Going AIPS:
A Programmers Guide to the NRAO Astronomical Image Processing System

W. D. Cotton and a cast of AIPS

Version 15 April 87

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 contributing to this manual:

John Benson - VLBI
Bill Cotton - Array processors and applications routines
Eric Greisen - Head knocker, TVs and AIPS system integration
Kerry Hilldrup - UNIX, CRAY COS implementation
Fred Schwab - Mathematics, critic
Craig Walker - Gadfly and VLBI
Don Wells - Hardware implementations and image processing techniques

CONTENTS

CHAPTER 3 GETTING STARTED - TASKS

CHAPTER 7 HIGH LEVEL UTILITY ROUTINES

CHAPTER 8 WAWA ("EASY") I/O

VOLUME 2 CONTENTS

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 used in conjunction with the AIPS manual, especially volumes 2 and 3 and should 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, 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 (hopefully 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 to 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 adverbs 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 except that the data is normally in floating point format. Associated with each main data file may be up to 10 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.

A third type of file is the text file. At the moment AIPS only reads text files. The files are used mostly for documentation although the file which defines the list of adverbs to be sent to a task is a text file. More details about the I/O routines can be found in the chapter on I/O.

1.5 STYLE

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. As a simple example, consider:

```
      SUBROUTINE COPY (N, KFROM, KTO)
C-------------------------------------------------------
C   COPY copies integer words from one array to another.
C   Inputs:  N      I*2     number of words to be copied
C            KFROM  I*2(N)  source array
C   Outputs: KTO    I*2(N)  destination array
C-------------------------------------------------------
      INTEGER*2  N, KFROM(1), KTO(1)
C-------------------------------------------------------
C                                          no copy: N <= 0
      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 are prime candidates for indentation. As an example, consider:

```
C                                              Multiply by transform matrix
      DO 10 I = 1,3
         VEC(I) = 0.0
         DO 10 J = 1,3
            VEC(I) = VEC(I) + TMATX(I,J)*VECO(J)
 10         CONTINUE
C                                              Unit vector to polar
C                                              Case at pole
      IF ((X.NE.0.0) .OR. (Y.NE.0.0)) GO TO 20
         ALPHA = 0.0
         DELTA = 0.0
         GO TO 30
 20   CONTINUE
         ALPHA = ATAN (X, Y)
         DELTA = SQRT (X*X + Y*Y)
 30   PDIST = ATAN2 (Z, DELTA)
C                                              Swap to increasing order
      IF (A.LT.B) GO TO 40
         C = A
         A = B
         B = C
 40   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. A branch other than those caused by DO loops should be to executable statements.

1.5.5 Statement Numbers

The use of GO TO statements is the cause of most logic errors in programming. However, with the use of standard indentation and statement numbering schemes, errors can be reduced and readability enhanced. 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.

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)) GO TO 20
```

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 16-bit computers. The task should be designed in a modular way to allow it to be overlaid on the "smaller" machines. Although this consideration is, we hope, diminishing, programmers must remember that 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. Programmers for the system must be aware of this requirement and avoid the easy assumptions about such matters as word and character lengths. The basic common /DCHCOM/ contains parameters giving the number of bits/word, words/floating point, words/double precision floating point, and characters/floating point. These must be used, rather than simple

equivalence statements, when dealing with "data structures" (arrays containing a mixture of integer, character, and floating point variables). One may use DATA statements to assign two characters to an integer and four characters to a real and then use formats A2 and A4, respectively, to print them. However, one cannot regard these variables as being fully packed with characters. The technique one must use to handle data structures, such as the map catalog data block described later in this manual, goes as follows: One equivalences integer, real, and double precision arrays to the full structure. Then one computes, using the parameters in /DCHCOM/, the subscripts needed with the three types of arrays to extract the desired quantities. The routine VHDRIN performs this computation for catalog blocks, storing its results in the common /HDRCOM/. Programmers will find this routine instructive. There are a wide variety of service routines to manipulate characters and to compute addresses.

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, except for the addition of INCLUDE, ENCODE, and DECODE statements and the use of a minimum number of local assembly language (or C) Z routines when absolutely required. I cannot review the entire language here, but I urge programmers 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, I remind programmers 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 of no more than 8 characters, 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.

2. ENCODE (‹nchar› , ‹format› , ‹array›) ‹list›

   where ‹nchar› is the total number of characters to be encoded, ‹format› is the format number, ‹array› is the variable into which the data are to be encoded, and ‹list› is an optional list of the variables whose values are to be encoded. The value of ‹nchar› may exceed the actual number of characters to be encoded, but may not exceed the number of characters which will fit in ‹array›. ENCODE performs a formatted write into memory.

3. DECODE (‹nchar› , ‹format› , ‹array›) ‹list›

   where ‹nchar› is the total number of characters to be decoded, ‹format› is the format number, ‹array› is the variable from which the data are to be decoded, and ‹list› is the list of variables to receive the decoded values. DECODE performs a formatted read from memory.

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 or so minus signs). 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, some of the simpler compilers do not. Therefore, in AIPS, we will use the following order:

1. Data type and dimension statements: INTEGER*2, INTEGER*4, LOGICAL*2, REAL*4, REAL*8 and COMPLEX in any order. We prohibit DIMENSION, INTEGER(see below), REAL, DOUBLE PRECISION, INTEGER*3, LOGICAL*1, LOGICAL*4, REAL*6, COMPLEX*8, and COMPLEX*16 statements 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. Also, INTEGER is to be used to declare variables to be used for variable dimension statements. This INTEGER statement should appear before the statement using the variable dimension.

   PARAMETER statements should be included with (usually before) the declaration statements.

2. Common statements: COMMON. 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.

3. Equivalence statements: EQUIVALENCE.

4. Data initialization statements: DATA. We prohibit the use of DATA statements to initialize variables in commons (as do the FORTRAN standards and many compilers). Character data must be typed correctly. Thus, although

```
INTEGER*2 IC(2)
DATA IC /'IAMC'/
```

   will work on many computers, we prohibit it. The use of octal and hexadecimal numbers in data statements is strongly discouraged.

5. 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 a preprocessor on some systems), 'a' is D, C, E, and V for the above types 1, 2, 3, and 4, respectively and 'ccc' is a one to three character name for the INCLUDE. In addition, a prefix P indicates a parameter include; PARAMETER includes must precede all other types of include. Since the statement order is fixed, an include text file may contain statements of only one of the above types. For example,

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

causes the text:

```
C                                                 Include DBWT
      INTEGER*2 BWTNUM, BWTLUN, BWTIND, BWTREC, BWTDAT(1)
      LOGICAL*2 WASERR
      REAL*4    BWTNAM(6)
C                                                 End DBWT
C                                                 Include CBWT
      COMMON /BWTCH/ BWTNAM, BWTNUM, BWTLUN, BWTIND, BWTREC,
     *   WASERR, BWTDAT
C                                                 End CBWT
```

to be inserted.

Note: I is sometimes used as the first letter of a declaration include which contains an array which should be explicitly declared in the routine. An example is IDCH.INC for the device characteristics common. The CDCH.INC file includes a common used to carry an array called FTAB which is used for I/O tables. The size of FTAB depends on the number of I/O streams desired concurrently and should be declared in the main routine of the program. Thus in the main routine of each program the includes INCS:IDCH.INC and INCS:CDCH.INC should appear as well as an INTEGER*2 declaration for FTAB. (See the chapter on I/O for a discussion of the required dimension of FTAB.)

1.6.4 Variable Declaration

The programmer is required to declare every variable in the routine. This will avoid any problems with the various default data types in various computer systems. Of particular importance, in

this regard, are those variables and constants which appear in CALL statements. Using the example of the subroutine COPY given above, the statement

```
CALL COPY (2, KF, KT)
```

will work on some machines, but will not work on computers which default to INTEGER*4 with an address which points to the high-order byte. The right way to code this is:

```
INTEGER*2  KF(n), KT(n), N2
...
DATA  N2 / 2 /
...
CALL COPY (N2, KF, KT)
```

All declaration statements must begin in column 7.

1.6.5 Literals And Expressions In CALL Statements

Literals (e.g., 2) and expressions should NEVER be used in CALL sequences. The data type received by the routine called may not be what it (or the programmer) expects.

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

1.7.1.1 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.

1.7.1.2 AIPS Manual And Cookbook - The AIPS manual and especially the AIPS Cookbook are employed by many AIPS users as a guide to using AIPS. In particular, many users are unaware of the existence of any feature in AIPS not advertised in the Cookbook; unfortunately, the Cookbook only covers the most elementary portions of the AIPS system. The AIPS manual and the Cookbook are maintained by Eric Greisen in Charlottesville.

1.7.2 Programmer Documentation

1.7.2.1 Precursor Comments - The most fundamental source of detailed programmer documentation in the AIPS system are comments in the source code, especially the precursor comments. A listing of all of the precursor comments in the AIPS system can be found in the AIPS manual volume 3. 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.

1.7.2.2 Shopping Lists - There are a number of list of AIPS routines with one-line descriptions of their functions. These lists are a good place to discover what utility routines are available. Unfortunately, since these lists are currently maintained by hand, they are woefully out of date. They are still useful, however.

1.7.2.3 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 CHANGE.DOC file associated with the master version of the AIPS system is published quarterly in the AIPSletter.

1.7.2.4 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 obtained from Eric Greisen in Charlottesville.

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.

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.

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 ENCODE to
C   record up to 64 characters per entry into array HISCRD.  Format:
C          ENCODE (64,format #,HISCRD(1,entry #)) list
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 COMMON /MSGCOM/
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*4  Visibility number, -1 => final call, no data
```

```
C                        passed but allows any operations to be completed
C      U          R*4    U in wavelengths
C      V          R*4    V in wavelengths
C      W          R*4    W in wavelengths
C      T          R*4    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 R*8 form by:
C                        CALL JULDAY (CAT4(K4DOB),XDAY) where XDAY will
C                        be the Julian day number.
C      IA1        I*2    First antenna number
C      IA2        I*2    Second antenna number
C      RPARM(*)   I*2    Random parameter array which includes U,V,W etc
C                        but also any other random parameters.
C      VIS(INCX,*) R*4    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(3)   R*4    Name of the aux. file (12 char)
C      CLAS2(2)   R*4    Class of the aux. file (6 char)
C      SEQ2       I*2    Sequence number of the aux. file.
C      DISK2      I*2    Volume number of the aux. file.
C      APARM(10)  R*4    User array.
C      BPARM(10)  R*4    User array.
C      BOX(4,10)  R*4    User array.
C      RA         R*8    Right ascension (1950) of phase center. (deg)
C      DEC        R*8    Declination (1950) of phase center. (deg)
C      FREQ       R*8    Frequency of observation (Hz)
C      NRPARM     I*2    # random parameters.
C      NCOR       I*2    # correlators
C      CATBLK(256)I*2    Catalog header record. See Going Aips for
C                        details.
C
C   Output:
C      U          R*4    U in wavelengths
C      V          R*4    V in wavelengths
C      W          R*4    W in wavelengths
C      T          R*4    Time in same units as input.
C      RPARM      R*4    Modified random parameter array.  N.B.  U,V,W,
C                        time, baseline should not be modified in RPARM
C      VIS        R*4    Visibilities
C      IRET       I*2    Return code  -1 => don't write
C                                       0 => OK
C                                      >0 => error, terminate.
C
C   Output in COMMON
C      NUMHIS             I*2  # history entries (max. 10)
C      HISCRD(16,NUMHIS) R*4  History records
C      CATBLK             I*2  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, and integer or floating format 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 statements 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; i.e.,
C      CATBLK(K2DIM)   = the number of axes,
C      CATBLK(K2NAX+i) = the dimension of each axis.
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 statements.
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) Setting 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 statement.
C
C    Input:
C     CATBLK(256)     I*2   Output catalog header, also CAT4, CAT8
C     CATOLD(256)     I*2   Input catalog header, also OLD4, OLD8
C    Output:
C     CATBLK(256)     I*2   Modified output catalog header.
C     IRET            I*2   Return error code, 0=>OK, otherwise abort.
C-----------------------------------------------------------------------
      INTEGER*2 LIMIT, I, FIRSTI, FIRSTO, N1, N4, N8, IRET, IFPC
      REAL*4    CAT4(128), OLD4(128)
      REAL*8    CAT8(64), OLD8(64)
      INTEGER*2 SEQIN, SEQOUT, DISKIN, DISKO, NEWCNO, OLDCNO,
     *   CATOLD(256), CATBLK(256), NUMHIS, JBUFSZ, ICODE
      LOGICAL*2 DROP1
      REAL*4    NAMEIN(3), CLAIN(2), XSEQIN, XDISKI, NAMOUT(3),
     *   CLAOUT(2), XSEQO, XDISKO, BLC(7), TRC(7), OPCODE, CPARM(10),
     *   DPARM(10), HISCRD(16,10), FBLANK
      INTEGER*2 NCODE, NTYPES, IOFF, IERR, INDXI, INC, INDEX,
     *   NCHTYP(10)
      REAL*4    CODES(10), UNITS(2,10), ATYPES(2,10), BLANK(2), TEMP,
     *   FCHARS(3)
      LOGICAL*2 LDROP1
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CHDR.INC'
      COMMON /INPARM/ NAMEIN, CLAIN, XSEQIN, XDISKI, NAMOUT, CLAOUT,
     *   XSEQO, XDISKO, BLC, TRC, OPCODE, CPARM, DPARM
      COMMON /PARMS/ FBLANK,
     *   DROP1,
     *   SEQIN, SEQOUT, DISKIN, DISKO, NEWCNO, OLDCNO,
     *   CATOLD, JBUFSZ, ICODE
      COMMON /HISTRY/ HISCRD, NUMHIS
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE  (CATBLK, CAT4, CAT8),   (CATOLD, OLD4, OLD8)
      DATA FCHARS /'FREQ','VELO','FELO'/
      DATA N1, N4, N8  /1,4,8/,          BLANK /2*'    '/
C                                        User definable values
C                                        # and value of OPCODEs
      DATA NCODE /0/
      DATA CODES /10*'    '/
C                                        Output units for each OPCODE.
C                                        Two R*4 words with 4 char. ea.
      DATA UNITS /'UNDE','FINE',18*'    '/
```

```
C                                          Allowed number of axis types
C                                          and types.
      DATA NTYPES /0/
      DATA ATYPES /20*'    '/
      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 DATA are Real*4 with blanking if necessary; output values
C   are R*4 which may also be blanked.  The calling routine keeps track
C   of max., min. and the occurrence of blanking.  If DROP1 is .TRUE.,
C   the calling routine expects 1 value returned per call;
C   otherwise, CATBLK(K2NAX) values per call are expected returned.
C   NOTE: blanked values are denoted by the value of the common
C   variable 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 by using ENCODE to
C   record up to 64 characters per entry into array HISCRD. Ex:
C         ENCODE (64,format #,HISCRD(1,entry #)) 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       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(7)    I*2  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*4  Input row, magic value blanked.
C   Values from commons:
C      ICODE      I*2  Opcode number from list in NEWHED.
C      FBLANK     R*4  Value of blanked pixel.
C      CPARM(10)  R*4  Input adverb array.
C      DPARM(10)  R*4  Input adverb array.
C      CATBLK     I*2  Output catalog header (also CAT4, CAT8)
C      CATOLD     I*2  Input catalog header (also OLD4, OLD8)
C      DROP1      L*2  True if one output value per call.
C   Output:
C      RESULT(*)  R*4  Output row.
C      IRET       I*2  Return code   0 => OK
C                                   >0 => error, terminate.
C   Output in COMMON
C     NUMHIS      I*2  # history entries (max. 10)
C     HISCRD(16,NUMHIS) R*4 History records
C     CATBLK      I*2  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.

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 (GETUV 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. Since clones of this task are likely to be specialized, some of the AIPS transportability requirements may be relaxed. In particular, the source code for UVFIL expects the names of external text files to be opened and read by normal Fortran calls.

UVFIL comes with specific example code reading such a file. The first routine, NEWHED, which the user may need to modify is needed to enter information used 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 the routine in which the catalog header is constructed.
C   Necessary values can be read in in the areas marked "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; i.e.,
C      CAT3(K3GCN)   = I*4 number of visibility records
C      CATBLK(K2PCN) = Number of random parameters.
C      CATBLK(K2DIM)= the number of axes,
C      CATBLK(K2NAX+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, see the chapter in GOING AIPS on Calibration and
C   Editing.
C
C    Input:
C     CATBLK(256)    I*2   Output catalog header, also CAT4, CAT8
C                          The OUTNAME, OUTCLASS, OUTSEQ are entered
C                          elsewhere.
C    Output:
C     CATBLK(256)    I*2   Modified output catalog header.
C     IRET           I*2   Return error code, 0=>OK, otherwise abort.
C    Also the antenna information can be filled into a common.
C-----------------------------------------------------------------------
      INTEGER*2 CATBLK(256), SEQOUT, I, NAXIS, NRAN, ANTSYM(30), NCHAN,
     *   NPOLN, DISKO, JBUFSZ, IERR, NANT, NDIM(7), INDEX, INC, ISTAR,
     *   NO, N1, N2, N8, N256, IFPC, IRET
      INTEGER*4 CAT3(128), XCOUNT
      REAL*4    INFILE(12), IN2FIL(12), TYPES(2,7), RTYPES(2,7),
     *   NAMOUT(3), CLAOUT(2), XSOUT, XDISO, APARM(10), BPARM(10),
     *   BUFFER(1600), CAT4(128), ANTNAM(2,30), XIAT, CRPIX(7),
     *   CRINC(7), UNITS(2), BANDW, TELE(2), OBSR(2), INSTR(2),
     *   BLANK(2), XUT1, OBSDAT(2), EPOCH
      REAL*8    CAT8(64), ANTLOC(3,30), GSTO, CRVAL(7)
C***********************************************************************
C   SAMPLE CODE
C
C      CHARACTER*48 FLNAME
C***********************************************************************
      INCLUDE 'INCS:PUVD.INC'
      INCLUDE 'INCS:DDCH.INC'
```

```
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DUVH.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CHDR.INC'
      INCLUDE 'INCS:CUVH.INC'
      COMMON /ANTS/ ANTLOC, GSTO, XIAT, XUT1, ANTNAM, NANT,
     *   ANTSYM
      COMMON /BUFRS/ BUFFER, JBUFSZ
      COMMON /INPARM/ INFILE, IN2FIL, NAMOUT, CLAOUT, XSOUT, XDISO,
     *   APARM, BPARM, SEQOUT, DISKO
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE  (CATBLK, CAT3, CAT4, CAT8)
      DATA NO, N1, N2, N8, N256 /0,1,2,8,256/,  BLANK /2*'    '/
      DATA ISTAR /'**'/
C                                          User definable values:
C                                          Random parameters.
C                                          No. random parameters.
      DATA NRAN /5/
C                                          Rand. parm. names.
      DATA RTYPES /'UU-L','    ','VV-L','    ','WW-L','    ',
     *   'TIME','1   ','BASE','LINE',4*'    '/
C                                        Uniform axes.
C                                          No. axes.
      DATA NAXIS /5/
C                                          Axes names.
      DATA TYPES /'COMP','LEX ','STOK','ES  ','FREQ','    ',
     *   'RA  ','    ','DEC ','    ',4*'    '/
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 ENCODE to
C  record up to 64 characters per entry into array HISCRD. Ex:
C     ENCODE (64,format #,HISCRD(1,entry #)) list
C  The history is written after the last call to FIDDLE.
C
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   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 and 5
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*4  Visibility number, -1 => final call, no data
C                   passed but allows any operations to be completed.
C
C  Inputs from COMMON
C  IN2FIL(12) R*4  Name of the aux. file (48 char)
C  APARM(10)  R*4  User array.
C  BPARM(10)  R*4  User array.
C  RA         R*8  Right ascension (1950) of phase center. (deg)
C  DEC        R*8  Declination (1950) of phase center. (deg)
C  FREQ       R*8  Frequency of observation (Hz)
C  NRPARM     I*2  # random parameters.
C  NCOR       I*2  # correlators
C  CATBLK(256)I*2  Catalog header record. See [DOC]HEADER for details
C
C  Output:
```

```
C  U           R*4  U in wavelengths at the reference frequency.
C  V           R*4  V in wavelengths
C  W           R*4  W in wavelengths
C  T           R*4  Time in days since the midnight at the start of
C                   the reference date.
C  IA1         I*2  Antenna number of the first antenna.
C  IA2         I*2  Antenna number of the second antenna.
C                   NOTE: IA2 MUST be greater that IA1
C  RPARM       R*4  Modified random parameter array. NB U,V,W,
C                   time and baseline should not be modified in RPARM
C  VIS(3,*)    R*4  Visibilities.  The first dimension is the COMPLEX
C                   axis in the order Real part, Imaginary part, weight.
C                   The order of the following visibilities is defined
C                   by variables in COMMOM /UVHDR/ (originally
C                   specified in NEWHDR).  The order number for Stokes
C                   parameters is JLOCS and the order number for
C                   frequency is given by JLOCF.  The lower order number
C                   increases faster in the array.
C                   See precursor comments in UVPGET for more details.
C  IRET        I*2  Return code  -1 => End of data.
C                                 0 => OK
C                                >0 => error, terminate.
C
C  Output in COMMON
C  NUMHIS      I*2  # history entries (max. 10)
C  HISCRD(16,NUMHIS) R*4 History records
C  CATBLK      I*2  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.

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 the 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 statements 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; i.e.,
C      CATBLK(K2DIM)= the number of axes,
C      CATBLK(K2NAX+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 statement.
C
C    Input:
C     CATBLK(256)     I*2   Output catalog header, also CAT4, CAT8
C                           The OUTNAME, OUTCLASS, OUTSEQ are entered
C                           elsewhere.
C    Output:
C     CATBLK(256)     I*2   Modified output catalog header.
C     IRET            I*2   Return error code, 0=>OK, otherwise abort.
C-----------------------------------------------------------------------
      INTEGER*2 LIMIT, I, NAXIS, N1, N8, IRET, IFPC, IROUND
      REAL*4    CAT4(128)
      REAL*8    CAT8(64)
      INTEGER*2 SEQOUT, DISKO, NEWCNO, CATBLK(256), NUMHIS, JBUFSZ,
     *   ICODE
      REAL*4    FILEIN(12), SOURCE(2), XMSIZE(2), CELLS(2),
     *          NAMOUT(3), CLAOUT(2), XSEQO, XDISKO,
     *          OPCODE, CPARM(10), DPARM(10),
     *          HISCRD(16,10), FBLANK
      INTEGER*2 NCODE, NTYPES, IOFF, IERR, INDXI, NX, NY,
     *   INC, INDEX
      REAL*4    CODES(10), UNITS(2,10), ATYPES(2,7),
     *   BLANK(2), TEMP, FCHARS(3)
C***********************************************************************
C   SAMPLE CODE
C
C      CHARACTER*48 INFILE
C***********************************************************************
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CHDR.INC'
      COMMON /INPARM/ FILEIN, SOURCE, XMSIZE, CELLS,
     *   NAMOUT, CLAOUT, XSEQO, XDISKO,
```

```
      *   OPCODE, CPARM, DPARM
       COMMON /PARMS/ FBLANK, SEQOUT, DISKO, NEWCNO,
      *   JBUFSZ, ICODE
       COMMON /HISTRY/ HISCRD, NUMHIS
       COMMON /MAPHDR/ CATBLK
       EQUIVALENCE  (CATBLK, CAT4, CAT8)
       DATA FCHARS /'FREQ','VELO','FELO'/
       DATA N1, N8 /1,8/,  BLANK /2*'    '/
C                                          User definable values
C                                          # and value of OPCODEs
       DATA NCODE /0/
       DATA CODES /10*'    '/
C                                          Output units for each OPCODE.
C                                          Two R*4 words with 4 char. ea.
       DATA UNITS /'UNDE','FINE',18*'    '/
C                                          Number of axes and types.
C                                          (Set for two axes = Ra, Dec.)
       DATA NAXIS  /2/
       DATA ATYPES /'RA--','-SIN','DEC-','-SIN',
      *   'STOK','ES  ','FREQ','    ',
      *   6*'    '/
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  to user to create an image, one row at a time.
C  Output values are R*4 which may  be blanked.
C  The calling routine keeps track of max., min. and the occurrence of
C  blanking.  NOTE: blanked values are denoted by the value of the
C  common variable FBLANK
C  CATBLK(K2NAX) values per call are expected returned.
C
C       Up to 10 history entries can be written by using ENCODE to
C  record up to 64 characters per entry into array HISCRD. Ex:
C     ENCODE (64,format #,HISCRD(1,entry #)) 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 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 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(7)    I*2  BLC (input image) of first value in DATA
C  Values from commons:
C   ICODE      I*2  Opcode number from list in NEWHED.
C   FBLANK     R*4  Value of blanked pixel.
C   CPARM(10) R*4  Input adverb array.
C   DPARM(10) R*4  Input adverb array.
C   CATBLK     I*2  Output catalog header (also CAT4, CAT8)
C  Output:
C   RESULT(*) R*4  Output row.
C   IRET       I*2  Return code   0 => OK
C                                >0 => error, terminate.
C
C  Output in COMMON
C  NUMHIS      I*2  # history entries (max. 10)
C  HISCRD(16,NUMHIS) R*4 History records
C  CATBLK      I*2  Catalog header block
C
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 all contributions, make Eric Greisen's life easier and 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 the section in the chapter on tasks about installing a new task. Basically this means getting the proper logical assignments for the include files and the subroutine libraries so that you can compile and link edit the task. Then you're all set (on a VAX at least). The VAX/VMS, NORD and UNIX versions of AIPS (at least) support the use of an adverb VERSION which specifies the directory in which the load module and help file are to be found. Simply set VERSION to the proper value, set the necessary adverbs and tell AIPS 'GO'.

2.4 HINTS FOR USING THE DEBUGGER IN AIPS

The symbolic debugger in VAX/VMS systems is a very powerful tool for debugging AIPS tasks. The Convex Debugger csd is also a competent debugging tool. In the following section there are a few hints about using the debugger in AIPS tasks.

- The AIPS compile and link edit command procedures will accept an argument 'DEBUG' after the name of the task and link a load module with the debugger.

- Use WAITTASK on VAXes to keep AIPS from trying to talk to the terminal at the same time as the debugger. In UNIX AIPS this is done automatically.

- To invoke csd on a Convex one must enter "%setenv DBUGR 'which csd'" BEFORE starting AIPS. All tasks are linked with the debugger in so all tasks will be run under the debugger. Note: Convex routines are normally not compiled with the debugger; if detailed debugging of a routine is desired, then that routine must be compiled with the DEBUG option (the -g compiler switch).

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, how many characters per floating point word, 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. Most 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.

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. Several of these, which will be discussed in detail below, are character handling, integers and call arguments for subroutines and functions.

3.2.1 Character Strings

One of the more serious problems with Fortran is its handling of characters. In Fortran 66, there was no distinct character data type, but characters can be put into other data type variables. These variables can be equivalenced in various ways to form data structures; that is, arrays which contain data of various types. Fortran 77 introduced explicit character variables and formally forbids storing characters in other data types. Unfortunately, the internal storage format for character variables is not defined and

varies from machine to machine. There is even a deliberate attempt to make it difficult to determine the exact internal structure of character variables. This means that character variables cannot be equivalenced in any way to other data types and most compilers check.

The net effect of the changes to Fortran 77 is that data structures are formally forbidden, although compilers allow the Fortran 66 conventions. As a result, the AIPS system primarily uses the Fortran 66 conventions and stores characters in REAL or INTEGER words. We strongly discourage the use of double precision words to hold 8 characters, since this will not work on some machines like Dec-10's.

Limited use of CHARACTER data types is allowed if discouraged. Many character functions are machine specific and the prohibition on equivalencing CHARACTER variables with other data types (either explicitly or implicitly) MUST be observed. CHARACTER variables are different from characters encoded in REALs or INTEGERs and the two CANNOT be interchanged (even if your VAX lets you think otherwise).

Different machines can store different numbers of characters in a REAL word. We take care of this problem with two types of character strings, packed and unpacked. Unpacked character strings contain 4 characters per REAL word and packed character strings contain as many characters as possible. (A third type, "loose" strings may be encountered in input parsing routines; these strings have one character per integer.) The number of characters per REAL is a parameter carried in a common. A number of character manipulation routines are available. A list follows; detailed descriptions of the call sequence can be found at the end of this chapter.

- CHCOPY moves characters from one packed string to another.
- CHCOMP compares two packed character strings.
- CHFILL fills a packed string with a character.
- CHPACK takes 4 characters per real word and packs them into a string.
- CHPAC2 takes 2 characters per integer and packs them into a string.
- CHXPND expands a packed character string into a real array 4 characters per word.
- CHXPN2 expands a packed character string into a integer array 2 characters per word.
- CHLTOU converts any lower case characters in a packed string to upper case.

- CHMATC searches one packed string for the occurrence of another.

- 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.

3.2.2 Integers

The number of bits in an integer word is also a problem. In particular, PDP 11 computers do not support 32 bit integers and Fortran 77 formally does not allow 16 bit integers. AIPS uses both short (16 bit) and long (32 bit) integers where appropriate, but all variables should be explicitly declared. On machines where these data types are not supported (e.g., all integers are 64 bits) a preprocessor is necessary to convert INTEGER*2 and INTEGER*4 to INTEGER. The ratio of the length of a short integer to a long integer is kept in the DCH common as NWDPLI; the ratio of the number of bits in a short integer to those in a single precision value is NWDPFP; the ratio for double precision (REAL*8) is NWDPDP.

Until recently the use of Integer*4 was forbidden in AIPS which caused the adoption of the rather unwieldy concept of "pseudo INTEGER*4" (usually denoted P I4) in which an array of two INTEGER*2 words are used to represent a larger integer. The first word contains the lowest order bits and the second word contains the higher order bits. There are two basic routines for handling pseudo INTEGER*4 integers, ZR8P4 and ZMATH4. A short description of each is given here and details of the call sequences are given at the end of this chapter. Use of Pseudo I*4 is being phased out, but it still appears in places.

- ZR8P4 converts between pseudo I*4 and R*8. Pseudo I*4 has the form of two short integers with the least significant half at the lower I*2 index. IBM I*4 has the form of a 2's complement, 32-bit integer with the most significant 16 bits in the I*2 word of lower index and the least significant 16 bits in the I*2 word of higher index.

- ZMATH4 does I*4 arithmetic on pseudo I*4 arguments

3.2.3 Call Arguments

Most machines have several lengths of integers or reals and in general AIPS routines will be using the shorter form. This can lead to problems if the default type is the longer form. In this case, if the call statement includes a literal or an expression, the value

passed will be the long form whereas the routine being called probably expects a short integer. Similar problems arise if the default is the short type and the long type is expected by the routine being called.

To avoid the problems resulting from expressions and literal values in call arguments, we prohibit all expressions and literals in call arguments. For instance, if a value of 1 is needed for a subroutine call, a variable named N1 is declared and DATAed a value of 1. The call argument used is then N1. Literal character strings should never be used in calls to AIPS system routines.

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 PRTTP follows:

