGGPL.FOR	plots model slice on Graphics
AIPSUB:TKGMPL.FOR	plot model fit to slice on the graphics terminal
APLSUB:TKLAB.FOR	labels axes on plot directly to a TK graphics device, draw ticks
AIPSUB:TKRSPL.FOR	plots residuals between slice and its model on Graphics device
AIPSUB:TKSLAC.FOR	activates and reads TEK cursor, converts result to image coordinates
APLSUB:TKSLIN.FOR	initialize parameters for plotting a slice directly on a TK graphics
AIPSUB:TKSLPL.FOR	plot a slice on graphics device
APLSUB:TKTICS.FOR	writes tick marks and labels directly to TK graphics device
APLGEN:ZTKBUF.FOR	flush TK buffer if needed, then store 8-bit byte in buffer
APLGEN:ZTKCL2.FOR	close a Tektronix device
APLGEN:ZTKCLS.FOR	close the TK device
APLGEN:ZTKFI2.FOR	read/write from/to a Tektronix device
APLGEN:ZTKOP2.FOR	read/write from/to a Tektronix device
APLGEN:ZTKOPN.FOR	open a TK device

B.1.14 HEADER

AIPSUB:AU3.FOR	verbs to display contents of catalogs and headers: CATA, IMHE ...
AIPSUB:AU7.FOR	verbs to print history, rescale image, alter axis descriptions
AIPSUB:AU7A.FOR	verbs to put/get header values, to put values into images
APLSUB:AXEFND.FOR	finds axis number for specified axis type
APLSUB:BLDSNM.FOR	builds a name for a scratch file
APLSUB:CATIO.FOR	reads/writes header blocks in the catalog file
APLSUB:CATKEY.FOR	reads/writes the Keyword section of an AIPS header file
APLNOT:COINC.FOR	Checks if two maps are exactly coincident.
APLNOT:DGHEAD.FOR	Fills output CATBLK for UVGET
APLNOT:FRQTAB.FOR	Fill Frequency table in common for IFs and channels
APLNOT:GETCTL.FOR	Determine Stokes' type of Clean map and other modeling info.
APLNOT:IMCREA.FOR	Fills catalog header for an image and optionally creates and catalogs
APLSUB:JULDAY.FOR	converts a character-encoded calendar date to Julian day number
AIPSUB:KWIKHD.FOR	list header contents in abbreviated, image centered form
APLSUB:LMPIX.FOR	returns pixel location corresponding to specified coordinates
APLSUB:LSTHDR.FOR	lists header contents in standard form with MSGWRT
AIPSUB:MSGHDR.FOR	lists header contents for standard header plus random parameters
APLSUB:NAMEST.FOR	packs image name in string with leading and trailing blanks removed
APLSUB:ROTFND.FOR	find the coordinate rotation angle from the catalog header
APLSUB:SUBHDR.FOR	changes input to output header correcting for subimaging
APLSUB:SWAPAX.FOR	swaps the values for two axes
APLSUB:UVPGET.FOR	determines pointers to UV data from the header
APLSUB:VHDRIN.FOR	computes pointers (subscripts) to address components of the header

B.1.15 HISTORY

AIPSUB:AU7.FOR	verbs to print history, rescale image, alter axis descriptions
APLSUB:EXTHIS.FOR	adds to history file for contents of FITS extension file being read
APLSUB:HENCO1.FOR	Adds INNAME, INCLASS, INSEQ, INDISK to an open history file
APLSUB:HENCO2.FOR	Adds IN2NAME, IN2CLASS, IN2SEQ, IN2DISK to an open history file
APLSUB:HENCO3.FOR	Adds IN3NAME, IN3CLASS, IN3SEQ, IN3DISK to an open history file
APLSUB:HENCOO.FOR	adds OUTNAME, OUTCLASS, OUTSEQ, OUTDISK to an open history file
APLSUB:HIAD80.FOR	puts an 80-character card image into a history file as required
APLSUB:HIADD.FOR	adds a history record ("card" = 72 characters) to a history file
APLSUB:HIADDN.FOR	Writes one history line to several history files
APLSUB:HICLOS.FOR	closes a history file, flushing the buffer if desired

```
APLSUB:HICOPY.FOR     copies one history file to the end of a second
APLSUB:HICREA.FOR     open a history file, creating one if needed
APLSUB:HIINIT.FOR     initializes the history common area - must be called before history
APLSUB:HIIO.FOR       does IO and file expansion (if needed) on HI files
APLSUB:HILOCT.FOR     manipulates the history table, opening, closing, located an entry
APLSUB:HIMERG.FOR     creates several new history files by merging several old ones.
APLSUB:HIOPEN.FOR     opens a history file, preparing common pointers and reading record 1
APLSUB:HIPLOT.FOR     places a record in the history file concerning a plot file creation
APLHOT:HIREAD.FOR     Reads next history card from a history file
APLSUB:HISCOP.FOR     creates new history file and copies an old one to it
```

B.1.16 IO-APPL

```
APLHOT:FQMATC.FOR     Check if selection criteria match FQ table entries.
APLHOT:MAKGAU.FOR     *TESS routine: Make a Gaussian convolution function.
APLHOT:MLTMAP.FOR     *TESS routine: multiplies an image by a value, writes another.
APLHOT:PLHPUT.FOR     Copies a subregion of a scratch file image to a cataloged image.
APLHOT:RESID.FOR      *TESS routine: Computes residual image.
APLHOT:SDGET.FOR      Reads single dish data with optional calibration and flagging
APLHOT:SDTCRD.FOR     Parse "SIHGLDSH" FITS table headers, get some keywords.
APLHOT:STEP.FOR       *TESS routine: adds a fraction of one image to another.
APLHOT:SUBMAP.FOR     *TESS Routine: Subtract two images.
APLHOT:TVFOAD.FOR     TVFLG routine to load and image with smoothing converting to display.
APLHOT:UVDOUT.FOR     Divides uv model in one half of a record into other, writes result.
APLHOT:UVDPAD.FOR     Reformat UV data record, doubling size and zero extra words.
APLHOT:UVGET.FOR      Read UV data with optional calibration, editing, selection, etc.
```

B.1.17 IO-BASIC

```
APLSUB:FSERCH.FOR     determines type and number of entries in the common file table (FTAB)
APLSUB:IAMOK.FOR      decides if a disk file type is allowed for the user on a disk
APLSUB:LSERCH.FOR     opens, locates, closes entries in the common file table (FTAB)
APLSUB:MDISK.FOR      reads or writes a row from an image
APLSUB:MINIT.FOR      intializes IO and pointers for quick-return image IO via MDISK
APLSUB:UVDISK.FOR     reads/writes records of arbitrary length, esp UV data, see UVINIT
APLSUB:UVINIT.FOR     initializes IO for arbitrary length records via UVDISK, esp UV data
APLGEN:ZCLOSE.FOR     closes open devices: disk, line printer, terminal
APLGEN:ZCMPR2.FOR     truncate a disk file, returning blocks to the system
APLGEN:ZCMPRS.FOR     release space from the end of an open disk file
APLGEN:ZCREA2.FOR     create the specified disk file
APLGEN:ZCREAT.FOR     creates a disk file
APLGEN:ZDACLS.FOR     close a disk file
APLGEN:ZDAOPN.FOR     open the specified disk file
APLGEN:ZDEST2.FOR     destroy a closed disk file
APLGEN:ZDESTR.FOR     destroy a closed disk file
APLGEN:ZEXIS2.FOR     return size of disk file and if  it exists
APLGEN:ZEXIST.FOR     return file size and, consequently, whether file exists
APLGEN:ZEXPN2.FOR     expand an open disk file
APLGEN:ZEXPND.FOR     expand an open disk file --- either map or non-map now allowed
APLGEN:ZFI2.FOR       read/write one 256-integer record from/to a non-map disk file
APLGEN:ZFIO.FOR       reads and writes single 256-integer records to non-map disk files
APLGEN:ZFRE2.FOR      return AIPS data disk free space information
APLGEN:ZMI2.FOR       read/write large blocks of data from/to disk, quick return
APLGEN:ZMIO.FOR       random-access, quick return (double buffer) disk IO for large blocks
```

APLGEN:ZMKTMP.FOR	convert a "temporary" file name into a unique name
APLGEN:ZMSGCL.FOR	close Message file or terminal
APLGEN:ZMSGDK.FOR	disk IO to message file
APLGEN:ZMSGOP.FOR	open a message file or message terminal
APLGEN:ZMSGXP.FOR	expand the message file
APLGEN:ZOPEN.FOR	open binary disk files and line printer and TTY devices
APLGEN:ZPATH.FOR	convert a file name
APLGEN:ZPHFIL.FOR	construct a physical file or device name from AIPS logical parameters
APLGEN:ZPHOLV.FOR	construct a physical file - version for UPDAT
APLGEN:ZRENA2.FOR	rename a file
APLGEN:ZTFILL.FOR	zero-fill, initialize a file IO table (FTAB) entry
APLGEN:ZTPOP2.FOR	open a tape device for double-buffer, asymchronous IO
APLGEN:ZTPWA2.FOR	wait for read/write from/to a tape device
APLGEN:ZWAI2.FOR	wait for read/write large blocks of data from/to disk
APLGEN:ZWAIT.FOR	wait for asynchronous ("MAP") IO to finish

B.1.18 IO-TV

APLGEN:ZARGC2.FOR	close an ARGS TV device
APLGEN:ZARGCL.FOR	close an ARGS TV device
APLGEN:ZARGMC.FOR	issues a master clear to an ARGS TV
APLGEN:ZARGO2.FOR	open ARGS TV device
APLGEN:ZARGOP.FOR	open ARGS TV device
APLGEN:ZIPACK.FOR	pack/unpack long integers into short integer buffer
APLGEN:ZM70CL.FOR	close an IIS Model 70 TV device, flushing any buffer
APLGEN:ZM70M2.FOR	issues a master clear to an IIS Model 70 TV
APLGEN:ZM70MC.FOR	issues a master clear to an IIS Model 70 TV
APLGEN:ZM70OP.FOR	open IIS Model 70 TV device
APLGEN:ZM70XF.FOR	read/write data to IIS Model 70 TV with buffering
APLGEN:ZV20CL.FOR	close a Comtal Vision 1/20 TV device
APLGEN:ZV20OP.FOR	open Comtal Vision 1/20 TV device
APLGEN:ZV20XF.FOR	read/write data to Comtal Vision 1/20 TV device

B.1.19 IO-UTIL

APLNOT:AKCESS.FOR	*TESS routine to read or write files
APLNOT:AKCLOS.FOR	*TESS routine to close files
APLNOT:AKOPEN.FOR	*TESS I/O routine to open files.
QNOT:APIO.FOR	Copies image-like data between disk and "AP memory".
APLNOT:APPLPB.FOR	*TESS routine to apply a taper to an image
QSUB:APROLL.FOR	Copies AP "memory" to disk, gives up AP then reloads AP
APLNOT:BGTOSM.FOR	*TESS routine to copy a subset of a large image to a small one
APLSUB:COMOFF.FOR	determines start block number of a plane in an N-dimensional image
APLNOT:COPMAP.FOR	*TESS routine to copy an image.
APLSUB:DBINIT.FOR	checks map window and initializes for map double buffer IO
APLSUB:DIE.FOR	closes down tasks which use DFIL.INC to maintain status of files
APLNOT:DIVMAP.FOR	Tim Corwell routine: Divide one image by another.
APLNOT:FILSWP.FOR	*TESS routine: switch file info
APLNOT:FLAT.FOR	*TESS routine: initialize an image to a value.
APLSUB:FSWTCH.FOR	switches names and addresses of two files
APLNOT:GETROW.FOR	Read row of an image opened with INTMIO
APLNOT:GTBWRT.FOR	Routine used by GRIDTB to write buffers.
APLNOT:GTF3D.FOR	Copies real-world bytes from a tape buffer, reading if necessary.
APLNOT:HIREAD.FOR	Reads next history card from a history file

```
APLNOT:INTMIO.FOR     Open an image file for use with GETROW
APLNOT:LINIO.FOR      Reads/writes line to/from an image.
APLNOT:MAKCVM.FOR     *TESS routine: Make image with residuals added.
APLSUB:MAPCLS.FOR     closes cataloged file, updating header and catalog status if needed
APLSUB:MAPOPN.FOR     open file pointed to by catalog entry and mark the entry busy
APLSUB:MDESTR.FOR     deletes a catalog entry and all files assocated with it
APLNOT:REIMIO.FOR     Reinitialize for image I/O using INTMIO
APLNOT:SCINTP.FOR     Interpolates bandpass tables in time.
APLNOT:SCLOAD.FOR     Copies part of one bandpass scratch file to another for efficiency.
APLSUB:SCREAT.FOR     create an AIPS-standard scratch file w common DFIL.INC, ...
APLNOT:SMTOBG.FOR     *TESS routine: Copies small image to a large one.
APLSUB:SNDY.FOR       closes all files, then deletes all scratch files
APLNOT:TABSRT.FOR     Sorts the entries in an AIPS table.
APLNOT:TPIOHD.FOR     Reads tape header and tests if FITS, tape labels etc.
APLNOT:VECWIN.FOR     Interpretes BLC and TRC into useable values as a vector.
APLGEN:ZDIR.FOR       build a full path name to files in AIPS-standard areas (HE, RU, ...)
APLGEN:ZFULLN.FOR     convert file name to full pathname with no logicals
APLGEN:ZRENAM.FOR     rename a disk file
```

B.1.20 IO-WAWA

```
APLSUB:A2WAWA.FOR     packs WaWa IO NameString from its components
APLSUB:CLENUP.FOR     closes all open files and deletes all scratch files for this task
APLSUB:FILCLS.FOR     close file opened by FILOPN, flushing write buffers, clearing catalog
APLSUB:FILCR.FOR      create associated or scratch non-map file
APLNOT:FILDEF.FOR     Fills in default values in WAWA namestring
APLSUB:FILDES.FOR     destroy the specified file or associated file
APLSUB:FILIO.FOR      reads/writes 256-integer record to non-map file opened by FILOPN
APLSUB:FILNUM.FOR     finds the FILTABle entry number for an open file
APLSUB:FILOPN.FOR     open image, associated, or scratch file (WaWa system)
APLSUB:GETHDR.FOR     get the catalog header for an open file (WaWa)
APLNOT:GETWIN.FOR     Get current window of file open in WAWA IO system.
APLNOT:GTNAME.FOR     WAWA IO routine to fill in a namestring for an open file
APLSUB:H2WAWA.FOR     packs AIPS adverb values into WaWa IO NameString
APLSUB:HDRINF.FOR     returns consecutivew items of specified type from header for WaWa
APLSUB:HDRWIN.FOR     sets image corners via WINDOW, revises header to that of output
APLNOT:IMOPEN.FOR     Open the TV under the system set up by IOSET
APLNOT:IMWIN.FOR      Set up window on TV device
APLSUB:IOSET.FOR      initialize tables and set buffer space for WaWa IO
APLNOT:MADD.FOR       Routine to add windows of open images.
APLNOT:MAKNAM.FOR     Constructs WAWA namestring (Now use H2WAWA or A2WAWA)
APLSUB:MAPCOP.FOR     Copy a map
APLSUB:MAPCR.FOR      create and catalog an image in the WaWa package
APLSUB:MAPIO.FOR      reads or writes a file opened by FILOPN (WaWa IO)
APLSUB:MAPMAX.FOR     determine extrema of image opened by FILOPN and update header
APLSUB:MAPWIN.FOR     set/reset the window parameters for an open file (in WaWa)
APLSUB:MAPXY.FOR      sets WaWa windows for a window in the top plane of an image
APLNOT:MCOPY.FOR      Copies a window in one image to another.
APLNOT:MFILL.FOR      Fill a window in an image with a given value.
APLSUB:OPENCF.FOR     opens a cataloged file (main file only), simplifies call to FILOPN
APLSUB:PRENAM.FOR     checks name-string for WaWa IO package - fills in some defaults
APLNOT:PRTERR.FOR     Prints standard WaWa error message and namestring of file.
APLSUB:PRTNAM.FOR     prints the contents of a WaWa-IO file Namestring
APLNOT:SAVHDR.FOR     Save catalog header for an open file.
```

APLSUB:SCRNAM.FOR	build scratch file name string in the WaWa form
APLSUB:TSKBEG.FOR	task start up operations (common inits, GTPARM, RELPOP) for WaWa
APLSUB:TSKEND.FOR	closes down a task and its files in the WaWa system
APLSUB:UNSCR.FOR	delete all scratch files belonging to this tasl
APLSUB:WAWA2A.FOR	unpacks WaWa IO NameString into its components

B.1.21 MAP

QNOT:APCONV.FOR	Disk based 2-D convolution using FFTs.
QNOT:APIO.FOR	Copies image-like data between disk and "AP memory".
AIPSUB:AU9.FOR	verbs to fit or interpolate the image intensity (MAXFIT, IMVAL)
APLNOT:BMSHP.FOR	*TESS routine to fit an elliptical Gaussian to a dirty beam
QNOT:CCSGRD.FOR	Transforms CLEAN components to a grid.
APLNOT:COINC.FOR	Checks if two maps are exactly coincident.
APLNOT:COMCLR.FOR	Scale and map complex array into RBG space, amp=inten. phase=hue.
QNOT:CONV.FOR	*TESS routine: Convolve a map with a beam.
QNOT:CONV1.FOR	First of four routines to convolve two real images.
QNOT:CONV2.FOR	Second of four routines to convolve two real images.
QNOT:CONV3.FOR	Third of four routines to convolve two real images.
QNOT:CONV4.FOR	Fourth of four routines to convolve two real images.
APLNOT:COPMAP.FOR	*TESS routine to copy an image.
QNOT:DISPTV.FOR	*TESS routine: Display an image on a TV
APLNOT:DSKFFT.FOR	2-D disk based FFT using AP.
QNOT:FFTIM.FOR	FFTs an image for uv interpolation.
APLNOT:GETCTL.FOR	Determine Stokes' type of Clean map and other modeling info.
APLNOT:GETROW.FOR	Read row of an image opened with INTMIO
QNOT:GRDCOR.FOR	Normalizes and corrects image for gridding convolution fn.
QNOT:GRDCRM.FOR	Loads CLEAN components into AP for uv model computation.
APLNOT:GRDFLT.FOR	Sets default gridding convolution functions.
QNOT:GRDSUB.FOR	Subtracts transform of CLEAN components from uv data.
QNOT:GRDTAB.FOR	Computes Fourier transform of gridding convolution function.
APLNOT:GRIDTB.FOR	Makes a gridded image of the UV data in TB order.
APLNOT:IMCREA.FOR	Fills catalog header for an image and optionally creates and catalogs
APLNOT:INTMIO.FOR	Open an image file for use with GETROW
QNOT:INTPFN.FOR	Computes interpolation kernals and put them into "AP memory".
APLNOT:LINIO.FOR	Reads/writes line to/from an image.
APLNOT:MADD.FOR	Routine to add windows of open images.
APLNOT:MAKCVM.FOR	*TESS routine: Make image with residuals added.
APLNOT:MAKGAU.FOR	*TESS routine: Make a Gaussian convolution function.
QNOT:MAKMAP.FOR	Makes image or beam from uv data set.
APLNOT:MCOPY.FOR	Copies a window in one image to another.
APLNOT:MFILL.FOR	Fill a window in an image with a given value.
APLNOT:MLTMAP.FOR	*TESS routine: multiplies an image by a value, writes another.
QSUB:PASS1.FOR	First of two routines to FFT an image file.
QSUB:PASS2.FOR	Second of two routines to FFT an image file.
APLNOT:PLNPUT.FOR	Copies a subregion of a scratch file image to a cataloged image.
APLNOT:REIMIO.FOR	Reinitialize for image I/O using INTMIO
APLNOT:RESID.FOR	*TESS routine: Computes residual image.
APLNOT:SAVHDR.FOR	Save catalog header for an open file.
APLNOT:SMTOBG.FOR	*TESS routine: Copies small image to a large one.
APLNOT:SUBMAP.FOR	*TESS Routine: Subtract two images.
APLNOT:SUMARY.FOR	Accumulates and lists CLEAN components
QNOT:UVMDIV.FOR	Divides a uv data set by the Fourier transform of a model.
QNOT:UVMSUB.FOR	Subtracts the Fourier transform of a model from a uv data set.

```
APLNOT:VECWIN.FOR     Interpretes BLC and TRC into useable values as a vector.
QNOT:VISDFT.FOR       Compute DFT of model and subtract/divide from/into uv data.
APLNOT:VMBLKD.FOR     *TESS Routine: Initialize constants in common.
APLNOT:VTTELL.FOR     *TESS Routine: checks TELL file.
```

B.1.22 MAP-UTIL

```
APLNOT:ADDMAP.FOR     *TESS routine to add images
APLNOT:APLPBI.FOR     *TESS routine to apply a taper to an image. VLA only!
APLSUB:BLTGLE.FOR     returns angle from A through a test position to B
APLSUB:BLTLIS.FOR     lists any segments of current row which fall inside blotch regions
APLSUB:COMOFF.FOR     determines start block number of a plane in an N-dimensional image
AIPSUB:CUBINT.FOR     does 2-dimensional cubic interpolation of array values to position
APLSUB:DBINIT.FOR     checks map window and initializes for map double buffer IO
APLSUB:HDRWIN.FOR     sets image corners via WINDOW, revises header to that of output
APLSUB:MAPSIZ.FOR     returns the file size needed to hold the specified image in AIPS
APLSUB:MAPSNC.FOR     creates a scratch image file of specified dimensionality
APLSUB:MCREAT.FOR     create and catalog a map file
APLSUB:MDISK.FOR      reads or writes a row from an image
APLSUB:MINIT.FOR      intializes IO and pointers for quick-return image IO via MDISK
APLSUB:MINSK.FOR      inits use of MSKIP to read noncontiguous, evenly spaced rows in a map
APLSUB:MSKIP.FOR      reads noncontiguous, but evenly spaced rows in a map (see also MINSK)
APLSUB:PEAKFN.FOR     returns location of maximum within 5 pixels of image plane center
APLSUB:PLNGET.FOR     reads subimage of a plane and writes it to scratch file with shifts
APLSUB:RESCAL.FOR     Scales and offsets a cataloged image, updates CATBLK
APLSUB:SETBSC.FOR     determines scaling/offset parameters to convert image to integer
APLSUB:SNRVAL.FOR     substitutes specified value for magic blank value in a buffer
APLSUB:SUBHDR.FOR     changes input to output header correcting for subimaging
APLSUB:WINDOW.FOR     translates user BLC, TRC parameters into usable window arrays
APLSUB:WRBLNK.FOR     write blanked pixels at all pixels corresponding to specified pixel
APLSUB:WRPLAN.FOR     copies an N dimensional plane to a N or N+1 dimensional image
```

B.1.23 MATH

```
APLNOT:APPLPB.FOR     *TESS routine to apply a taper to an image
APLNOT:BOXBSM.FOR     Box car smoothing of an irregularly spaced array with blanking
APLNOT:BOXSMO.FOR     Does boxcar smoothing of an irregularly spaced array.
APLNOT:BSC.FOR        Computes Besselian star constants
APLNOT:CALRES.FOR     *TESS routine to calculate the residuals of an image.
APLNOT:CAXPY.FOR      Linpack routine: Complex constant times a vector plus a vector
APLNOT:CD.FOR         Computes Besselian day numbers C and D for aberration
APLNOT:CGEDI.FOR      Linpack routine: Determinant and inverse of a complex matrix
APLNOT:CGEFA.FOR      Linpack routine: Factors complex matrix by Gaussian Elimination
APLNOT:CLD.FOR        Converts Julian date to civil date
QNOT:CONV.FOR         *TESS routine: Convolve a map with a beam.
QNOT:CONV1.FOR        First of four routines to convolve two real images.
QNOT:CONV2.FOR        Second of four routines to convolve two real images.
QNOT:CONV3.FOR        Third of four routines to convolve two real images.
QNOT:CONV4.FOR        Fourth of four routines to convolve two real images.
QNOT:CONVFN.FOR       Computes convolving fn. kernels and stores them in "AP memory"
APLSUB:COVAR.FOR      Determines the covariance matrix of an M x N matrix
APLNOT:CSCAL.FOR      Linpack routine: Complex constant times vector
APLNOT:CSWAP.FOR      Linpack routine: Swaps two complex vectors
AIPSUB:CUBINT.FOR     does 2-dimensional cubic interpolation of array values to position
```