```
PRTTP     LLLLLLLLLLLLUUUUUUUUUUUU CCCCCCCCCCCCCCCCCCCCCCCCCCCC
PRTTP:   Task to print contents of tapes (UV data, maps, ...)
INTAPE          0.0          9.0   Tape unit # (0=> 1)
NFILES          0.0      32000.0   # files to advance from
                                   beginning of tape.  > 1000 ->
                                   start where tape is now.
PRTLEV         -1.0          2.0   Amount of print (2 -> a lot)
                                      (0 -> summaries)
                                      (-1 -> very brief print)
DOCRT          -1.0        132.0   > 0 => print on CRT, else
```

```
                                        use line printer
                                    > 72 => terminal width
-------------------------------------------------------------------
PRTTP
Type:  Task  (interactive only)
Use:   To print on the line printer or terminal a fairly
       detailed summary of the contents of a tape.  The program
       begins by rewinding the tape and then advances the tape
       by the user specified number of files.  PRTTP then
       reports on the contents of all files until a double
       end-of-file mark is found.  The tape is finally
       positioned after the first of the two end-of-files.  The
       task can recognize the FITS formats (map and UV), the IBM
       map format, and the VLA UV-data export format.
Adverbs:
  INTAPE.......Input tape drive number.  0 => 1.
  NFILES.......Number of files to advance from the beginning
               of the tape.  <= 0 => 0.  To have the program
               begin at the current tape position, give
               NFILES any number > 1000.  The relative file
               numbers (n) will appear on the print as
               1000+n.
  PRTLEV.......Amount of print desired (for FITS format only):
               -1 => minimal information, 0 => summaries in
               IMHEADER form, 1 => add non-History cards,
               2 => add History cards too.
  DOCRT........True (> 0) means to put the display on the user
               terminal.  False (<= 0) means use the line
               printer.  If 72 < DOCRT <= 132, the task will
               assume that the terminal is DOCRT characters wide
-------------------------------------------------------------------
```

On the first line 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 if 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 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 variables. Characters are in packed strings, but require (NCHAR+4)/3 REAL variables.

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 at compile time. Since many compilers are fussy about the order of declaration statements, the declarations for most commons are divided up into several parts.

The INCLUDE files names have the form nxxx.INC where n indicates the type of include file: D indicates that type declarations are included, C indicates that COMMON statements are included, E indicates that EQUIVALENCE statements are included, V indicates that DATA statements are included, Z indicates that machine dependent declarations are included, and I is a special version of D in which a particular declaration is omitted. 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
cannot be done in UNIX. In UNIX another logical must be defined, used in the file during debugging and finally removed from the source code and replaced with INCS:.

Many tasks also have their own includes; this greatly reduces the problems in developing and maintaining tasks.

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 common is the Device Characteristics Common; this common is obtained from the INCLUDE files IDCH.INC, DDCH.INC and CDCH.INC. The text of these includes are to be found at the end of this chapter.

The only difference between IDCH.INC and DDCH.INC is the declaration of the INTEGER array FTAB. This array is used to keep system tables for the I/O routines. The contents of FTAB are normally of little interest to the programmer, but the size of this array is determined by the number of different types of files to be open at the same time. Thus, in the main routine, the include IDCH.INC should be used and space reserved for FTAB by an explicit declaration. In subsequent routines, the INCLUDE DDCH.INC is used to declare the variables in the common. In all cases CDCH.INC is used for the COMMON statement.

The FTAB array is used to keep AIPS and system I/O tables so the size of the array depends on the computer. On Modcomps, which require the largest tables, the dimension of the FTAB should be

(# devices open) * 2

+ (# of regular (extension) files open) * 14

+ (# of map (image and uv data) files open) * 80 bytes.

Note that a byte is defined here as half a short integer. The number of files open refers to the maximum number open in each category at any time. In general, it is probably a good idea to double these values to reduce problems with future installations.

The contents of the Device Characteristics common 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 chapter on disk I/O.

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 a short 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 CHDR.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. In the future, the catalog header will probably be expanded to include arbitrary keyword/value pairs to allow storage of information not currently allocated space in the header.

3.7.3 History Common

The routines that write HIstory files carry information in pointers in commons invoked with the INCLUDEs DHIS.INC and CHIS.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, DTVD.INC, CTVC.INC and CTVD.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 INCLUDEs DUVH.INC and CUVH.INC. This common is filled in by a call to UVPGET which analyzes the current catalog header in common /MAPHDR/. Details of the call arguments and the pointers etc. set are found at the end of this chapter.

3.7.6 Files Common, /CFILES/

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 includes DFIL.INC and CFIL.INC and use of the termination routine DIE (which will be discussed in a later section). The contents of the DFIL.INC and CFIL.INC are 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 /CFILES/ 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') 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 the history file.
- HIAD80 adds an 80-character card image into an open history file.
- HICLOS closes a history file, flushing the buffer if requested.

Once the history file is open, entries can be made in it by first ENCODEing the message (up to 72 characters) into a real array dimensioned to be at least 72 bytes and calling HIADD. An example:

```
      REAL*4    CARD(18)
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:CMSG.INC'
              .
              .
              .
      ENCODE (72,2000,CARD) TSKNAM, FACTOR
 2000 FORMAT (2A3,' FACTOR=',F5.2,' / CORRECTION FACTOR')
      CALL HIADD (HLUN, CARD, 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. The precursor comments for these routines can be found in AIPS manual volume 3.

3.9.2 Extension Tables (ALLTAB)

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.

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 (<= 64 is best) is first encoded into array MSGTXT in the message common, which is invoked by the includes DMSG.INC and CMSG.INC. Then a call is made to the routine MSGWRT with a single INTEGER*2 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:

```
INTEGER*2  N4
INCLUDE 'INCS:DMSG.INC'
  .
  .
INCLUDE 'INCS:CMSG.INC'
  .
  .
DATA    N4 /4/
  .
  .
ENCODE (80,1000,MSGTXT)
```

```
      CALL MSGWRT (N4)
        .
        .
 1000 FORMAT ('FINISHED READING THE DATA')
```

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 common /MSGCOM/ (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*2 LPLUN, LPIND, N1, N2, BUFFER(256), IPCNT
      LOGICAL*2 T,F
      REAL*4    LPNAME(6), VALUE1, VALUE2
      INCLUDE 'INCS:DDCH.INC'
          .
          .
      INCLUDE 'INCS:CHCH.INC'
          .
          .
      DATA  LPLUN /1/, LPNAME /6*'     '/, N1, N2/1,2/
      DATA  T, F /.TRUE.,.FALSE./
          .
          .
C                                        Open the printer.
      CALL ZOPEN (LPLUN, LPFIND, N1, LPNAME, F, T, T, IERR)
        (handle error condition if detected)
C                                        Header page if batch
      CALL BATPRT (N1, 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 (N2, 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 and CDCH.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 and CDCH.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 PRTTP 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. Encode messages to be sent into an array and send the array to ZTTYIO. Lines read from the terminal will be returned by ZTTYIO as an unpacked character string. An example of the use of ZTTYIO is the following:

```
      INTEGER*2  N1, N72
      INTEGER*2  TTYLUN, TTYIND, IRET
      LOGICAL*2  T, F
      REAL*4     READ, WRITE, LINE(18)
                    .
                    .
      DATA N1, N72 /1,72/,   TTYLUN /5/
      DATA T, F /.TRUE.,.FALSE/, READ, WRITE /'READ','WRIT'/
                    .
                    .
C                                        Open the terminal
      CALL ZOPEN (TTYLUN, TTYIND, N1, LINE, F, T, T, IERR)
C                                        Error if IERR .NE. 0
                    .
                    .
C                                        Encode message for terminal
      ENCODE (72,1000,LINE)
C                                        Send to terminal
C                                        Set here to read and write
C                                        up to 72 characters per
C                                        transmission.
      CALL ZTTYIO (WRITE, TTYLUN, TTYIND, N72, LINE, IERR)
C                                        Error if IERR .NE. 0
                    .
                    .
C                                        Read from terminal.
C                                        Up to 72 characters.
      CALL ZTTYIO (READ, TTYLUN, TTYIND, N72, 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 now cataloged as type 'SC' so that the user can directly delete them. The /CFILES/ common described in a previous section is designed for this purpose and is obtained by the INCLUDEs DFIL.INC and CFIL.INC.

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*2 IRET, NX, NY, NP(2), BP, N2, BUFF(512)
      INTEGER*4 SIZE
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DDCH.INC'
            .
            .
      INCLUDE 'INCS:CFIL.INC'
      INCLUDE 'INCS:CDCH.INC'
            .
            .
      DATA N2 /2/
            .
            .
C                                    NX, NY are the size of an
C                                    image.  Make a scratch file
C                                    big enough for a REAL copy
C                                    of the image.
C
C                                    Compute the size in bytes.
C                                    Note: NWDPFP is from the
C                                    /DCHCOM/ and is the size of
C                                    a REAL word in terms of
C                                    short integers. 1 short
```

```
C                                              integer = 2 bytes
      BP = 2 * NWDPFP
      NP(1) = NX
      NP(2) = NY
C                                              Compute size needed
      CALL MAPSIZ (N2, NP, BP, 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 /CFILES/ common as 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*2 LUN, IND, SC, N1
      REAL*4    FILE(6)
              ...
      DATA N1 /1/, SC /'SC'/, T /.TRUE./
              ...
C                                              ISCR = /CFILES/ slot number.
      CALL ZPHFIL (SC, SCRVOL(ISCR), SCRCNO(ISCR), N1, FILE, IRET)
      CALL ZOPEN (LUN, IND, SCRVOL(ISCR), FILE, T, T, T, 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 /CFILES/ 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 /CFILES/ 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 and CDCH.INC) with NPOPS (the AIPS number of the initiating task) from the message common (DMSG.INC and CMSG.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. The following sections will describe the procedure for several operating systems.

3.14.1 Under VMS

The AIPS installation on a VAX makes heavy use of the directory structures, command files, and the logical name capability. AIPS files are kept in a hierarchal directory structure with logical names for each of the subdirectories. The current baroque structure and programmer instructions are described in Appendix A.

3.14.2 Under UNIX

The AIPS directory structure under UNIX is the same as that used for VMS. There are a few differences in the procedures needed to compile and link tasks. These are summarized in the following:

- COMLNK: COMLNK is used for all compiles and links, COMTST does not exist. The argument list to COMLNK may include (after the program name) compiler or linker options.

3.14.3 Developing Software (VMS And UNIX)

For many purposes, it is convenient to leave a task in the programmer's own directory. The directory in which to find the HELP file and the executable module can be specified in AIPS using the adverb VERSION, e.g.,

```
VERSION='UMA0:[MYAIPS.PGM]'
```

In this case, source and object modules may be kept in any directory. The HELP file and the execute module must be put in the same directory.

UNIX systems distinguish between upper and lower case letters. This is a problem for AIPS, since the AIPS command parser converts all characters to upper case, whereas the programmer's directory name is very likely to contain some lower case letters. In this case, a logical (shell variable) symbol must be defined which points to the correct directory; e.g., before starting AIPS:

```
%setenv MYDIR `pwd`
```

then inside AIPS use

```
VERSION='MYDIR'
```

to specify the directory

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.
- Cxxx.INC. These files contain Fortran COMMON statements.
- Exxx.INC. These contain Fortran EQUIVALENCE statements.
- Vxxx.INC. These contain Fortran DATA statements.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations, but the declaration of some variable is omitted. This type of include is used in the main program to reserve space for the omitted variable in the appropriate common. The omitted variable must be declared and dimensioned separately.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

3.15.1 CDCH.INC

```
C                                                          Include CDCH
      COMMON /DCHCOM/ XPRDMM, XTKDMM, SYSNAM, VERNAM, RLSNAM, TIMEDA,
     *    TIMESG, TIMEMS, TIMESC, TIMECA, TIMEBA, TIMEAP, RFILIT,
     *    NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3, NTAB3,
     *    NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR, CSIZPR, NINTRN,
     *    KAPWRD, NCHPFP, NWDPFP, NWDPDP, NWDPLI, NWDPLO, NBITWD,
     *    NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC, NTKACC,
     *    UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT, KAP2WD
      COMMON /FTABCM/ DEVTAB, FTAB
C                                                          End CDCH.
```

3.15.2 CFIL.INC

```
C                                                               Include CFIL
      COMMON /CFILES/ RQUICK, NSCR, SCRVOL, SCRCNO, NCFILE, FVOL, FCNO,
     *    FRW, CCNO, IBAD, LUNS
C                                                               End CFIL
```

3.15.3 CMSG.INC

```
C                                                               Include CMSG
      COMMON /MSGCOM/ MSGCNT, TSKNAM, NPOPS, NLUSER, MSGTXT, NACOUN,
     *    MSGSUP, MSGREC, MSGKIL, ISBTCH, MSGDM1, MSGDM2, MSGDM3
C                                                               End CMSG.
```

3.15.4 CUVH.INC

```
C                                                               Include CUVH
      COMMON /UVHDR/ FREQ, RA, DEC, SOURCE, NVIS, ILOCU, ILOCV,
     *    ILOCW, ILOCT, ILOCB, ILOCSU, JLOCC, JLOCS, JLOCF, JLOCR,
     *    JLOCD, JLOCIF, INCS, INCF, INCIF, ICORO, NRPARM, LREC, NCOR,
     *    ISORT
C                                                               End CUVH
```

3.15.5 DDCH.INC

```
C                                                               Include DDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2), TIMEDA(15),
     *    TIMESG, TIMEMS, TIMESC, TIMECA, TIMEBA(4), TIMEAP(3),
     *    RFILIT(14)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *    CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NWDPLI,
     *    NWDPLO, NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *    NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *    KAP2WD, DEVTAB(50), FTAB(1)
C                                                               End DDCH.
```

3.15.6 DFIL.INC

```
C                                                               Include DFIL
      INTEGER*2 NSCR, SCRVOL(64), SCRCNO(64), IBAD(10), LUNS(10),
     *    NCFILE, FVOL(50), FCNO(50), FRW(50), CCNO
      LOGICAL*2 RQUICK
C                                                               End DFIL
```

3.15.7 DMSG.INC

```
C                                                               Include DMSG
      INTEGER*2 MSGCNT, TSKNAM(3), NPOPS, NLUSER, MSGSUP, MSGREC,
     *    MSGKIL, ISBTCH, MSGDM1, MSGDM2, MSGDM3
      INTEGER*4 NACOUN
      REAL*4    MSGTXT(20)
C                                                               End DMSG.
```

3.15.8 DUVH.INC

```
C                                                               Include DUVH
      INTEGER*4 NVIS
      INTEGER*2 ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, JLOCC,
     *    JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC, NCOR, ISORT,
     *    INCS, INCF, INCIF, ICORO
      REAL*4 SOURCE(2)
      REAL*8 FREQ, RA, DEC
C                                                               End DUVH
```

3.15.9 IDCH.INC

```
C                                                               Include IDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2), TIMEDA(15),
     *    TIMESG, TIMEMS, TIMESC, TIMECA, TIMEBA(4), TIMEAP(3),
     *    RFILIT(14)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *    CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NWDPLI,
     *    NWDPLO, NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *    NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *    KAP2WD, DEVTAB(50)
C                                                               End IDCH.
```

3.15.10 PUVD.INC

```
C                                                       Include PUVD
C                                       Parameters for uv data
      INTEGER*4 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=100)
C                                       MAXFLD=max. no fields
      PARAMETER (MAXFLD=16)
C                                       MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                                       End PUVD.
```

3.16 ROUTINES

3.16.1 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*2   Number of "Forbidden" types to copy.
 NOTTYP(*)   I*2   Table types to ignore.
 LUNOLD      I*2   LUN for old file
 LUNNEW      I*2   LUN for new file
 VOLOLD      I*2   Disk number for old file.
 VOLNEW      I*2   Disk number for new file.
 CNOOLD      I*2   Catalog slot number for old file
 CNONEW      I*2   Catalog slot number for new file
In/out:
 CATNEW(256)I*2   Catalog header for new file.
Output:
 BUFF1(1024) I*2   Work buffer
 BUFF2(1024) I*2   Work buffer
 IRET        I*2   Return error code  0 => ok, otherwise TABCOP
                                         or 10*CATIO error.
```

3.16.2 CHCOPY - moves characters from one string to another.

```
CHCOPY (NCHAR, NP1, STR1, NP2, STR2)
Inputs: NCHAR   I*2     Number of characters to move
        NP1     I*2     Start char position in input string
        STR1    R*4(*)  Input string
        NP2     I*2     Start char position in output string
Output: STR2    R*4(*)  Output string
```

3.16.3 CHCOMP - compares two character strings.

```
CHCOMP (NCHAR, KP1, STR1, KP2, STR2, EQUAL)
Inputs: NCHAR  I*2     # characters to compare
        KP1    I*2     starting character in string 1
        STR1   R*4(*)  string 1
        KP2    I*2     starting character in string 2
        STR2   R*4(*)  string 2
Output: EQUAL  L*2     T => strings are same
```

3.16.4 CHFILL - fills a string with a character.

```
CHFILL (NCHAR, CHAR, NBP, STRING)
Inputs: NCHAR   I*2     Number of char positions to fill
        CHAR    I*2     Char in char position 1
        NBP     I*2     Start char position to fill
Output: STRING  R*4(*)  Filled string
```

3.16.5 CHLTOU - converts any lower case characters in a packed string to upper case.

```
CHLTOU (N, STRING)
Inputs: N       I*2     Number of characters
In/out: STRING  R*4(*)  Packed string to be converted.
```

3.16.6 CHMATC - searches one string for the occurrence of another string.

```
CHMATC (NA, JA, CA, NB, JB, CB, NP)
Inputs: NA   I*2      Number of characters in CA (start at JA)
        JA   I*2      Start at char position JA in CA
        CA   R*4(*)   Packed substring to be found in CB
        NB   I*2      Number of characters in CB (n.b. TOTAL)
        JB   I*2      Start search at offset in CB
        CB   R*4(*)   Packed string.
Output: NP   I*2      start position in CB of CA, 0 if none.
                      w.r.t. start of string
```

3.16.7 CHPACK - takes characters 4 / real and packs them into a string.

```
CHPACK (NCH, ISTR, NP, OSTR)
Inputs: NCH   I*2     number of characters
        ISTR  R*4(*)  real array 4 char / word
        NP    I*2     start position in output string
Output: OSTR  R*4(*)  output packed string
```

3.16.8 CHPAC2 - takes characters 2 / integer and packs them into a string.

```
CHPAC2 (NCH, ISTR, NP, OSTR)
Inputs: NCH   I*2      number of characters
        ISTR  I*2(*)   integer array 2 char / word
        NP    I*2      start position in output string
Output: OSTR  R*4(*)   output packed string
```

3.16.9 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.

```
CHWMAT (NPM, PS, IPT, NTS, TS, EQUAL)
Inputs: NPM    I*2        Length of test string (not incl NTS-1
                          characters)
        PS     R*4(*)     Packed pattern string
        IPT    I*2(NPM)   Pattern array prepared by PSFORM
        NTS    I*2        Start char position in TS for testing
        TS     R*4(*)     Packed test string
Output: EQUAL  L*2        T => they match
```

3.16.10 CHXPND - expands a packed character string into a real array with four characters per word.

```
CHXPND (NCH, KIN, KFIRST, KOUT)
Inputs:  NCH     I*2     Number of characters to unpack
         KIN     R*4(*)  Packed string
         KFIRST  I*2     First character to unpack
Outputs: KOUT    R*4(*)  Looser string
```

3.16.11 CHXPN2 - expands a packed character string into an integer array with two characters per word.

```
CHXPN2 (NCH, KIN, KFIRST, KOUT)
Inputs:  NCH     I*2     Number of characters to unpack
         KIN     R*4(*)  Packed string
         KFIRST  I*2     First character to unpack
Outputs: KOUT    I*2(*)  Looser string
```

3.16.12 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. A call to DIE should be the last executable statement before the STOP statement. NOTE: DIE should be used only by tasks using common /CFILES/ (obtained from includes CFIL.INC and DFIL.INC).

```
DIE (ICODE, BUFF)
Inputs: ICODE    I*2     Return code: 0 => good, other => bad end
        BUFF     I*2(256)    Work buffer
```

Locations in catalog are communicated by COMMON /CFILES/

```
NCFILE    I*2  Number of files marked in catalog.
FVOL(50)  I*2  Volume numbers of the maps.
FCNO(50)  I*2  Slot numbers of the maps.
FRW(50)   I*2  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*2  Number of scratch files to be destroyed
SCRVOL(20) I*2 Scratch file volume numbers
SCRCNO(20) I*2 Scratch file catalog numbers
```

3.16.13 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 is called by DIE).

```
DIETSK (IRET, RQUICK, IBUF)
Inputs:  IRET     I*2       0 => ok, else bad end
         RQUICK   L*2       T => initiator already resumed
Output:  IBUF     I*2(256)  Scratch buffer
```

3.16.14 EXTCOP - copies a 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        I*2   Extension file type eg 'CC','AN'
 INVER       I*2   Version number to copy, 0=>copy all.
 OUTVER      I*2   Version number on output file, if more than one
                   copied (INVER=0) this will be the no. of the first
                   file.  If OUTVER=0 the EXTINI defaults are used.
 LUNOLD      I*2   LUN for old file
 LUNNEW      I*2   LUN for new file
```

```
 VOLOLD      I*2   Disk number for old file.
 VOLNEW      I*2   Disk number for new file.
 CNOOLD      I*2   Catalog slot number for old file
 CNONEW      I*2   Catalog slot number for new file
 CATNEW(256)I*2    Catalog header for new file.
In/out:
 BUFF1(>512)I*2    Work buffer: 256 words + n * 256 words (enough
                   to hold at least one logical record)
 BUFF2(>512)I*2    Work buffer: as BUFF1
 BUFF3(*)   I*2    Buffer large enough to hold one logical record.
Output:
 IRET        I*2   Return error code  0 => ok
                                      1 => files the same, no copy.
                                      2 => no input files exist
                                      3 => failed
                                      4 => no output files created.
```

3.16.15 GTPARM - obtains the activator (AIPS) 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    I*2(3)  Task name (ASCII) 2 chars / integer
         NPARMS  I*2     number of real variables wanted
Outputs: RQUICK  L*2     T => release POPs as soon as possible
                         F => wait until you have finished
         RPARM   R*4(NPARMS)  parameters received
         SCRTCH  I*2(256)     scratch buffer
         IERR    I*2          error code: 0 -> ok
                                          1 -> initiator not found
                                          2 -> disk troubles
                                          3 -> initiator zeroed
```

3.16.16 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*2       Number R*4 parameters
 Output: OPTELL   R*4       Opcode from TELL
         PARMS    R*4(*)    Returned parameters
         SCRTCH   I*2(256)  Scratch buffer
         IERR     I*2       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.

3.16.17 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*2       lun of HI file (must be open!!)
        CARD    I*2(*)    new card
IN/out: BUFFER  I*2(256)  HI work buffer
Output: IERR    I*2       0 => ok, other set by HIIO
```

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

```
HIAD80 (HLUN, IST, CARD, HBLK, IERR)
Inputs: HLUN    I*2       LUN of HI file (must be open!!)
        IST     I*2       Start character position in card
        CARD    R*4(20)   80-character packed "card"
In/out: HBLK    I*2(256)  HI I/O buffer
Output: IERR    I*2       Error code of HIADD
```

3.16.19 HICLOS - closes a history file updating it if requested.

```
HICLOS (HLUN, UPDATE, BUFFER, IERR)
Inputs: HLUN    I*2       file lun (already open!!)
        UPDATE  L*2       T => write last record & update pointers
In/out: BUFFER  I*2(256)  HI work buffer
Output: IERR    I*2       error code : 0 - ok
                                       1 - LUN not open
                                       2-6 - ZFIO errors
```

3.16.20 HIINIT - initializes the history common area /HICOM/.

```
HIINIT (NFILES)
Inputs:  NFILES   I*2   number of HI files open at once (max)
                       at least 3 are available via DHIS.INC
```

3.16.21 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. 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.

```
HISCOP (LUNOLD, LUNNEW, VOLOLD, VOLNEW, CNOOLD,
  *   CNONEW, CATBLK, BUFER1, BUFER2, IERR)
Inputs:  LUNOLD          I*2  LUN for old history file.
         LUNNEW          I*2  LUN for new history file.
         VOLOLD          I*2  Vol. number for old history file.
         VOLNEW          I*2  Vol. number for new history file.
         CNOOLD          I*2  Catalog slot number of old history file.
         CNONEW          I*2  Catalog slot number of new history file.
In/Out:  CATBLK(256)     I*2  Catalog header of map for new file.
         BUFER1(256)     I*2  Work buffer, used for old file.
         BUFER2(256)     I*2  Work buffer, new file; must be used in
                              further HIADD calls until file is closed.
Output:  IERR            I*2  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.
```

3.16.22 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     R*4(*)   Input file name 12 packed chars
        INCL    R*4(*)   Input file class 6 packed chars
        INS     I*2      Input file sequence number
        DEFCLS  R*4(*)   Default output file class 6 packed chars
                         if 1st 4 chars blank, use task name
In/Out: OUTN    R*4(*)   User-supplied OUTNAME adverb
        OUTCL   R*4(*)   User-supplied OUTCLASS adverb
        OUTS    I*2      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.
```

3.16.23 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*2       LUN for print device (open)
          OUTIND  I*2       FTAB pointer for print device
          DOCRT   R*4       › 0. => use CRT, else line printer
          NC      I*2       Number characters in line
          T1      R*4(*)    Page title line 1 : packed chars
          T2      R*4(*)    Page title line 2 : packed chars
          LINE    R*4(*)    Text line: packed chars
  In/out: NLINE   I*2       Number lines so far on page
                            › 1000 => just ask about continuing
                            = 999 => just start new page
          IPAGE   I*2       Current page number
                            = 0 => just start new page
  Output: SCRTCH  R*4(›33)  scratch core
          IERR    I*2       Error code: 0 => OK, -1 user asks to quit
```

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

```
PSFORM (NC, PS, IPT)
Inputs: NC   I*2       Number characters in pattern possible
        PS   R*4(*)    Pattern string (packed)
Output: IPT  I*2(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
```

incl present following

3.16.25 RELPOP - releases the held POPS (AIPS) task, passing it a return code.

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

3.16.26 SCREAT - is intended to replace all previous scratch file creation routines in AIPS (beginning on February 11, 1985). It uses the Common included via the new DFIL.INC, CFIL.INC pair 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 SCvcccO1 where v is the disk number and ccc is the catalog slot number. Their logical names are determined from the routine BLDSNM.

```
  SCREAT (SIZE, WBUFF, IERR)

Input:  SIZE    I*4        Desired size in bytes  (NOTE real I*4)
Output: WBUFF   I*2(512)   Scratch buffer    (NOTE 512 integers)
        IERR    I*2        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
Note: this common uses IBAD to specify BADDISKs which are avoided.
```

3.16.27 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       I*2   Extension file type (e.g., 'CC','AN')
```

```
    INVER       I*2   Version number to copy, 0 => copy all.
    OUTVER      I*2   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*2   LUN for old file
    LUNNEW      I*2   LUN for new file
    VOLOLD      I*2   Disk number for old file.
    VOLNEW      I*2   Disk number for new file.
    CNOOLD      I*2   Catalog slot number for old file
    CNONEW      I*2   Catalog slot number for new file
   In/out:
    CATNEW(256)I*2    Catalog header for new file.
   Output:
    BUFF1(256) I*2    Work buffer
    BUFF2(256) I*2    Work buffer
    IRET       I*2    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
```

3.16.28 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+IABS(ICOR-ICOR0)*INCS+(IF-1)*INCIF

     UVPGET (IERR)

    Inputs: From common /MAPHDR/
       CATBLK(256)    I*2  Catalog block
       CAT4           R*4  same as CATBLK
       CAT8           R*8  same as CATBLK
    Output: In common /UVHDR/
       SOURCE(2)      R*4  Packed source name.
       ILOCU          I*2  Offset from beginning of vis record of U
       ILOCV          I*2                      "                  V
       ILOCW          I*2                      "                  W
       ILOCT          I*2                      "               Time
       ILOCB          I*2                      "           Baseline
       ILOCSU         I*2                      "          Source id.
       JLOCC          I*2  Order in data of complex values
       JLOCS          I*2  Order in data of Stokes' parameters.
       JLOCF          I*2  Order in data of Frequency.
       JLOCR          I*2  Order in data of RA
       JLOCD          I*2  Order in data of dec.
       JLOCIF         I*2  Order in data of IF.
       INCS           I*2  Increment in data for stokes (see above)
```

INCF	I*2	Increment in data for freq. (see above)
INCIF	I*2	Increment in data for IF.
ICORO	I*2	Stokes value of first value.
NRPARM	I*2	Number of random parameters
LREC	I*2	Length in values of a vis record.
NVIS	I*4	Number of visibilities
FREQ	R*8	Frequency (Hz)
RA	R*8	Right ascension (1950) deg.
DEC	R*8	Declination (1950) deg.
NCOR	I*2	Number of correlators
ISORT	C*2	Sort order
IERR	I*2	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

3.16.29 ZDCHIN - initializes the disk characteristics common. If NDISK ‹ 0, ZDCHIN uses ABS (NDISK) but skips reading parameters from the parameter disk file. Otherwise, ZDCHIN starts by hard-coded parameter values and then resets some based on values on an alterable disk file.

```
ZDCHIN (NDEV, NDISK, NMAP, IOBLK)
Inputs: NDISK   max number regular disk files open at once
        NMAP    max number of map (double buf) files open at once
        NDEV    max number of devices open at once
        IOBLK   I*2(256) I/O block for reading values off disk.
```

3.16.30 ZMATH4 - does I*4 arithmetic on pseudo I*4 arguments

```
ZMATH4 (ARG1, OP, ARG2, RESULT)

Inputs:
   ARG1 P I*4  First P I*4 argument
   OP     I*2  OPeration ='PL'(+);'MI'(-);'MU'(x);'DI'(/)
                         'MN'(min); 'MX'(max)
   ARG2  P I*4 Second P I*4 argument
Outputs:
   RESULT P I*4 Result
```

3.16.31 ZR8P4 - converts between pseudo I*4 and R*8. Pseudo I*4 has the form of two short integers with the least significant half at the lower I*2 index. IBM I*4 has the form of a 2's complement, 32-bit integer with the most significant 16 bits in the I*2 word of lower index and the least significant 16 bits in the I*2 word of higher index.

```
ZR8P4 (OP, INTG, DX)
Inputs: OP     R*4      '4TO8' Pseudo I*4 to R*8
                        '8TO4' R*8 to pseudo I*4
                        '4IB8' IBM I*4 to R*8
                        '8IB4' R*8 to IBM I*4
In/out: INTG   I*2(2)   the I*4
        DX     R*8      the R*8
```

3.16.32 ZTTYIO - performs I/O to a terminal.