APLNOT:DA13.FOR	Computes arguments A1, A2 and A3 of the mean motion of the sun.
APLNOT:DA46.FOR	Computes arguments A4, A5 and A6 of the mean motion of the moon
APLNOT:DAPM.FOR	Converts apparent to mean positions.
APLNOT:DCUV.FOR	Computes unit vector for a given celestial position.
APLNOT:DDOT.FOR	Linpack routine: Form dot product of two vectors (DOUBLE)
APLNOT:DERF.FOR	Double precision erf function
APLNOT:DIVMAP.FOR	Tim Corwell routine: Divide one image by another.
APLNOT:DMACH.FOR	Linpack? routine: Sets machine precision parameters. (DOUBLE)
APLNOT:DMAP.FOR	Compute apparent position from mean position
APLNOT:DNRM2.FOR	Compute Euclidean norm of N-Vector
APLSUB:DPMPAR.FOR	returns machine precision or smallest or largest magnitude
APLNOT:DPRE.FOR	Compute General precession matrix
APLNOT:DTRC.FOR	Transforms spherical coordinates given transform matrix.
APLNOT:DUVC.FOR	Converts unit vector to celestial coordinates.
APLNOT:DVDMIN.FOR	Davidon
APLSUB:ENORM.FOR	computes the Euclidean norm of a N-vector
APLNOT:EPS.FOR	Computes mean obliquity of the Ecliptic for a Julian date.
APLNOT:ERF.FOR	error function.
APLNOT:FNDVAR.FOR	*TESS routine: Convert errors in Jy/beam to Jy per cell
APLNOT:FOURG.FOR	Cooley-Tukey fast fourier transform.
APLNOT:FOURYF.FOR	Fast Fourier transform by W. Newman - vectorizes.
APLNOT:GNFSMO.FOR	Boxcar smooths and ASCAL solution (GA) file.
APLNOT:GNSMO.FOR	Optimized spline smoothing of amplitudes in ASCAL (GN) file.
APLNOT:GRD.FOR	Compute the general relativity displacements in RA and DEC.
QNOT:GRDCOR.FOR	Normalizes and corrects image for gridding convolution fn.
APLNOT:GRDFLT.FOR	Sets default gridding convolution functions.
QNOT:GRDTAB.FOR	Computes Fourier transform of gridding convolution function.
APLNOT:GSTROT.FOR	Computes GST at UT=0 and earth rotation rate.
APLNOT:ICAMAX.FOR	Linpack routine: Index of complex element with max. abs. value
APLNOT:ICSORT.FOR	Two key in memory sort by one of several methods
APLSUB:JABER.FOR	Compute vectors needed for J2000 aberation and GR light bending.
APLSUB:JNUT.FOR	Computes nutation from IAU 1980 series
APLSUB:JPOLAR.FOR	Correct rectangular position for polar motion.
APLSUB:JPRECS.FOR	Precess between apparent and J2000 epoch positions.
APLSUB:JPRENU.FOR	Compute rotation matrix for precession and nutation IAU 1980 series.
APLNOT:L1.FOR	Compute L1 solution to an overdetermined system of linear equations
APLNOT:LG2BIT.FOR	Converts between bit arrays and logical arrays
APLSUB:LMDER.FOR	minimize the sum of squares of M nonlinear functions in N variables
APLSUB:LMDER1.FOR	minimize the sum of squares of M nonlinear functions in N variables
APLSUB:LMSTR.FOR	minimize sum of squares of M nonlinear functions in N variables
APLSUB:LMSTR1.FOR	minimize sum of squares of M nonlinear functions in N variables
APLNOT:MACMIN.FOR	Returns the smallest positive value that added to 1.0 is .gt. 1.0.
APLNOT:MAKGAU.FOR	*TESS routine: Make a Gaussian convolution function.
APLSUB:MATVMU.FOR	multiplies a matrix and a vector
APLNOT:MLTMAP.FOR	*TESS routine: multiplies an image by a value, writes another.
APLNOT:NULB.FOR	Finds a root of a function in an interval.
APLNOT:NUT2.FOR	Computes nutation in longitude and obliquity for a Julian date.
APLNOT:NUT4.FOR	Computes nutation using a non ridgid earth model
APLNOT:PARANG.FOR	Computes antenna parallactic angles
QSUB:PASS1.FOR	First of two routines to FFT an image file.
QSUB:PASS2.FOR	Second of two routines to FFT an image file.
APLSUB:PERMAT.FOR	permutes rows or columns of matrix according to permutation vector
APLNOT:PRECES.FOR	Convert between mean and apparent positions (B1950 only)
APLNOT:QKSORT.FOR	Two key "quick" sort routine to sort arrays.

```
APLSUB:QRFAC.FOR      computes a QR factorization of an MxN matrix
APLSUB:QRSOLV.FOR     completes the least squares matrix solution
APLSUB:RANDIN.FOR     initializes tables for random number routine RANDUM
APLSUB:RANDUM.FOR     generates random number between 0 and 1; initialized by RANDIN
APLNOT:RESID.FOR      *TESS routine: Computes residual image.
APLNOT:RFFTF.FOR      Vectorizable, table lookup Fast Fourier transform (non-AP)
APLSUB:RWUPDT.FOR     computes the QR decomposition of an upper triangular matrix + a row
APLSUB:SLAEVP.FOR     Earth position and motion ephemeris (J2000)
APLNOT:SOUELV.FOR     Computes source hour angles and elevations
APLNOT:SPHFN.FOR      Evaluate rational approx. to selected spheriodial functions.
APLNOT:STEP.FOR       *TESS routine: adds a fraction of one image to another.
APLNOT:SUBMAP.FOR     *TESS Routine: Subtract two images.
```

B.1.24 MESSAGES

```
APLGEN:ZMSGCL.FOR     close Message file or terminal
```

B.1.25 MODELING

```
QNOT:ALGSUB.FOR       Interpolates model visibility grom a grid and subtracts from uv data.
APLNOT:BMSHP.FOR      *TESS routine to fit an elliptical Gaussian to a dirty beam
QNOT:CCSGRD.FOR       Transforms CLEAN components to a grid.
APLSUB:COVAR.FOR      Determines the covariance matrix of an M x N matrix
APLSUB:DECONV.FOR     deconvolves two gaussians
QNOT:FFTIM.FOR        FFTs an image for uv interpolation.
APLNOT:FRQTAB.FOR     Fill Frequency table in common for IFs and channels
APLSUB:GETERR.FOR     calculates the errors on the fitted parameters
APLNOT:GRDAT.FOR      Getn info about CLEAN components for GRDSUB.
QNOT:GRDCRM.FOR       Loads CLEAN components into AP for uv model computation.
APLNOT:GRDSET.FOR     Creates scratch files and sets up for GRDSUB
QNOT:GRDSUB.FOR       Subtracts transform of CLEAN components from uv data.
QNOT:INTPFN.FOR       Computes interpolation kernals and put them into "AP memory".
APLSUB:LMDER.FOR      minimize the sum of squares of M nonlinear functions in N variables
APLSUB:LMDER1.FOR     minimize the sum of squares of M nonlinear functions in N variables
APLSUB:LMPAR.FOR      completes solution of the MxN matrix least squares problem
APLSUB:LMSTR.FOR      minimize sum of squares of M nonlinear functions in N variables
APLSUB:LMSTR1.FOR     minimize sum of squares of M nonlinear functions in N variables
APLSUB:MOM.FOR        calculates moments in a 16x16 data array
AIPSUB:PFIT.FOR       parabolic fit to 3x3 matrix
APLSUB:QRFAC.FOR      computes a QR factorization of an MxN matrix
APLSUB:QRSOLV.FOR     completes the least squares matrix solution
APLSUB:RWUPDT.FOR     computes the QR decomposition of an upper triangular matrix + a row
APLNOT:SETGDS.FOR     Sets up for UV model computation, fills common in DGDS.INC
APLNOT:UVDOUT.FOR     Divides uv model in one half of a record into other, writes result.
APLNOT:UVDPAD.FOR     Reformat UV data record, doubling size and zero extra words.
QNOT:UVMDIV.FOR       Divides a uv data set by the Fourier transform of a model.
QNOT:UVMSUB.FOR       Subtracts the Fourier transform of a model from a uv data set.
QNOT:VISDFT.FOR       Compute DFT of model and subtract/divide from/into uv data.
```

B.1.26 PARSING

```
APLSUB:CH2NUM.FOR     converts string containing an integer in ASCII form into the integer
APLSUB:CHLTOU.FOR     converts a CHARACTER string to all upper case letters
APLNOT:CITC2D.FOR     KEYIN routine: parses Double precision value from a character string
```

```
APLNOT:CITC2I.FOR     KEYIN routine: parse an integer from a character string.
APLNOT:CITC2R.FOR     KEYIN routine: parses a floating value from a character string
APLNOT:CITCPR.FOR     KEYIN routine: character compare with wild cards.
APLNOT:CITEXP.FOR     KEYIN routine: evaluate an expression in a character string
APLNOT:CITSKP.FOR     KEYIN routine: Find next non blank character in a string.
APLNOT:DCODEF.FOR     Decodes data from a character string using a format.
APLSUB:FPARSE.FOR     interprets card image from FITS header into AIPS header format
APLSUB:GETCRD.FOR     parses card image from FITS header, returns recognized keyword
APLNOT:GETKEY.FOR     Parses symbol = value from a character string.
APLSUB:GETLOG.FOR     returns value of logical variable from character buffer
APLSUB:GETNUM.FOR     returns numeric field from character buffer
APLSUB:GETSTR.FOR     returns a string value (was enclosed by quotes) from character buffer
APLSUB:GETSYM.FOR     returns next symbol in character-form card image
APLSUB:GTWCRD.FOR     returns allowed keyword from FITS header card image
APLSUB:IDWCRD.FOR     returns allowed keyword from FITS header card image
APLNOT:KEYIN.FOR      AIPS version of CIT parsing routine
APLNOT:SDTCRD.FOR     Parse "SINGLDSH" FITS table headers, get some keywords.
APLNOT:TABF3D.FOR     Determines repeat count and data type for FITS 3-D tables entries.
APLNOT:TABFRM.FOR     Parses format for FITS ASCI table entries.
APLNOT:TABSPC.FOR     Determines repeat count and data type for FITS 3-D tables entries.
```

B.1.27 PLOT-APPL

```
APLNOT:AITOFF.FOR     writes vectors for Aitoff projection grid to plot file
AIPSUB:AU8A.FOR       verb EXTLIST to list contents of plot files and other extension files
APLNOT:COMCLR.FOR     Scale and map complex array intp RBG space, amp=inten. phase=hue.
```

B.1.28 PLOT-UTIL

```
APLSUB:AXSTRN.FOR     encodes axis type and value in a string
APLSUB:CHNTIC.FOR     counts characters to the left of a plot (for labeling vertical axis)
APLSUB:CLAB1.FOR      puts axis labels in plot file and calls CTICS to draw and label ticks
APLSUB:CLAB2.FOR      puts axis labels in plot file and calls CTICS to draw and label ticks
APLSUB:COMLAB.FOR     initializes line drawing and labels plot with text, contour levels
APLSUB:CONDRW.FOR     writes contour plot to a plot file
APLSUB:CTICS.FOR      writes tick marks and tick labels to a plot file
APLSUB:GCHAR.FOR      writes a draw character string command record into a plot file
APLSUB:GFINIS.FOR     writes the end of plot record into a plot file and closes it down
APLSUB:GINIT.FOR      creates, opens, initializes plot file (does not catalog it)
APLSUB:GINITG.FOR     writes an initialize-for-grey-scale record into a plot file
APLSUB:GINITL.FOR     writes an initialize-for-line-drawing command into a plot file
APLSUB:GMCAT.FOR      writes a copy-misc-image-catalog-info records into a plot file
APLSUB:GPHWRT.FOR     write plot buffer to file, prepares buffer for more commands
APLSUB:GPOS.FOR       write a position-"pen" command into a plot file
APLSUB:GRAYPX.FOR     writes an array of grey values into a plot file
APLSUB:GVEC.FOR       writes a move-pen-down (or write vector) command in a plot file
APLSUB:HIPLOT.FOR     places a record in the history file concerning a plot file creation
APLSUB:INTEDG.FOR     returns intersections of a line with the edges of a box
APLSUB:ISCALE.FOR     scale a buffer by various functions to an integer buffer (ie for TV)
APLSUB:LABINI.FOR     initializes commons for labeling of plots (calls SETLOC)
APLSUB:LABNO.FOR      write a tick mark numeric label in a plot file
APLSUB:LINLIM.FOR     clips X,Y values at edges of rectangular area with interpolation
APLNOT:PLEND.FOR      End-of-plot clean-up functions: Gary plot package.
APLSUB:PLGRY.FOR      draws grey scale commands in the plot file: Gary plot package
```