```
  SUBROUTINE ZTTYIO (OPER, LUN, FIND, NBYTES, BUFFER, IERR)
Inputs: OPER     R*4      'READ' or 'WRIT'
        LUN      I*2      LUN of open device
        FIND     I*2      Pointer to FTAB for open device
        NBYTES   I*2      # bytes (characters) to transmit (<= 132)
In/out: BUFFER   R*4(*)   Message to or from terminal, unpacked
                          string.
Output: IERR     I*2      Error code: 0 => ok
                                      1 => file not open
                                      2 => input parameter 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.

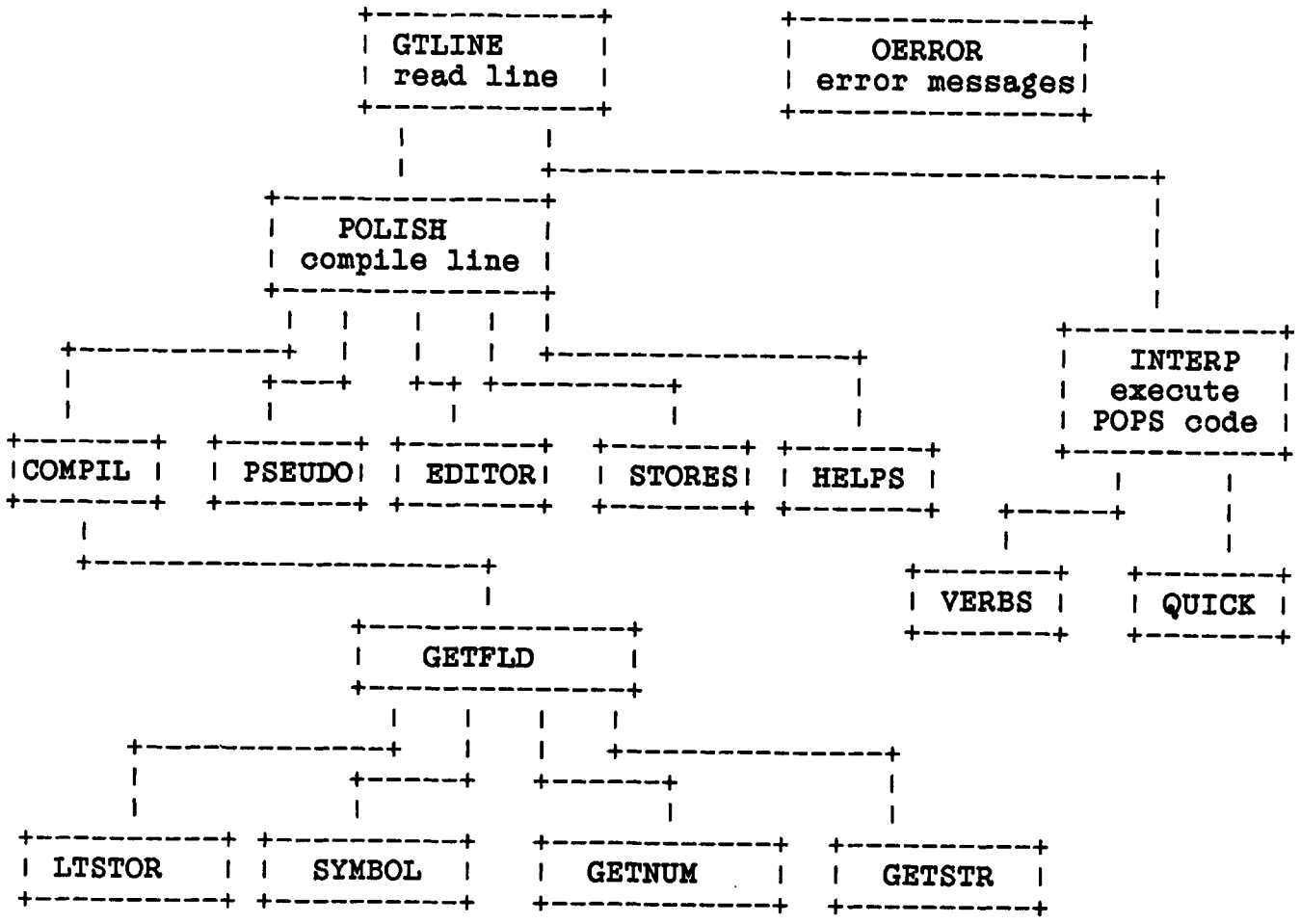
4.2.1 The POPS Processor

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:



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

- GTLINE this 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 this routine 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 QUICK 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.

- QUICK 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 REAL*8 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.

4.2.2.1 /CORE/ - This common is obtained by the includes DCON.INC, CCON.INC and ECON.INC and contains the basic POPS "memory" or K array, i.e., the symbol tables, adverb values, procedures etc. This common consists of two equivalenced, I*2 (K) and R*4 (C), 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 includes DAPL.INC and CAPL.INC and follow a shortened declaration of the K array in common /CORE/. 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.

4.2.2.2 /POPS/ - This common carries the various stacks, stack pointers and other values. This common is obtained from includes DPOP.INC and CPOP.INC. The contents of this common are described in the following:

V(60)	R*4	Operand stack for REAL variables.
XX	R*4	Intermediate REAL value
KT	I*2	Starting location in K array of KKT (temporary) area.
LPGM	I*2	Start address of an entry in the K array. Used while allocating storage.
LLIT	I*2	Not used
LAST	I*2	Last token (opcode); if zero, finished with line. Used by COMPIL.
IDEBUG	I*2	A debug flag used in various places. If true (.GE.0) then debug info about POPS is given.
MODE	I*2	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*2	= 1 if an operator has been found in the current instruction; 0 otherwise.
LINK	I*2	A link (pointer in K array)
L	I*2	Another link (pointer in K array)
NAMEP	I*2	Pointer in K array to a name in the symbol table.
IP	I*2	Pointer in K array
LP	I*2	Pointer in K array
SLIM	I*2	Maximum allowed index in the stacks (currently 60)
AP	I*2	Pointer to last entry in STACK
BP	I*2	Pointer to last entry in CSTACK
ONE	I*2	Pointer in C to value of 1.0
ZERO	I*2	Pointer in C to value of 0.0
TRUE	I*2	Pointer in C to value .TRUE.
FALSE	I*2	Pointer in C to value .FALSE.
STACK(60)	I*2	Instruction stack
CSTACK(60)	I*2	Second (temporary) instruction stack
SP	I*2	Pointer in STACK
CP	I*2	Pointer in CSTACK
SPO	I*2	Another pointer in STACK
MPAGE	I*2	Number of pages (512 bytes) in the Memory file. (LISTF + K array)
LPAGE	I*2	Number of pages (512 bytes) of the memory file which contain LISTF (procedure source code)

4.2.2.3 /SMSTUF/ - This common contain various important values passed between routines. This common is obtained with the includes DSMS.INC and CSMS.INC. The contents of this common follow.

KPAK(5)	R*4	Temporary array for storing a symbol name. A packed character string.
NKAR	I*2	The number of characters in KPAK
KBPTR	I*2	A character pointer in KARBUF, the input line buffer.
NEWCOD	I*2	Tag given by SYMBOL when allocating space for a new adverb.
TYPE	I*2	Symbol type. See section on TAG and TYPE.
SKEL	R*4	Not used.
TAG	I*2	Symbol number. See section on TAG and TYPE.
LEVEL	I*2	Precedence level bias.
LX	I*2	Number of integer words in character string X.
NEXTP	I*2	Precedence level of next item on A-stack.
X(15)	R*4	Temporary storage for character strings.
LOCSYM	I*2	Location in symbol or literal table of current symbol.

4.2.2.4 /IO/ - This common contains short I/O buffers and related information. This common can be obtained from includes DIO.INC and CIO.INC. The contents of this common follow.

ILF	I*2	Not used.
ICRLF	I*2	Not used.
IPT	I*2	Prompt character
IPAGE	I*2	Not used.
IVEC	I*2	Not used.
NBYTES	I*2	Number of valid characters in KARBUF, number of last non-blank character.
KARBUF(80)	I*2	An unpacked buffer containing the current input line.
JBUFF(40)	I*2	Buffer used to read user input as a packed string.
IPRT	I*2	Not used.
KARLIM	I*2	Number of characters in KARBUF
IUNIT	I*2	Input unit number for PREAD; 1=› user terminal, 2=› text editor, 3=›batch input file 4=›text entered during screen hold.
HOLDUF(40)	I*2	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 common /ERRORS/ to an error code known to the routine OERROR, increments ERRLEV in /ERRORS/, and, if ERRLEV .LE. 5, copies the name of the subroutine (two characters per integer) into /ERRORS/ 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. Common /ERRORS/ is obtained from includes DERR.INC and CERR.INC.

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.

4.2.6.1 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.

4.2.6.2 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.

4.2.6.3 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 overlays 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

Step	Token being 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. Update the overlay structure on machines with limited address space.

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

6. Recompile and link edit AIPS.

7. 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 QUICK,
C   that is, it should be called from an AU routine.
C-----------------------------------------------------------------------
      REAL*4    V1, V2, RESULT
      INTEGER*2 POTERR, N3, PRGNAM(3)
      INCLUDE 'INCS:DPOP.INC'
      INCLUDE 'INCS:DERR.INC'
      INCLUDE 'INCS:CPOP.INC'
      INCLUDE 'INCS:CERR.INC'
      DATA N3 /3/,   PRGNAM /'TE','ST','XX'/
C-----------------------------------------------------------------------
C                                          Set potential error number,
C                                          7 = 'STACK LIMIT'
      POTERR = 7
C                                          Check that stack not
C                                          exhausted.
      IF (SP.LT.2) GO TO 980
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                                          Finished OK
      GO TO 999
C                                          Set error code
 980  ERRNUM = POTERR
C                                          Fill in /ERRORS/.
      ERRLEV = ERRLEV + 1
```

```
      IF (ERRLEV.LE.5) CALL COPY (N3, PRGNAM, PNAME(3*ERRLEV-2))
C                                          Return
 999  RETURN
      END
```

The stack contents are as follows when TESTXX 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                 N
        3                 TAG
        4                 2
  where for TYPE = 2,7  N = 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 CAPL.INC. Note that the order of adverbs is really defined in the POPSDAT.HLP file and the order in CAPL.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 packed, but 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 includes DAPL.INC and CAPL.INC. The new adverbs should be entered in the same relative location among the other adverbs in CAPL.INC 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 (3I2,4A2,5A4)"

The response 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 = 2 * 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 'NEW' area; 'OLD' is 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 (5A2,1X,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 QUICK. 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 dimension; 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:

```
C-                                          This module is POPSDAT.
-------------------------------------------------------------------
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              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              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
C-                                FORMAT
C-C-C-C-C-  Nch  Typ  ITAG ????  ____.__ ____.__
PRTMSG         6    4   100                            \ AU1
EXIT           4    4   101
RESTART        7    4   102
CLRMSG         6    4   103
C-HELP                   110                            \ AU1A
C-INP                    111
C-INPUTS                 112
C-EXPLAIN                113
C-SHOW                   114
C-GO           2    4   120                            \ AU2
SPY            3    4   121
C-WAITTASK               122
C-ABORTASK     8    4   123
C-TPUT         4    4   124
C-TELL         4    4   125
STQUEUE        7    4   126
```

C-TGET			130	\ AU2A
C-SGDESTR			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		

C-				FORMAT	
C-C-C-C-C-	Nch	Typ	Ndim	???? _____.__	_____.__
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	
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	1.00
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	
C-C-C-C-C-	Nch	Typ	Ndim	????	____.__ ____.__
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          16.00
PBSIZE        6    2    1          16.00
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
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
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;GROFF(1234);J=GRCH;GRCH=24;GRCL;
GRCH=J;TVCL;TVON(TVCH);TVLOD;TVWED(16);TVWLAB;TVFID;RETURN
FINISH
*-------------------------------------------------------------
```

4.7 INCLUDES

4.7.1 CAPL.INC

```
C                                                               Include CAPL
      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,
     *    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,
     *    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, XANTWT, XSOLIN, XCALCO, XREFA, XSMODE, XSOLTY, XSOLMO,
     *    XSOLCO, XWTUV, XDODEL,
     *         STRA1, STRA2, STRA3, STRB1, STRB2, STRB3, STRC1, STRC2,
     *         STRC3, ARRAY1, ARRAY2, ARRAY3, SCALR1, SCALR2, SCALR3
C                                                               End CAPL
```

4.7.2 CBAT.INC

```
C                                                               Include CBAT
      COMMON /BATCH/ BATLUN, BATIND, BATREC, BATDUM, BATDAT
C                                                               End CBAT
```

4.7.3 CBWT.INC

```
C                                                               Include CBWT
      COMMON /BWTCH/ BWTNAM, BWTNUM, BWTLUN, BWTIND, BWTREC,
     *    WASERR, BWTDAT
C                                                               End CBWT
```

4.7.4 CCON.INC

```
C                                                               Include CCON
      COMMON /CORE/  K
C                                                               End CCON.
```

4.7.5 CERR.INC

```
C                                                               Include CERR
      COMMON /ERRORS/ ERRNUM, IERROR, ERRLEV, PNAME
C                                                               End CERR.
```

4.7.6 CIO.INC

```
C                                                               Include CIO
      COMMON /IO/ ILF, ICRLF, IPT, IPAGE, IVEC, NBYTES, KARBUF,
     *    JBUFF, IPRT, KARLIM, IUNIT, HOLDUF
C                                                               End CIO.
```

4.7.7 CPOP.INC

```
C                                                               Include CPOP
      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, SPO, MPAGE, LPAGE
C                                                               End CPOP.
```

4.7.8 CSMS.INC

```
C                                                               Include CSMS
      COMMON /SMSTUF/ KPAK, NKAR, KBPTR, NEWCOD, TYPE, SKEL,
     *    TAG, LEVEL, LX, NEXTP, X, LOCSYM
C                                                               End CSMS.
```

4.7.9 DAPL.INC

```
C                                                               Include DAPL
      INTEGER*2 K(14770)
C                                             character strings
      REAL*4 INNAM(3), INCLS(2), INTYP, IN2NAM(3), IN2CLS(2), IN2TYP,
     *    IN3NAM(3), IN3CLS(2), IN3TYP, OUTNAM(3), OUTCLS(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,
     *          STRA1, STRA2(2), STRA3(3), STRB1, STRB2(2), STRB3(3),
     *          STRC1, STRC2(2), STRC3(3)
C                                             numeric variables
      REAL*4 XTRUE, XFALSE, USERID, INSEQ, INDSK, IN2SEQ, IN2DSK,
     *    IN3SEQ, IN3DSK, OUTSEQ, OUTDSK, 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*4 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),
     *    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(16), PBSIZE(16),
     *    OUTVER, DOCELL, PIX2XY(7), PIX2VL, STFCTR, CUTOFF, TAMROF,
```

```
*    XBLV, XANTWT(30), XSOLIN, XCALCO, XREFA, XSMODE(7), XSOLTY,
*    XSOLMO, XSOLCO, XWTUV, XDODEL,
*          ARRAY1(10), ARRAY2(20,2), ARRAY3(3), SCALR1, SCALR2,
*          SCALR3
C                                                              End DAPL
```

4.7.10 DBAT.INC

```
C                                                              Include DBAT
      INTEGER*2 BATLUN, BATIND, BATREC, BATDUM, BATDAT(256)
C                                                              End DBAT
```

4.7.11 DBWT.INC

```
C                                                              Include DBWT
      INTEGER*2 BWTNUM, BWTLUN, BWTIND, BWTREC, BWTDAT(1)
      LOGICAL*2 WASERR
      REAL*4    BWTNAM(6)
C                                                              End DBWT
```

4.7.12 DCON.INC

```
C                                                              Include DCON
      INTEGER*2 K(18944), KXORG
      REAL*4    C(9472)
C                                                              End DCON.
```

4.7.13 DERR.INC

```
C                                                              Include DERR
      INTEGER*2 ERRNUM, IERROR(5), ERRLEV, PNAME(15)
C                                                              End DERR.
```

4.7.14 DIO.INC

```
C                                                               Include DIO
      INTEGER*2 ILF, ICRLF, IPT, IPAGE, IVEC, NBYTES, KARBUF(80),
     *   JBUFF(40), IPRT, KARLIM, IUNIT, HOLDUF(40)
C                                                               End DIO.
```

4.7.15 DPOP.INC

```
C                                                               Include DPOP
      INTEGER*2 KT, LPGM, LLIT, LAST, IDEBUG, MODE, IFFLAG, LINK,
     *   L, NAMEP, IP, LP, SLIM, AP, BP, ONE, ZERO, TRUE, FALSE,
     *   STACK(60), CSTACK(60), SP, CP, SPO, MPAGE, LPAGE
      REAL*4 V(60), XX
C                                                               End DPOP.
```

4.7.16 DSMS.INC

```
C                                                               Include DSMS
      INTEGER*2 NKAR, KBPTR, NEWCOD, TYPE, TAG,
     *   LEVEL, LX, NEXTP, LOCSYM
      REAL*4 SKEL, X(15), KPAK(5)
C                                                               End DSMS.
```

4.7.17 ECON.INC

```
C                                                               Include ECON
      EQUIVALENCE (K(1),C(1)),  (K(8),KXORG)
C                                                               End ECON.
```

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 nearly as flexible. The catalog header describes the order and amount of data, its format, scaling information, 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 7 random parameters have labels kept in the catalog header. More than 7 random parameters can be used, but the labels for the eighth 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 24 characters long.

An example from a VAX system with private catalogs is "DAOn:ttrcccvv.uuu" , where n is the zero-relative disk drive number, DAOn: 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	LENGTH	TYPE	DESCRIPTION
0	1	I*2	Volume number of disk containing this catalog
1	1	I*2	Unused
2	1	I*2	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 = 6 + IWPC(20), NLPR = 256/NWPL, and IWPC(20) = number of words to hold twenty packed characters.

The description of a directory entry is as follows:

OFFSET	LENGTH	TYPE	DESCRIPTION
0	1	I*2	User ID number; or -1 if slot is empty
1	1	I*2	Map file activity status
2	3	I*2(3)	Date/Time file was cataloged
5	1	I*2	User defined sequence number 1 to 9999
6	(6)	C*12	User defined map name, 12 characters
(12)	(3)	C*6	User defined map class, 6 characters
(15)	(1)	C*2	Map type, 2 characters
...			

where numbers in ( ) are those appropriate to normal, 16-bit machines.

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 packed characters together with the 2-character string which identifies the "physical" map type, in their slots in the directory. The sequence number is similarly an arbitrary I*2 reference number.

The Map Status is an I*2 number 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 insure 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 indicate use:

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 a short integer). The catalog header block contains double and single precision floating point numbers, short and long integers , and character strings. The catalog header record is accessed by equivalencing integer, real and double precision arrays, and obtaining the information from the array of the appropriate data type. Since the amount of storage for different data types varies from machine to machine, and the contents of the catalog header record occasionally change, we use pointers for the different arrays that are computed by VHDRIN. These pointers are kept in a common invoked with the INCLUDEs DHDR.INC and CHDR.INC.

The uses of the pointers and values on a VAX 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.

OFFSET	LENGTH	TYPE	POINTER	DESCRIPTION
0	8	C*8	K4OBJ- 1	Source name
8	8	C*8	K4TEL- 3	Telescope, i.e., 'VLA'
16	8	C*8	K4INS- 5	e.g., receiver or correlator
24	8	C*8	K4OBS- 7	Observer name
32	8	C*8	K4DOB- 9	Observation date in format 'DD/MM/YY'
40	8	C*8	K4DMP- 11	Date map created in format 'DD/MM/YY'
48	8	C*8	K4BUN- 13	Map units, i.e., 'JY/BEAM '
56	7*8	C*8(7)	K4PTP- 15 (K2PTPN- 7)	Random Parameter types
112	7*8	C*8(7)	K4CTP- 29 (K2CTPN- 7)	Coordinate type, i.e., 'LL'
168	8	R*8	K8BSC- 22	Map scaling factor
176	8	R*8	K8BZE- 23	Map offset factor: Real value = BSCALE * pixel + BZERO
184	56	R*8(7)	K8CRV- 24 (K2CTPN- 7)	Coordinate value at reference pixel
240	28	R*4(7)	K4CIC- 61 (K2CTPN- 7)	Coordinate value increment along axis

268	28	R*4(7) K4CRP= 68 (K2CTPN= 7)	Coordinate Reference Pixel
296	28	R*4(7) K4CRT= 75 (K2CTPN= 7)	Coordinate Rotation Angles
324	4	R*4 K4EPO= 82	Epoch of coordinates (years)
328	4	R*4 K4DMX= 83	Real value of data maximum
332	4	R*4 K4DMN= 84	Real value of data minimum
336	4	R*4 K4BLK= 85	Value of indeterminate pixel (real maps only)
340	4	I*4 K3GCN=	Number of random par. groups. This is the number of uv data records.
344	2	I*2 K2PCN=173	Number of random parameters
346	2	I*2 K2DIM=174	Number of coordinate axes
348	14	I*2(7) K2NAX=175 (K2CTPN= 7)	Number of pixels on each axis
362	2	I*2 K2BPX=182	Code for pixel type: 1 integer, 2 real, 3 dbl prec, 4 complex, 5 dbl prec complex
364	2	I*2 K2INH=183	For integer maps: ‹ 0 the value of an indeterminate pixel, › 0 the number of bits used to represent noise est. = 0 no blanking of pixels
366	2	I*2 K2IMS=184	Image sequence no.
368	12	C*12 K4IMN= 93 (K4IMNO= 1)	Image name Character offset in packed string
380	6	C*6 K4IMC= 93 (K4IMCO=13)	Image class Character offset in packed string
386	2	C*2 K4PTY= 93 (K4PTYO=19)	Map physical type (i.e., 'MA','UV') Character offset in packed string
388	2	I*2 K2IMU=195	Image user ID number
390	4	I*4 K3NIT=	# clean iterations
392	4	R*4 K4BMJ= 99	Beam major axis in degrees
396	4	R*4 K4BMN=100	Beam minor axis in degrees
400	4	R*4 K4BPA=101	Beam position angle in degrees
404	2	I*2 K2TYP=203	Clean map type: 1-4 => normal, components, residual, points. For uv data this word contains a two character sort order code.
406	2	I*2 K2ALT=204	Velocity reference frame: 1-3 => LSR, Helio, Observer + 256 if radio definition.
408	8	R*8 K8ORA= 52	Antenna pointing Right Ascension
416	8	R*8 K8ODE= 53	Antenna pointing Declination
424	8	R*8 K8RST= 54	Rest frequency of line (Hz)
432	8	R*8 K8ARV= 55	Alternate ref pixel value (frequency or velocity)
440	4	R*4 K4ARP=111	Alternate ref pixel location (frequency or velocity)
444	4	R*4 K4XSH=112	
448	4	R*4 K4YSH=113	
452	20	I*2(10) K2EXT=227 (K2EXTN=10)	Names of subsidiary file types (i.e., 'PL') 2 char unpacked form
472	20	I*2(10) K2VER=237	Number of versions of corresponding

		(K2EXTN=10)	subsidiary file
492	28	I*2(10)	Reserved

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: K2nnn indicates the short integer array, K3 indicates the long integer array, K4nnn indicates the real array, and K8nnn indicates the double precision array. Most of the character strings are obtained from the real array and many require special handling. The Name, class, and physical type are contained in a packed string and the labels of the regular and random axes are each kept in a packed character string. This is best explained by an example:

```
      INTEGER*2 CATBLK(256), NDIM1, N1, N6, N8, INDEX, IFPC
      REAL*4    CAT4(128), CRPIX2, CLASS(2), ALABE2(2)
      REAL*8    CAT8(64), CRVAL3
      INCLUDE 'INCS:DHDR.INC'
           .
           .
           .
      INCLUDE 'INCS:CHDR.INC'
      COMMON /MAPHDR/ CATBLK
           .
           .
           .
      EQUIVALENCE (CATBLK, CAT4, CAT8)
      DATA N1, N6, N8 /1,6,8/
           .
           .
           .
C                                        Get the dimension of
C                                        the first axis (I*2)
      NDIM1 = CATBLK(K2NAX)
C                                        Get reference pixel
C                                        of second axis (R*4)
      CRPIX2 = CAT4(K4CRP+1)
C                                        Get coordinate at reference
C                                        pixel on third axis. (R*8)
      CRVAL3 = CAT8(K8CRV+2)
C                                        Copy axis label for second
C                                        axis (R*4 array).
C                                        Note: IFPC is an AIPS utility
C                                        function that returns the
C                                        number of R*4 words for, in
C                                        this case, 8 characters.
      INDEX = K4PTP + (2-1) * IFPC (N8)
      CALL CHCOPY (N8, N1, CAT4(INDEX), N1, ALABE2)
C                                        Copy image class.
```

```
CALL CHCOPY (N6, K4IMCO, CAT4(K4IMC), N1, CLASS)
```

In the example above the catalog header block is obtained from a common named /MAPHDR/. Many AIPS utility routines get the catalog header record from this common, so it is a good place to store it.

5.4.5.1 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 (K2DIM), the number of pixels on each axis (K2NAX) and the format of the data (K2BPX). The scaling parameters for either integer or floating disk formats are kept in the header record (K8BSC, K8BZE).

The label for each axis is in a packed character string array pointed to by K4CTP. The coordinate increment between pixels must be a constant on each axis, and the array of axis increments is obtained using the pointer K4CIC. The array of coordinate reference pixels (the pixel at which the coordinate value is that pointed to by K8CRV) is pointed to by K4CRP; 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 K8CRV.

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 scaled integer images is obtained using the pointer K2INH (usually -32768) and for floating point images by K4BLK (always 'INDE').

Each row of an image (first dimension) starts on a disk sector boundary 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.

5.4.5.2 Uv Data Files - Uv data files consist of a sequence of visibility records each of which contains all data measured on a given baseline in a given integration period. The number of visibility records is given in the catalog header record by the INTEGER*4 value pointed to by K3GCN. The order of the visibility records are given by the two character code pointed to by K2TYP. (More details of the sort order can be found in the chapter on disk I/O). All values are in floating point.

Each visibility record consists of a number (K2PCN) 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 7 can be kept in the header. These labels are kept in packed character strings pointed to by K4PTP. 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 K4XSH and K4YSH. 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

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.

5.4.6 Routines To Access The Data Catalog

5.4.6.1 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.

5.4.6.2 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 IC10000n, where 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 grey-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 grey-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/).

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 (Grey-scale image)

OFFSET	LENGTH	TYPE	DESCRIPTION
0	2	I*2	Sequence number of last sub-image cataloged on this plane
2	2	I*2	Seq. no. of sub-image in slot 1; 0 if slot empty
4	8	I*2(4)	TV pixel positions of corners of 1st sub-image, x1,y1,x2,y2
12	2	I*2	Seq. no. of sub-image in slot 2; 0 if empty
14	8	I*2(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 is replaced in the IC by:

OFFSET	LENGTH	TYPE	POINTER	DESCRIPTION
408	8	R*4(2)	I4RAN=103	Map values displayed as min & max brightness (units are those of file, not the physical ones)
416	2	I*2	I2VOL=209	Disk volume from which map came
418	2	I*2	I2CNO=210	Catalog slot number of orig. map
420	8	I*2(4)	I2WIN=211	Map pixel positions of corners of displayed image (rel. to orig. map)
428	10	I*2(5)	I2DEP=215	Depth of displayed image in 7 - dimensional map (axes 3 - 7)
438	8	I*2(4)	I2COR=220	TV pixel positions of corners of image on screen
446	2	I*2	I2TRA=224	2-char code for transfer function used to compute TV brightness from map intensity values.
448	2	I*2	I2PLT=225	Code for type of plot.
450	62	I*2(31)	I2OTH=226	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 includes DHDR.INC and CHDR.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 grey-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 > NGRAY+NGRAPH.

5.5.5 Image Catalog Commons

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

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

The listings of DTVC.INC and CTVC.INC are given at the end of this chapter.

The common /DCHCOM/ 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.

5.6.1.1 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.

5.6.1.2 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:

K8RST	Rest frequency (Hz)
K4ARP	Alternate reference pixel
K8ARV	Alternate reference value
K2ALT	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.

5.6.2.1 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 the WRST.
- -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.

5.6.2.2 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 the common /LOCATI/ which is obtained using the INCLUDEs DLOC.INC and CLOC.INC. Listings of these includes can be found at the end of this chapter. The /LOCATI/ common in initialized by the routine SETLOC.

5.6.2.2.1 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 /LOCATI/ common based on the current catalog header block in the /MAPHDR/ 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.

5.6.2.2.2 Common /LOCATI/ - This common is used by the position routines and the plot labeling routines to keep constants needed for the coordinate transformation. The contents of this common are described in the following:

RPVAL	R*8(4)	Reference pixel values
COND2R	R*8	Degrees to radians multiplier = pi/180
AXDENU	R*8	delta(nu) / nu(x) when a FELO axis is present.
GEOMD1	R*8	Storage for parameter needed for geometry.
GEOMD2	R*8	"
GEOMD3	R*8	"
GEOMD4	R*8	"
RPLOC	R*4(4)	Reference pixel locations
AXINC	R*4(4)	Axis increments
CTYP	R*4(2,4)	Axis types
CPREF	R*4(2)	x,y axis prefixes for labeling
ROT	R*4	Rotation angle of position axes
SAXLAB	R*4(5,2)	Labels for axes 3 and 4 values (4 characters per floating word)
ZDEPTH	I*2(5)	Value of Idepth from SETLOC call
ZAXIS	I*2	1 relative number of z axis
AXTYP	I*2	Position axis code
CORTYP	I*2	Which position is which
LABTYP	I*2	Special x,y label request
SGNROT	I*2	Extra sign to apply to rotation
AXFUNC	I*2(7)	Kind of axis code
KLOCL	I*2	0-rel axis number-longitude axis
KLOCM	I*2	0-rel axis number-latitude axis
KLOCF	I*2	0-rel axis number-frequency axis
KLOCS	I*2	0-rel axis number-stokes axis
KLOCA	I*2	0-rel axis number-"primary axis" 3
KLOCB	I*2	0-rel axis number-"primary axis" 4
NCHLAB	I*2(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 KLOCn 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(,1). 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

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.
- Cxxx.INC. These files contain Fortran COMMON statements.
- Exxx.INC. These contain Fortran EQUIVALENCE statements.
- Vxxx.INC. These contain Fortran DATA statements.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations, but the declaration of some variable is omitted. This type of include is used in the main program to reserve space for the omitted variable in the appropriate common. The omitted variable must be declared and dimensioned separately.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

5.7.1 CHDR.INC

```
C                                                              Include CHDR
      COMMON /HDRVAL/ K4OBJ, K4TEL, K4INS, K4OBS, K4DOB, K4DMP,
     *    K4BUN, K4PTP, K4CTP, K4CIC, K4CRP, K4CRT, K4EPO,
     *    K4DMX, K4DMN, K4BLK, K4IMN, K4IMC, K4PTY, K4BMJ,
     *    K4BMN, K4BPA, K4ARP, K4XSH, K4YSH, K4IMNO, K4IMCO, K4PTYO,
     *    K8BSC, K8BZE, K8CRV, K8ORA, K8ODE, K8RST, K8ARV,
     *    K3GCN, K3NIT,
     *    K2PTPN, K2CTPN, K2EXTN,
     *    K2PCN, K2DIM, K2NAX, K2BPX, K2INH, K2IMS, K2IMU, K2TYP,
     *    K2ALT, K2EXT, K2VER,
     *    I4RAN, I2VOL, I2CNO, I2WIN, I2DEP, I2COR, I2TRA, I2PLT, I2OTH,
     *    K2RES, K2RESN
C                                                              End CHDR.
```

5.7.2 CLOC.INC

```
C                                                               Include CLOC
      COMMON /LOCATI/ RPVAL, COND2R, AXDENU, GEOMD1, GEOMD2, GEOMD3,
     *    GEOMD4, RPLOC, AXINC, CTYP, CPREF, ROT, SAXLAB, ZDEPTH,
     *    ZAXIS, AXTYP, CORTYP, LABTYP, SGNROT, AXFUNC, KLOCL, KLOCM,
     *    KLOCF, KLOCS, KLOCA, KLOCB, NCHLAB
C                                                               End CLOC
```

5.7.3 CTVC.INC

```
C                                                               Include CTVC
      COMMON /TVCHAR/ NGRAY, NGRAPH, NIMAGE, MAXXTV, MAXINT, SCXINC,
     *    SCYINC, MXZOOM, NTVHDR, CSIZTV, GRPHIC, ALLONE, MAXXTK,
     *    CSIZTK, TYPSPL, TVALUS, TVXMOD, TVYMOD, ISUNUM, LUTOUT,
     *    OFMINP, OFMOUT, TVDUMS,
     *    TVZOOM, TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC,
     *    TYPMOV, YBUFF
C                                                               End CTVC
```

5.7.4 DHDR.INC

```
C                                                               Include DHDR
      INTEGER*2 K4OBJ, K4TEL, K4INS, K4OBS, K4DOB, K4DMP, K4BUN,
     *    K4PTP, K4CTP, K4CIC, K4CRP, K4CRT, K4EPO, K4DMX, K4DMN,
     *    K4BLK, K4IMN, K4IMC, K4PTY, K4BMJ, K4BMN, K4BPA, K4ARP,
     *    K4XSH, K4YSH, K4IMNO, K4IMCO, K4PTYO
      INTEGER*2 K8BSC, K8BZE, K8CRV, K8ORA, K8ODE, K8RST, K8ARV
      INTEGER*2 K3GCN, K3NIT
      INTEGER*2 K2PTPN, K2CTPN, K2EXTN
      INTEGER*2 K2PCN, K2DIM, K2NAX, K2BPX, K2INH, K2IMS, K2IMU,
     *    K2TYP, K2ALT, K2EXT, K2VER
      INTEGER*2 I4RAN, I2VOL, I2CNO, I2WIN, I2DEP, I2COR, I2TRA,
     *    I2PLT, I2OTH
      INTEGER*2 K2RES, K2RESN
```

5.7.5 DLOC.INC

```
C                                                               Include DLOC
      REAL*8     RPVAL(4), COND2R, AXDENU, GEOMD1, GEOMD2, GEOMD3,
     *   GEOMD4
      REAL*4     RPLOC(4), AXINC(4), CTYP(2,4), CPREF(2,2), ROT,
     *   SAXLAB(5,2)
      INTEGER*2 ZDEPTH(5), ZAXIS, AXTYP, CORTYP, LABTYP, SGNROT,
     *   AXFUNC(7), KLOCL, KLOCM, KLOCF, KLOCS, KLOCA, KLOCB,
     *   NCHLAB(2)
C                                                               End DLOC
```

5.7.6 DTVC.INC

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

5.8 ROUTINES

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

```
      AXEFND (NCHC, TYPE, NAXIS, CAT4, IOFF, IERR)
   Inputs:
      NCHC        I*2  Compare only first NCHC characters of axis type
      TYPE(2)     R*4  Unpacked char. axis type.
      NAXIS       I*2  the number of axes to search,
                          for uniform axes use: K2CTPN
                          for random axes use:  K2PTPN
      CAT4(*)     R*4  Catalog axis name list,
                          for uniform axes use: CAT4(K4CTP)
                          for random axes use : CAT4(K4PTP)
   Output:
      IOFF        I*2  Axis offset ( zero relative axis number)
      IERR        I*2  Return error code, 0=›OK, 1=›could not find.
```

5.8.2 CATDIR - manipulates catalog directory and will fill in the defaults used for NAME, CLASS, SEQ etc. if requested.

```
      CATDIR (OP, IVOL, CNO, NAME, CLASS, SEQ, PTYPE, USID,
     *   STAT, BUFF, IERR)
   Inputs:
     OP     R*4  specifies the desired operation:
                 searches find entry with specified data:
                    'SRCH' high seq # (if SEQ 0), return values
                    'SRNH' high seq # (if SEQ 0), NOT return values
                    'SRCN' next match, return values
                    'SRNN' next match, NOT return values
                  'OPEN' - create a new slot
                  'CLOS' - destroy a slot
                  'INFO' - return contents of a slot
                  'CSTA' - modify status of a slot
     IVOL   I*2   Disk volume containing catalog
                  0 -› all on searches, OPEN
     CNO    I*2   Slot number to begin: SRCN, SRNN, OPEN
                  Ignored if IVOL = 0 : searches, OPEN
                  Slot number to examine (solely): CLOS, INFO, CSTA
     NAME   R*4(3) File name: searches, OPEN, CLOS (12 packed chars)
     CLASS  R*4(2) File class: searches, OPEN, CLOS (6 packed chars)
     SEQ    I*2   File sequence number: searches, OPEN, CLOS
     PTYPE  I*2   File physical type (2 chars): searches, OPEN, CLOS
     USID   I*2   User identification #: searches, OPEN, CLOS
     STAT   R*4   Status (OP=CSTA): READ, WRIT, CLRD, or CLWR
   Outputs:
     CNO    I*2   Slot number found: searches, OPEN
     IVOL   I*2   If 0 on input, value actually used: searches, OPEN
     NAME   R*4(3) File name: SRCH, SRCN, INFO (12 packed chars)
```

```
CLASS R*4(2) File type: SRCH, SRCN, INFO (6 packed chars)
SEQ   I*2    File sequence number: SRCH, SRCN, INFO
PTYPE I*2    File physical file type (2 chars): SRCH, SRCN, INFO
USID  I*2    User identification #: SRCH, SRCN, INFO
STAT  R*4    Status: INFO
BUFF  I*2(256)  Working buffer
IERR  I*2    Error return
              1 =>  can't open cat file
              2 =>  input error
              3 =>  can't read catalog 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 CATIO - reads or writes blocks in the map catalog.

```
    CATIO (OP, IVOL, CNO, CATBLK, STAT, BUFF, IERR)
Inputs:  OP     R*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*2       Disk volume containing catalog (1 rel)
         CNO    I*2       Slot number of interest
         CATBLK I*2(256)  Array to be written on disk: WRIT, UPDT
         STAT   R*4       Status desired for slot after operation
                          'READ','WRIT','REST' where REST => no
                          change of status is desired
Outputs: CATBLK I*2(256)  Array read from disk: READ
         BUFF   I*2(256)  Working buffer
         IERR   I*2       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.
```

5.8.4 COORDT - translates between types of coordinates:

```
      COORDT (ITI, ITO, LONGI, LATI, EPOK, LONGO, LATO, IERR)
   Inputs: ITI    I*2   Input type (1 Ra, Dec; 2 gal, 3 ecliptic)
           ITO    I*2   Output type
           LONGI  R*8   Input longitude
           LATI   R*8   Input latitude
           EPOK   R*4   Epoch of positions (used very simply with
                        ecliptic coords only)
                        1950 assumed in Galactic conversions!!!!!!!!!
   Output: LONGO  R*8   Output longitude
           LATI   R*8   Output latitude
           IERR   I*2   error: 0 ok, 1 input error, 2 conversion fails
```

5.8.5 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*4   X pixel position
           YVAL   R*8   Y coordinate value
   Output: XVAL   R*8   X coordinate value
           IERR   I*2   0 ok, 1 out of range,
                                2 bad type, 3 undefined
   Common: /LOCATI/ position parameters must have been set
            up by SETLOC
```

5.8.6 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).

```
      SUBROUTINE FNDY (YPIX, XVAL, YVAL, IERR)
   Inputs: YPIX   R*4   Y pixel position
           XVAL   R*8   X coordinate value
   Output: YVAL   R*8   Y coordinate value
           IERR   I*2   0 ok, 1 out of range,
                                2 bad type, 3 undefined
   Common: /LOCATI/ position parameters must have been set
            up by SETLOC
```

5.8.7 MAPCLS - closes a map file and clears the catalog status.

```
      MAPCLS (OP, IVOL, CNO, LUN, IND, CATBLK, CATUP,
     *   WBUFF, IERR)
   Inputs:
      OP       R*4    OPcode used by MAPOPN to open this file
      IVOL     I*2    Disk volume containing map file
      CNO      I*2    Catalog slot number of file
      LUN      I*2    Logical unit # used for file
      IND      I*2    FTAB pointer for LUN
      CATBLK   I*2(256) New catalog header which can optionally
                         be written into header if OP=WRIT or INIT
                         Dummy argument if OP=READ
      CATUP    L*2       If TRUE write CATBLK into catalog,
                         ignored if OP = READ
   Outputs:
      IERR      I*2   0 = O.K.
                      1 = CATDIR couldn't access catalog
                      5 = illegal OP code
```

5.8.8 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           R*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*2   Logical unit # to use
  In/Out:
     NAMEIN(3)    R*4   Image name (name) (12 packed chars)
     CLASIN(2)    R*4   Image name (class) (6 packed chars)
     SEQIN        I*2   Image name (seq.#)
     USID         I*2   User identification #
     IVOL         I*2   Input disk unit
     TYPIN        I*2   Physical type of file (2 packed chars)
   Outputs:
     IND          I*2   FTAB pointer
     CNO          I*2   Catalog slot containing map
     CATBLK(256)I*2     Buffer containing current catalog block
     IERR         I*2   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
```

```
Buffer:
   WBUFF(256) I*2  Working buffer for CATIO and CATDIR
```

5.8.9 ROTFND - finds the map rotation angle from a given catalog block

```
   ROTFND (CAT4, ROT, IERR)
Inputs:
    CAT4(*)    R*4  File catalog header
Outputs:
    ROT        R*4  File rotation angle (degrees)
    IERR       I*2  Error code. 0=›OK, 1=›couldn't find axis.
```

5.8.10 SETLOC - uses the catalog header to build the values of the position common /LOCATI/ for use by position finding and axis labeling routines (at least).

```
   SETLOC (DEPTH, SWAPOK)
Inputs: DEPTH   I*2(5)   Position of map plane axes 3 - 7
        SWAPOK  L*2      T =› okay to swap axes if rotation
                         angle is near 90 degrees.
Common: /MAPHDR/ catalog block (not modified)
        /LOCATI/ position parms - created here
```

5.8.11 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. Currently this routine is in the AIPSUB area (AIPS program).

```
   TVFIND (MAXPL, TYPE, IPL, UNIQUE, CATBLK, SCRTCH,
  *   IERR)
Inputs: MAXPL   I*2        Highest plane number allowed (i.e., do
                           graphics count?)
        TYPE     I*2       2-char image type to restrict search
Output: IPL     I*2        Plane number found
        UNIQUE  L*2        T =› only one image visible now
                           (all types)
        CATBLK  I*2(256)   Image catalog block found
        SCRTCH  I*2(256)   Scratch buffer
        IERR    I*2        Error code: 0 =› ok
                               1 =› no image
                               2 =› IO error in image catalog
                               3 =› TV error
```

5.8.12 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+IABS(ICOR-ICORO)*INCS+(IF-1)*INCIF

   UVPGET (IERR)
Inputs: From common /MAPHDR/
   CATBLK(256)    I*2  Catalog block
   CAT4           R*4  same as CATBLK
   CAT8           R*8  same as CATBLK
Output: In common /UVHDR/
   SOURCE(2)      R*4  Packed source name.
   ILOCU          I*2  Offset from beginning of vis record of U
   ILOCV          I*2                    "                     V
   ILOCW          I*2                    "                     W
   ILOCT          I*2                    "                  Time
   ILOCB          I*2                    "              Baseline
   ILOCSU         I*2                    "             Source id.
   JLOCC          I*2  Order in data of complex values
   JLOCS          I*2  Order in data of Stokes' parameters.
   JLOCF          I*2  Order in data of Frequency.
   JLOCR          I*2  Order in data of RA
   JLOCD          I*2  Order in data of dec.
   JLOCIF         I*2  Order in data of IF.
   INCS           I*2  Increment in data for stokes (see above)
   INCF           I*2  Increment in data for freq. (see above)
   INCIF          I*2  Increment in data for IF.
   ICORO          I*2  Stokes value of first value.
   NRPARM         I*2  Number of random parameters
   LREC           I*2  Length in values of a vis record.
   NVIS           I*4  Number of visibilities
   FREQ           R*8  Frequency (Hz)
   RA             R*8  Right ascension (1950) deg.
   DEC            R*8  Declination (1950) deg.
   NCOR           I*2  Number of correlators
   ISORT          C*2  Sort order
   IERR           I*2  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
```

5.8.13 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 common /LOCATI/ which requires a previous call to SETLOC.

```
      XYPIX (X, Y, XPIX, YPIX, IERR)
   Inputs:  X    R*8   X-coordinate value (header units)
            Y    R*8   Y-coordinate value (header units)
   Output:  XPIX R*4   x-coordinate pixel location
            YPIX R*4   y-coordinate pixel location
            IERR I*2   0 ok, 1 out of range,
                             2 bad type, 3 undefined
```

5.8.14 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 common /LOCATI/ requiring a previous call to SETLOC. This program is the inverse of XYPIX.

```
      XYVAL (XPIX, YPIX, X, Y, Z, IERR)
   Inputs:
      XPIX     R*4  Pixel location, x-coordinate
      YPIX     R*4  Pixel location, y-coordinate
   Outputs:
      X        R*8  X-coordinate value at pixel location
      Y        R*8  Y-coordinate value at pixel location
      Z        R*8  Z-coordinate value (if part of a position
                    pair with either X or Y)
     IERR      I*2  0 ok, 1 out of range,
                          2 bad type, 3 undefined
   COMMON Inputs:
      /LOCATI/ position parms deduced from the map header by
      subroutine SETLOC
   Units are as in the map header: degrees for position coordinates.
```

5.8.15 YCINIT - Initialize image catalog for plane IPLANE. Generic Y routine: for local host operated TV device.

```
      YCINIT (IPLANE, BUFF)
   Input:  IPLANE     I*2     Image plane to initialize
   Output: BUFF(256)  I*2     Working buffer
```

5.8.16 YCOVER - checks to see if there are partially replaced images in any of the TV planes currently visible by quadrant. Generic Y routine: for local host-operated TV device.

```
      YCOVER (OVER, BUF, IERR)
   Outputs: OVER   L*2(4)   T => there are in quadr. I
            BUF    I*2(512) scratch
            IERR   I*2      Error code: 0 => ok, other catlg IO error
```

5.8.17 YCWRIT - Write image catalog block to image catalog. Generic Y routine: for local host-operated TV device.

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

5.8.18 YCREAD - Read image catalog block. Generic Y routine: for local host-operated TV device.

```
      YCREAD (IPLANE, IX, IY, ICTBL, IERR)
   Inputs:
      IPLANE I*2      plane containing image whose block is wanted
      IX     I*2      X pixel coordinate of a point within image
      IY     I*2      Y pixel coordinate of point within image
   Outputs:
      ICTBL  I*2(256) Image catalog block
      IERR   I*2       error codes: 0 => ok
                          1 => IX, IY lies outside image
                          2 => Catalog i/o errors
```

CHAPTER 6

DISK FILES

6.1 OVERVIEW

Most images, uv data sets, and other information in the AIPS system are kept in disk files. Image and uv data files to be kept longer than the execution of a single task are stored in cataloged files, although tasks may use scratch files for temporary storage. 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 the AIPS manual Volume 2. 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. This type of file may be expanded and contracted and physical I/O is always done in 512-byte blocks. The second type of file, known as "map" files, is used for image and uv data files. This type of file can be contracted, but not expanded, and I/O 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.)

There are several occasions when the programmer must be aware of the distinction between these two types of files. The first is in the setup and initialization of the CDCH.INC commons. This common must be declared and initialized to handle the largest number of each type of file which will be open at any given time. A description of this process is given in the chapter on tasks.

The other places where there is a 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/. 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/. 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/ 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 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*2 SYM, IRET, NX, NY, NP(2), BP, N2
INTEGER*4 NBYTE
LOGICAL*2 MAP
REAL*4    PHNAME(6)
REAL*8    XSIZE
INCLUDE 'INCS:DDCH.INC'
     ...
```

```
      INCLUDE 'INCS:CDCH.INC'
            ...
      DATA SYM /'MA'/,  MAP /.TRUE./,  N2 /2/
            ...
C                                        NX, NY are the size of an
C                                        image.  Make a  file
C                                        big enough for a REAL copy
C                                        of the image.
C
C                                        Compute the size in bytes.
C                                        Note: NWDPFP is from the
C                                        /DCHCOM/ and is the size of
C                                        a REAL word in terms of
C                                        short integers. 1 short
C                                        integer = 2 bytes
      BP = 2 * NWDPFP
      NP(1) = NX
      NP(2) = NY
      CALL MAPSIZ (N2, NP, BP, NBYTE)
C                                        Size now in NBYTE
C                                        Make physical name.
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 (SYM, IVOL, CNO, IVER, PHNAME, IERR)
C                                        filename now in PHNAME.
C                                        (error if IERR not 0)
C                                        Create file of type 'MA'
      CALL ZCREAT (IVOL, PHNAME, NBYTE, MAP, IERR)
C                                        Test for errors...
```

In the example above, a map file was created large enough to hold a NX by NY floating point image using the routine MAPSIZ to compute the correct size for the file. If this file is to be cataloged, then a catalog header record should be constructed and calls made to CATDIR and CATIO before the call to ZCREAT 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. 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 catalog file destroyed.

6.3.4 Expansion And Contraction Of Files

Regular (extension) files can be both expanded and compressed. Map (data) files can be compressed, but not expanded. 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. 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 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 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.

- MINI3 initializes I/O for image files; can specify a subimage for reads.

- MDIS3 does double buffered I/O for image files; requires initialization by MINI3.

- 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 (INCLUDEs DDCH.INC and CDCH.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 Common

The device Characteristics common, obtained from the INCLUDEs DDCH.INC and CDCH.INC contains a number of useful parameters about the host system.

XPRDMM	R*4	Printer points per millimeter
XTKDMM	R*4	Graphics points per millimeter
SYSNAM	R*4(5)	System name (20 char)
VERNAM	R*4	Version ID (4 char)
RLSNAM	R*4(2)	Release name (8 characters)
TIMEDA	R*4(15)	Min. TIMDEST time for each disk (days)
TIMESG	R*4	Min. TIMDEST time for SAVE/GET files (days)

TIMEMS	R*4	Min. automatic destruction time for messages
TIMESC	R*4	Min. automatic destruction time for scratch
TIMECA	R*4	Min. destruction time for empty catalogs.
TIMEBA	R*4(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*4(3)	1 => time between rolls (min) 2,3 polynomial terms for determining how long a job must wait before grabbing the AP.
RFILIT	R*4(14)	Spare
NVOL	I*2	Number of disk drives available to AIPS
NBPS	I*2	Number of bytes per disk sector
NSPG	I*2	Number of disk sectors per allocation granule
NBTB1	I*2	Number bytes in FTAB / non-FTAB device
NTAB1	I*2	Max number of non-FTAB devices open at once
NBTB2	I*2	Number bytes in FTAB / slow I/O device
NTAB2	I*2	Max number of slow I/O devices open at once
NBTB3	I*2	Number bytes in FTAB / fast I/O device
NTAB3	I*2	Max number of fast I/O devices open at once
NTAPED	I*2	Number of tape drives available to AIPS
CRTMAX	I*2	Number lines / CRT terminal page
PRTMAX	I*2	Number lines / printer page
NBATQS	I*2	Number batch AIPSs in system
MAXXPR	I*2(2)	Number of plotter dots / page in X, Y
CSIZPR	I*2(2)	Number of plotter dots / character in X, Y
NINTRN	I*2	Maximum # simultaneous interactive AIPSs
KAPWRD	I*2	# 1024s of words of array processor memory
NCHPFP	I*2	# characters / floating point
NWDPFP	I*2	# words / floating point
NWDPDP	I*2	# words / double-precision floating point
NWDPLI	I*2	# words / long integer
NWDPLO	I*2	# words / logical
NBITWD	I*2	# bits / word
NWDLIN	I*2	# words in a POPS input line
NCHLIN	I*2	# characters in a POPS input line
NTVDEV	I*2	# television display devices available
NTKDEV	I*2	# graphics display devices available
BLANKV	I*2	Integer magic value => blanked pixel
NTVACC	I*2	Number POPS programs allowed access to TV devices
NTKACC	I*2	Number POPS programs allowed access to graphics
UTCSIZ	I*2	Private catalog size (0=>public)
BYTFLP	I*2	Byte flip, 0=none, 1=bytes, 2=words, 3=both
USELIM	I*2	Maximum user number
NBITCH	I*2	# bits per character
NCHPRT	I*2	Width of line printer in characters.
KAP2WD	I*2	# 1024s words of secondary AP memory.
DEVTAB	I*2(50)	Device type code numbers
FTAB	I*2(*)	I/O driving tables

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. The pixels may be one of several types, but in practice, they are floating point values (a few applications allow scaled integers). 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.

6.4.4.1 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*2 IRET, CNO, IVOL, IVER, MA, LUN, IND
      LOGICAL*2 MAP, EXCL, WAIT
      REAL*4    PHNAME(6)
           .
           .
      DATA MAP, EXCL, WAIT /.TRUE.,.TRUE.,.TRUE./
      DATA IVER /1/,    MA /'MA'/,    LUN /16/
           .
           .
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)
```

6.4.4.2 MINI3 And MDIS3 - Once the image file is opened, I/O is normally initialized by a call to MINI3; I/O is done by calls to MDIS3, with a final call to MDIS3 to flush the buffer, if necessary. MINI3 sets up the bookkeeping for one plane of an image at a time; if multiple planes are to be read, multiple calls to MINI3 must be made. A rectangular window in a given plane can be specified to MINI3, 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, MDIS3 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 MINI3 and MDIS3 are given at the end of this chapter.

Note: eventually MINI3 and MDIS3 will be replaced by MINIT and MDISK. In this new version, MINIT will accept true I*4 arguments rather that pseudo I*4 arguments.

6.4.4.3 Multi-plane Images (COMOF3) - If the image has more than two dimensions, planes parallel to the first plane can be accessed using the block offset argument to MINI3. The subroutine COMOF3 can be used to compute the block offset. The block offset is an I*4 number whose value for the first plane is 1. COMOF3 returns a value which is to be added to the block offset for the first plane. COMOF3 will eventually be replaced by COMOFF which will take true rather than pseudo I*4 arguments.

An example of the use of COMOF3 to compute the block offset:

```
      INTEGER*2 CATBLK(256), BP, BLKOF(2), ONE(2), PLARR(5),
     *   IERR, PLUS
      INCLUDE 'DHDR.INC'
      INCLUDE 'DDCH.INC'
            .
            .
      INCLUDE 'CHDR.INC'
      INCLUDE 'CDCH.INC'
      COMMON /MAPHDR/ CATBLK
            .
            .
      DATA ONE /1,0/, PLUS /'PL'/
            .
            .
C                                          Compute bytes / per pixel.
C                                          assume REAL format file.
C                                          NWDPFP = # short integers
C                                            per floating value.
C                                            Obtained from DDCH.INC and
C                                            CDCH.INC includes.
      BP = 2 * NWDPFP
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 /MAPHDR/
      CALL COMOF3 (CATBLK(K2DIM), CATBLK(K2NAX), PLARR, BP,
     *   BLKOF, IERR)
C                                          Add block offset for first
C                                          plane.
      CALL ZMATH4 (BLKOF, PLUS, ONE, BLKOF)
C                                          BLKOF now contains the value
C                                          to send to MINI3 to get the
C                                          specified plane.
```

A detailed description of the call sequence for COMOF3 is given at the end of this chapter.

6.4.4.4 Example Of MINI3 And MDIS3 - 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   number ISCR1 and ISCR2 and writes them in the /CFILES/ scratch
C   file number ISCR3
C   Inputs:
C    NX, NY  I*2  Number of pixels per row and number of rows
C    ISCR1   I*2  /CFILES/ scratch file number of first input file
C    ISCR2   I*2  /CFILES/ scratch file number of second input file
C    ISCR3   I*2  /CFILES/ scratch file number of output file
C   Output:
C    IERR    I*2  Return code, 0=>OK, otherwise error.
C-----------------------------------------------------------------------
      INTEGER*2 N1, N2, N8
      INTEGER*2 FIND1, FIND2, FIND3, BIND1, BIND2, BIND3, BO(2), BP,
     *   WIN(4), NX, NY, BUFSZ1, BUFSZ2, BUFSZ3, LUN1, LUN2, LUN3, SC
      LOGICAL*2 T,F
      REAL*4 READ, WRITE, FINI, FILE(6)
      REAL*4 BUFF1(4096), BUFF2(4096), BUFF3(4096)
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CFIL.INC'
      DATA T, F /.TRUE.,.FALSE./
      DATA READ, WRITE, FINI, SC /'READ','WRIT','FINI','SC'/
      DATA BO, WIN /1,0, 4*0/
      DATA N1, N2, N8 /1,2,8/,
C                                          Use LUNs 16, 17, 18
      DATA LUN1, LUN2, LUN3 /16,17,18/
C-----------------------------------------------------------------------
C                                          Set bytes per pixel (floating)
      BP = 2 * NWDPFP
C                                          Set buffer sizes
      BUFSZ1 = 4096 * BP
      BUFSZ2 = 4096 * BP
      BUFSZ3 = 4096 * BP
C                                           Open and init ISCR1
      CALL ZPHFIL (SC, SCRVOL(ISCR1), SCRCNO(ISCR1), N1, FILE, IERR)
      CALL ZOPEN (LUN1, FIND1, SCRVOL(ISCR1), FILE, T, F, T, IERR)
C                                           Check for error
      IF (IERR.EQ.0) GO TO 10
         ENCODE (80,1000,MSGTXT) IERR, READ, N1
         GO TO 990
 10   CALL MINI3 (READ, LUN1, FIND1, NX, NY, WIN, BUFF1, BUFSZ1,
     *   BP, BO, IERR)
C                                           Check for error
      IF (IERR.EQ.0) GO TO 20
         ENCODE (80,1010,MSGTXT) IERR, READ, N1
```

```
         GO TO 990
C                                          Open and init ISCR2
 20    CALL ZPHFIL (SC, SCRVOL(ISCR2), SCRCNO(ISCR2), N1, FILE, IERR)
       CALL ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), FILE, T, F, T, IERR)
C                                          Check for error
       IF (IERR.EQ.0) GO TO 30
          ENCODE (80,1000,MSGTXT) IERR, READ, N1
          GO TO 990
 30    CALL MINI3 (READ, LUN2, FIND2, NX, NY, WIN, BUFF2, BUFSZ2,
      *    BP, BO, IERR)
C                                          Check for error
       IF (IERR.EQ.0) GO TO 40
          ENCODE (80,1010,MSGTXT) IERR, READ, N2
          GO TO 990
C                                          Open and init ISCR3
 40    CALL ZPHFIL (SC, SCRVOL(ISCR3), SCRCNO(ISCR3), N1, FILE, IERR)
       CALL ZOPEN (LUN3, FIND3, SCRVOL(ISCR3), FILE, T, F, T, IERR)
C                                          Check for error
       IF (IERR.EQ.0) GO TO 50
          ENCODE (80,1000,MSGTXT) IERR, WRITE
          GO TO 990
 50    CALL MINI3 (WRITE, LUN3, FIND3, NX, NY, WIN, BUFF3, BUFSZ3,
      *    BP, BO, IERR)
C                                          Check for error
       IF (IERR.EQ.0) GO TO 60
          ENCODE (80,1010,MSGTXT) IERR, WRITE
          GO TO 990
C                                          Loop, adding rows.
 60    DO 110 I = 1,NY
C                                          Read ISCR1
          CALL MDIS3 (READ, LUN1, FIND1, BUFF1, BIND1, IERR)
C                                          Check for error
          IF (IERR.EQ.0) GO TO 70
             ENCODE (80,1060,MSGTXT) IERR, READ, N1
             GO TO 990
C                                          Read ISCR2
 70       CALL MDIS3 (READ, LUN2, FIND2, BUFF2, BIND2, IERR)
C                                          Check for error
          IF (IERR.EQ.0) GO TO 80
             ENCODE (80,1060,MSGTXT) IERR, READ, N2
             GO TO 990
C                                          Write ISCR3
 80       CALL MDIS3 (WRITE, LUN3, FIND3, BUFF3, BIND3, IERR)
C                                          Check for error
          IF (IERR.EQ.0) GO TO 90
             ENCODE (80,1060,MSGTXT) IERR, WRITE
             GO TO 990
C                                          Add row.
 90       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 MDIS3 (FINI, LUN3, FIND3, BUFF3, BIND3, IERR)
C                                          Check for error
      IF (IERR.EQ.0) GO TO 120
         ENCODE (80,1060,MSGTXT) IERR, FINI
         GO TO 990
C                                          Close files.
 120  CALL ZCLOSE (LUN1, FIND1, IERR)
      CALL ZCLOSE (LUN2, FIND2, IERR)
      CALL ZCLOSE (LUN3, FIND3, IERR)
C                                         Finished OK.
      IERR = 0
      GO TO 999
C                                         An error has occurred - send
C                                         message
 990  CALL MSGWRT (N8)
 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
```

6.4.4.5 MINS3 And MSKI3 - 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 MINS3 and MSKI3 will do this operation. Descriptions of these routines can be found at the end of this chapter.

Note: MINS3 AND MSKI3 will eventually be replaced by MINSK and MSKIP. In this new form, MINSK will accept true I*4 arguments.

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 with or without various format conversions. 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

6.4.6.1 Single- And Multi-source Files -

AIPS has traditionally had single-source data files, containing data from a single source which had already been calibrated and had most of the bad data flagged. In order to allow the development of calibration and editing software, a new "type" of data file is allowed which may 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 this "new" type of data file is very similar to the single source file so existing software can access these files. 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.

6.4.6.2 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.

6.4.6.3 Visibility Record Structure - AIPS uv data is organized in the data file in the same way that similar data is organized on a FITS 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 of the visibility logical record is fixed in a given data base, but may vary from one data base to another. All values are in floating point format, and records may span disk sector boundaries.

The random parameters can be in any order, but the names of only the first seven 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", "VV-L", "WW-L" indicating that the coordinates correspond to the tangent point of the data 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.

The regular portion of the array is like an image array in that the order of the axes is arbitrary. In practice, 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 CH table which gives the

frequency offset from the reference frequency for that IF. The CH table is accessed by the routine CHNDAT, whose call sequence is given at the end of this chapter.

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

```
| u, v, w, t, b| R1, I1, W1, R2, I2, W2, R3, I3, W3, R4, I4, W4|
  random              RR          LL          RL          LR
   parameters          rectangular data array
```

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).
- 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. For the VLA this is usually the integration time in tens of seconds. In general, it is arbitrary. Data with Wn <= 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

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(K2TYP)) as a two-character 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 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.

6.4.6.4 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/. These pointers are placed in a common which is obtained by the DUVH.INC and CUVH.INC INCLUDEs. 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 +
         IABS (ICOR-ICORO) * INCS
```

6.4.6.5 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.