APLNOT:PLMAKE.FOR	creates & opens plot file, puts into map header, writes first record
APLSUB:PLPOS.FOR	puts a position vector command in a plot file: Gary plot package
APLSUB:PLVEC.FOR	puts a draw vector command in a plot file: Gary plot package
APLSUB:RNGSET.FOR	set plat intensity range from image header and user parameters
APLSUB:SETLOC.FOR	sets location common for coordinate computations and display
APLSUB:SCALMM.FOR	computes plot scaling factors and plot scale in arc sec per mm
APLSUB:SLBINI.FOR	initializes labeling for slice plots
APLSUB:STARPL.FOR	adds to plot plus signs at coordinates given in an ST (star) file
APLSUB:TICCOR.FOR	correct tick lengths from increments in dir cosines to coordinates
APLSUB:TICINC.FOR	determines tick mark lengths and increments for CTICS, ...
APLSUB:TKLAB.FOR	labels axes on plot directly to a TK graphics device, draw ticks
APLSUB:TKSLIN.FOR	initialize parameters for plotting a slice directly on a TK graphics
APLSUB:TKTICS.FOR	writes tick marks and labels directly to TK graphics device
APLGEN:ZDOPRT.FOR	reads bit file and causes it to be plotted on printer/plotter
APLGEN:ZLASC2.FOR	spool a closed laser printer print/plot file
APLGEN:ZLASCL.FOR	close and spool a laser printer print/plot file
APLGEN:ZLASIO.FOR	open, write to, close and spool a laser printer print/plot file
APLGEN:ZLASOP.FOR	open a laser printer print/plot file
APLGEN:ZLWIO.FOR	open, write to, close and spool a PostScript print/plot file
APLGEN:ZLWOP.FOR	open a PostScript (LaserWriter) print/plot file

B.1.29 POPS-APPL

AIPSUB:AU1.FOR	prints and clears the message file, sets up for EXIT and RESTART
AIPSUB:AU1A.FOR	does parameter display: INPUTS, SHOW, HELP, EXPLAIN
AIPSUB:AU2.FOR	handles task-related activities: GO, TELL, WAIT, ABORT, SPY, TPUT
AIPSUB:AU2A.FOR	verb functions on task save and Save/Get files: TGET, SGdestr, index
AIPSUB:AU3.FOR	verbs to display contents of catalogs and headers: CATA, IMHE ...
AIPSUB:AU3A.FOR	verbs for disk management: FREE, ALLDEST, TIMDEST, etc.
AIPSUB:AU3B.FOR	verbs to rearrange the entries in the catalog file: RECAT, RENUMBER
AIPSUB:AU4.FOR	verbs to handle basic tape operations: TPHEAD, MOUNT, AVFILE, ...
AIPSUB:AU5.FOR	basic TV verbs to do on/off, read cursor position, init the TV, ...
AIPSUB:AU5A.FOR	verbs to load images to the TV including ROAM
AIPSUB:AU5B.FOR	verbs to anotate TV images
AIPSUB:AU5C.FOR	verbs to draw wedges on TV, erase images, set corners with TV cursor
AIPSUB:AU5D.FOR	verbs to load and run TV movie sequences
AIPSUB:AU6.FOR	verbs to manipulate TV scroll, zoom, color tables, and TVHUEINT
AIPSUB:AU6A.FOR	verbs to set the TV blank and white LUT linearly and to blink planes
AIPSUB:AU6B.FOR	verb to display image value at pixel indicated by TV cursor (CURVAL)
AIPSUB:AU6C.FOR	verb to alter zoom and enhance image in standard way: TVFIDDLE
AIPSUB:AU6D.FOR	verbs to do image statistics in blotch regions: TVSTAT, IMSTAT
AIPSUB:AU7.FOR	verbs to print history, rescale image, alter axis descriptions
AIPSUB:AU7A.FOR	verbs to put/get header values, to put values into images
AIPSUB:AU8.FOR	verbs to get or clear name adverbs, destroy extension files
AIPSUB:AU8A.FOR	verb EXTLIST to list contents of plot files and other extension files
AIPSUB:AU9.FOR	verbs to fit or interpolate the image intensity (MAXFIT, IMVAL)
AIPSUB:AU9A.FOR	verbs to read TEK cursor and display pixel, sky, image values
AIPSUB:AU9B.FOR	verbs to plot slices and models on graphics
AIPSUB:AU9C.FOR	verbs to set initial guesses for slice model fits using TEK graphics
AIPSUB:AUA.FOR	verb to submit batch jobs to AIPSC and the QMNGR queues
AIPSUB:AUB.FOR	verbs to prepare, edit, and review batch jobs and queues
AIPSUB:AUC.FOR	verbs to enter, list, drop gripes, enter password
AIPSUB:AUT.FOR	site-specific test verbs
AIPSUB:PRTALN.FOR	prints line on CRT orprinter, handles page full AND POPS type-ahead

AIPSUB:TASKWT.FOR	waits for tasks to begin, send resumption signal, and/or terminate

B.1.30 POPS-LANG

AIPSUB:BCLEAN.FOR	Pops items from B-stack to A-stack until BPR-stack precedence < NEXTP
AIPSUB:CHUNT.FOR	searches symbol table for character string accepting min match
AIPSUB:COMPIL.FOR	parses line of input with GETFLD, builds stacks for execution
AIPSUB:CONCAT.FOR	creates temporary literal on stack = concatanation of 2 strings
AIPSUB:EDITOR.FOR	does operations needed at start and end of editing existing procedure
AIPSUB:EQUIV.FOR	checks whether two variables are logically equivalent
AIPSUB:GETFLD.FOR	finds the next symbol in KARBUF and determines its pointers
AIPSUB:GETNME.FOR	gets the next name in the input character buffer
AIPSUB:HELPS.FOR	executes "pseudoverb name" -> hidden verb w name on stack (INP, RUN)
AIPSUB:HUNT.FOR	searches a linked list for words to be matched
AIPSUB:INIT.FOR	initializes symbol, procedure text tables, and commons for POPS
AIPSUB:KWICK.FOR	verbs: math, assignment, comparison, looping, branching, proc calls
AIPSUB:LLOCAT.FOR	allocates space in linked-list array and handles link pointers
AIPSUB:LTSTOR.FOR	allocate storage for literal if needed, return pointer in any case
AIPSUB:MASSGN.FOR	handles array = value(s) constructs
AIPSUB:OERROR.FOR	gives user error message, resets parameters to read next input line
AIPSUB:POLISH.FOR	parses the input text buffer, building stacks; executes pseudoverbs
AIPSUB:POP.FOR	pops item from stack
AIPSUB:PREAD.FOR	reads an input line from current input source (CRT, RUN file, batch)
AIPSUB:PSEUDO.FOR	compiles pseudoverbs: PROC, declarations, IF, THEN, WHILE, FINISH, ..
AIPSUB:PUSH.FOR	pushes item onto stack advancing the stack pointer
AIPSUB:RLOCAT.FOR	allocates space in linked-list array and handles link pointers
AIPSUB:SETTYP.FOR	replaces the symbol type code in the data description structure
APLSUB:STLTOU.FOR	converts any characters beween single quotes to upper case
AIPSUB:STORES.FOR	stores proc code; pseudoverbs: SAVE, GET, RESTORE, STORE, LIST, ...
AIPSUB:SUBS.FOR	converts variable with subscript to the appropriate scalar
AIPSUB:SYMBOL.FOR	obtains symbol identification from symbol table; creates new symbols
AIPSUB:VERBS.FOR	calls verbs subroutines (AUnc) by verb number - interactive version
AIPSUB:VERBSB.FOR	calls verbs subroutines (AUnc) by verb number - batch version
AIPSUB:VERBSC.FOR	calls verbs subroutines (AUnc) by verb number - Checker version

B.1.31 POPS-UTIL

AIPSUB:ASSGN.FOR	performs the assignment functions of scalar/vector = scalar/vector
AIPSUB:CONFRM.FOR	asks user to respond yes or no to some question
APLSUB:GETNUM.FOR	returns numeric field from character buffer
APLSUB:GETSTR.FOR	returns a string value (was enclosed by quotes) from character buffer
AIPSUB:PRTMSG.FOR	prints and deletes messages from the MS file
AIPSUB:RDUSER.FOR	reads the user number from the terminal
AIPSUB:SCHOLD.FOR	wait for user input on screen full, allows type ahead, quit, continue
AIPSUB:SGLAST.FOR	does a SAVE or GET of the K array cataloged as LASTEXIT.
AIPSUB:SGLOCA.FOR	locates a Save/Get file by name in catalog of SG files
AIPSUB:UINIT.FOR	general, non-language initialization routine; only calls VHDRIN

B.1.32 PRINTER

APLSUB:BATPRT.FOR	prints header/trailer messages for printer tasks when run in batch
APLSUB:DATDAT.FOR	converts "DD/MM/YY" form of date to "dd-mmm-yyyy" for printing
AIPSUB:PRTALN.FOR	prints line on CRT orprinter, handles page full AND POPS type-ahead
APLSUB:PRTLIN.FOR	prints line on printer or terminal with page-full handling, headers

```
APLGEN:ZDOPRT.FOR      reads bit file and causes it to be plotted on printer/plotter
APLGEN:ZENDPG.FOR      advance printer if needed to avoid electrostatic-printer "burn-out"
APLGEN:ZLASC2.FOR      spool a closed laser printer print/plot file
APLGEN:ZLASCL.FOR      close and spool a laser printer print/plot file
APLGEN:ZLASIO.FOR      open, write to, close and spool a laser printer print/plot file
APLGEN:ZLASOP.FOR      open a laser printer print/plot file
APLGEN:ZLPCL2.FOR      queue a file to the line printer and delete
APLGEN:ZLPCLS.FOR      close an open printer device
APLGEN:ZLPOP2.FOR      open a line-printer text file - actual OPEN call
APLGEN:ZLPOPN.FOR      open a line-printer text file
APLGEN:ZLWIO.FOR       open, write to, close and spool a PostScript print/plot file
APLGEN:ZLWOP.FOR       open a PostScript (LaserWriter) print/plot file
```

B.1.33 SDISH

```
APLNOT:CSINI.FOR       Create/Open/Init Single dish calibration (CS) table
APLNOT:DCALSD.FOR      Apply Single dish calibration to data.
APLNOT:DGETSD.FOR      Reads, selects single dish data, calibrates and edits.
APLNOT:GACSIN.FOR      Initializes CS file, and prepares table to be applied.
APLNOT:SDCGET.FOR      Sets up to interpolate in Single dish calibration (CS) table.
APLNOT:SDCSET.FOR      Interpolates single dish calibration data for current time.
APLNOT:SDGET.FOR       Reads single dish data with optional calibration and flagging
APLNOT:SDTCRD.FOR      Parse "SINGLDSH" FITS table headers, get some keywords.
APLNOT:TABCS.FOR       Does I/O to single dish calibration (CS) table opened by CSINI
```

B.1.34 SERVICE

```
AIPSUB:AIPINI.FOR      does all AIPS initializations for a stand-alone program
AIPSUB:DESCR.FOR       destroys all scratch files for tasks which are no longer active
APLSUB:INQFLT.FOR      inquire of the user for specified number of floating-point values
APLSUB:INQGEN.FOR      inquire of user for specified list of integer, float, & char values
APLSUB:INQINT.FOR      inquire of user for specified number of integer values
APLSUB:INQSTR.FOR      request character string from user (1st N characters of line)
APLGEN:ZADDR.FOR       determine if 2 addresses inside computer are the same
APLGEN:ZDELA2.FOR      delay current process a specified interval
APLGEN:ZDELAY.FOR      delay current process a specified interval
APLGEN:ZERRO2.FOR      return system error message for given system error code
APLGEN:ZGTBIT.FOR      get array of bits from a word
APLGEN:ZHEX.FOR        encode an integer into hexadecimal characters
APLGEN:ZKDUMP.FOR      display portions of an array in various Fortran formats
APLGEN:ZMSGWR.FOR      call MSGWRT based on call arguments - for C routines to call MSGWRT
APLGEN:ZMYVER.FOR      returns OLD, NEW, or TST based on translation of logical AIPS_VERSION
APLGEN:ZPTBIT.FOR      put array of bits into a word
APLGEN:ZTIME.FOR       return the local time of day
```

B.1.35 SLICE

```
AIPSUB:AU9B.FOR        verbs to plot slices and models on graphics
AIPSUB:AU9C.FOR        verbs to set initial guesses for slice model fits using TEK graphics
AIPSUB:SET1DG.FOR      sets initial guess parameters with the TEK for fitting slices
AIPSUB:SLOCIN.FOR      initialize location common for slice (on the TEK) model fitting
AIPSUB:TKGGPL.FOR      plots model slice on Graphics
AIPSUB:TKGMPL.FOR      plot model fit to slice on the graphics terminal
AIPSUB:TKRSPL.FOR      plots residuals between slice and its model on Graphics device
```

APLSUB:TKSLIN.FOR	initialize parameters for plotting a slice directly on a TK graphics
AIPSUB:TKSLPL.FOR	plot a slice on graphics device

B.1.36 SORT

APLSUB:LSORT.FOR	sort a data buffer minimizing number times records are switched
APLSUB:MERGE.FOR	sorts by merging previously sorted blocks of records
APLSUB:OSORT.FOR	does quick sort on array of vectors, then reorders by calling PERMAT
APLSUB:PERMAT.FOR	permutes rows or columns of matrix according to permutation vector
APLSUB:SHSORT.FOR	Shell sort of an array or records on two keys

B.1.37 SPECTRAL

APLNOT:DATBND.FOR	Applies the bandpass correction to data.
APLNOT:FQMATC.FOR	Check if selection criteria match FQ table entries.
APLNOT:FRQTAB.FOR	Fill Frequency table in common for IFs and channels
APLNOT:IOBSRC.FOR	Search for antennas in the current bandpass buffer.
APLNOT:SCINTP.FOR	Interpolates bandpass tables in time.
APLNOT:SCLOAD.FOR	Copies part of one bandpass scratch file to another for efficiency.
APLNOT:SETSM.FOR	Determines type of spectral smoothing and sets up look up table.
APLNOT:SMOSP.FOR	Convolves a spectrum with a tabulated function.
APLNOT:TABBP.FOR	Does I/O to bandpass (BP) table opened by BPINI
APLNOT:UVGET.FOR	Read UV data with optional calibration, editing, selection, etc.