- 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.

6.4.6.6 UVINIT And UVDISK - UV data files may be located and opened using routine MAPOPN and read or written using UVINIT and UVDISK in much the same manner in which image files are read with MINI3 and MDIS3. One significant difference between UVDISK and MDIS3 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 MINI3 and UVINIT is that, unlike MINI3, 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 transfers (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 => -NIO or 0, in which case the entire remaining data in the buffer is 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 MDIS3 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. This avoids having a counter for the visibilities (remember that I*2 variables can only count to 32767). 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.

6.4.6.7 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  number ISCR1 and
C   writes them in the /CFILES/ scratch  file number ISCR2.
C   The current values in the /UVHDR/ common are assumed to describe
C   the uv data files.
C   Inputs:
C    ISCR1     I*2  /CFILES/ scratch file number of input file
C    ISCR2     I*2  /CFILES/ scratch file number of output file
C    LUN1      I*2  Logical unit number to use for file 1
C    LUN2      I*2  Logical unit number to use for file 2
C    BUFF1(*) R*4  I/O buffer to use for file 1
C    BUFF2(*) R*4  I/O buffer to use for file 2
C    BUFSZ1    I*2  Size of BUFF1 in bytes
C    BUFSZ2    I*2  Size of BUFF2 in bytes
C   Inputs from common /UVHDR/
C    NVIS      I*4  Number of visibility records
C    LREC      I*2  logical record length.
C    NRPARM    I*2  number of random parameters.
C    ICORO     I*2  (signed) value of first Stokes' parameter.
C    JLOCF     I*2  zero relative order of the frequency axis in the
C                   data array.
C    JLOCS     I*2  relative order of the Stokes' axis.
C    JLOCIF    I*2  relative order of the IF axis.
C    INCF      I*2  word increment in the data array between
C                   successive frequencies at the same location on all
C                   other axes.
C    INCS      I*2  word increment in the data array between
C                   successive Stokes' values.
C    INCS      I*2  word increment in the data array between
C                   successive IF values.
C   Inputs from common /MAPHDR/
C    CATBLK(256)  I*2  Catalog header record
C   Output:
C    IERR      I*2  Return code, 0=>OK, otherwise error.
C-----------------------------------------------------------------
      INTEGER*2 N1, N2, N8
      INTEGER*2 FIND1, FIND2, BIND1, BIND2, BP, NFREQ, NSTOKE, NIF,
     *   BUFSZ1, BUFSZ2, LUN1, LUN2, I, IV, IFQ, IST, IIF,
     *   NIOIN, NIOUT, CATBLK(256), INDEX, JCORO, SC, NILIM
      INTEGER*4 BO, VO
      LOGICAL*2 T,F
      REAL*4 READ, WRITE, FLUSH, FILE(6)
      REAL*4 BUFF1(1), BUFF2(1)
C                                        Listings of the standard
C                                        INCLUDE files are at the end
C                                        of the chapter on tasks.
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DUVH.INC'
```

```
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CUVH.INC'
      INCLUDE 'INCS:CHDR.INC'
C                                          Catalog header block common
      COMMON /MAPHDR/ CATBLK
      DATA T, F /.TRUE.,.FALSE./
      DATA READ, WRITE, FLUSH, SC  /'READ','WRIT','FLSH','SC'/
      DATA VO, BO /0,1/
      DATA N1, N2, N8 /1,2,8/
C-----------------------------------------------------------------------
C                                          Set bytes per pixel (floating)
      BP = 2 * NWDPFP
C                                          Take absolute value of first
C                                          Stokes' value.
      JCORO = IABS (ICORO)
C                                          Find dimension of freq
C                                          and Stokes axes.
      NFREQ = CATBLK(K2NAX+JLOCF)
      NSTOKE = CATBLK(K2NAX+JLOCS)
C                                          May not have IF axis
      NIF = 1
      IF (JLOCIF.GT.0) NIF = CATBLK(K2NAX+JLOCIF)
C                                            Open and init ISCR1
      CALL ZPHFIL (SC, SCRVOL(ISCR1), SCRCNO(ISCR1), N1, FILE, IERR)
      CALL ZOPEN (LUN1, FIND1, SCRVOL(ISCR1), FILE, T, F, T, IERR)
C                                            Check for error
      IF (IERR.EQ.0) GO TO 10
         ENCODE (80,1000,MSGTXT) IERR, READ, N1
         GO TO 990
C                                          Let UVINIT determine #/call
 10   NIOIN = 0
      CALL UVINIT (READ, LUN1, FIND1, NVIS, VO, LREC, NIOIN,
     *   BUFSZ1, BUFF1, BO, BP, BIND1, IERR)
C                                            Check for error
      IF (IERR.EQ.0) GO TO 20
         ENCODE (80,1010,MSGTXT) IERR, READ, N1
         GO TO 990
C                                            Open and init ISCR2
 20   CALL ZPHFIL (SC, SCRVOL(ISCR2), SCRCNO(ISCR2), N1, FILE, IERR)
      CALL ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), FILE, T, F, T, IERR)
C                                            Check for error
      IF (IERR.EQ.0) GO TO 30
         ENCODE (80,1000,MSGTXT) IERR, WRITE, N2
         GO TO 990
 30   NIOUT = 8
      CALL UVINIT (WRITE, LUN2, FIND2, NVIS, VO, LREC, NIOUT,
     *   BUFSZ2, BUFF2, BO, BP, BIND2, IERR)
      NILIM = NIOUT
      NIOUT = 0
C                                            Check for error
      IF (IERR.EQ.0) GO TO 60
         ENCODE (80,1010,MSGTXT) IERR, WRITE, N2
```

```
         GO TO 990
C                                          Loop through data file.
C                                          Read input file
 60      CALL UVDISK (READ, LUN1, FIND1, BUFF1, NIOIN, BIND1, IERR)
C                                           Check for error
         IF (IERR.EQ.0) GO TO 70
            ENCODE (80,1060,MSGTXT) IERR, READ, N1
            GO TO 990
C                                          Check if data all read.
 70      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
            IF (NIOUT.LT.NILIM) GO TO 100
               CALL UVDISK (WRITE, LUN2, FIND2, BUFF2, BIND2,
     *            NIOUT, IERR)
               NIOUT = 0
C                                           Check for error
               IF (IERR.EQ.0) GO TO 100
                  ENCODE (80,1060,MSGTXT) IERR, WRITE
                  GO TO 990
 100        CONTINUE
C                                          Loop back for more data
 110     GO TO 60
C                                          Finished, flush buffer.
C                                          No more output records.
 120  IF (NIOUT.GT.0) CALL UVDISK (WRITE, LUN2, FIND2, BUFF2,
     *    BIND2, NIOUT, IERR)
      NIOUT = 0
      IF (IERR.EQ.0) CALL UVDISK (FLUSH, LUN2, FIND2, BUFF2,
     *    BIND2, NIOUT, IERR)
C                                           Check for error
      IF (IERR.EQ.0) GO TO 130
         ENCODE (80,1060,MSGTXT) IERR, FINI
```

```
         GO TO 990
C                                                Close files.
 130  CALL ZCLOSE (LUN1, FIND1, IERR)
      CALL ZCLOSE (LUN2, FIND2, IERR)
      IERR = 0
      GO TO 999
C                                                Error.
 990  CALL MSGWRT (N8)
 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
```

6.4.7 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. The details of the extension file structure are described in the AIPS manual volume 2 for most types of extension files.

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 tables manipulating routines, including writing to, and reading from, tape automatically, are available.

6.4.7.1 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.

6.4.7.2 EXTINI And EXTIO - NOTE: TABINI and TABIO are strongly preferred over EXTINI and EXTIO.

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.

6.4.8 Text Files

AIPS uses a number of text files, such as the HELP and RUN files. At the moment, the text file capability is read only. There are several routines which allow access to text files: ZTOPEN, ZTREAD, ZTCLOS, and KEYIN.

- 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.
- 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.

AIPS I/O routines have a number of standard places that they can find text files. These include the RUN file area, the HELP file area, and various source code areas. If a programmer wishes to read an arbitrary text file, the best thing to do is to put the file 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. An example of the use of ZTREAD to read a file named "INDATA" from the RUN area follows:

```
      INTEGER*2 LUN, FIND, LINE(70), IERR, RU, N1, N8, N24
      LOGICAL*2 WAIT
      REAL*4    PNAME(6), MNAME(2), XNAME(6), YNAME(2)
      INCLUDE 'INCS:DDCH.INC'
                     .
                     .
      INCLUDE 'INCS:CDCH.INC'
      DATA WAIT /.TRUE./, YNAME /'INDA','TA  '/,  LUN /10/
      DATA RU /'RU'/
      DATA N1, N8, N24 /1,8,24/
                     .
                     .
C                                          Pack MNAME
      CALL CHPACK (N8, YNAME, N1, MNAME)
C                                          Make file name
      CALL ZPHFIL (RU, N1, N1, N1, PNAME, IERR)
C                                          Open file
C                                          VERNAM is from the common
C                                          in INCLUDE CDCH.INC
      CALL ZTOPEN (LUN, FIND, N1, PNAME, MNAME, VERNAM, WAIT, IERR)
C                                          Error if IERR .NE. 0
                     .
                     .
C                                          Read line from file.
      CALL ZTREAD (LUN, FIND, LINE, IERR)
C                                          Error if IERR .NE. 0
C                                          Next line of test from file
C                                          is now in array LINE
                     .
                     .
C                                          Close file.
      CALL ZTCLOS (LUN, FIND, IERR)
```

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

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 buffers 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 common (INCLUDEs DDCH.INC and CDCH.INC) contains 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 short integers 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 MINI3, MDIS3, MINS3, MSKI3, 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.

For non-disk transfers, the number of bytes transferred by ZFIO is arbitrary. 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 ROUTINES

6.6.1 CALCOP - copys 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)
```

Input:

DISK	I*2	Disk number for cataloged output file. If .LE. 0, then the output file is a /CFILES/ scratch file.
CNOSCR	I*2	Catalog slot number for cataloged file; /CFILES/ scratch file number if a scratch file, IF DISK=CNOSCR=0, then the scratch is created.
BUFFER(*)	R*4	Work buffer for writing.
BUFSZ	I*2	Size of BUFFER in bytes.

Input via common:

CATBLK(256)	I*2	Catalog header block from UVGET
NVIS	I*2	(/UVHDR/) Number of vis. records.
LREC	I*2	(/UVHDR/) length of vis. record in R*4 words.
NRPARM	I*2	(/UVHDR/) number of (R*4) random parameters.

Output:

CNOSCR	I*2	Scratch file number if created.
IRET	I*2	Error code: 0 => OK, >0 => failed, abort process.

Output via common:

CATBLK(256)	I*2	Catalog header block with actual no. records.
NVIS	I*2	(/UVHDR/) Actual number of vis. records.

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

6.6.2 CHNDAT - creates and fills or reads CH (IF descriptor) extension tables.

```
      CHNDAT (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
     *   NIF, FOFF, ISBAND, IERR)
```

Inputs:

OPCODE	R*4	Operation code: 'WRIT' = create/init for write or read; 'READ' = open for read only
BUFFER(512)	I*2	I/O buffer and related storage, also defines file

```
                    if open.
     DISK           I*2 Disk to use.
     CNO            I*2 Catalog slot number
     VER            I*2 CH file version
     CATBLK(256)    I*2 Catalog header block.
     LUN            I*2 Logical unit number to use
    Input/Output:
     NIF            I*2 Number of IFs.
     FOFF(*)        R*8 Frequency offset in Hz from ref. freq.
                        True = reference + offset.
     ISBAND(*)      I*2 Sideband of each IF.
                        -1 => 0 video freq. is high freq. end
                         1 => 0 video freq. is low freq. end
    Output:
     IERR           I*2 Return error code, 0=>OK, else TABINI or TABIO
                        error.
```

6.6.3 COMOF3 - Computes the block offset BLKOF of a 2-D map plane in a NAX-dimensional map from the beginning of the map.

```
      COMOF3 (NAX, SAX, PLARR, BYTPIX, BLKOF, IERR)

   Inputs:
      NAX       I*2  Number of axes in map
      SAX(7)    I*2  Number of pixels on each axis
      PLARR(5)  I*2  Depth of required plane along other axes
      BYTPIX    I*2  Bytes per pixel in map
   Outputs:
      BLKOF(2)  I*2  Pseudo I4 block offset
      IERR      I*2  Error return 0 = OK, 1= error in NAX
```

6.6.4 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, BP, NREC, BUFFER, IERR)
   Input:
      OPCODE          R*4   Operation code, 'READ' => read only,
                                            'WRIT' => read/write
      PTYP            I*2   Physical extension type (e.g., 'CC')
      VOL             I*2   Volume number
      CNO             I*2   Catalog slot number
      VER             I*2   Version number: (<= 0 => write a new one,
                            read the latest one)
      CATBLK(256)     I*2   Catalog block of cataloged file.
```

LUN	I*2	Logical unit number to use.
LREC	I*2	Record length in units of BP (write new)
BP	I*2	Bytes per value. 0=› Use existing value
NREC	I*2	(used for 'WRIT' only) Number of logical records to create in the initial file and/or the number of records by which to extend the file when it fills up.
BUFFER(*)	I*2	Work buffer, at least 1024 bytes in size, more if logical record longer than 512 bytes

Output:

LREC	I*2	Logical record length (in units of BP) for read/write old files
BP	I*2	BP if input value = 0 and a file exists.
VER	I*2	Version number used.
CATBLK(256)	I*2	Catalog block updated if necessary.
IND	I*2	FTAB pointer.
BUFFER(*)	I*2	Header info.
IERR	I*2	Return error code. 0 =› OK 1 =› bad input. 2 =› could not find or open 3 =› create/I/O problem.

Usage 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' in the catalog, 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*2 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 (2 char per word)
11 - 16	Creation date, time
17 - 28	File name (packed character string)
29	Volume number on which file resides.
30 - 32	Last write-access task (2 char per word)
33 - 38	Last write-access time, date

39 - 56	reserved. (53-56 used by EXTIO:
	53 = # I*2 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.

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

```
   EXTIO (OPCODE, LUN, IND, IRNO, RECORD, BUFFER, IERR)
Inputs:
   OPCODE          R*4  Opcode 'READ','WRIT','CLOS'
   LUN             I*2  Logical unit number
   IND             I*2  FTAB pointer
   IRNO            I*2  Logical record no. 0=› next.
   RECORD( )       I*2  Array containing record to be written
   BUFFER( )       I*2  Work buffer = 512 bytes + enough 512 byte
                        blocks for at least one full logical record.
Output:
   RECORD( )       I*2  Array containing record read.
   BUFFER( )       I*2  buffer.
   IERR            I*2  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. EXTINI MUST be called before EXTIO.

6.6.6 GETVIS - gets and reformats uv data. May return multiple Stokes types. Requires setup by SETVIS.

```
   GETVIS (MODE, MVIS, JADR, SFACT, ALLWT, DATA, WT,
  *   VIS, IERR)

Inputs:
   MODE            I*2  Operation number (see SETVIS).
                        When MODE = 2 or 3 and RL and LR are given
                        the U visibility is multiplied by i.
   MVIS            I*2  Number of visibilities wanted.
   JADR(2,MVIS)    I*2  Pointers set by SETVIS.
   SFACT(2,MVIS)   R*4  Factors set by SETVIS.
   ALLWT           L*2  Flag set by SETVIS, if .TRUE. all relevant
                        weights must be positive.
   DATA(3,*)       R*4  Visibility portion of input data.
Outputs:
   WT              R*4  Average weight.
   VIS(MVIS)       CMPX Visibilities.
```

IERR	I*2	Error code, 0=›OK, 1 =› bad weights.(data flagged). 2 = bad input.

6.6.7 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*2	Operation number (see SET1VS). When MODE = 3 and RL and LR are given, the U visibility is multiplied by i.
MVIS	I*2	Number of visibilities wanted.
JADR(2)	I*2	Pointers set by SET1VS.
JINC	I*2	Increment between vis.
SFACT(2)	R*4	Factors set by SET1VS.
ALLWT	L*2	If true all vis are required.
STOKES	L*2	True if input data true Stokes'. Used for UPOL only.
DATA(3,*)	R*4	Visibility portion of input data.

Outputs:

WT	R*4	Average weight.
VIS(MVIS)	CMPX	Visibilities.
IRET	I*2	Error code, 0=›OK, 1 =› bad weights.(data flagged).

6.6.8 KEYIN - Standard Fortran version of the CIT VLBI KEYIN subroutines. This subroutine reads keyed parameters on card images. The text file should be opened via a call to ZTOPEN before the first call to KEYIN and closed via a call to ZTCLOS after the last call. (HINT: use LUN = 10 for the RUN area.)
Note: in this version, time-like entries in the form hh:mm:ss will be returned in hours.

```
KEYIN (KEYS, VALUES, N, ENDMRK, MODE, LUN, FIND, IERR)
```

Inputs:

KEYS(2,N)	R*4	array of parameter names (packed characters) Array names should have the last characters indicate the element number. Should all be in upper case characters.
VALUES(N)	R*8	array to receive values or defaults, each value corresponds to a KEY.
N	I*2	number of parameters (dimension of keys, values)
ENDMRK(2)	R*4	special keyword to indicate end of input
MODE	I*2	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*2    LUN to read from (used in call to ZTOPEN)
    FIND         I*2    FTAB pointer for input. (from ZTOPEN)
   Outputs:
    VALUES(N)    R*8    Values or input defaults.
    N            I*2    (MODE=3 only) number of values found
    IERR         I*2    error code, 0=>OK, 1=>EOF found, 2=>Error
```

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

```
      MAPSIZ (NAX, NP, NB, ISIZE)
  Inputs:  NAX    I*2       # axes
           NP     I*2(NAX)  # pixels on each axis.
           NB     I*2       # bytes / pixel
  Output:  ISIZE  I*4       file size in bytes
```

6.6.10 MAPCLS - closes a map file and clears the catalog status.

```
      MAPCLS (OP, IVOL, CNO, LUN, IND, CATBLK, CATUP,
     *   WBUFF, IERR)
   Inputs:
      OP       R*4    OPcode used by MAPOPN to open this file
      IVOL     I*2    Disk volume containing map file
      CNO      I*2    Catalog slot number of file
      LUN      I*2    Logical unit # used for file
      IND      I*2    FTAB pointer for LUN
      CATBLK   I*2(256) New catalog header which can optionally
                         be written into header if OP=WRIT or INIT
                         Dummy argument if OP=READ
      CATUP    L*2      If TRUE write CATBLK into catalog,
                        ignored if OP = READ
   Outputs:
      IERR      I*2   0 - O.K.
                      1 - CATDIR couldn't access catalog
                      5 - illegal OP code
```

6.6.11 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           R*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*2   Logical unit # to use
  In/Out:
     NAMEIN(3)    R*4   Image name (name) (12 packed chars)
     CLASIN(2)    R*4   Image name (class) (6 packed chars)
     SEQIN        I*2   Image name (seq.#)
     USID         I*2   User identification #
     IVOL         I*2   Input disk unit
     TYPIN        I*2   Physical type of file (2 packed chars)
   Outputs:
     IND          I*2   FTAB pointer
     CNO          I*2   Catalog slot containing map
     CATBLK(256)I*2     Buffer containing current catalog block
     IERR         I*2   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
   Buffer:
      WBUFF(256) I*2    Working buffer for CATIO and CATDIR
```

6.6.12 MCREAT - creates and catalogs an image data file based on a catalog header block in common /MAPHDR/.

```
      MCREAT (IVOL, CNO, WBUFF, IERR)
   In/Outs:
      IVOL        I*2     Volume # on which to put file: 0 => ALL
                          on output has volume used
      WBUFF       I*2(256) Working buffer
   Outputs:
      CNO         I*2       Catalog slot number
      IERR        I*2       Error code; 0 => o.k.
                                        1 => couldn't create, no room
                                        2 => no create, duplicate name
                                        3 => no room in catalog
                                        4 => I/O problem on catalog
                                        5 => Other Create errors
```

```
Common: (in/out)
   CATBLK     I*2(256) Catalog block (via common MAPHDR)
   CATB4      R*4(128) Catalog block (equivalenced to CATBLK)
```

The file created 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 => lowest unique). The extension file areas of the CATBLK are cleared and the "DATE-MAP" string is filled in.

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

6.6.14 MDIS3 - reads or writes image data to/from disks and other devices.

```
    MDIS3 (OP, LUN, FIND, BUFF, BIND, IERR)
Inputs:
  OP    I*4  Op code char string 'WRIT', 'READ','FINI'
  LUN   I*2  logical unit number
  FIND  I*2  Pointer to FTAB returned by ZOPEN
Input and/or output:
  BUFF  ??   Buffer holding data, you better know  specification
Output:
  BIND  I*2  Pointer to position in buffer of first pixel in window
             in the present line
  IERR  I*2  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
```

MDIS3 sets array index to the start of the next line wanted.

NOTE: the line sequence is set by the WIN parameter in MINI3, if the values of WIN(2) and Win(4) are switched, then the file will be accessed backwards.
A call with OP = 'FINI' flushes the buffer when writing.
MINI3 MUST be called before MDIS3.

6.6.15 MINI3 - initializes the I/O tables for MDIS3.

```
    MINI3 (OP, LUN, IND, LX, LY, WIN, BUFF, BFSZ, BYTPIX,
   *   BLKOF, IERR)
```

Inputs:

OP	R*4	Operation code character string: 'READ','WRIT'
LUN	I*2	logical unit number
IND	I*2	pointer to FTAB, returned by ZOPEN when file is opened
LX	I*2	Number of pixels per line in X-direction for whole plane
LY	I*2	Number of lines in whole plane.
WIN	I*2(4)	Xmin,Ymin,Xmax,Ymax defining desired subrectangle in the plane. A subimage may NOT be specified for 'WRIT'.
BFSZ	I*2	Size of total available buffer in bytes, should be even Special case: BUFSZ=32767 is treated as though BUFSZ=32768 to allow double buffering of 16Kbyte records.
BYTPIX	I*2	Number of bytes per pixel in stored map
BLKOF	I*2(2)	Pseudo I*4 block number, 1 relative, of first map pixel in the desired plane. Use COMOF3 + ZMATH4 to set.

Outputs:

IERR	I*2	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

MINI3 sets up 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. One can run the planes together iff the rows are not blocked: iff NBPS / (LX * BYTPIX) < 2.
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 I/O 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 transferred per I/O op
2 =>	No. of major times a buffer has been accessed
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 (lsb  I*4)
 8 =>  msb of pseudo I*4 block offset
 9 =>  Block increment in file for each operation
10 =>  No. of bytes transferred
11 =>  I/O op code 1=> read, 2 => write.
12 =>  BYTPIX
13 =>  # rows / major line (>= 1)
14 =>  # times this major line has been accessed
15 =>  # pixels to increment for next row (= LX)
```

6.6.16 MINS3 - initializes the I/O tables for the "scatter read" I/O routine MSKI3.

```
      SUBROUTINE MINS3 (LUN, FIND, LROW, NROW, ISTRT, NSKIP, BUFF,
     *   BUFSZ, BP, BO, NBUF, IERR)
   Input:
     LUN    I*2 = Logical unit number.
     FIND   I*2 = pointer to FTAB returned by ZOPEN.
     LROW   I*2 = Length of a row pixels.
     NROW   I*2 = Total number of rows.
     ISTRT  I*2 = First row for read.
     NSKIP  I*2 = Number of rows to skip.
     BUFF(1) I*2 = Output buffer.
     BUFSZ   I*2 = Buffer size in bytes.
     BP      I*2 = bytes/pixel.
     BO(2)   I*2 = Block offset, pseudo I*4.
     NBUF    I*2 = factor times which LROW ( if LROW .GE. 32768)
                   normally = 1.
   Output:
     NBUF    I*2 = number of buffer fulls to complete read of row.
                   MSKI3 must be called this number of times to
                   complete the read.
     IERR    I*2 = 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 assignments:
              0 = LUN
              1 = BP bytes/pixel
              2 = BO(1) block offset
              3 = BO(2)
              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 MSKI3 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.
```

6.6.17 MSKI3 - 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 .LE. BUFSZ/2, 2) between BUFSZ/2 and BUFSZ or 3).GT.BUFSZ. For case 3) multiple calls (NBUF from MINS3) are required to read each row. Each call returns LROW*BP/NBUF bytes and I/O is single buffered. IFIN = 0 indicates a row is completed. See MINS3 for more details.

```
      SUBROUTINE MSKI3 (LUN, FIND, BUFF, BIND, IFIN, IERR)
   Input:
     LUN     I*2 - Logical unit number.
     FIND    I*2 - pointer for FTAB
     BUFF(1)I*2 - Buffer
   Output:
     BIND    I*2 - Pointer for BUFF
     IFIN    I*2 - 0 if row complete, 1 otherwise.
     IERR    I*2 - error code: 0 - OK
                   1 - file not open
                   2 - attempt to read past end of map.
                  10+- I/O error - 10 +  ZWAIT error.

     MINS3 MUST be called before MSKI3.
```

6.6.18 PLNGET - reads a selected portion of a selected plane from a cataloged file 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. Output file is REAL*4, but the input may be either INTEGER*2 or REAL*4. 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.

```
      PLNGET (IDISK, ICNO, CORN, JWIN, XOFF, YOFF,
     *    NOSCR, NX, NY, BUFF1, IBUFF1, BUFF2, BUFSZ1, BUFSZ2,
     *    LUN1, LUN2, IRET)
   Inputs:
   IDISK          I*2  Input image disk number.
   ICNO           I*2  Input image catalog slot number.
   CORN(7)        I*2  BLC in input image (1 & 2 ignored)
```

JWIN(4)	I*2	Window in plane.
XOFF	I*2	offset in cells in first dimension of the center from MX/2+1 (MX 1st dim. of input win.)
YOFF	I*2	offset in cells in second dimension of the center from MY/2+1 (MY 2nd dim. of input win.)
NOSCR	I*2	Scratch file number in common /CFILES/ for output.
NX, NY	I*2	Dimensions of output file.
BUFF1(*)	R*4	Work buffer
IBUFF1(*)	I*2	Work buffer (should be the same as BUFF1)
BUFF2(*)	R*4	Work buffer.
BUFSZ1	I*2	Size in bytes of BUFF1/IBUFF1
BUFSZ2	I*2	Size in bytes of BUFF2
LUN1, LUN2	I*2	Log. unit numbers to use.

Output:

IRET	I*2	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.

Usage notes:
CATBLK in COMMON /MAPHDR/ is set to the input file CATBLK.

6.6.19 PLNPUT - writes a subregion of a REAL*4 scratch file image into a cataloged image (either I*2 or R*4).

```
      PLNPUT (IDISK, ICNO, CORN, JWIN, NOSCR, NX, NY,
     *   BUFF1, BUFF2, IBUFF2, BUFSZ1, BUFSZ2, LUN1, LUN2, IRET)
```

Input:

IDISK	I*2	Output image disk number.
ICNO	I*2	Output image catalog slot number.
CORN(7)	I*2	BLC in Output image (1 & 2 ignored)
JWIN(4)	I*2	Window in plane in input image.
NOSCR	I*2	Scratch file number in common /CFILES/ for input scratch file.
NX, NY	I*2	Dimensions of input file.
BUFF1(*)	R*4	Work buffer
BUFF2(*)	R*4	Work buffer.
IBUFF2(*)	I*2	Work buffer (should be the same as BUFF2)
BUFSZ1	I*2	Size in bytes of BUFF1.
BUFSZ2	I*2	Size in bytes of BUFF2/IBUFF2
LUN1, LUN2	I*2	Log. unit numbers to use.

Output:

IRET	I*2	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 and the scaling and offset parameters are set. Of particular importance is the data max/min values which must apply to the real*4 map. As this is read from the catalog, it must be updated by a call to CATIO etc. before calling this routine.

6.6.20 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*2	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*2	First line channel desired.
IFNUM	I*2	IF number wanted.

Output:

MVIS	I*2	Number of visibilities in requested output format.
JADR(2,*)	I*2	Pointers to the first and second visibility input records to be used in the output record.
SFACT(2,*)	R*4	Factors to be multiplied by the first and second input vis's to make the output vis.
ALLWT	L*2	Flag, = .TRUE. if all visibilities must have positive weight.
IERR	I*2	Error flag. 0 =>OK, otherwise data unavailable.

6.6.21 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*2	Desired output data format: 1 => I 2 => Q 3 => U 4 => V 5 => RCP 6 => LCP
NCH	I*2	First line channel desired.
Output:		
JADR(2)	I*2	Pointers to the first and second visibility input records to be used in the output record.
SFACT(2)	R*4	Factors to be multiplied by the first and second input vis's to make the output vis.
ALLWT	L*2	If true no flagged data is allowed.
JINC	I*2	Visibility increment.
IRET	I*2	Error flag. 0 =>OK, otherwise data unavailable.

6.6.22 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	R*4	Operation code, 'READ' => read only, 'WRIT' => read/write
PTYP	I*2	Physical extension type (e.g., 'CC')
VOL	I*2	Disk volume number
CNO	I*2	Catalog slot number
CATBLK(256)	I*2	Catalog block of cataloged file.
LUN	I*2	Logical unit number to use.
NREC	I*2	Number of logical rec. for create/extend
NBUF	I*2	Number I*2 words in BUFFER
In/out:		
VER	I*2	Version number: (<= 0 => write a new one, read the latest one), returns one used.
NKEY	I*2	Maximum number of keyword/value pairs input: used in create, checked on write old (0 => any); output: actual
NCOL	I*2	Number of logical columns (does not include selection column). Input: used in create,

		checked on write old (0=›any); output: actual
DATP(128,2)	I*2	DATP(*,1) address pointers (output only) DATP(*,2) column data type codes. Input: used in create only; output: actual.
BUFFER(*)	I*2	Work buffer, at least 1024 bytes in size, more if logical record longer than 512 bytes Output: control info, lookup table, ...

Output:

IERR	I*2	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*2 word(s)		Description
1 - 2	(I*4)	Number 512-byte records now in file
3 - 4	(I*4)	Max number rows allowed in current file
5 - 6	(I*4)	Number rows (logical records) now in file
7		Number of bytes/value (2 for TA files)
8		Number values / logical (# I*2s / 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		Physical file name (set on each TABINI call)
29 - 31		Creation task name (2 chars / integer)
32		Disk number
33 - 38		Last access date: ZDATE(33), ZTIME(36)
39 - 41		Last access task name (2 chars / integer)
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 - 59         "*AIPS TABLE*" packed string to verify that table.
 60              If 1 then then table cannot be written as FITS ASCII
 61              Number of selection strings now in file
 62              Next available R*4 address for a selection string
 63              First R*4 address of selection string 1
 64              First R*4 address of selection string 2
 65              First R*4 address of selection string 3
 66              First R*4 address of selection string 4
 67              First R*4 address of selection string 5
 68              First R*4 address of selection string 6
 69              First R*4 address of selection string 7
 70              First R*4 address of selection string 8
********** for TABIO / TABINI use only **********
 71              IOP : 1 => read, 2 => writ
 72              Number I*2 words per logical record
 73 - 74   (I*4) Current table row physical record in BUFFER
 75 - 76   (I*4) Current table row logical record in BUFFER
 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         Table title (4 chars / real)
129 -256         lookup table as COLPTR(logical column) = phys column
```

6.6.23 TABIO - does random access I/O to Tables extension files. Mixed reads and writes are allowed if TABINI was called 'WRIT'. Writes are limited by the size of the structure (i.e., columns for units and titles) or to the current maximum logical record plus one. Files opened for WRITe are updated and compressed on CLOS.

```
      TABIO (OPCODE, IRCODE, IRNO, RECORD, BUFFER, IERR)

   Inputs:
      OPCODE          R*4  Opcode 'READ','CLOS'
                           'WRIT' : write data as selected
                           'FLAG' : write data as de-selected
      IRCODE          I*2  0 => Table row
                           1 => DATPTR/DATYPE record
                           2 => data selection string
                           3 => title
                           4 => units
                           5 => keyword/value pair
      IRNO            I*4  Logical record number.   NOTE***** I*4 ***
                           0 => next (can work with row data and with
```

```
                              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*2       Array containing record to be written
   BUFFER( )        I*2       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*2       Array containing record read.
   BUFFER( )        I*2       buffer.
   IERR             I*2       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.
```

6.6.24 UVCREA - creates and catalogs a uv data file using the catalog header record in the common /MAPHDR/.

```
   UVCREA (IVOL, CNO, WBUFF, IERR)
In/Outs:
   IVOL       I*2    Volume # on which to put file.  0 => any
                     on output is volume used (IERR = 0)
Outputs:
   WBUFF      I*2(256) Working buffer
   CNO        I*2      Catalog slot number
   IERR       I*2      Error code; 0 => o.k.
                                   1 => couldn't create, no room
                                   2 => no create, duplicate name
                                   3 => no room in catalog
                                   4 => I/O problem on catalog
                                   5 => Other Create errors
COMMON: /MAPHDR/ catalog block used a lot, final seq # on output
```

6.6.25 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 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 transferred to the beginning of buffer 1 if that is the next buffer to be filled. 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 half 1. One exception to this is when NIO => -NIO or 0, in which case the entire remaining data in the buffer is 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 MDIS3 with opcode 'FINI'.

NOTE: A call to UVINIT is REQUIRED prior to calling UVDISK.

```
    UVDISK (OP, LUN, FIND, BUFFER, NIO, BIND, IERR)
Inputs:
 OP          R*4  Opcode 'READ','WRIT','FLSH' are legal
 LUN         I*2  Logical unit number
 FIND        I*2  FTAB pointer returned by ZOPEN
 BUFFFER()   I*2  Buffer for I/O
 NIO         I*2  For writes, the number of visibilities added to the
                  buffer; not used for reads.
Output:
 NIO         I*2  For reads, the number of visibilities ready in the
                  buffer;
                  For writes, the maximum number which can be added to
                  the buffer.  If zero for read or write then the file
                  is completely read or written.
 BIND        I*2  The pointer in the buffer to the first word of the
                  next record for reads, or the first word of the next
                  record to be copied into the buffer for writes.
 IERR        I*2  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.

6.6.26 UVGET - obtains data from a data base with optional application of flagging 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 /UVHDR/ common to reflect the output rather than input data.

UVGET (OPCODE, RPARM, VIS, IERR)

Input:

OPCODE	R*4	Opcode 4 char. 'INIT' => Open files Initialize I/O. 'READ' => Read next specified record. 'CLOS' => Close files.

Inputs via common /SELCAL/ (Includes DSEL,CSEL.INC)

UNAME(3)	R*4	AIPS name of input file.
UCLAS(2)	R*4	AIPS class of input file.
UDISK	R*4	AIPS disk of input file.
USEQ	R*4	AIPS sequence of input file.
SOURCS(4,30)	R*4	Names (16 char) of up to 30 sources, *=>all First character of name '-' => all except those specified.
TIMRNG(8)	R*4	Start day, hour, min, sec, end day, hour, min, sec. 0's => all
UVRNG(2)	R*4	Minimum and maximum baseline lengths in 1000's wavelengths. 0's => all
STOKES	R*4	Stokes types wanted. 'I','Q','U','V','R','L','IQU','IQUV' ' '=> Leave data in same form as in input.
BCHAN	I*2	First channel number selected, 1 rel. to first channel in data base. 0 => all
ECHAN	I*2	Last channel selected. 0=>all
BIF	I*2	First IF number selected, 1 rel. to first IF in data base. 0 => all
EIF	I*2	Last IF selected. 0=>all
DOCAL	L*2	If true apply calibration, else not. the CL table
DXTIME	R*4	Integration time (days). Used when applying delay corrections to correct for delay error.
ANTENS(50)	I*2	List of antennas selected, 0=>all, any negative => all except those specified
SUBARR	I*2	Subarray desired, 0=>all
FGVER	I*2	FLAG file version number, if < 0 then NO flagging is applied. 0 => use highest numbered table.
CLUSE	I*2	Cal (CL or SN) file version number to apply.

Output:

RPARM(*)	R*4	Random parameter array of datum.
VIS(3,*)	R*4	Regular portion of visibility data.
IERR	I*2	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.

UVFREQ	R*8	Frequency corresponding to u,v,w
CATBLK(256)	I*2	Catalog header block, describes the output data rather than input.
NPRMIN	I*2	Number or random parameters in the input data.
TRANSL	L*2	If true translate data to requested Stokes'
CNTREC(2,3)	I*4	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) Includes DSEL.INC and CSEL.INC should be declared in the main program or at a level that they will not be overlaid while UVGET is in use (i.e., between the 'INIT' and 'CLOS' calls)
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.
3) OPCODE = 'INIT' does the following:
   - The catalog 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.
   - 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, 28, 29, 30

     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.

6.6.27 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*BP is an integral number of blocks. Otherwise partial writes or oversize reads will have to be done. Minimum disk I/O is one block. 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.

```
      UVINIT (OP, LUN, FIND, NVIS, VISOFF, LREC, NPIO,
     *   BUFSZ, BUFFER, BO, BP, BIND, IERR)
```

Inputs:

OP	R*4	OP code, 'READ' or 'WRIT' for desired operation.
LUN	I*2	Logical unit number of file.
FIND	I*2	FTAB pointer for file returned by ZOPEN.
NVIS	I*4	Total number of visibilities to read. NVIS+VISOFF must be no greater than the total number in the file.
VISOFF	I*4	Offset in vis. rec. of first vis. rec. from BO.
LREC	I*2	Number of values in a visibility record.
NPIO	I*2	Number of visibilities per call to UVDISK.
BUFSZ	I*2	Size in bytes of the buffer. If 32767 given, 32768 is assumed.
BUFFER( )	I*2	Buffer
BO	I*4	Block offset to begin transfer from (1-relative)
BP	I*2	Bytes per value in the vis. record.

Output:

NPIO	I*2	For WRITE, the max. number of visibilities which can be accepted.
BIND	I*2	Pointer in BUFFER for WRITE operations.
IERR	I*2	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
2-3	= # vis. records left to transfer. I*4 For double buffer read, 1 more I/O will have been done than indicated.
4-5	= Block offset for next I/O. I*4
6	= byte offset of next I/O
7	= bytes per value
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.(values)
Used for WRIT only; includes any data carried over
from the last write.
15 = Buffer pointer for call (values)

6.6.28 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+IABS(ICOR-ICORO)*INCS+(IF-1)*INCIF

UVPGET (IERR)
```

Inputs: From common /MAPHDR/

CATBLK(256)	I*2	Catalog block
CAT4	R*4	same as CATBLK
CAT8	R*8	same as CATBLK

Output: In common /UVHDR/

SOURCE(2)	R*4	Packed source name.
ILOCU	I*2	Offset from beginning of vis record of U
ILOCV	I*2	" V
ILOCW	I*2	" W
ILOCT	I*2	" Time
ILOCB	I*2	" Baseline
ILOCSU	I*2	" Source id.
JLOCC	I*2	Order in data of complex values
JLOCS	I*2	Order in data of Stokes' parameters.
JLOCF	I*2	Order in data of Frequency.
JLOCR	I*2	Order in data of RA
JLOCD	I*2	Order in data of dec.
JLOCIF	I*2	Order in data of IF.
INCS	I*2	Increment in data for Stokes (see above)
INCF	I*2	Increment in data for freq. (see above)
INCIF	I*2	Increment in data for IF.
ICORO	I*2	Stokes value of first value.
NRPARM	I*2	Number of random parameters
LREC	I*2	Length in values of a vis record.
NVIS	I*4	Number of visibilities
FREQ	R*8	Frequency (Hz)
RA	R*8	Right ascension (1950) deg.
DEC	R*8	Declination (1950) deg.
NCOR	I*2	Number of correlators
ISORT	C*2	Sort order
IERR	I*2	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

6.6.29 ZCLOSE - closes file associated with LUN removing any EXCLusive use state and clears up the FTAB. NO LONGER DOES TAPES!

```
      ZCLOSE (LUN, FIND, IERR)
  Inputs:  LUN    logical unit number
           FIND   FTAB pointer from ZOPEN
  Output:  IERR   error code: 0 -> no error
                              1 -> Deaccess or Deassign error
                              2 -> file already closed in FTAB
                              3 -> both errors
                              4 -> erroneous LUN
```

6.6.30 ZCMPRS - releases unused disk space from a non-map file. Will also allow "map" files. File must be open. "Byte" defined as 1/2 of a small integer. Note: it is dangerous to compress files written by TABIO unless the bookkeeping information kept in the first record of the file is changed to reflect the new size of the file. See the description of TABINI in the section on extension file I/O in this chapter.

```
      ZCMPRS (IVOL, PNAME, LUN, LSIZE, SCRTCH, IERR)

  Inputs:  IVOL   I*2       volume number.
           PNAME  R*4(6)    physical file name
           LUN    I*2       logical unit number under which file is
                            open.
  In/Out:  LSIZE  I*4       (In) desired final size in bytes.
                            (out) actual final size in bytes.
  Outputs: SCRTCH I*2(256)  Scratch buffer
           IERR   I*2       error code: 0 => ok
                                        1 => input data error
                                        2 => compress error FMGR
```

6.6.31 ZCREAT - creates a disk file for reading/writing.

```
      ZCREAT (IVOL, PNAME, ISIZE, MAP, ASIZE, SCRTCH, IERR)
  Inputs:
     IVOL     I*2       Disk drive unit number.
                        The disk drive name
                        (volume) is encoded in the file name.
     PNAME    R*4(6)    Physical file name given by ZPHFIL.(ASCII)
                        left justified, padded with blanks.
     ISIZE    I*4       Requested size of the file in bytes. Will be
                        rounded to next higher granule.
     MAP      L*2       True if map file.
  Outputs:
     ASIZE    I*4       Actual number of bytes in the new file.
```

```
      SCRTCH  I*2(256) Scratch buffer.
      IERR    I*2      Error return code.  The values mean
                       0 - success.
                       1 - file already exists.
                       2 - volume is not available.
                       3 - space is not available.
                       4 - Other.
```

6.6.32 ZDESTR - destroys the file associated with PNAME. The file must already be closed.

```
         ZDESTR (IVOL, PNAME, IERR)
  Input:
    IVOL   I*2     Volume number of disk.
    PNAME  R*4(6) Physical file name. 24 characters max.
  Output:
    IERR   I*2      Completion code. 0-good.
                                     1-file not found
                                     2-failed
```

6.6.33 ZEXPND - expands the space allocated to a regular (non-map) file.

```
      ZEXPND (LUN, IVOL, PHNAME, NREC, IERR)
   Inputs: LUN    I*2      LUN of file (already open)
           IVOL   I*2      disk volume number of file
           PHNAME R*4(6)   physical file name of file
   In/Out: NREC   I*2      # 256-integer records requested/received
   Output: IERR   I*2      error code 0 => ok
                                      1 => input error
                                      2 => FMGR error
```

6.6.34 ZFIO - reads or writes one logical record between core and device LUN. For disk devices, the record length is always 256 integers. NREC gives the random access record number (in units of 256-integers). For non-disk devices, NREC contains the number of bytes. The I2 version has been renamed ZFI3. 15APR87: ZFIO will refuse to do tape i/o.

```
       ZFIO (OPER, LUN, FIND, NREC, BUF, IERR)
   Inputs:
      OPER   R*4  Operation = 'READ' or 'WRIT'
      LUN    I*2  logical unit number
```

```
      FIND    I*2  pointer to file area in FTAB
      NREC    I*4  record number in file: starts with 1 (DISKS)
                   number of bytes (Sequential DEVICES)
      BUF     I*2  (256) array to hold record
   Output: IERR  I*2   error code: 0 -> ok
                                   1 -> file not open
                                   2 -> input error
                                   3 -> IO error
                                   4 -> end of file
                                   5 -> begin of medium
                                   6 -> end of medium
   No longer performs IO to television devices (15mar84).
```

6.6.35 ZMIO - a low level random access, large record, double buffered device I/O. No longer performs I/O to TV or tape devices.

```
      ZMIO (OP, LUN, FIND, BLKNO, NBYTES, BUFF, IBUFF, IERR)
   Inputs:
      OP      R*4  Operation - 'READ', 'WRIT'. 4 characters.
      LUN     I*2  Logical unit number of a previously opened map.
      FIND    I*2  Pointer to FTAB returned by ZOPEN.
      BLKNO   I*4  One relative beginning block number. The size of a
                   block is given by NBPS in COMMON/DCHCOM/.
      NBYTES  I*2  Number of bytes to transfer.
      BUFF    R*4  The I/O buffer.
      IBUFF   I*2  Buffer number to be used - 1 or 2.
   Outputs:
      IERR    I*2  Error return code:
                      0 = Success.
                      1 = File not open.
                      2 = Operation incorrectly specified.
                      3 = I/O error.
                      4 = end of file (no messages)
```

6.6.36 ZOPEN - opens logical files, performing full open on disk files and sets up an FTAB entry for double buffering if required. ZOPEN does not open TV and tape devices.

```
      ZOPEN (LUN, IND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)
   Inputs:
      LUN     I*2      Logical unit number.
      IVOL    I*2      Disk volume containing file, 1,2,3,...
      PNAME   R*4(6)   24-character physical file name, left-justified,
                       packed, and padded with blanks.
      MAP     L*2      is this a map file ?
      EXCL    L*2      desire exclusive use?
      WAIT    L*2      I will wait?
   Output:
```

```
      IND    I*2       Index into FTAB for the file control block.
      IERR   I*2       Error return code:
                           0 = no error
                           1 = LUN already in use
                           2 = file not found
                           3 = volume not found
                           4 = excl requested but not available
                           5 = no room for lun
                           6 = other open errors
```

6.6.37 ZPHFIL - constructs a physical file name in PNAM from ITYPE, IVOL, NSEQ, and IVER. New version designed either for public data files or user specific files.

```
   Example: If ITYPE='MA', AIPSVER=C, NSEQ=321, IVER=99,NLUSER=762 then
               PNAME='DA07:MAC32199;1' for public data or
               PNAME='DA07:MAC32199.762;1' for private data
   ITYPE = 'MT' leads to special name for tapes
   ITYPE = 'TK' leads to special name for TEK4012 plotter CRT
   ITYPE = 'TV' leads to special name for TV device
   ITYPE = 'ME' leads to special logical for POPS memory files

      ZPHFIL (ITYPE, IVOL, NSEQ, IVER, PNAM, IERR)
   Inputs:
      ITYPE  I*2  Two characters denoting type of file. For example,
                  'MA' for map file.
      IVOL   I*2  Disk number
      NSEQ   I*2  User supplied sequence number. 000-4095.
      IVER   I*2  User supplied version number. 00-255.
   Outputs:
      PNAM(6)R*4  >= 24-byte field to receive the physical file name,
                  left justified (packed) and padded with blanks.
      IERR   I*2  Error return code.
              0 = good return.   1 = error.
```

6.6.38 ZTCLOS - closes a text file.

```
      ZTCLOS (LUN, FIND, IERR)
  Inputs:   LUN     I*2  logical unit number.
            FIND    I*2  Not used with this routine.
  Output:   IERR    I*2  Error code.
                             0 => no error.
                             1 => RMS error.
                             2 => file not open.
```

6.6.39 ZTOPEN - opens a text file.

```
      ZTOPEN (LUN, FIND, IVOL, PNAME, MNAME, VERSON, WAIT,
     *   IERR)
  Inputs:   LUN       I*2 logical unit number.
            IVOL      I*2 disk drive number
            PNAME     R*4(6) disk-file type.  Only type ('HE' etc.)
                      used. Should be generated by ZPHFIL.
            MNAME     R*4(2) file name.
            VERSON    R*4(5) Version (determines in which dir/subdir
                      to look for the file).
            WAIT      L*2 T  => wait until file is available.
  Output:   IERR      I*2 error 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
                                  6  => Other open errors.
            FIND      I*2 pointer to FTAB location.
```

6.6.40 ZTREAD - reads the next sequential card image from a text file.

```
      ZTREAD (LUN, FIND, BUF, IERR)
   Inputs:   LUN       I*2 logical unit number
             FIND      I*2 FTAB pointer for LUN
   Output:   BUF       I*2 array card image.(> -  80 chars packed)
             IERR      I*2 Error code:
                               0  => No error
                               1  => File not open.
                               2  => End of file.
                               4  => Other.
```

6.6.41 ZWAIT - waits until I/O operation is complete - not for tapes.

```
      ZWAIT (LUN, IND, IBUF, IERR)
   Inputs:
      LUN  I*2  logical unit number
      IND  I*2  Pointer to FTAB
      IBUF I*2  Wait for 1st or 2nd buffer in double buffered I/O
   Output:
      IERR I*2  Error return  0 => ok
                              1 => LUN not open
                              3 => I/O error
                              4 => end of file
```

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 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, the new 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.

7.3 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.

7.4 UV MODEL CALCULATIONS

A system of routines is available to compute the Fourier transform of a model, given as either CLEAN 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 and the array processor. 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 components or images into a uv data set.
- UVMSUB subtracts model visibilities derived from CLEAN components or images from a uv data set.

7.5 IMAGE FORMATION

A system of routines is available to form a dirty image from a uv data set. These routines make heavy use of COMMONs and the array processor. A short description is given here and the details of the subroutine calls and interface COMMONs are given at the end of this chapter.

- 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.
- UVGRID grids a uv data file.
- UVUNIF applies the uniform weighting corrections to uv data.

- GRDCOR normalizes and corrects an image for the gridding convolution used to grid the image. Used in conjunction with UVGRID and DSKFFT.

7.6 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.
- Cxxx.INC. These files contain Fortran COMMON statements.
- Exxx.INC. These contain Fortran EQUIVALENCE statements.
- Vxxx.INC. These contain Fortran DATA statements.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations, but the declaration of some variable is omitted. This type of include is used in the main program to reserve space for the omitted variable in the appropriate common. The omitted variable must be declared and dimensioned separately.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

7.6.1 PUVD.INC

```
C                                                       Include PUVD
C                                       Parameters for uv data
      INTEGER*4 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=100)
C                                       MAXFLD=max. no fields
```

```
      PARAMETER (MAXFLD=16)
C                                          MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                                          End PUVD.
```

7.6.2 CFIL.INC

```
C                                                          Include CFIL
      COMMON /CFILES/ RQUICK, NSCR, SCRVOL, SCRCNO, NCFILE, FVOL, FCNO,
     *   FRW, CCNO, IBAD, LUNS
C                                                          End CFIL
```

7.6.3 CGDS.INC

```
C                                                          Include CGDS
C                                          Local include for uv modeling
      COMMON /MAPDES/  SCRBLK, KLNBLK,
     *   CELLSG, SCLUG, SCLVG, SCLWG, SCLUM, SCLVM,
     *   DXCG, DYCG, DZCG, FLUXG, TFLUXG, XPOFF, YPOFF, SSROT, CCROT,
     *   FACGRD, OSFX, OSFY, PTFLX, PTRAOF, PTDCOF,
     *   NCLNG, NSUBG,
     *   DOFFT, NONEG, DOPTMD, NGRDAT,
     *   MFIELD, FLDSZ, CCDISK, CCCNO, CCVER, CNOBEM, BEMVOL,
     *   KSTOK, SCTYPE, VOFF, NSTOK
C                                                          End CGDS.
C                                                          Include CGDS
C                                          Local include for uv modeling
      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,
     *   NCLNG, NSUBG,
     *   DOFFT, NONEG, DOPTMD, NGRDAT,
     *   MFIELD, FLDSZ, CCDISK, CCCNO, CCVER, CNOBEM, BEMVOL,
     *   KSTOK, SCTYPE, VOFF, NSTOK, NCHANG
C                                                          End CGDS.
```

7.6.4 CMPR.INC

```
C                                                               Include CMPR
C                                       Local include for gridding
C                                       and correction routines.
      COMMON /GRDCOM/ FREQUV,
     *   XFLD, YFLD, XPARM, YPARM, TAPERU, TAPERV, ZEROSP,
     *   BMMAX, BMMIN, FLDMAX, FLDMIN,
     *   BEMMAX, XSHIFT, YSHIFT, BLMAX, BLMIN,
     *   MNAME, MCLASS,
     *   DOZERO, DOTAPE, DOUNIF,
     *   NXBEM, NYBEM, NXUNF, NYUNF, NXMAX, NYMAX, ICNTRX, ICNTRY,
     *   CTYPX, CTYPY, NUVCH, CHUV1, NCHAVG, UNFBOX,
     *   TVFLD, BORES, BOBEM, MDISK, MSEQ
C                                                               End CMPR
```

7.6.5 CSEL.INC

```
C                                                               Include CSEL
C                                       Common for UVGET use
C                                       Data selection and control
      COMMON /SELCAL/ UVFREQ,
     *   UNAME, UCLAS, USEQ, UDISK, UFILE,
     *   SOURCS, CALSOU, TIMRNG, UVRNG, STOKES, INTFN, INTPRM,
     *   UVRA, TSTART, TEND, UBUFF,
     *   SELFAC,
     *   INXRNO, NINDEX, FSTVIS, LSTVIS,
     *   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL,
     *   CLBUFF, FGBUFF, NXBUFF,
     *   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN, IXLUN,
     *   CATUV, ANTENS, NANTSL,
     *   NSOUWD, SOUWAN, SOUWTN, NCALWD, CALWAN, CALWTN,
     *   SUBARR, SMOTYP, CURSOU, NXKOLS, NXNUMV, MVIS, JADR, PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU
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, BLFAC,
     *   ICLRNO, NCLINR, MAXCLR, CNTREC,
     *   DOCAL, DOAPPL,
     *   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CLKOLS, CLNUMV,
     *   LCLTAB, LCUCAL, ICALP1, ICALP2, POLOFF
      COMMON /MAPHDR/ CATBLK
C                                                               End CSEL
```

7.6.6 CUVH.INC

```
C                                                               Include CUVH
      COMMON /UVHDR/ FREQ, RA, DEC, SOURCE, NVIS, ILOCU, ILOCV,
     *   ILOCW, ILOCT, ILOCB, ILOCSU, JLOCC, JLOCS, JLOCF, JLOCR,
     *   JLOCD, JLOCIF, INCS, INCF, INCIF, ICORO, NRPARM, LREC, NCOR,
     *   ISORT
C                                                               End CUVH
```

7.6.7 DFIL.INC

```
C                                                               Include DFIL
      INTEGER*2 NSCR, SCRVOL(20), SCRCNO(20), IBAD(10), LUNS(10),
     *   NCFILE, FVOL(50), FCNO(50), FRW(50), CCNO
      LOGICAL*2 RQUICK
C                                                               End DFIL
```

7.6.8 DGDS.INC

```
C                                                               Include DGDS
C                                                 Local include for uv modeling
      INTEGER*2 SCRBLK(256), KLNBLK(256), MFIELD, FLDSZ(2,MAXFLD),
     *   CCDISK(MAXFLD), CCCNO(MAXFLD), CCVER(MAXFLD), CNOBEM,
     *   BEMVOL, KSTOK, SCTYPE, VOFF, NSTOK, NCHANG
      LOGICAL*2 DOFFT, NONEG, DOPTMD, NGRDAT
      INTEGER*4 NSUBG(MAXFLD), NCLNG(MAXFLD)
      REAL*4    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
      REAL*8    FREQG(MAXCHA)
C                                                               End DGDS
```

7.6.9 DMPR.INC

```
C                                                               Include DMPR
C                                          Local include for gridding
C                                          and correction routines.
C                                          NOTE uses PARAMETER in DGDS.INC
      INTEGER*2 NXBEM, NYBEM, NXUNF, NYUNF, NXMAX, NYMAX,
     *   ICNTRX(MAXFLD), ICNTRY(MAXFLD), CTYPX, CTYPY, NUVCH, CHUV1,
     *   NCHAVG, UNFBOX, TVFLD, BORES(2,MAXFLD), BOBEM(2), MDISK, MSEQ
      LOGICAL*2 DOZERO, DOTAPE, DOUNIF
      REAL*4    XFLD(MAXFLD), YFLD(MAXFLD), XPARM(10), YPARM(10),
     *   TAPERU, TAPERV, ZEROSP(5), BMMAX, BMMIN,
     *   FLDMAX(MAXFLD), FLDMIN(MAXFLD), BEMMAX, MCLASS(2), MNAME(3),
     *   XSHIFT(MAXFLD), YSHIFT(MAXFLD), BLMAX, BLMIN
      REAL*8    FREQUV
C                                                               End DMPR
```

7.6.10 DSEL.INC

```
C                                                               Include DSEL
C                                          Commons for UVGET use
      INTEGER*4 XCTBSZ
C                                          XCTBSZ=internal gain table size
      PARAMETER (XCTBSZ=2500)
C                                          Data selection and control
      INTEGER*2 ANTENS(50), NANTSL, NSOUWD, SOUWAN(30), SOUWTN(30),
     *   NCALWD, CALWAN(30), CALWTN(30), SUBARR, SMOTYP, CURSOU,
     *   NXKOLS(6), NXNUMV(6), MVIS, JADR(2,1024), PMODE, LRECIN,
     *   UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU
      LOGICAL*2 DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL
      INTEGER*4 INXRNO, NINDEX, FSTVIS, LSTVIS
      REAL*4    SOURCS(4,30), CALSOU(4,30), TIMRNG(8),  UVRNG(2),
     *   STOKES, INTFN, INTPRM(3), UVRA(2), TSTART, TEND,
     *   SELFAC(2,1024)
      REAL*8    UVFREQ
C                                          Flag table info
      REAL*4    TMFLST, FLGTND(MAXFLG)
      INTEGER*4 IFGRNO
      LOGICAL*2 DOFLAG, FLGPOL(4,MAXFLG)
      INTEGER*2 FGVER, NUMFLG, FGKOLS(8), FGNUMV(8),
     *   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*4    GMMOD, CURCAL(XCTBSZ), LCALTM, CALTAB(XCTBSZ,2),
     *   CALTIM(3), RATFAC(MAXIF), DELFAC(MAXIF), DXTIME, DXFREQ,
     *   BLFAC(2,MXBASE,MAXIF)
      INTEGER*4 ICLRNO, NCLINR, MAXCLR, CNTREC(2,3)
      LOGICAL*2 DOCAL, DOAPPL
```

```
      INTEGER*2 CLVER, CLUSE, NUMANT, NUMPOL, NUMIF,
     *   CLKOLS(39), CLNUMV(39), LCLTAB, LCUCAL, ICALP1, ICALP2,
     *   POLOFF(4,2)
C                                          File specification.
      INTEGER*2 IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN,
     *   IXLUN, CATUV(256), CATBLK(256)
      REAL*4    UNAME(3), UCLAS(2), USEQ, UDISK, UFILE(6)
C                                          I/O buffers
      INTEGER*2 CLBUFF(1024), FGBUFF(512), NXBUFF(512)
      REAL*4    UBUFF(8192)
C                                                        End DSEL
```

7.6.11 DUVH.INC

```
C                                                        Include DUVH
      INTEGER*4 NVIS
      INTEGER*2 ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, JLOCC,
     *   JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC, NCOR, ISORT,
     *   INCS, INCF, INCIF, ICORO
      REAL*4 SOURCE(2)
      REAL*8 FREQ, RA, DEC
C                                                        End DUVH
```

7.7 ROUTINES

7.7.1 CALCOP - copys selected data from one data file to another optionally applying calibration and editing information. The input file should have been opened with UVGET. An output, scratch file will be created if requested; 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)

    Input:
      DISK          I*2  Disk number for cataloged output file.
                         If .LE. 0 then the output file is a /CFILES/
                         scratch file.
      CNOSCR        I*2  Catalog slot number for if cataloged file;
                         /CFILES/ scratch file number if a scratch file,
                         IF DISK-CNOSCR-0 then the scratch is created.
      BUFFER(*)     R*4  Work buffer for writing.
      BUFSZ         I*2  Size of BUFFER in bytes.
    Input via common:
      CATBLK(256)   I*2  Catalog header block from UVGET
      NVIS          I*2  (/UVHDR/) Number of vis. records.
      LREC          I*2  (/UVHDR/) length of vis. record in R*4 words.
      NRPARM        I*2  (/UVHDR/) number of (R*4) random parameters.
    Output:
      CNOSCR        I*2  Scratch file number if created.
      IRET          I*2  Error code: 0 => OK,
                                    >0 => failed, abort process.
    Output via common:
      CATBLK(256)   I*2  Catalog header block with actual no. records.
      NVIS          I*2  (/UVHDR/) Actual number of vis. records.
   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
```

7.7.2 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.

NOTE: this routine uses the array processor.