B.1.38 SYSTEM

APLSUB:ACOUNT.FOR	Writes beginning and final entries in the AIPS accounting file
AIPSUB:AIPINI.FOR	does all AIPS initializations for a stand-alone program
APLSUB:BATQ.FOR	performs operations on batch queue control file such as OPEN RUN CLOS
APLSUB:DIE.FOR	closes down tasks which use DFIL.INC to maintain status of files
APLSUB:DIETSK.FOR	closes a task: restarting AIPS, settling the accounting, issuing msg
APLSUB:PASENC.FOR	encrypts a 12-character password into 3 Holleriths
APLSUB:PASWRD.FOR	prompts for and checks password if the user has a non-blank PW entry
APLSUB:RELPOP.FOR	places a return code in the task data file, thereby resuming AIPS
APLSUB:WHOAMI.FOR	given root task name, gets actual task name and finds NPOPS number
APLGEN:ZABOR2.FOR	establishes or carries out (when appropriate) abort handling
APLGEN:ZABORT.FOR	establishes or carries out (when appropriate) abort handling
APLGEN:ZACTV8.FOR	activate the requested program, returning process ID information
APLGEN:ZCPU.FOR	return current process CPU time and IO count
APLGEN:ZDATE.FOR	return the local date
APLGEN:ZDCNI2.FOR	initialize device and Z-routine characteristics commons - local vals
APLGEN:ZDCHIC.FOR	set more system parameters; make them available to C routines
APLGEN:ZDCHIN.FOR	initialize message, device and Z-routine characteristics commons
APLGEN:ZFREE.FOR	display available disk space
APLGEN:ZGNAME.FOR	get name of current process
APLGEN:ZPRI2.FOR	raise or lower the process priority
APLGEN:ZPRIO.FOR	raise or lower the process priority
APLGEN:ZPRPAS.FOR	prompt user and read 12-character password (invisible) from CRT
APLGEN:ZSETUP.FOR	performs system-level operations after VERNAM, TSKNAM, NPOPS known
APLGEN:ZSTAI2.FOR	does any system cleanup needed at the end of interactive AIPS session
APLGEN:ZSTAIP.FOR	does any system cleanup needed at the end of interactive AIPS session
APLGEN:ZTACT2.FOR	inquires if a task is currently active on the local computer
APLGEN:ZTACTQ.FOR	inquires if a task is currently active on the local computer
APLGEN:ZTKILL.FOR	deletes (or kills) the specified process

APLGEN:ZTQSP2.FOR display AIPS account or all processes running on the system
APLGEN:ZTQSPY.FOR display AIPS account or all processes running on the system
APLGEN:ZTRLOG.FOR translate a logical name
APLGEN:ZWHOMI.FOR determines AIPSxn task name; sets NPOPS, assigns TV and TK devices

B.1.39 TAPE

APLSUB:DWRITE.FOR translate "DEC" format map header and display parameters
APLSUB:EXTHIS.FOR adds to history file for contents of FITS extension file being read
APLSUB:EXTREQ.FOR parse FITS tape record for required extension file FITS keywords
APLSUB:FNDEOT.FOR advances tape to logical end of information (2 consecutive EOFs)
AIPSUB:FWRITE.FOR converts FITS header to AIPS header and displays it with MSGWRT
APLNOT:GTF3D.FOR Copies real-world bytes from a tape buffer, reading if necessary.
APLSUB:MLREOF.FOR advances tape to end of file and reports records read in TAPIO system
APLNOT:PTF3D.FOR Copies 8-bit bytes to tape.
APLNOT:R3DTAB.FOR Read data from FITS 3-D table and write AIPS table.
APLNOT:RWTAB.FOR Read FITS ASCII table data and write AIPS table file.
APLNOT:SDTCRD.FOR Parse "SINGLDSH" FITS table headers, get some keywords.
APLSUB:SKPBLK.FOR find next non-blank card image in a FITS header, read tape if needed
APLSUB:SKPEXT.FOR finishes reading FITS extension header, skips the extension data
APLNOT:TABAXI.FOR parse FITS tape record for required extension file FITS keywords
APLNOT:TABF3D.FOR Determines repeat count and data type for FITS 3-D tables entries.
APLNOT:TABFRM.FOR Parses format for FITS ASCI table entries.
APLSUB:TABLIN.FOR reads a line from the data portion of a FITS extension of type TABLE
APLNOT:TABSPC.FOR Determines repeat count and data type for FITS 3-D tables entries.
APLSUB:TAPIO.FOR read/writes tape and FITS disk files
APLSUB:TPHEAD.FOR reads a tape record, advances over label file, decides if it
APLNOT:TPIOHD.FOR Reads tape header and tests if FITS, tape labels etc.
AIPSUB:UWRITE.FOR writes summary of UV Export-format tape
APLSUB:VBOUT.FOR writes variable length, blocked records of 16-bit integers to tape
APLGEN:ZBKLD1.FOR initialize environment for BAKLD
APLGEN:ZBKLD2.FOR does BACKUP operation: load images from tape to directory
APLGEN:ZBKLD3.FOR clean up system things for BAKLD ending
APLGEN:ZBKTP1.FOR initialize BACKUP to tape operation for BAKTP
APLGEN:ZBKTP2.FOR write a cataloged file plus extensions to BACKUP tape in BAKTP
APLGEN:ZBKTP3.FOR clean up host environment at end of BAKTP
APLGEN:ZBYTF2.FOR interchange bytes in buffer if needed to go between local & standard
APLGEN:ZBYTFL.FOR interchange bytes in buffer if needed to go between local & standard
APLGEN:ZMCACL.FOR convert Modcomp compressed ASCII to Hollerith characters (for FILLR)
APLGEN:ZMOUN2.FOR mount or dismount magnetic tape device
APLGEN:ZMOUNT.FOR mount or dismount magnetic tape device
APLGEN:ZR8P4.FOR converts pseudo I*4 to double precision - for tape handling only
APLGEN:ZRDMF.FOR convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers
APLGEN:ZRM2RL.FOR convert Modcomp to local single precision floating point
APLGEN:ZTAP2.FOR position (forward/back record/file), write EOF, etc. for tapes
APLGEN:ZTAPE.FOR mount, dismount, position, write EOF, etc. for tapes
APLGEN:ZTAPIO.FOR tape operations for IMPFIT (compressed FITS transport tape)
APLGEN:ZTPCL2.FOR close a tape device
APLGEN:ZTPCLD.FOR close pseudo-tape disk file
APLGEN:ZTPCLS.FOR closes a tape device (real or pseudo-tape disk)
APLGEN:ZTPMI2.FOR tape read/write
APLGEN:ZTPMID.FOR pseudo-tape disk read/write for 2880-bytes records
APLGEN:ZTPMIO.FOR read/write tape devices with quick return IO methods
APLGEN:ZTPOP2.FOR open a tape device for double-buffer, asymchronous IO

```
APLGEN:ZTPOPD.FOR      open a pseudo-tape, sequential disk file for FITS
APLGEN:ZTPOPN.FOR      open tape or pseudo-tape device
APLGEN:ZTPWA2.FOR      wait for read/write from/to a tape device
APLGEN:ZTPWAD.FOR      "wait" for IO operation to complete on pseudo-tape disk file (ZTPMID)
APLGEN:ZTPWAT.FOR      wait for asynchronous IO to finish on tape or pseudo-tape disk
```

B.1.40 TERMINAL

```
APLGEN:ZPRMPT.FOR      prompt user and read 80-characters from CRT screen
APLGEN:ZPRPAS.FOR      prompt user and read 12-character password (invisible) from CRT
APLGEN:ZTTBUF.FOR      reads terminal input with no prompt or wait - simulates TV trackball
APLGEN:ZTTCLS.FOR      close a terminal device
APLGEN:ZTTOP2.FOR      open a message terminal
APLGEN:ZTTOPN.FOR      open a terminal device
APLGEN:ZTTYIO.FOR      read/write buffer to terminal
```

B.1.41 TEXT

```
APLSUB:TXTMAT.FOR      min match handling for text files (calls ZTXMAT, does messages)
APLSUB:VERMAT.FOR      min match handling for text file names incl sequence of directories
APLGEN:ZDIR.FOR        build a full path name to files in AIPS-standard areas (HE, RU, ...)
APLGEN:ZTCLOS.FOR      close text file opened with ZTOPEN
APLGEN:ZTOPE2.FOR      open text file for ZTOPEN
APLGEN:ZTOPEN.FOR      open text file - logical area, version, member name as arguments
APLGEN:ZTREAD.FOR      read next 80-character record in sequential text file (ZTOPEN type)
APLGEN:ZTXCLS.FOR      clos text file opened via ZTXOPN
APLGEN:ZTXIO.FOR       read/write a line to a text file
APLGEN:ZTXMA2.FOR      find all file names matching a given wildcard specification
APLGEN:ZTXMAT.FOR      return list of files in specified area beginning with specified chars
APLGEN:ZTXOP2.FOR      translate the file name and open a text file
APLGEN:ZTXOPN.FOR      open a text file for read or write
```

B.1.42 TV

```
APLNOT:IMWIN.FOR       Set up window on TV device
```

B.1.43 TV-APPL

```
AIPSUB:AU5.FOR         basic TV verbs to do on/off, read cursor position, init the TV, ...
AIPSUB:AU5A.FOR        verbs to load images to the TV including ROAM
AIPSUB:AU5B.FOR        verbs to anotate TV images
AIPSUB:AU5C.FOR        verbs to draw wedges on TV, erase images, set corners with TV cursor
AIPSUB:AU5D.FOR        verbs to load and run TV movie sequences
AIPSUB:AU6.FOR         verbs to manipulate TV scroll, zoom, color tables, and TVHUEINT
AIPSUB:AU6A.FOR        verbs to set the TV blank and white LUT linearly and to blink planes
AIPSUB:AU6B.FOR        verb to display image value at pixel indicated by TV cursor (CURVAL)
AIPSUB:AU6C.FOR        verb to alter zoom and enhance image in standard way: TVFIDDLE
AIPSUB:AU6D.FOR        verbs to do image statistics in blotch regions: TVSTAT, IMSTAT
QNOT:DISPTV.FOR        *TESS routine: Display an image on a TV
YSUB:GRBOXS.FOR        sets rectangular boxes or diagonal line with TV cursor and graphics
AIPSUB:GRLUTS.FOR      interactive piecewise linear LUT using graphic plane and cursor
AIPSUB:HIENH.FOR       interactive linear enhancement of 2-image hue-intensity TV display
AIPSUB:HILUT.FOR       calculates new LUTs for hue-intensity display and sends them to TV
YSUB:IAXIS1.FOR        draws axis labels and tick marks (via ITICS) on TV
```

APLNOT:IMOPEN.FOR	Open the TV under the system set up by IOSET
YSUB:ITICS.FOR	draws tick marks and labels on TV
AIPSUB:TVBLNK.FOR	blinks two TV channels, cursor controls rate
APLNOT:TVFOAD.FOR	TVFLG routine to load and image with smoothing converting to display.
AIPSUB:TVMOVI.FOR	runs movie algorithm on pre-loaded images, with interactions
AIPSUB:TVROAM.FOR	does interactive multi-channel "ROAM" display on pre-loaded images
YGEN:YCUCOR.FOR	correct cursor position for scroll; return image coordinates, header
YGEN:YISDRM.FOR	read/write data memory of NRAO-ISU device
YGEN:YISDSC.FOR	read/write micro-processor memory of NRAO-ISU device
YGEN:YISJMP.FOR	cause microprocessor jump toaddress in NRAO-ISU device
YGEN:YISLOD.FOR	loads/unloads program memory of NRAO-ISU device
YGEN:YISMPM.FOR	reads/writes microprocessor memory of the NRAO-ISU device
YGEN:YMKCUR.FOR	selects the form of the cursor to be displayed

B.1.44 TV-BASIC

YGEN:YALUCT.FOR	drives the TV arithmetic logic unit - not to be used much
YGEN:YCONST.FOR	controls the constant registers added to the TV picture - not used
YGEN:YCRCTL.FOR	controls the TV cursor visibility, position; reads trackball buttons
YGEN:YFDBCK.FOR	causes a feedback operation in the TV
YGEN:YGGRAM.FOR	controls the TV graphics color assignments
YGEN:YGRAFE.FOR	controls the graphics control register (IIS function)
YGEN:YGRAPH.FOR	turns TV graphics planes on and off
YGEN:YIFM.FOR	read/write TV Input look-up-table
YGEN:YIMGIO.FOR	read/write data to the TV grey and graphics memories
YGEN:YINIT.FOR	initialize everything about the TV
YGEN:YLUT.FOR	read/write channel-based look-up-table
YGEN:YMNMAX.FOR	read 3 min/max values from TV data paths (IIS only, not used)
YGEN:YOFM.FOR	read/write all-channel look-up-table ("output function memory")
YGEN:YRHIST.FOR	read the histogram of the selected TV output color
YGEN:YSCROL.FOR	write the scroll registers (shift location of 1 or more TV channels)
YGEN:YSHIFT.FOR	read/write the shift (bias) registers of the TV (IIS M70, not used)
YGEN:YSPLIT.FOR	set channel selection by split-screen quadrant
YGEN:YSTCUR.FOR	reads/writes the cursor pattern array
YGEN:YTVCIN.FOR	initialize TV characteristics common (not needed much - see TVOPEN)
YGEN:YTVCLS.FOR	close the TV, including TV device and TV control/parameter disk file
YGEN:YTVOPN.FOR	open the TV device and the TV disk control/parameter file.
YGEN:YZOOMC.FOR	set the TV zoom magnification and center

B.1.45 TV-IO

YGEN:YTVCL2.FOR	close actual TV device (called by YTVCLS)
YGEN:YTVMC.FOR	issue a master clear to reinitialize IO to the TV
YGEN:YTVOP2.FOR	open actual TV device (called by YTVOPN)
APLGEN:ZARGS.FOR	sends command to/from the ARGS TV device
APLGEN:ZARGXF.FOR	translates IIS Model 70 commands into calls to ZARGS for ARGS TV
APLGEN:ZDEAC2.FOR	close DeAnza TV device
APLGEN:ZDEACL.FOR	close DeAnza TV device
APLGEN:ZDEAMC.FOR	issue a master clear to the TV - for DeAnzas this is a No-Op
APLGEN:ZDEAO2.FOR	opens DeAnza TV device
APLGEN:ZDEAOP.FOR	opens DeAnza TV device
APLGEN:ZDEAX2.FOR	do actual read/write from/to DeAnza device
APLGEN:ZDEAXF.FOR	do IO to DeAnza TV
APLGEN:ZIVSOP.FOR	opens IVAS TV device - using the IIS package

APLGEN:ZM70C2.FOR	close IIS Model 70/75 TV device
APLGEN:ZM70O2.FOR	opens IIS Model 70.75 TV device
APLGEN:ZM70X2.FOR	read/write from/to IIS Model 70/75 device
APLGEN:ZTTBUF.FOR	reads terminal input with no prompt or wait - simulates TV trackball
APLGEN:ZV20C2.FOR	close Comtal Vision 1/20 TV device
APLGEN:ZV20MC.FOR	issue a master clear to the TV - for Comtal this is a No-Op
APLGEN:ZV20O2.FOR	opens Comtal Vision 1/20 TV device
APLGEN:ZV20X2.FOR	does I/O to Comtal Vision 1/20 TV device
APLGEN:ZVTVC2.FOR	close virtual TV connection to remote, real-TV computer
APLGEN:ZVTVC3.FOR	close connection in real-TV computer to client, virtual-TV computer
APLGEN:ZVTVCL.FOR	close connection in client (virtual-TV) to server (remote, real-TV)
APLGEN:ZVTVGC.FOR	close & reopen connection in server (real-TV) to client (virtual-tv)
APLGEN:ZVTVO3.FOR	open connection in server (real-TV) to client (virtual-TV)
APLGEN:ZVTVOP.FOR	opens connection from client (virtual-TV) to server (real-TV)
APLGEN:ZVTVRC.FOR	closes channel in server (real-TV) to client (virtual-TV)
APLGEN:ZVTVRO.FOR	open socket in server (real-TV) to any client (virtual-TV)
APLGEN:ZVTVRX.FOR	does IO for server (real TV) to client (Virtual-TV) incl close/reopen
APLGEN:ZVTVX2.FOR	writes/reads to/from server for the client (virtual TV) machine
APLGEN:ZVTVX3.FOR	reads/writes from/to client (virtual TV) for the server (real TV)
APLGEN:ZVTVXF.FOR	sends data from the client (virtual TV) to server (real TV)

B.1.46 TV-UTIL

YSUB:BLTFIL.FOR	fills in closed polygons on a tv "blotch" plane
APLSUB:CHAVRT.FOR	converts between local HOLL and local INT binary forms for transport
APLSUB:DECBIT.FOR	converts decimal coded number to bit coded (e.g. 13 -> 0000101)
YSUB:DLINTR.FOR	interactive delays, cursor tests, prevent wraparound
AIPSUB:GRPOLY.FOR	uses TV graphics to let user develop polygonal blotch regions
APLSUB:HDRBUF.FOR	translates AIPS header to/from FITS-standard integer form
YSUB:IENHNS.FOR	interactive linear enhancement of TV black & white LUTs
YSUB:ILNCLR.FOR	computes and loads a piecewise linear OFM to the TV
APLSUB:IMA2MP.FOR	converts pixel numbers in a TV-image into real image pixels
YSUB:IMANOT.FOR	draws a character string with black background to graphics
YSUB:IMCCLR.FOR	write color contour OFM to TV from standard sets
YSUB:IMCHAR.FOR	writes character string to TV
YSUB:IMLCLR.FOR	continuous colors fom blue thru green to red (or rotations thereof)
YSUB:IMPCLR.FOR	writes OFM with color contour helix in lightness-hue-saturation space
YSUB:IMVECT.FOR	draws connected line segments on TV
APLSUB:ISCALE.FOR	scale a buffer by various functions to an integer buffer (ie for TV)
APLSUB:MKYBUF.FOR	packages a command line into machine-independent form
APLSUB:MOVIST.FOR	sets/resets the movie status parameters in the TV common
APLSUB:MP2IMA.FOR	convert image pixel positions to TV pixel positions
APLSUB:REALOG.FOR	converts numbers between floating and an integer version of their log
APLSUB:REAVRT.FOR	converts between local REAL and local INT binary forms for transport
YSUB:TVCLOS.FOR	does error checks on device open, then closes the TV via YTVCLS
YSUB:TVFIDL.FOR	standard, simple interactive B&W LUT and color enhancements, zooming
AIPSUB:TVFIND.FOR	determines which of the visibile images on the TV the user desires
YSUB:TVLOAD.FOR	load image to a TV memory from open MA file
YSUB:TVOPEN.FOR	sets LUNs, calls YTVOPN to open the TV device, does error messages
YSUB:TVWHER.FOR	turns on cursor, waits for button, returns quadrant, position, button
APLSUB:TVWIND.FOR	determines image windows for TV, including for interpolation & Roam
APLSUB:UNYBUF.FOR	unpacks a machine-independent integer buffer into local command line
YGEN:YCHRW.FOR	writes characters into image and graphics planes
YGEN:YCINIT.FOR	initialize image catalog for specified TV memory plane

```
YGEN:YCNECT.FOR      write line segment between 2 points on TV
YGEN:YCOVER.FOR      checks for overlapped images on the TV by quadrant
YGEN:YCREAD.FOR      read the image catalog, return image header for TV only
YGEN:YCURSE.FOR      read and control TV cursor
YGEN:YCWRIT.FOR      write image header to image catalog, update image catalog directory
YGEN:YFILL.FOR       fill rectangle of TV memory with a constant value
YGEN:YFIND.FOR       determines the unique TV image of desired type, returns catalog block
YGEN:YGYHDR.FOR      builds basic TV IO header to write gray scale data
YGEN:YLOCAT.FOR      return TV positions for set of image positions
YGEN:YLOWOH.FOR      select least on bit in a bit mask integer
YGEN:YMAGIC.FOR      initialize graphics, zoom, scroll units for IIS Model 75 (level 3)
YGEN:YMKHDR.FOR      builds standard TV-IO header, used for IIS Models 70 and 75
YGEN:YSLECT.FOR      turn gray and graphics planes on and off
YGEN:YTCOMP.FOR      decide if a parameter has changed
YGEN:YZERO.FOR       fill a TV memory plane with zeros
APLGEN:ZVTVO2.FOR    open connection in client (virtual-TV) to server (remote, real-TV)
```

B.1.47 UTILITY

```
APLSUB:AP2SIZ.FOR    returns largest power of 2 not exceeding 1024 times first argument
APLSUB:BLDSNM.FOR    builds a name for a scratch file
APLSUB:BLDTNM.FOR    constructs full task name by appending NPOPS to task root name
APLSUB:BLTGLE.FOR    returns angle from A through a test position to B
APLSUB:BOUNDS.FOR    prints message if 1 or 2 values are outside a specified range
APLSUB:CATIME.FOR    stores current, or recovers previous, date and time in packed format
APLSUB:COMPAR.FOR    compares two integer arrays and returns .TRUE. if they are equal
APLSUB:COORDD.FOR    converts angles between degrees and sexagesimal format
APLSUB:COPY.FOR      copies integer words from one array to another
APLSUB:DAT2JD.FOR    converts date and time to a Julian date
APLSUB:DATDAT.FOR    converts "DD/MM/YY" form of date to "dd-mmm-yyyy" for printing
APLSUB:DTINIT.FOR    inits parameters for displaying elapsed CPU and real time w DTTIME
APLSUB:DTTIME.FOR    displays elapsed CPU and real times since last call to DTINIT
APLSUB:FILL.FOR      fills an integer array with an integer constant
APLSUB:FMATCH.FOR    returns pointer to location of small array in a bigger array
APLSUB:FRMT.FOR      encode floating number removing trailing zeros, alter accuracy if nec
APLSUB:GETRLS.FOR    returns the name of the current release (edited each quarter in CV)
APLSUB:GREG.FOR      converts Julian day number to date in character form
APLSUB:GTPARM.FOR    starts tasks, getting parameters and task ID number, does accounting
APLSUB:GTTELL.FOR    gets any parameters sent to task by AIPS verb TELL
APLSUB:H2CHR.FOR     convert AIPS Hollerith string to Fortran CHARACTER variable
APLSUB:IROUND.FOR    rounds a REAL to the nearest INTEGER
APLSUB:ITRIM.FOR     returns length of CHARACTER variable to last non-blank
APLSUB:JD2DAT.FOR    converts Julian day number to calendar date and time
APLSUB:JTRIM.FOR     clears nulls, returns length of CHARACTER variable to last non-blank
APLSUB:JULDAY.FOR    converts a character-encoded calendar date to Julian day number
APLSUB:LINTER.FOR    does linear interpolation of a 1-D INTEGER array
APLSUB:LSORT.FOR     sort a data buffer minimizing number times records are switched
APLSUB:METSCA.FOR    scale a value to the range 1-999 and provide a metric prefix to match
APLSUB:MSGWRT.FOR    writes messages to log file and/or terminal - a fundamental routine!
APLSUB:NAMEST.FOR    packs image name in string with leading and trailing blanks removed
APLSUB:NMATCC.FOR    returns next character in a string not matching a specified constant
APLSUB:NMATCH.FOR    returns next word in INTEGER array not matching a specified constant
APLSUB:RCOPY.FOR     copies one real array into another
APLSUB:REALOG.FOR    converts numbers between floating and an integer version of their log
```

APLSUB:RFILL.FOR	fills a real array with a constant
APLSUB:SETUP.FOR	does several task start up chores for non-interactive tasks
APLSUB:STRLIN.FOR	computes integer array as linear interpolation between two points
APLSUB:TIMDAT.FOR	convert integer time and date to character form for display
APLGEN:ZERROR.FOR	prints strings associated with system error codes for Z routines
APLGEN:ZMSGER.FOR	prints strings associated with system error codes for ZMSG routines