```
 DSKFFT (NR, NC, IDIR, HERM, LI, LW, LO,
*   JBUFSZ, BUFF1, BUFF2, SMAX, SMIN, IERR)
```

Inputs:

NR	I*2	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*2	The number of columns in input array (# rows in output).
IDIR	I*2	1 for forward (+i) transform, -1 for inverse (-i) transform. If HERM = .TRUE. the following are recognized: IDIR=1 keep real part only. IDIR=2 keep amplitudes only. IDIR=3 keep full complex (half plane)
HERM	L*2	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*2	File number in /CFILES/ of input.
LW	I*2	File number in /CFILES/ of work file (may equal LI).
LO	I*2	File number in /CFILES/ of output.
JBUFSZ	I*2	Size of BUFF1, BUFF2 in bytes. Should be large at least 4096 R*4 words.

Output:

BUFF1	R*4	Working buffer
BUFF2	R*4	Working buffer
SMAX	R*4	For HERM=.TRUE. the maximum value in the output file.
SMIN	R*4	For HERM=.TRUE. the minimum value in the output file.
IERR	I*2	Return error code, 0=›OK, otherwise error.

7.7.3 GRDCOR - normalizes and corrects an image file for the gridding convolution function used in gridding uv data to make the image. Used in conjunction with UVGRID.
Uses AIPS LUNs 18 and 19

NOTE: This routine uses the Array Processor

```
   GRDCOR (IFIELD, DOGCOR, DISKI, CNOSCI, DISKO, CNOSCO,
  *    MAPMAX, MAPMIN, JBUFSZ, BUFF1, BUFF2, BUFF3, IRET)
```

Input:

IFIELD	I*2	The subfield number, if = 1 the histogram is zero filled first. If IFIELD = 0 the input is assumed to be a beam.
DOGCOR	L*2	If TRUE, do gridding convolution correction.
DISKI	I*2	Input file disk number for cataloged files, .LE. 0 => /CFILES/ scratch file.
CNOSCI	I*2	Input file catalog slot number or /CFILES/ scratch file number.
DISKO	I*2	Output file disk number for cataloged files, .LE. 0 => /CFILES/ scratch file.
CNOSCO	I*2	Output file catalog slot number or /CFILES/ scratch file number.
JBUFSZ	I*2	Size in bytes of buffers. Dimension of BUFF1,2,3 must be at least 4096 R*4.

From commons: (Includes DGDS, DMPR, DUVH, CGDS, CMPR, CUVH)

BEMMAX	R*4	Sum of the weights used in gridding, used to normalize images.
CTYPX,CTYPY	I*2	Convolving function types for RA and Dec
XPARM(10)	R*4	Convolving function parameters for RA XPARM(1) = support half width.
YPARM(10)	R*4	Convolving function parameters for Dec.
BORES(2,16)	PI*4	Block offset desired in output file for an image, 1 per field. (1 rel.)
BOBEM(2)	P I*4	Block offset desired in output file for an beam. (1 rel.)
NGRDAT	L*2	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*2	Dimension of map in RA, Dec (cells)
ICNTRX,ICNTRY(*)	I*2	The center pixel in X and Y for each field.

Output:

MAPMAX	R*4	The maximum value in the resultant image.
MAPMIN	R*4	The minimum value in the resultant image.
BUFF1	R*4	Working buffer
BUFF2	R*4	Working buffer
BUFF3	R*4	Working buffer
IRET	I*2	Return error code. 0=>OK, error otherwise.

7.7.4 MAKMAP - makes a 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. The weights of the data may (optionally) have the uniform weighting correction made.

The visibilities are convolved onto a 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 in Q1GRD. The bandwidth synthesis (BS) process may use the SCRWRK file. For bandwidth synthesis both the CNOSCO and SCRWRK files should be big enought 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.
Input uv data in UV file CNOSCI, DISKI;
Output image in image file CNOSCO, DISKO
Uses buffer UBUFF from the UVGET commons (include C/DSEL.INC)

NOTE: This routine uses the Array Processor

```
 MAKMAP (IFIELD, DISKI, CNOSCI, DISKO, CNOSCO,
*    SCRGRD, SCRWRK, CHANN,
*    DOCREA, DOINIT, DOBEAM, DOSEL, DOGCOR,
*    JBUFSZ, BUFFER, IRET)
```

Inputs:

IFIELD	I*2	Field number to map, if 0 then make a beam.
DISKI	I*2	Input file disk number for cataloged files, .LE. 0 => /CFILES/ scratch file.
CNOSCI	I*2	Input file catalog slot number or /CFILES/ scratch file number.
DISKO	I*2	Output file disk number for cataloged files, .LE. 0 => /CFILES/ scratch file.
CNOSCO	I*2	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*2	Grid scratch file number, will be set if the file is created, (DOINIT=TRUE)
SCRWRK	I*2	Work scratch file number, will be set if the file is created, (DOINIT=TRUE)
CHANN	I*2	Channel number to grid. If DOSEL=TRUE then this is 1-rel wrt the selected data.
DOCREA	L*2	If TRUE, Create/catalog output image file.
DOINIT	L*2	If TRUE, initialize scratch files, set defaults for convolving functions. Should be TRUE on first call, and FALSE there after.

DOBEAM	L*2	If TRUE a grid the beam before gridding the field. See usage notes.
DOSEL	L*2	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*2	If TRUE, correct image for gridding convolution correction function. (Normally .TRUE.)
JBUFSZ	I*2	Size in bytes of buffers. Dimension of BUFFER must be at least 4096 R*4.

From commons: (Includes DGDS, DMPR, CGDS, CMPR)

MFIELD	I*2	The number of fields which are going to to be imaged (excluding any beam). MUST be filled in.
FLDSZ(2,*)	I*2	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*2	If TRUE, apply Uniform weighting. Should be TRUE on only the first call, otherwise it will be applied again.
NCHAVG	I*2	Number of channels to grid together for bandwidth synthesis.
UNFBOX	I*2	Half width of unif. wt. counting box size.
CTYPX,CTYPY	I*2	Convolving function types for RA and Dec
XPARM(10)	R*4	Convolving function parameters for RA XPARM(1) = support half width.
YPARM(10)	R*4	Convolving function parameters for Dec.
UVRNG(2)	R*4	Minimum and maximum baseline lengths in 1000's wavelengths. 0's => all
XSHIFT(16)	R*4	Shift in X (after rotation) in asec. in projected coordinates. 1 per field.
YSHIFT(16)	R*4	Shift in Y (after rotation) in asec. in projected coordinates. 1 per field.
STOKES	R*4	Stokes types wanted. 'I','Q','U','V','R','L'
DOZERO	L*2	If true then do zero spacing flux.
ZEROSP(5)	R*4	Zero spacing flux, 1=>flux density (Jy) 5 => weight to use. polarization.
TFLUXG	R*4	The total flux density removed from the data, this will be subtracted from the zero spacing flux before gridding.
DOTAPE	L*2	True if taper requested.
TAPERU,TAPERV	R*4	TAPER (to 30%) in u and v (kilolamda)
NXUNF,NYUNF	I*2	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(common /MAPHDR/):

CATBLK(256)	I*2	Catalog header for uv data input file. (only used on DOINIT=TRUE call)

The following must be provided if DOCREA is TRUE (includes D/CMPR,

```
 D/CGDS)
   MNAME(3)          R*4  Output image name (12 char. packed)
   MCLASS(2)         R*4  Output image class (6 char. packed)
                          (If more than 1 field the last 2 char
                          are used to encode the field number)
   MDISK             I*2  Desired image file output disk
   MSEQ              I*2  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*2  Dimension of map in RA, Dec (cells)
   NXBEM,NYBEM       I*2  Dimension (cells) of beam.
   CELLSG(2)         R*4  The cell spacing in X and Y in arcseconds.
   XSHIFT(16)        R*4  Shift in X (after rotation) in asec.
                          in projected coordinates. 1 per field.
   YSHIFT(16)        R*4  Shift in Y (after rotation) in asec.
                          in projected coordinates. 1 per field.
   ICNTRX,ICNTRY(*) I*2 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 D/CGDS).
   CCDISK(16)   I*2  Disk numbers of the output images.
                     (Must be zeroed if not filled in.)
   CCCNO(16)    I*2  Catalog slot numbers of output images.
                     (Must be zeroed if not filled in.)
The following must be provided if DOSEL is true.
(Includes D/CSEL.INC)
   UNAME(3)     R*4  AIPS name of input file.
   UCLAS(3)     R*4  AIPS class of input file.
   UDISK        R*4  AIPS disk of input file.
   USEQ         R*4  AIPS sequence of input file.
   FGVER        I*2  FLAG file version number, if < 0 then
                     NO flagging is applied.
   SOURCS(2,1)  R*4  Names (8 char) of desired source.
   TIMRNG(8)    R*4  Start day, hour, min, sec, end day, hour,
                     min, sec. 0's => all
   STOKES       R*4  Stokes types wanted.
                     'I','Q','U','V','R','L'
   BCHAN        I*2  First channel number selected, 1 rel. to first
                     channel in data base. 0 => all
   ECHAN        I*2  Last channel selected. 0=>all
   BIF          I*2  First IF number selected, 1 rel. to first
                     IF in data base. 0 => all
   EIF          I*2  Last IF selected. 0=>all
   DOCAL        L*2  If true apply calibration, else not.
The following must be provided if DOCAL is TRUE.
   ANTENS(50)   I*2  List of antennas selected, 0=>all,
                     any negative => all except those specified
   GAUSE        I*2  GAIN (CL or SN) file version number to use.
Output:
   DISKI       I*2   UV data file disk if data reformatted.
   CNOSCI      I*2   Reformatted uv data scratch file number
                     to be used in subsequent calls.
   DISKO       I*2   Output image file disk number if output file.
                     created and/or cataloged (DOCREA=TRUE
```

		or input DISKO=0 and CNOSCO=0).
CNOSCO	I*2	Output image file catalog slot number or scratch file number if output file created.
SCRGRD	I*2	Grid scratch file number, will be set if the file is created, (DOINIT=TRUE)
SCRWRK	I*2	Work scratch file number, will be set if the file is created, (DOINIT=TRUE)
DOSEL	L*2	Set to FALSE if data reformatted.
DOBEAM	L*2	Set to FALSE.
DOINIT	L*2	Set to FALSE.
BUFFER(*)	R*4	Working buffer
IRET	I*2	Return error code. 0=›OK, error otherwise.

Output in Common:

DOUNIF	L*2	Set to FALSE if uniform weighting applied.
UBUFSZ	I*2	Buffer size for UBUFF (UVGET buffer)
MNAME(3)	R*4	Output image name (12 char. packed) (defaults applied)
MCLASS(2)	R*4	Output image class (6 char. packed) (defaults applied)
MDISK	I*2	Desired image file output disk (defaults applied)
MSEQ	I*2	Desired image file output sequence no. (defaults applied)
FLDMAX(*)	R*4	Maximum pixel value in field.
FLDMIN(*)	R*4	Minimum pixel value in field.

The following are filled in if a output file is created:

CCDISK(16)	I*2	Disk numbers of the output images.
CCCNO(16)	I*2	Catalog slot numbers of output images.

Usage Notes:

1) The input uvdata file is, with one exception, assumed to be accurately described by the contents of CAT4 and the common /UVHDR/ (includes DUVH, CUVH). 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 (includes D/CSEL). 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. As many channels as possible

will be gridded in the same pass as allowed by the "AP" memory available. Several passes may be required under some circumstances.

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, CAT4 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
CHDR.INC, CUVH.INC, CFIL.INC, CMPR.INC, CGDS.INC, CSEL.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.

7.7.5 UVGET - obtains data from a data base with optional application of flagging 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 /UVHDR/ common to reflect the output rather than input data.

```
      UVGET (OPCODE, RPARM, VIS, IERR)

   Input:
      OPCODE     R*4  Opcode 4 char.
                      'INIT' => Open files Initialize I/O.
                      'READ' => Read next specified record.
                      'CLOS' => Close files.
   Inputs via common /SELCAL/  (Includes DSEL,CSEL.INC)
      UNAME(3)      R*4  AIPS name of input file.
      UCLAS(2)      R*4  AIPS class of input file.
      UDISK         R*4  AIPS disk of input file.
      USEQ          R*4  AIPS sequence of input file.
      SOURCS(4,30) R*4  Names (16 char) of up to 30 sources, *=>all
```

		First character of name '-' => all except those specified.
TIMRNG(8)	R*4	Start day, hour, min, sec, end day, hour, min,sec. 0's => all
UVRNG(2)	R*4	Minimum and maximum baseline lengths in 1000's wavelengths. 0's => all
STOKES	R*4	Stokes types wanted. 'I','Q','U','V','R','L','IQU','IQUV' ' '=> Leave data in same form as in input.
BCHAN	I*2	First channel number selected, 1 rel. to first channel in data base. 0 => all
ECHAN	I*2	Last channel selected. 0=>all
BIF	I*2	First IF number selected, 1 rel. to first IF in data base. 0 => all
EIF	I*2	Last IF selected. 0=>all
DOCAL	L*2	If true apply calibration, else not. the CL table
DXTIME	R*4	Integration time (days). Used when applying delay corrections to correct for delay error.
ANTENS(50)	I*2	List of antennas selected, 0=>all, any negative => all except those specified
SUBARR	I*2	Subarray desired, 0=>all
FGVER	I*2	FLAG file version number, if < 0 then NO flagging is applied. 0 => use highest numbered table.
CLUSE	I*2	Cal (CL or SN) file version number to apply.

Output:

RPARM(*)	R*4	Random parameter array of datum.
VIS(3,*)	R*4	Regular portion of visibility data.
IERR	I*2	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.

UVFREQ	R*8	Frequency corresponding to u,v,w
CATBLK(256)	I*2	Catalog header block, describes the output data rather than input.
NPRMIN	I*2	Number or random parameters in the input data.
TRANSL	L*2	If true translate data to requested Stokes'
CNTREC(2,3)	I*4	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) Includes DSEL.INC and CSEL.INC should be declared in the main program or at a level that they will not be overlaid while UVGET is in use (i.e., between the 'INIT' and 'CLOS' calls)
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.
3) OPCODE = 'INIT' does the following:
   - The catalog 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.
- 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, 28, 29, 30
  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.

7.7.6 UVGRID - convolves uv data onto a grid using AP routines. The visibilities are convolved onto the grid using the convolving function specified by CTYPX,CTYPY,XPARM,YPARM. The defaults for these values must be 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 in AP1GRD. If more than one channels are to be gridded together, UVGRID loops over the frequency channels in an outer loop, reading the grid and uv data several times and writing the grid several times. This bandwidth synthesis (BS) process will use the SCRWRK file. For bandwidth synthesis both the SCROUT and SCRWRK files should be big enough for an extra m rows, where m is the half width of the X convolving function.

This version uses a scrolling buffer as large as possible using both the primary and secondary AP memory; 1/4 of the memory is used as the uv data buffer when possible. Bandwidth synthesis is done using as many channels as possible. The number of channels is determined by the width of the scrolling buffer, the channel separations and the highest row number on which data occurs.
Zero spacing flux densities are gridded if provided.
Uses AIPS LUNs 18, 20, 21
Input uv data file in UV file CNOSCI.
Output grid file in image file SCROUT although SCROUT and SCRWRK may be switched if bandwidth synthesis is used.

NOTE: This routine uses the Array Processor

```
      UVGRID (IFIELD, SCRWRK, DISKI, CNOSCI, SCROUT,
     *    UVH4, JBUFSZ, BUFF1, BUFF2, BUFF3, IRET)
```

Inputs:

IFIELD	I*2	Field number to grid, if 0 then grid a beam.
SCRWRK	I*2	/CFILES/ file number for work file, (used for bandwidth synthesis)
DISKI	I*2	Input file disk number for cataloged files, .LE. 0 => /CFILES/ scratch file.
CNOSCI	I*2	Input file catalog slot number or /CFILES/ scratch file number.
DISKO	I*2	Output file disk number for cataloged files, .LE. 0 => /CFILES/ scratch file.
SCROUT	I*2	/CFILES/ scratch file number of output file
UVH4(128)	R*4	UV data catalog header record.
JBUFSZ	I*2	Size in bytes of buffers. Dimension of BUFF1,2,3 must be at least 4096 R*4.

From commons: (Includes DGDS, DMPR, DUVH, CGDS, CMPR, CUVH)

NVIS	I*4	Number of visibility records (/UVHDR/)
LREC	I*2	Number of (real) words per visibility record (/UVHDR/)
NCHAVG	I*2	Number of frequency channels to grid together.
FLDSZ(2,*)	I*2	Dimension of map in RA, Dec (cells)
CELLSG(2)	R*4	The cell spacing in X and Y in arcseconds.
CHUV1	I*2	First channel number in file to grid (1 relative)
FREQ	R*8	Reference frequency (Hz) (/UVHDR/)
JLOCF	I*2	0 relative number of the frequency axis, (/UVHDR/)
TFLUXG	R*4	The total flux density removed from the data, this will be subtracted from the zero spacing flux before gridding.
CTYPX,CTYPY	I*2	Convolving function types for RA and Dec
XPARM(10)	R*4	Convolving function parameters for RA XPARM(1) = support half width.
YPARM(10)	R*4	Convolving function parameters for Dec.
BLMAX	R*4	Maximum baseline length allowed in 1000s of wavelengths.
BLMIN	R*4	Minimum baseline length allowed in 1000s of wavelengths.
DOZERO	L*2	If true then do zero spacing flux.
ZEROSP(5)	R*4	Zero spacing flux, 1=>flux density (Jy) 5 => weight to use. polarization.
DOTAPE	L*2	True if taper requested.
TAPERU,TAPERV	R*4	TAPER (to 30%) in u and v (kilolamda)
NXBEM,NYBEM	I*2	The size of the BEAM in pixels.
FREQG(*)	R*8	Frequencies of the channels
FREQUV	R*8	Reference frequency for u,v, and w.
NGRDAT	L*2	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.

XFLD,YFLD(*)	R*4	Field of view in RA and Dec (arcseconds)
DXCG,DYCG,DZCG	R*4	2*pi*(delta ra, delta dec, and delta z)

		to be used in AP1GRD to shift positions. (u,v and w are in cells). one per field.
SCLUG,SCLVG,SCLWG	R*4	Conversion factors for u,v and w from wavelengths at the reference frequency to cells. one set per field.
ICNTRX,ICNTRY(*)	I*2	The center pixel in X and Y for each field.

The following must be provided if NGRDAT is .FALSE.

CCDISK(16)	I*2	Disk numbers of the output images.
CCCNO(16)	I*2	Catalog slot numbers of output images.

Output:

SCRWRK	I*2	/CFILES/ file number for work file, (used for bandwidth synthesis)
SCROUT	I*2	/CFILES/ scratch file number of output file
BUFF1	R*4	Working buffer
BUFF2	R*4	Working buffer
BUFF3	R*4	Working buffer (buffers should be contiguous in memory)
IRET	I*2	Return error code. 0=›OK, error otherwise.

Output via common:

BEMMAX	R*4	Sum of weights = normalization factor

Usage Notes:

1) The input uvdata file is, with one exception, assumed to be accurately described by the contents of UVH4 and the common /UVHDR/ (includes DUVH, CUVH). The exception is that the frequencies of the channels are given by the common array FREQG. The u,v, and w are assumes to be given by the common variable FREQUV.

2) The contents of common /UVHDR/ (=includes DUVH, CUVH) are filled in by UVPGET from the catalog header; UVPGET should be called before calling UVGRID.

3) If NGRDAT is .FALSE. then the properties (e.g., shift) of the desired output image are assumed to be described in the catalog header of the existent file pointed to by CCDISK,CCCNO(IFIELD).

4) Only one polarization will be processed and the input data is assumed to be in the desired Stokes' type (i.e., I, Q, U, V etc.) In the general case this will require reformatting the data. This can be accomplished via CALCOP to do the whole file or UVGET or SET1VS & GET1VS which work a record at a time. Multiple channels may be gridded together a la bandwidth synthesis by specifying NCHAVG › 1. One channel of several channels may be gridded using CHUV1 › 1.

5) 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, UVH4 should tell which.

6) The necessary image normalization constant for proper normalization of the FFTed image is produced only by a call with IFIELD=0 to grid the sampling function. Therefore, UVGRID must be called to grid the sampling function IRREGARDLESS of whether or not a beam will be produced.

7) The gridding convolution function parameters must be completely specified. The defaults should be filled in by a call to GRDFLT before calling UVGRID.
8) Multiple IFs can be processed using the common frequency table FREQG

7.7.7 UVMDIV - divides model visibilities derived from CLEAN 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/ (includes DGDS.INC and CDGS.INC) contains most of the critical information about the CLEAN components files or images to be used. 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 regardless of the value of METHOD.

NOTE: This routine uses the Array Processor

```
 UVMDIV (DISKI, CNOSCI, DISKO, CNOSCO, MODEL, METHOD,
*   DOMSG, CHANNEL, NCHAN, CATBLK, JBUFSZ, BUFF1, BUFF2, BUFF3,
*   IRET)
```

Inputs:

DISKI	I*2	Input disk number. if .LE. 0 then input is a scratch file.
CNOSCI	I*2	Input file catalog slot number or /CFILES/ scratch file number.
DISKO	I*2	Output disk number. if .LE. 0 then output is a scratch file.
CNOSCO	I*2	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*2	1=› clean components, 2=›image.
METHOD	I*2	1=›gridded, -1=›DFT, 0=›chose.
DOMSG	L*2	If true give percent done messages for DFT.
CHANNEL	I*2	First uv data channel to subtract.
NCHAN	I*2	Number of frequency channels to subtract.
CATBLK(256)	I*2	Uv data catalog header record.
JBUFSZ	I*2	Size of BUFF1,2,3 in bytes, must be at least 4096 real words.
BUFF1,2,3	R*4	Work buffers.

Inputs from COMMON /MAPDES/:

MFIELD	I*2	Number of fields
NSUBG(*)	I*4	Number of components already sub.
NCLNG(*)	I*4	Number of components per field.
CCDISK(*)	I*2	Disk numbers for CC files
CCCNO(*)	I*2	Catalog slot numbers for CC files.
CCVER(*)	I*2	CC file version number for each field.
FACGRD	R*4	Value to multiply clean component fluxes by before subtraction (negative for sum).
NONEG	L*2	Stop reading comps. from a file past the first negative component. (DFT modeling ONLY)
DOPTMD	L*2	Use the point model specified by PTFLX, PTRAOF, PTDCOF (DFT modeling ONLY)
PTFLX	R*4	Point model flux density (Jy) (I pol. only)
PTRAOF	R*4	Point model RA offset from uv phase center (asec)
PTDCOF	R*4	Point model Dec. offset from uv phase center

Input from COMMON /UVHDR/:

LREC	I*2	Length of visibility record.
NVIS	I*4	Number of visibility records.
NRPARM	I*2	"Random" parameters before data, can be used to skip observed values when computing model.

Output:

CNOSCO	I*2	Output file catalog slot number or /CFILES/ scratch file number. Value returned if not specified in call.
IRET	I*2	Return error code. 0=›OK, otherwise failed.

7.7.8 UVMSUB - subtracts a clean model or an image from a set of uv data. Extensive use is made of commons to communicate with UVMSUB, in particular /MAPDES/ (includes DGDS.INC and CDGS.INC) contains most of the critical information about the CLEAN 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 regardless of the value of METHOD.

NOTE: This routine uses the Array Processor

```
UVMSUB (DISKI, CNOSCI, DISKO, CNOSCO, MODEL, METHOD,
*   CHANNEL, NCHAN, DOSUM, DOMSG, CATBLK, JBUFSZ, BUFF1, BUFF2,
*   BUFF3, IRET)
```

Inputs:

DISKI	I*2	Input disk number. if .LE. 0 then input is a scratch file.
CNOSCI	I*2	Input file catalog slot number or /CFILES/ scratch file number.
DISKO	I*2	Output disk number. if .LE. 0 then output is a scratch file.

```
   CNOSCO      I*2 Output file catalog slot number or /CFILES/
                   scratch file number.
   MODEL       I*2 1=› clean components, 2=›image.
   METHOD      I*2 1=›gridded, -1=›DFT, 0=›chose.
   CHANNEL     I*2 First uv data channel to subtract.
   NCHAN       I*2 Number of frequency channels to subtract.
   DOSUM       L*2 If true then sum component fluxes in FLUXG,
                   TFLUXG.
   DOMSG       L*2  If true give percent done messages for DFT.
   CATBLK(256)I*2 Uv data catalog header record.
   JBUFSZ      I*2 Size of BUFF1,2,3 in bytes, must be at least 4096
                   real words.
Inputs from COMMON /MAPDES/:
   MFIELD       I*2   Number of fields
   NSUBG(*)     I*4   Number of components already sub.
   NCLNG(*)     I*4   Number of components per field.
   CCDISK(*)    I*2   Disk numbers for CC files
   CCCNO(*)     I*2   Catalog slot numbers for CC files.
   CCVER(*)     I*2   CC file version number for each field.
   FACGRD       R*4   Value to multiply clean component fluxes
                      by before subtraction (negative for sum).
   NONEG        L*2  Stop reading comps. from a file past the first
                     negative component. (DFT modeling ONLY)
   DOPTMD       L*2  Use the point model specified by PTFLX, PTRAOF,
                     PTDCOF (DFT modeling ONLY)
   PTFLX        R*4  Point model flux density (Jy) (I pol. only)
   PTRAOF       R*4  Point model RA offset from uv phase center
                     (asec)
   PTDCOF       R*4  Point model Dec. offset from uv phase center
Input from COMMON /UVHDR/:
   LREC         I*2   Length of visibility record.
   NVIS         I*4   Number of visibility records.
   NRPARM       I*2   "Random" parameters before data, can be used
                      to skip observed values when computing model.
   BUFF1,2,3  R*4 Work buffers.
Output:
   IRET        I*2 Return error code. 0=›OK, otherwise failed.
```

7.7.9 UVUNIF - computes uniform weighting corrections and applies them to the weights in the visibility data base. The visibility weights are divided by the number of visibilities occurring in cells within a box of half width UNFBOX centered on the cell in which a given visibility resides. Does the uniform weighting correction for the uv cellsize defined by CELLSG, NXUNF, NYUNF and UNFBOX
Input uv data file in uv file DISKI, CNOSCI.
Output uv data file in uv file DISKO, CNOSCO.
Uses AIPS LUNs 18, 20, 21 (all files closed on successful return)

NOTE: This routine uses the Array Processor

```
UVUNIF (DISKI, CNOSCI, DISKO, CNOSCO, SCRWRK, CAT4,
```

```
     *    JBUFSZ, BUFF1, BUFF2, IBUFF3, IRET)

   Inputs:
     SCRWRK        I*2   /CFILES/ file number for work file,
     DISKI         I*2   Input file disk number for cataloged files,
                         if .LE. 0 => scratch file.
     CNOSCI        I*2   Output file catalog slot number or /CFILES/
                         scratch file number.
     DISKO         I*2   Output file disk number for cataloged files,
                         if .LE. 0 => scratch file.
     CNOSCO        I*2   Output file catalog slot number or /CFILES/
                         scratch file number.
     CAT4(128)     R*4   UV data catalog header record.
     JBUFSZ        I*2   Size in bytes of buffers. Dimension of
                         BUFF1,2,IBUFF3  must be at least 4096 R*4.
   From commons: (Includes DGDS, DMPR, DUVH, CGDS, CMPR, CUVH)
     UNFBOX        I*2   Half width of unif. wt. counting box size.
     NVIS          I*4   Number of visibility measurements. (/UVHDR/)
     LREC          I*2   Number of (real) words per visibility record
                         (/UVHDR/)
     NCHAVG        I*2   Number of continuum channels to grid
                         together. (Used to determine number of weights
                         per visibility to correct.)
     CHUV1         I*2   First channel number in file to correct weight
                         (1 relative) (first ch. to be gridded)
     FREQUV        R*8   Reference frequency of the u, v, w
     NGRDAT        L*2   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.
     CELLSG(2)     R*4   The cell spacing in X and Y in arcseconds.
     NXUNF,NYUNF   I*2   Dimensions ( cells) of the map in RA and Dec
                         to be used to determine uniform wt. counting box
   The following must be provided if NGRDAT is .FALSE.
     CCDISK(16)    I*2   Disk numbers of the output images.
     CCCNO(16)     I*2   Catalog slot numbers of output images.