B.1.48 UV

QNOT:ALGSUB.FOR	Interpolates model visibility grom a grid and subtracts from uv data.
APLNOT:CALREF.FOR	Adjusts the reference antenna in an SN table.
QNOT:CCSGRD.FOR	Transforms CLEAN components to a grid.
APLNOT:DGGET.FOR	Selects uv data and changes Stokes
APLNOT:DGNEAD.FOR	Fills output CATBLK for UVGET
APLNOT:DGINIT.FOR	Sets arrays for selecting data and changing Stokes
QNOT:FFTIM.FOR	FFTs an image for uv interpolation.
APLNOT:FQMATC.FOR	Check if selection criteria match FQ table entries.
APLNOT:GAININ.FOR	Initializes calibration table for application.
APLNOT:GET1VS.FOR	Extract desired uv data, 1 value per freq. channel.
APLNOT:GETANT.FOR	Reads AN table and stores the info in common.
APLNOT:GETCTL.FOR	Determine Stokes' type of Clean map and other modeling info.
APLNOT:GETFQ.FOR	Find info on a given frequency id.
APLNOT:GETSOU.FOR	Find info on a given source id.
APLNOT:GETSTN.FOR	Reads the VLB station list opened in VBLIN and VBCIT
APLNOT:GNFSMO.FOR	Boxcar smooths and ASCAL solution (GA) file.
APLNOT:GNSMO.FOR	Optimized spline smoothing of amplitudes in ASCAL (GN) file.
QNOT:GRDCRM.FOR	Loads CLEAN components into AP for uv model computation.
APLNOT:GRDFLT.FOR	Sets default gridding convolution functions.
APLNOT:GRDSET.FOR	Creates scratch files and sets up for GRDSUB
QNOT:GRDSUB.FOR	Subtracts transform of CLEAN components from uv data.
APLNOT:GRIDTB.FOR	Makes a gridded image of the UV data in TB order.
APLNOT:INDXIN.FOR	Initializes index (NX) file, finds first scan selected.
QNOT:INTPFN.FOR	Computes interpolation kernals and put them into "AP memory".
APLNOT:IOBSRC.FOR	Search for antennas in the current bandpass buffer.
APLNOT:LXYPOL.FOR	Fills polarization correction table for AT like linear polarization.
QNOT:MAKMAP.FOR	Makes image or beam from uv data set.
APLNOT:MULSDB.FOR	Determines if a uv file is multi- or single- source.
APLNOT:NDXINI.FOR	Create/open/init index (NX) table
APLNOT:NXTFLG.FOR	Manages flagging info in tables in common.
APLNOT:PARANG.FOR	Computes antenna parallactic angles
APLNOT:POLSET.FOR	Fills polarization correction table from info in AN table.
QPSAP:Q1FIN.FOR	Finish gridding a row of uv data.
QPSAP:Q1GRD.FOR	Grid a uv data.
QPSAP:QFINGR.FOR	Finish gridding row of uv data.
QPSAP:QGADIV.FOR	Divide Gaus. model vis. into uv data.
QPSAP:QGASUB.FOR	Subtract Gaus. model vis. from uv data.
QPSAP:QGRD1.FOR	Convolves visibility data onto a grid.
QPSAP:QGRD2.FOR	Convolves linear polarization data onto a grid.
QPSAP:QGRD3.FOR	Convolve visibility data onto a grid.
QPSAP:QGRD4.FOR	Convolves visibility data onto a grid.
QPSAP:QGRDFI.FOR	Finish griding a row of uv data.
QPSAP:QGRDMI.FOR	Combined complex vector in gridding uv data.
QPSAP:QGRID.FOR	Grid uv data into row.
QPSAP:QGRIDA.FOR	Grid visibility data.

```
QPSAP:QINT.FOR      Interpolates model visibilityes from a grid.
QPSAP:QINTP.FOR     Interpolates model visibilities from a grid.
QPSAP:QMCALC.FOR    Compute model visibility from point model.
QPSAP:QPTDIV.FOR    Divide point model visibility into uv data.
QPSAP:QPTFAZ.FOR    zCompute phase in model visibilities.
QPSAP:QPTSUB.FOR    Subtract point model visibility from uv data.
QPSAP:QSPDIV.FOR    Divide Gaussian model  visibility into uv data.
QPSAP:QSPSUB.FOR    Subtract Gaussian model visibility from uv data.
QPSAP:QUVIN.FOR     Interpolate visibility model from a grid.
QPSAP:QUVINT.FOR    Interpolate model visibility from grid.
QPSAP:QXXPTS.FOR    Subtract point model visibility from uv data.
APLNOT:SCINTP.FOR   Interpolates bandpass tables in time.
APLNOT:SCLOAD.FOR   Copies part of one bandpass scratch file to another for efficiency.
APLNOT:SELINI.FOR   Initialize data selection and control in commons in DSEL.INC
APLNOT:SELSMG.FOR   Selects calibrator data, smooths solutions.
APLNOT:SET1VS.FOR   Sets up pointer and weights arrays for selecting uv data.
APLNOT:SETGDS.FOR   Sets up for UV model computation, fills common in DGDS.INC
APLNOT:SETGRD.FOR   Sets up for gridding uv data.
APLNOT:SETSM.FOR    Determines type of spectral smoothing and sets up look up table.
APLNOT:SETSTK.FOR   Sets STOKES parameters correctly for plotting routines
APLNOT:SMOSP.FOR    Convolves a spectrum with a tabulated function.
APLNOT:SN2CL.FOR    Apply an SN to a CL table.
APLNOT:SNAPP.FOR    Append SN tables and keep track of reference antennas.
APLNOT:SNINI.FOR    Create/open/initialize solution (SN) tables.
APLNOT:SNSMO.FOR    Smooths solution (SN) tables
APLNOT:SOUELV.FOR   Computes source hour angles and elevations
APLNOT:SOUFIL.FOR   Fills in arrays of source numbers to be included or excluded.
APLNOT:SOURNU.FOR   Look up source numbers for a list of names.
APLNOT:TABBL.FOR    Do IO to Baseline (BL) table after setup by BLINI.
APLNOT:TABBP.FOR    Does I/O to bandpass (BP) table opened by BPINI
APLNOT:TABCAL.FOR   Does I/O to Calibration (CL) table opened by CALINI
APLNOT:TABCS.FOR    Does I/O to single dish calibration (CS) table opened by CSINI
APLNOT:TABFLG.FOR   Does I/O to Flag (FG) table opened by FLGINI
APLNOT:TABFQ.FOR    Does I/O to frequency (FQ) table opened by FQINI
APLNOT:TABGA.FOR    Does I/O to GAIN (GA) table opened by GAINI
APLNOT:TABNDX.FOR   Does I/O to Index (NX) table opened by NDXINI
APLNOT:TABSN.FOR    Does I/O to Solution (SN) table opened by SNINI
APLNOT:TABSOU.FOR   Does I/O to Source (SU) table opened by SOUINI
APLNOT:TABTY.FOR    Does I/O to Tsys (TY) table opened by TYINI
APLNOT:TYINI.FOR    Create/open/initialize Tsys (TY) table
APLNOT:UVDOUT.FOR   Divides uv model in one half of a record into other, writes result.
APLNOT:UVDPAD.FOR   Reformat UV data record, doubling size and zero extra words.
APLNOT:UVGET.FOR    Read UV data with optional calibration, editing, selection, etc.
QNOT:UVGRID.FOR     Grids uv data to be FFTed.
QNOT:UVMDIV.FOR     Divides a uv data set by the Fourier transform of a model.
QNOT:UVMSUB.FOR     Subtracts the Fourier transform of a model from a uv data set.
QNOT:UVTBGD.FOR     Grids uv data in arbitrary sort order to be FFTed.
QNOT:UVTBUN.FOR     Determines and applies uniform weighting to uv data in arb. order.
QNOT:UVUNIF.FOR     Determines and applies uniform weighting to a uv data set.
APLNOT:VISCNT.FOR   Determines number of visibility records requested of UVGET
QNOT:VISDFT.FOR     Compute DFT of model and subtract/divide from/into uv data.
```

B.1.49 UV-UTIL

APLNOT:AN1ORS.FOR	Determines a list of antenna pairs from adverbs ANTANNA, BASELINE
APLSUB:ANTDAT.FOR	Returns the reference date and frequency for each array in uv dataset
APLNOT:CALCOP.FOR	Copies selected uv data with calibration and editing
APLSUB:GETVIS.FOR	uses setup from SETVIS to get and reformat a visibility sample
APLSUB:MERGE.FOR	sorts by merging previously sorted blocks of records
APLSUB:OSORT.FOR	does quick sort on array of vectors, then reorders by calling PERMAT
APLSUB:REQBAS.FOR	Apply ANTENNA and BASELINE selection adverbs to a baseline
APLSUB:SETVIS.FOR	initializes pointers to select/convert uv data to desired form
APLSUB:SNSORT.FOR	Shell sort of an array or records on two keys
APLSUB:UVCREA.FOR	create and catalog a uv data base file
APLSUB:UVDISK.FOR	reads/writes records of arbitrary length, esp UV data, see UVINIT
APLSUB:UVINIT.FOR	initializes IO for arbitrary length records via UVDISK, esp UV data
APLSUB:UVPGET.FOR	determines pointers to UV data from the header
AIPSUB:UWRITE.FOR	writes summary of UV Export-format tape
APLSUB:VISCHK.FOR	checks if UV data sample is desired, returns it in RR, LL, RL, LR

B.1.50 VLA

APLNOT:APLPBI.FOR	*TESS routine to apply a taper to an image. VLA only!
APLNOT:APPLPB.FOR	*TESS routine to apply a taper to an image

B.1.51 Y0

YGEN:YCINIT.FOR	initialize image catalog for specified TV memory plane
YGEN:YCNECT.FOR	write line segment between 2 points on TV
YGEN:YCOVER.FOR	checks for overlapped images on the TV by quadrant
YGEN:YCREAD.FOR	read the image catalog, return image header for TV only
YGEN:YCUCOR.FOR	correct cursor position for scroll; return image coordinates, header
YGEN:YCURSE.FOR	read and control TV cursor
YGEN:YCWRIT.FOR	write image header to image catalog, update image catalog directory
YGEN:YFILL.FOR	fill rectangle of TV memory with a constant value
YGEN:YFIND.FOR	determines the unique TV image of desired type, returns catalog block
YGEN:YLOCAT.FOR	return TV positions for set of image positions
YGEN:YLOWON.FOR	select least on bit in a bit mask integer
YGEN:YSLECT.FOR	turn gray and graphics planes on and off
YGEN:YTCOMP.FOR	decide if a parameter has changed

B.1.52 Y1

YGEN:YCRCTL.FOR	controls the TV cursor visibility, position; reads trackball buttons
YGEN:YGRAPN.FOR	turns TV graphics planes on and off
YGEN:YIMGIO.FOR	read/write data to the TV grey and graphics memories
YGEN:YINIT.FOR	initialize everything about the TV
YGEN:YLUT.FOR	read/write channel-based look-up-table
YGEN:YOFM.FOR	read/write all-channel look-up-table ("output function memory")
YGEN:YSCROL.FOR	write the scroll registers (shift location of 1 or more TV channels)
YGEN:YSPLIT.FOR	set channel selection by split-screen quadrant
YGEN:YTVCIN.FOR	initialize TV characteristics common (not needed much - see TVOPEN)
YGEN:YTVCLS.FOR	close the TV, including TV device and TV control/parameter disk file
YGEN:YTVMC.FOR	issue a master clear to reinitialize IO to the TV
YGEN:YTVOPN.FOR	open the TV device and the TV disk control/parameter file.
YGEN:YZERO.FOR	fill a TV memory plane with zeros
YGEN:YZOOMC.FOR	set the TV zoom magnification and center

B.1.53 Y2

YGEN:YALUCT.FOR	drives the TV arithmetic logic unit - not to be used much
YGEN:YCONST.FOR	controls the constant registers added to the TV picture - not used
YGEN:YFDBCK.FOR	causes a feedback operation in the TV
YGEN:YIFM.FOR	read/write TV Input look-up-table
YGEN:YMNMAX.FOR	read 3 min/max values from TV data paths (IIS only, not used)
YGEN:YRHIST.FOR	read the histogram of the selected TV output color
YGEN:YSHIFT.FOR	read/write the shift (bias) registers of the TV (IIS M70, not used)

B.1.54 Y3

YGEN:YGGRAM.FOR	controls the TV graphics color assignments
YGEN:YGRAFE.FOR	controls the graphics control register (IIS function)
YGEN:YGYHDR.FOR	builds basic TV IO header to write gray scale data
YGEN:YISDRM.FOR	read/write data memory of NRAO-ISU device
YGEN:YISDSC.FOR	read/write micro-processor memory of NRAO-ISU device
YGEN:YISJMP.FOR	cause microprocessor jump toaddress in NRAO-ISU device
YGEN:YISLOD.FOR	loads/unloads program memory of NRAO-ISU device
YGEN:YISMPM.FOR	reads/writes microprocessor memory of the NRAO-ISU device
YGEN:YMAGIC.FOR	initialize graphics, zoom, scroll units for IIS Model 75 (level 3)
YGEN:YMKCUR.FOR	selects the form of the cursor to be displayed
YGEN:YMKHDR.FOR	builds standard TV-IO header, used for IIS Models 70 and 75
YGEN:YSTCUR.FOR	reads/writes the cursor pattern array
YGEN:YTVCL2.FOR	close actual TV device (called by YTVCLS)
YGEN:YTVOP2.FOR	open actual TV device (called by YTVOPN)

B.1.55 Z

APLGEN:ZABORT.FOR	establishes or carries out (when appropriate) abort handling
APLGEN:ZACTV8.FOR	activate the requested program, returning process ID information
APLGEN:ZADDR.FOR	determine if 2 addresses inside computer are the same
APLGEN:ZARGCL.FOR	close an ARGS TV device
APLGEN:ZARGMC.FOR	issues a master clear to an ARGS TV
APLGEN:ZARGOP.FOR	open ARGS TV device
APLGEN:ZARGXF.FOR	translates IIS Model 70 commands into calls to ZARGS for ARGS TV
APLGEN:ZBKLD1.FOR	initialize environment for BAKLD
APLGEN:ZBKLD2.FOR	does BACKUP operation: load images from tape to directory
APLGEN:ZBKLD3.FOR	clean up system things for BAKLD ending
APLGEN:ZBKTP1.FOR	initialize BACKUP to tape operation for BAKTP
APLGEN:ZBKTP2.FOR	write a cataloged file plus extensions to BACKUP tape in BAKTP
APLGEN:ZBKTP3.FOR	clean up host environment at end of BAKTP
APLGEN:ZBYMOV.FOR	move 8-bit bytes from in-buffer to out-buffer
APLGEN:ZBYTFL.FOR	interchange bytes in buffer if needed to go between local & standard
APLGEN:ZC8CL.FOR	convert packed ASCII buffer to local character string
APLGEN:ZCLC8.FOR	convert local character string to packed ASCII buffer
APLGEN:ZCLOSE.FOR	closes open devices: disk, line printer, terminal
APLGEN:ZCMPRS.FOR	release space from the end of an open disk file
APLGEN:ZCPU.FOR	return current process CPU time and IO count
APLGEN:ZCREAT.FOR	creates a disk file
APLGEN:ZDATE.FOR	return the local date
APLGEN:ZDCHIN.FOR	initialize message, device and Z-routine characteristics commons
APLGEN:ZDEACL.FOR	close DeAnza TV device
APLGEN:ZDEAOP.FOR	opens DeAnza TV device
APLGEN:ZDEAXF.FOR	do IO to DeAnza TV

```
APLGEN:ZDELAY.FOR      delay current process a specified interval
APLGEN:ZDESTR.FOR      destroy a closed disk file
APLGEN:ZDNPRL.FOR      convert 64-bit HP floating buffer to local DOUBLE PRECISION values
APLGEN:ZDM2DL.FOR      convert Modcomp REAL*6 and REAL*8 to local double precision
APLGEN:ZENDPG.FOR      advance printer if needed to avoid electrostatic-printer "burn-out"
APLGEN:ZERROR.FOR      prints strings associated with system error codes for Z routines
APLGEN:ZEXIST.FOR      return file size and, consequently, whether file exists
APLGEN:ZEXPND.FOR      expand an open disk file --- either map or non-map now allowed
APLGEN:ZFIO.FOR        reads and writes single 256-integer records to non-map disk files
APLGEN:ZFREE.FOR       display available disk space
APLGEN:ZGETCN.FOR      get a character from a REAL word
APLGEN:ZGNAME.FOR      get name of current process
APLGEN:ZGTBIT.FOR      get array of bits from a word
APLGEN:ZHEX.FOR        encode an integer into hexadecimal characters
APLGEN:ZIPACK.FOR      pack/unpack long integers into short integer buffer
APLGEN:ZKDUMP.FOR      display portions of an array in various Fortran formats
APLGEN:ZLASIO.FOR      open, write to, close and spool a laser printer print/plot file
APLGEN:ZLPCLS.FOR      close an open printer device
APLGEN:ZLPOPN.FOR      open a line-printer text file
APLGEN:ZLWIO.FOR       open, write to, close and spool a PostScript print/plot file
APLGEN:ZM70MC.FOR      issues a master clear to an IIS Model 70 TV
APLGEN:ZM70OP.FOR      open IIS Model 70 TV device
APLGEN:ZM70XF.FOR      read/write data to IIS Model 70 TV with buffering
APLGEN:ZMIO.FOR        random-access, quick return (double buffer) disk IO for large blocks
APLGEN:ZMKTMP.FOR      convert a "temporary" file name into a unique name
APLGEN:ZMOUNT.FOR      mount or dismount magnetic tape device
APLGEN:ZMSGCL.FOR      close Message file or terminal
APLGEN:ZMSGDK.FOR      disk IO to message file
APLGEN:ZMSGER.FOR      prints strings associated with system error codes for ZMSG routines
APLGEN:ZMSGOP.FOR      open a message file or message terminal
APLGEN:ZMSGXP.FOR      expand the message file
APLGEN:ZOPEN.FOR       open binary disk files and line printer and TTY devices
APLGEN:ZPNFIL.FOR      construct a physical file or device name from AIPS logical parameters
APLGEN:ZPNOLV.FOR      construct a physical file - version for UPDAT
APLGEN:ZPRIO.FOR       raise or lower the process priority
APLGEN:ZPRMPT.FOR      prompt user and read 80-characters from CRT screen
APLGEN:ZPRPAS.FOR      prompt user and read 12-character password (invisible) from CRT
APLGEN:ZPTBIT.FOR      put array of bits into a word
APLGEN:ZPUTCH.FOR      inserts 8-bit "character" into a word
APLGEN:ZRDMF.FOR       convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers
APLGEN:ZRENAM.FOR      rename a disk file
APLGEN:ZRLR64.FOR      convert buffer of local double precision values to IEEE 64-bit float.
APLGEN:ZRM2RL.FOR      convert Modcomp to local single precision floating point
APLGEN:ZSTAIP.FOR      does any system cleanup needed at the end of interactive AIPS session
APLGEN:ZTACTQ.FOR      inquires if a task is currently active on the local computer
APLGEN:ZTAPE.FOR       mount, dismount, position, write EOF, etc. for tapes
APLGEN:ZTAPIO.FOR      tape operations for IMPFIT (compressed FITS transport tape)
APLGEN:ZTIME.FOR       return the local time of day
APLGEN:ZTOPEN.FOR      open text file - logical area, version, member name as arguments
APLGEN:ZTPWAT.FOR      wait for asynchronous IO to finish on tape or pseudo-tape disk
APLGEN:ZTQSPY.FOR      display AIPS account or all processes running on the system
APLGEN:ZTTBUF.FOR      reads terminal input with no prompt or wait - simulates TV trackball
APLGEN:ZTXCLS.FOR      clos text file opened via ZTXOPN
APLGEN:ZTXIO.FOR       read/write a line to a text file
```

```
APLGEN:ZTXMAT.FOR     return list of files in specified area beginning with specified chars
APLGEN:ZTXOPN.FOR     open a text file for read or write
APLGEN:ZV20CL.FOR     close a Comtal Vision 1/20 TV device
APLGEN:ZV20MC.FOR     issue a master clear to the TV - for Comtal this is a No-Op
APLGEN:ZV20OP.FOR     open Comtal Vision 1/20 TV device
APLGEN:ZV20XF.FOR     read/write data to Comtal Vision 1/20 TV device
APLGEN:ZWHOMI.FOR     determines AIPSxn task name; sets NPOPS, assigns TV and TK devices
```

B.1.56 Z2

```
APLGEN:ZABOR2.FOR     establishes or carries out (when appropriate) abort handling
APLGEN:ZARGC2.FOR     close an ARGS TV device
APLGEN:ZARGO2.FOR     open ARGS TV device
APLGEN:ZARGS.FOR      sends command to/from the ARGS TV device
APLGEN:ZBYTF2.FOR     interchange bytes in buffer if needed to go between local & standard
APLGEN:ZCMPR2.FOR     truncate a disk file, returning blocks to the system
APLGEN:ZCREA2.FOR     create the specified disk file
APLGEN:ZDACLS.FOR     close a disk file
APLGEN:ZDAOPN.FOR     open the specified disk file
APLGEN:ZDCHI2.FOR     initialize device and Z-routine characteristics commons - local vals
APLGEN:ZDCHIC.FOR     set more system parameters; make them available to C routines
APLGEN:ZDEAC2.FOR     close DeAnza TV device
APLGEN:ZDEAO2.FOR     opens DeAnza TV device
APLGEN:ZDELA2.FOR     delay current process a specified interval
APLGEN:ZDEST2.FOR     destroy a closed disk file
APLGEN:ZDIR.FOR       build a full path name to files in AIPS-standard areas (HE, RU, ...)
APLGEN:ZERRO2.FOR     return system error message for given system error code
APLGEN:ZEXIS2.FOR     return size of disk file and if  it exists
APLGEN:ZEXPN2.FOR     expand an open disk file
APLGEN:ZFI2.FOR       read/write one 256-integer record from/to a non-map disk file
APLGEN:ZFRE2.FOR      return AIPS data disk free space information
APLGEN:ZLASC2.FOR     spool a closed laser printer print/plot file
APLGEN:ZLASCL.FOR     close and spool a laser printer print/plot file
APLGEN:ZLASOP.FOR     open a laser printer print/plot file
APLGEN:ZLPCL2.FOR     queue a file to the line printer and delete
APLGEN:ZLPOP2.FOR     open a line-printer text file - actual OPEN call
APLGEN:ZM70C2.FOR     close IIS Model 70/75 TV device
APLGEN:ZM70O2.FOR     opens IIS Model 70.75 TV device
APLGEN:ZMI2.FOR       read/write large blocks of data from/to disk, quick return
APLGEN:ZMOUN2.FOR     mount or dismount magnetic tape device
APLGEN:ZMSGWR.FOR     call MSGWRT based on call arguments - for C routines to call MSGWRT
APLGEN:ZPATH.FOR      convert a file name
APLGEN:ZPRI2.FOR      raise or lower the process priority
APLGEN:ZRENA2.FOR     rename a file
APLGEN:ZSTAI2.FOR     does any system cleanup needed at the end of interactive AIPS session
APLGEN:ZTACT2.FOR     inquires if a task is currently active on the local computer
APLGEN:ZTAP2.FOR      position (forward/back record/file), write EOF, etc. for tapes
APLGEN:ZTKCL2.FOR     close a Tektronix device
APLGEN:ZTKOP2.FOR     read/write from/to a Tektronix device
APLGEN:ZTOPE2.FOR     open text file for ZTOPEN
APLGEN:ZTPCL2.FOR     close a tape device
APLGEN:ZTPMI2.FOR     tape read/write
APLGEN:ZTPMID.FOR     pseudo-tape disk read/write for 2880-bytes records
APLGEN:ZTPOP2.FOR     open a tape device for double-buffer, asymchronous IO
```

```
APLGEN:ZTPOPD.FOR      open a pseudo-tape, sequential disk file for FITS
APLGEN:ZTPWA2.FOR      wait for read/write from/to a tape device
APLGEN:ZTQSP2.FOR      display AIPS account or all processes running on the system
APLGEN:ZTTCLS.FOR      close a terminal device
APLGEN:ZTTOP2.FOR      open a message terminal
APLGEN:ZTTOPN.FOR      open a terminal device
APLGEN:ZTXMA2.FOR      find all file names matching a given wildcard specification
APLGEN:ZTXOP2.FOR      translate the file name and open a text file
APLGEN:ZV20C2.FOR      close Comtal Vision 1/20 TV device
APLGEN:ZV20O2.FOR      opens Comtal Vision 1/20 TV device
APLGEN:ZV20X2.FOR      does I/O to Comtal Vision 1/20 TV device
APLGEN:ZWAI2.FOR       wait for read/write large blocks of data from/to disk
```

Index