    Output:
     BUFF1         R*4   Working buffer
     BUFF2         R*4   Working buffer
     IBUFF3        I*2   Working buffer
     IRET          I*2   Return error code, 0=>OK, error otherwise.
   Usage Notes:
    1) The input uvdata file is, with one exception, assumed to be
     accurately described by the contents of CAT4 and the common
     /UVHDR/ (includes DUVH, CUVH).  The exception is that the u, v and
     w may refer to a different frequency.  The common input variable
     FREQUV gives the reference frequency for the u, v, and w.
    2) The contents of common /UVHDR/ (-includes DUVH, CUVH)
     are filled in by UVPGET from the catalog header; UVPGET should
     be called before calling UVGRID.
    3) If NGRDAT is .FALSE. then the properties (e.g., cellsize) of the
     desired output image are assumed to be described in the catalog
```

header of the existent file pointed to by CCDISK,CCCNO(IFIELD).
4) The random parameters in the data should include, in order, u, v, w, weight (optional), time (optional) and baseline (optional). The weights are required but may be passed either as random parameters or as part of the regular data array, CAT4 should tell which.
5) The uniform correction made is to divide the weight of each visibility by the number of occurrences in its counting box regardless of the weights of the visibilities.

CHAPTER 8

WAWA ("EASY") I/O

8.1 OVERVIEW

We have created a fairly 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 core space 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 INCLUDEs DFIL.INC and CFIL.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 whose name reflects the number of simultaneous map type files the programmer wants open.

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 so that fewer format conversions are needed.

4. Many subroutine calls are combined so that e.g., ZPHFIL, CATDIR, CATIO, and MINIT, more or less disappear from sight.

5. Scratch files are cataloged along with regular maps, which makes destroying them easier, either within the task or externally.

6. A general clean-up subroutine for closing files and destroying scratch files is provided.

7. "Hidden" buffers large enough to hold a 2048-point Real*4 map 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 real array of length 9, e.g.,

```
REAL*4  NAMS (9)
where NAMS(1:3) contain the file NAME as 12 packed characters
      NAMS(4:5) contain the file CLASS as 6 packed characters
      NAMS(6)   contains SEQ as a real number
      NAMS(7)   contains the disk volume as a real number
      NAMS(8)   contains the file physical type as A2
      NAMS(9)   contains the file USID number as a real number
```

The formats match those provided by GTPARM. If you specify an [INPUTS] file with INNAME, INCLASS, INSEQ, INDISK, INTYPE and USERID the parameters will be inserted into your input array such that it forms a valid namestring.

Some null values are allowed that cause defaults to be invoked.

1. A leading double blank in NAMS(1) means "any NAME".

2. A leading double blank in NAMS(4) means "any CLASS".

3. A 0.0 in NAMS(6) means "any SEQ".

4. A 0.0 in NAMS(7) means "any DISK".

5. A leading double blank in NAMS(8) means a physical type of "MA".

6. A 0.0 in NAMS(9) means USID of NLUSER i.e., the task user. A 32,000.0 in NAMS(9) means "any USID"

A value of "SC" for the leading characters in NAMS(8) means "scratch". In this case, all the package subroutines substitute internally (some do not alter the calling namestring) a NAME, CLASS, and USID unique to the main task and AIPS initiator (i.e., interactive AIPS 1, 2 or BATCH AIPS 6, 7, ...) : NAME = 'SCRATCH FILE', CLASS = TSKNAM|NPOPS , USID = NLUSER

8.4 SUBROUTINES

The following is a list of the WaWa package of routines with a short description of each. Detailed descriptions of the function and call sequence of these routines can be found at the end of this chapter.

1. IOSETn - Setup I/O for n simultaneous map files.

2. FILOPN - Open a file, particularly associated files.

3. OPENCF - Open a cataloged file.

4. FILIO - Do I/O to a non-map file.

5. MAPWIN - Set a multi-dimensional window on an open map.

6. MAPXY - Set a 2-dim window on top plane of a map.

7. MAPIO - Read or write to a map.

8. FILCLS - Close a map or non-map file.

9. FILCR - Create a non-map file.

10. MAPCR - Create a map file.

11. FILDES - Destroy either a map or non-map file.

12. UNSCR - Destroy all scratch files.

13. CLENUP - Call UNSCR and close any still open files.

14. MAPMAX - Find MAX & MIN of a map and enter into catalog.

15. FILNUM - Find WaWa pointers to open file (used for history).

16. GETHDR - Retrieve catalog header for an open cataloged file.

17. SAVHDR - Save header in catalog for an open cataloged file.

18. HDRINF - Retrieve specified items from map header.

19. TSKBEn - Combination of IOSETn and some task startup chores.

20. 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 standard 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.

8.6.2.1 Information Common - The primary common in the WaWa package is obtained by the includes DITB.INC and CITB.INC. The text of these 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	R*4	'WRIT'	I/O control strings
REED	R*4	'READ'	
CLWR	R*4	'CLWR'	Catalog control strings
CLRD	R*4	'CLRD'	
REST	R*4	'REST'	
OPEN	R*4	'OPEN'	
CLOS	R*4	'CLOS'	
SRCH	R*4	'SRCH'	
INFO	R*4	'INFO'	
UPDT	R*4	'UPDT'	
FINI	R*4	'FINI'	I/O control string
CSTA	R*4	'CSTA'	Catalog control string
INDEF	R*4	'INDE'	Blanked floating point pixel
SUBNAM(3,8)	I*2		Subroutine names: CATDIR, CATIO, MINIT, MDISK, ZCLOSE, ZCREAT, ZDESTR, ZOPEN in the form of 2 char/word for error messages
LINT	I*2		Number integer values in one IO buffer
LREAL	I*2		Number real values in one IO buffer
NFIL	I*2		Number simultaneous open map files
EFIL	I*2		Size of FILTAB (5 + NFIL) - number of simultaneous files of all types
QUACK	I*2		0 => restart AIPS at end, 1 => already done
POLUN	I*2		FILTAB pointer for LUN value (1)
POFIN	I*2		FILTAB pointer for I/O table pointer value (2)
POVOL	I*2		FILTAB pointer for disk number value (3)
POCAT	I*2		FILTAB pointer for cat location value (4)
POIOP	I*2		FILTAB pointer for opcode number (5): values 1 => write, 2=> read, <0 => new win
POASS	I*2		FILTAB pointer for is it associated file (6): 1 => assoc, 0 => main file
POBPX	I*2		FILTAB pointer for bytes/pixel code (7)
PODIM	I*2		FILTAB pointer for # axes (8)
PONAX	I*2		FILTAB pointer for # points on each of 7 axes (9)
POBLC	I*2		FILTAB pointer for Bottom left corner (16)
POTRC	I*2		FILTAB pointer for Top right corner (23)
PODEP	I*2		FILTAB pointer for current depth in I/O on axes 2 - 7 (30), Area (36) used for integer map (input) blanking code.
POBL	I*2		FILTAB pointer for block offset start I/O in the current plane (37)
FILTAB(38,EFIL)	I*2		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!!)

8.6.2.2 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 CAT2, CAT4, and CAT8. 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 includes DCAT.INC, CCAT.INC, and ECAT.INC. The other common, called /WAWABU/, contains:

RMAX(10)	R*4	1-5 used by MAPIO for scale factor
RMIN(10)	R*4	1-5 used by MAPIO for offset
WBUFF(256)	I*2	scratch buffer for catalog access
RBUF(n*2048)	R*4	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.

8.6.2.3 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:

```
Declarations:
        INCLUDE 'INCS:ZFT5.INC'     File table space
        INCLUDE 'INCS:IBUn.INC'     WaWa buffer/table sizes
        INCLUDE 'INCS:IITB.INC'     WaWa I/O common
        INCLUDE 'INCS:IDCH.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
Commons:
        INCLUDE 'INCS:CBUF.INC'
        INCLUDE 'INCS:CITB.INC'
        INCLUDE 'INCS:CDCH.INC'
        INCLUDE 'INCS:CHDR.INC'
        INCLUDE 'INCS:CMSG.INC'
        INCLUDE 'INCS:CCAT.INC'
        INCLUDE 'INCS:CFIL.INC'
Equivalences:
        INCLUDE 'INCS:EBUF.INC'
        INCLUDE 'INCS:ECAT.INC'
```

8.6.3 Error Return Codes

A uniform system of error code numbers has been adopted in the WaWa I/O package. These code 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.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statements.
- Exxx.INC. These contain Fortran EQUIVALENCE statements.
- Vxxx.INC. These contain Fortran DATA statements.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations, but the declaration of some variable is omitted. This type of include is used in the main program to reserve space for the omitted variable in the appropriate common. The omitted variable must be declared and dimensioned separately.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

8.7.1 IBU1.INC

```
C                                                          Include IBU1
      REAL*4    RMAX(10), RMIN(10), RBUF(2048)
      INTEGER*2 WBUFF(256), IBUF(1)
      INTEGER*2 FILTAB(38,6)
C                                                          End IBU1
```

8.7.2 IBU2.INC

```
C                                                          Include IBU2
      REAL*4    RMAX(10), RMIN(10), RBUF(4096)
      INTEGER*2 WBUFF(256), IBUF(1)
      INTEGER*2 FILTAB(38,7)
C                                                          End IBU2
```

8.7.3 IBU3.INC

```
C                                                                  Include IBU3
      REAL*4    RMAX(10), RMIN(10), RBUF(6144)
      INTEGER*2 WBUFF(256), IBUF(1)
      INTEGER*2 FILTAB(38,8)
C                                                                  End IBU3
```

8.7.4 IBU4.INC

```
C                                                                  Include IBU4
      REAL*4    RMAX(10), RMIN(10), RBUF(8192)
      INTEGER*2 WBUFF(256), IBUF(1)
      INTEGER*2 FILTAB(38,9)
C                                                                  End IBU4
```

8.7.5 IBU5.INC

```
C                                                                  Include IBU5
      REAL*4    RMAX(10), RMIN(10), RBUF(10240)
      INTEGER*2 WBUFF(256), IBUF(1)
      INTEGER*2 FILTAB(38,10)
C                                                                  End IBU5
```

8.7.6 IITB.INC

```
C                                                                  Include IITB
      REAL*4  WRIT, REED, CLWR, CLRD, REST, OPEN, CLOS, SRCH,
     *   INFO, UPDT, FINI, CSTA, INDEF
      INTEGER*2  SUBNAM(3,8), LINT, LREAL, NFIL, EFIL, QUACK,
     *   POLUN, POFIN, POVOL, POCAT, POIOP, POASS, POBPX,
     *   PODIM, PONAX, POBLC, POTRC, PODEP, POBL
C                                                                  End IITB.
```

8.7.7 DCAT.INC

```
C                                                                  Include DCAT
      INTEGER*2 CAT2(256)
      REAL*4    CAT4(128)
      REAL*8    CAT8(64)
C                                                                  End DCAT.
```

8.7.8 DFIL.INC

```
C                                                                  Include DFIL
      INTEGER*2 NSCR, SCRVOL(20), SCRCNO(20), IBAD(10), LUNS(10),
     *   NCFILE, FVOL(50), FCNO(50), FRW(50), CCNO
      LOGICAL*2 RQUICK
C                                                                  End DFIL
```

8.7.9 CBUF.INC

```
C                                                                  Include CBUF
      COMMON /WAWABU/ RMAX, RMIN, WBUFF, RBUF
C                                                                  End CBUF.
```

8.7.10 CFIL.INC

```
C                                                                  Include CFIL
      COMMON /CFILES/ RQUICK, NSCR, SCRVOL, SCRCNO, NCFILE, FVOL, FCNO,
     *   FRW, CCNO, IBAD, LUNS
C                                                                  End CFIL
```

8.7.11 CITB.INC

```
C                                                                  Include CITB
      COMMON /WAWAIO/ WRIT, REED, CLWR, CLRD, REST, OPEN, CLOS,
     *   SRCH, INFO, UPDT, FINI, CSTA, INDEF, SUBNAM,
     *   LINT, LREAL, NFIL, EFIL, QUACK,
     *   POLUN, POFIN, POVOL, POCAT, POIOP, POASS, POBPX,
     *   PODIM, PONAX, POBLC, POTRC, PODEP, POBL, FILTAB
```

```
C                                                                    End CITB.
```

8.7.12 CCAT.INC

```
C
      COMMON /MAPHDR/ CAT2                                           Include CCAT
C                                                                    End CCAT.
```

8.7.13 EBUF.INC

```
C
      EQUIVALENCE (RBUF(1), IBUF(1))                                 Include EBUF
C                                                                    End EBUF.
```

8.7.14 ECAT.INC

```
C
      EQUIVALENCE (CAT2(1), CAT4(1), CAT8(1))                        Include ECAT
C                                                                    End ECAT.
```

8.7.15 ZFT5.INC

```
C
      INTEGER*2 FTAB(310)                                            Include ZFT5
C                                                                    End ZFT5.
```

8.8 DETAILED DESCRIPTIONS OF THE SUBROUTINES.

8.8.1 CLENUP - Close all files opened with FILOPN. Destroy scratch files.

```
CLENUP
   no arguments
```

8.8.2 FILCLS - Close a file, cleaning up any I/O pending to it and taking care of catalog bookkeeping.

```
FILCLS (LUN)
Inputs:
   LUN       I*2      Logical unit number
```

8.8.3 FILCR - Create a non-map of "file" type file associated with the cataloged file NAMS, and modify catalog block accordingly.

```
FILCR (NAMS, TYPE, NBLOCK, VER, ERROR)
Inputs:
   NAMS(9)   R*4      Specifies catalog slot
   TYPE      R*4      Extension file type (2 characters)
In/out:
   NBLOCK    I*2      Number of 512-byte blocks requested
Output:
   VER       R*4      Version of newly created file
```

8.8.4 FILDES - Destroy a cataloged or extension file and modify catalog appropriately.

```
FILDES (NAMS, EXT, TYPE, VER, ERROR)
Inputs:
   NAMS(9)   R*4      Specifies catalog entry
   EXT       L*2      Is file an extension file?
   TYPE      R*4      IF(EXT) what is extension type? (2 char)
   VER       R*4      IF(EXT) what is extension version?
```

8.8.5 FILIO - Transfer a specified 512 byte record between an open file associated with LUN, and the array DATA.

```
FILIO (OP, LUN, NREC, DATA, ERROR)
Inputs:
   OP         R*4      "READ" or "WRIT"
   LUN        I*2      Logical unit number
   NREC       I*2      record number
Inputs/Output:
   DATA(256)  I*2      data area
```

8.8.6 FILNUM - Find the FILTAB entry for an open file.

```
FILNUM (LUN, IFIL, ERROR)
Inputs:
   LUN      I*2      Logical unit number of file
Outputs:
   IFIL     I*2      Entry number (2nd subscript to FILTAB)
```

8.8.7 FILOPN - Find a cataloged or extension file in catalog, open file and associate it with the LUN.

```
FILOPN (LUN, NAMS, EXT, TYPE, VER, ERROR)
Inputs:
   LUN        I*2      logical unit number
   EXT        L*2      Desired file is an extension of a cataloged
                       file? Not for type 'SC..'
   TYPE       C*2      if EXT is true, EXT file TYPE
   VER        R*4      if EXT is true, EXT file version number
                       (VER = 0.0 => latest version)
In/out:
   NAMS(9)    R*4      namestring specifying catalog
```

8.8.8 GETHDR - Fetch the header block of a cataloged, open file.

```
GETHDR (LUN, HDR, ERROR)
Inputs:
   LUN        I*2      Logical Unit. No. of an open map
Outputs:
   HDR(256)   I*2      Map header of that map
```

8.8.9 HDRINF - Fetch a NUMBER of consecutive entries from the map header of an open map.

```
HDRINF (LUN, TYPE, START, NUMBER, DATA, ERROR)
Inputs:
   LUN      I*2     Logical Unit. No. of an open map
   TYPE     I*2     type of header information wanted
                    1=› I*2; 2=› R*4; 3=› R*8 6=› Ch*8
   START    I*2     Index of 1st item requested, in the system
                    specified by TYPE
   NUMBER   I*2     Number of items requested
Outputs:
   DATA(*)  ***     Receiving array; type specified by TYPE
```

8.8.10 IOSET1, IOSET2, IOSET3, IOSET4, And IOSET5 - These routines initialize the I/O tables; call ZDCHIN; allocate buffer space for map I/O to n files adequate for 2048 real or 1024 complex pixels per line, where n is the last character of the name.

```
IOSETn

no calling arguments
```

8.8.11 MAPCR - Create and catalog a map-type file. Only R*4 and complex*8 maps will be created.

```
MAPCR (ONAMS, NAMS, HDR, ERROR)
Input:
   ONAMS(9) R*4     Namestring of a related file to be used to
                    complete the defaults in NAMS. This is
                    typically the input file namestring. The
                    actual values must be filled in before call.
Input/output:
   NAMS(9)  R*4     Namestring NAME:CLASS:SEQ:VOL:TYPE:USID
                    of map to be created and may contain blanks
                    (null values) or wild-cards.
   HDR(256) I*2     Catalog block specifying enough
                    information to determine file size:
                    specifically # of axes and # of pixels
                    on each axis.
```

8.8.12 MAPIO - Transfer one line of data between core area DATA and a disk map-type file. On READ, data are scaled using the header scaling and offset factors. Integer "blanked" values are replaced with the R*4 value numerically equivalent to the string "INDE".

When you start writing, MAX and MIN in the header will be marked as "INDE" or indefinite. You can switch from "READ" to "WRIT" at any time.

```
MAPIO (OP, LUN, DATA, ERROR)
Inputs:
   OP        R*4      "READ"  "WRIT"
   LUN       I*2      Logical unit number
In or out:
   DATA(*)   R*4      data area
```

8.8.13 MAPMAX - Determine the maximum and minimum values of an R*4 map and enter values into map header.

```
MAPMAX (LUN, MAX, MIN, ERROR)
Inputs:
   LUN       I*2      Logical Unit No. of an open map
Outputs:
   MAX       R*4      Map maximum value
   MIN       R*4      Map minimum value
```

8.8.14 MAPWIN - Select a subarray of the (up to) 7-dimensional map array hypercube so that MAPIO (cf. above) only reads a subset of the hypercube. If MAPWIN is not called, the entire map will be delivered, line by line, by MAPIO. If it is, the lines in the subarray will be delivered line by line.

When WRITing, you cannot window in the x-direction (fastest varying coordinate) because of disk addressing problems, but you can window in the other dimensions.

MAPWIN can be called any number of times after opening a file, even if a previous window has not been completely transferred.

```
MAPWIN (LUN, BLC, TRC, ERROR)
Inputs:
   LUN       I*2      Logical unit number
   BLC(7)    R*4      Bottom left corner of subarray
   TRC(7)    R*4      Top Right Corner of subarray
```

8.8.15 MAPXY - Does the same as MAPWIN, but assumes you only want to talk to part, or all, of the top 2-dimensional plane of a possibly multi-dimensional map. If WIN(1) = 0.0, you get the entire top plane.

```
MAPXY (LUN, WIN, ERROR)
Inputs:
   LUN      I*2     Logical unit number
   WIN(4)   R*4     A 2-dimensional window
```

8.8.16 OPENCF - Same as FILOPN, but restricted to cataloged files (i.e., no associated files) to simplify call sequence.

```
OPENCF (LUN, NAMS, ERROR)
Inputs:
   LUN      I*2     Logical unit number
In/out:
   NAMS(9)  R*4     File namestring
```

8.8.17 SAVHDR - Save the catalog header block for a file that is already open (via FILOPN or OPENCF).

```
SAVHDR (LUN, CAT, ERROR)
Inputs:
   LUN        I*2    Logical unit number of file
   CAT(256)   I*2    Header block to be saved
```

8.8.18 TSKBE1, TSKBE2, TSKBE3, TSKBE4, And TSKBE5 - For n = 1,...5 this subroutine does several task startup chores:

1. Calls IOSETn to initialize I/O
2. Calls GTPARM to get parameters
3. If DOWAIT is false, calls RELPOP

```
TSKBEn (PRGNAM, NPARM, RPARM, ERROR)
Inputs:
   PRGNAM(3) I*2    Name of task we are starting up
   NPARM     I*2    Number of R*4 parameters we expect initiator
                    to pass
Outputs:
   RPARM(*)  R*4    Array to receive passed parameters
```

8.8.19 TSKEND - Combines some task ending chores:

1. Calls CLENUP to destroy scratch files & close other files
2. If DOWAIT was true, calls RELPOP with return code IRET

TSKEND (IRET)
Inputs:

IRET	I*2	return code back to initiator if DOWAIT true 0 => ok, > 0 => troubles

8.8.20 UNSCR - Destroy all scratch files created by this task.

UNSCR
no arguments

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 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 true 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. We should allow the previous directory structure to be easily reproduced, so that no changes are necessary on other working systems.

6. Preserve non-standard areas so that we can keep track of programs which are, or use, non-standard code.

7. 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.

8. Try to make the structure as logical and consistent as possible.

9. Use the minimum number of directories consistent with the above. There will, however, need to be a substantial increase in the number of directories.

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 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 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.

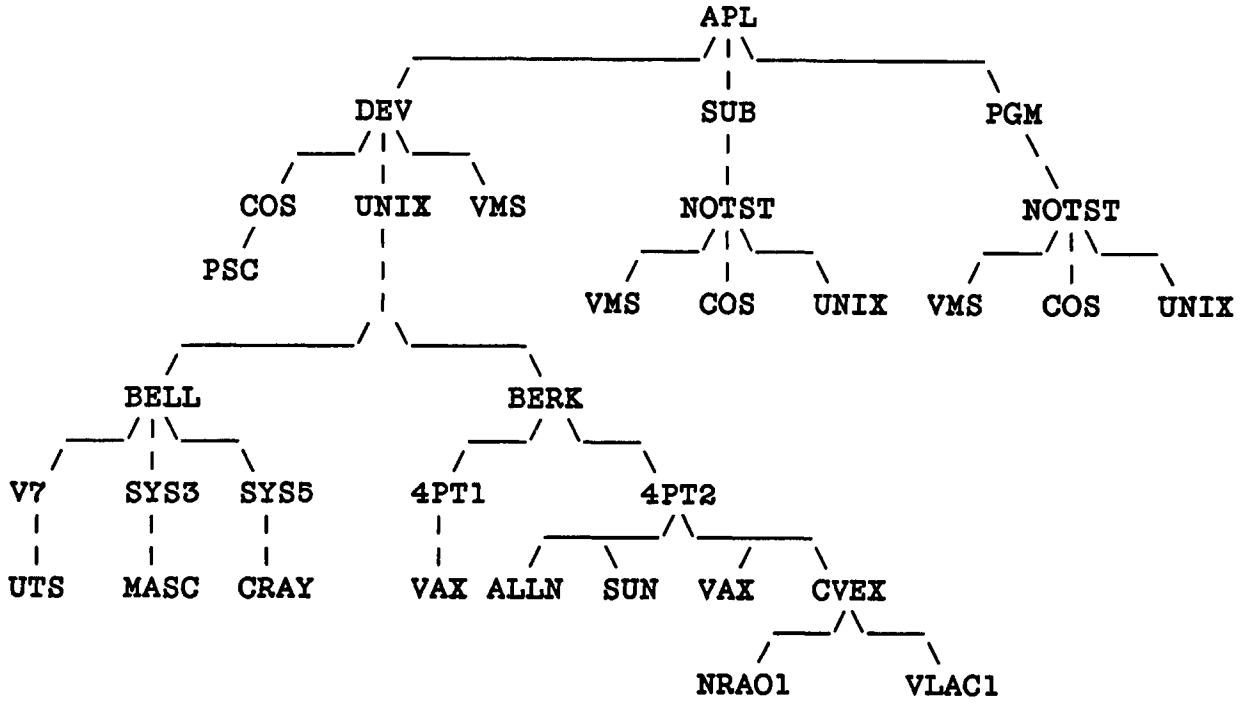
2. Utility routines - library subroutines that may call device-specific routines, but are themselves device independent. A non-standard area is also provided.

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.

A.2.2.1 APL

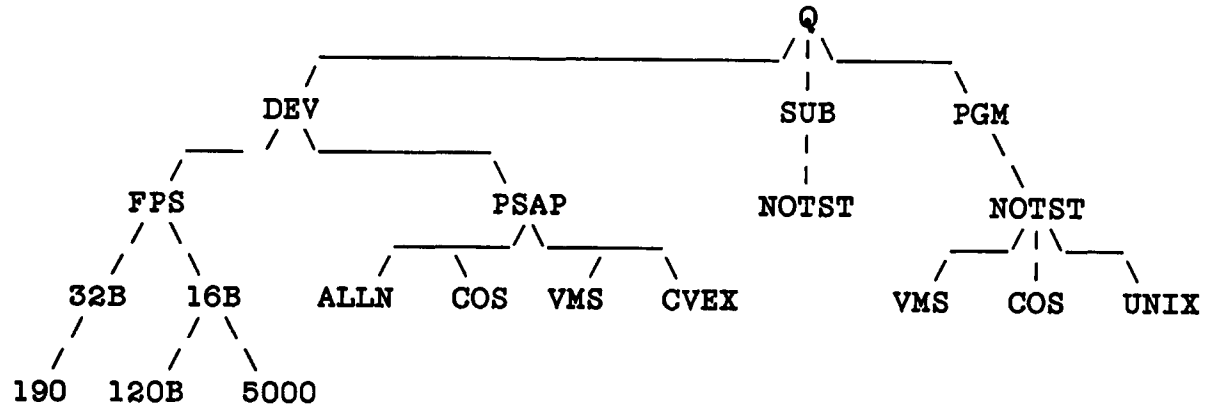
This area is for utility routines and programs that make no reference to an AP or TV device.



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 belong 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. The system-specific areas should not be considered an invitation to flood AIPS with machine-dependent code. Apart from the system-specific areas containing Z-routines, the UNIX and COS areas are essentially empty. The same cannot be said of the VMS areas. This is an unfortunate consequence of a VMS-dominated adolescence.

A.2.2.2 Q

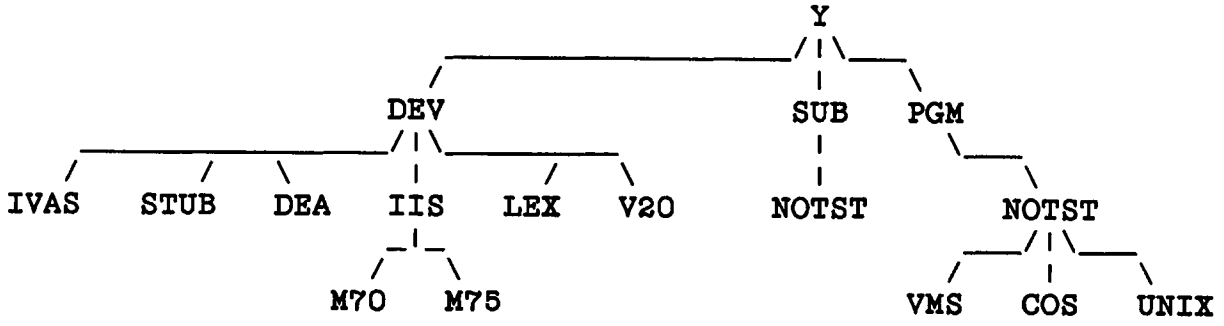
This area is for routines and programs that make use of the AP.



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 code. The lower level branches support a variety of different AP devices, 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 AP.

A.2.2.3 Y

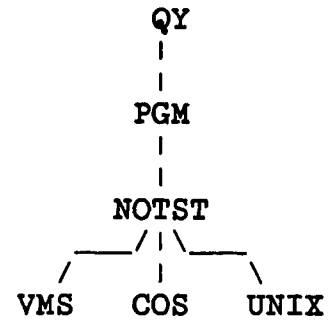
This area is for routines and programs that make use of the TV.



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.

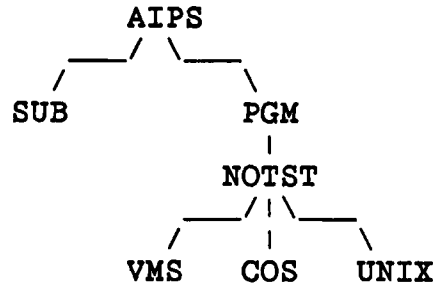
A.2.2.4 QY

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.



A.2.2.5 AIPS

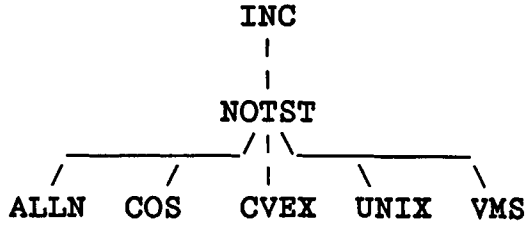
This area is for the POPS-level programs and related routines. Several of these make use of the TV device, but, as they are routines not accessible to tasks, they reside here. The stand-alone service programs are also stored in this area.



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

A.2.2.6 Include

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



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.

A.2.2.7 Help

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

A.2.2.8 Load

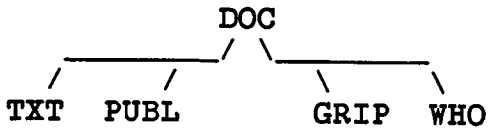
This area is for load modules, i.e., fully linked programs in a form ready to be run. This is split into a standard LOAD area and a few alternative areas immediately below (e.g., LOAD.ALT1). These alternate areas could, for example, be used to keep pseudo-AP versions of programs or versions linked for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

A.2.2.9 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.

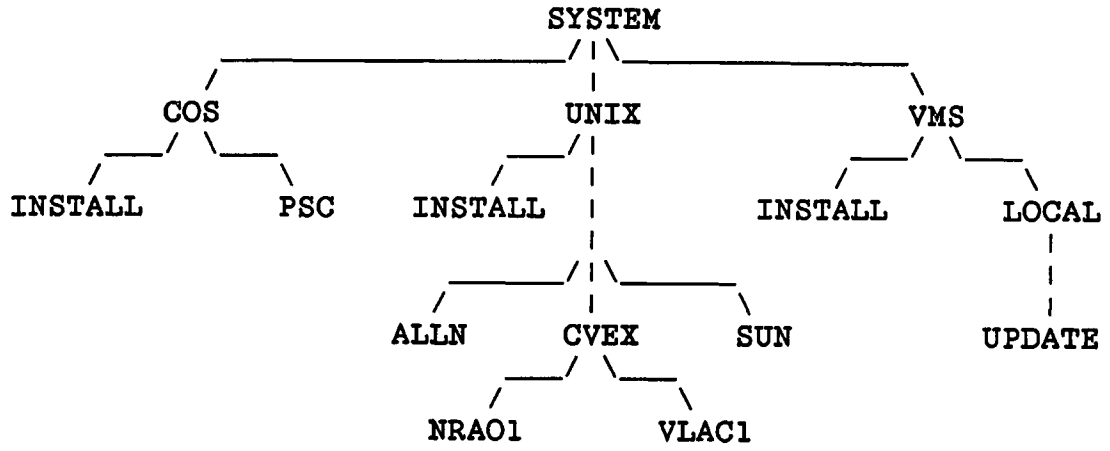
A.2.2.10 Documentation

The DOC area is used to store documentation files (this manual, other coding descriptions) in DOCTXT, the Cookbook and AIPSletters in DOCPUBL, received and answered gripes in DOCGRIP, and mailing lists in DOCWHO. The directory structure is simple:



A.2.2.11 System

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



The COS, 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

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 in its UNIX incarnation. (The VMS and UNIX versions of this file, called AREAS.DAT, differ, but the only substantive difference is in the definition of SYSLOCAL.) It can be used to assign the appropriate mnemonics, or to create a complete directory tree.

```
! "-------------------------------------------------------------------"
! " AREAS.DAT                                                         "
! "-------------------------------------------------------------------"
! " This file defines almost the entire AIPS directory structure in   "
! " terms of environment variables for use in UNIX/AIPS programming.  "
! " This may saturate the local environment space and you may find it "
! " necessary to comment out some or all of the ones not applicable to "
! " or unused in the host implementation of UNIX/AIPS (e.g., *MC4,    "
! " *COS, *VMS, DOCGRIP, DOCPUBL, DOCWHO, specific ...Y/DEV/... and   "
! " .../Q/DEV/...directories, etc.), especially if these directories  "
! " don't exist or are empty.                                         "
! "                                                                   "
! " This file should be largely system independent if the following   "
```

```
! " special characters can be dealt with:                                "
! "                                                                       "
! "      ! comment character                                              "
! "      . directory delimiter                                            "
! "                                                                       "
! " Generic UNIX version.                                                 "
! "-------------------------------------------------------------------"
! " 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
!                                       "Cray COS non-standard programs"
AIPGCOS         AIPS.PGM.NOTST.COS
!                                       "Modcomp non-standard programs"
AIPGMC4         AIPS.PGM.NOTST.MC4
!                                       "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
!                                       "Cray COS programs"
APGCOS          APL.PGM.NOTST.COS
!                                       "Modcomp programs"
APGMC4          APL.PGM.NOTST.MC4
!                                       "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
!                                       "Cray COS non-standard routines"
APLNCOS         APL.SUB.NOTST.COS
!                                       "Modcomp non-standard routines"
APLNMC4         APL.SUB.NOTST.MC4
!                                       "UNIX non-standard routines"
APLNUNIX        APL.SUB.NOTST.UNIX
!                                       "VMS non-standard routines"
APLNVMS         APL.SUB.NOTST.VMS
!
! "-------------------------------------------------------------------"
! " Z-routine areas                                                   "
! "-------------------------------------------------------------------"
!
!                                       "Generic"
APLGEN          APL.DEV
!                                       "Cray COS"
APLCOS          APL.DEV.COS
!                                       "Pittsburgh Supercomputer Center"
APLPSC          APL.DEV.COS.PSC
!                                       "Modcomp"
APLMC4          APL.DEV.MC4
!                                       "Generic UNIX"
APLUNIX         APL.DEV.UNIX
!                                       "Bell UNIX"
APLBELL         APL.DEV.UNIX.BELL
!                                       "Bell System III UNIX"
APLSYS3         APL.DEV.UNIX.BELL.SYS3
!                                       "Masscomp System III UNIX"
APLMASC         APL.DEV.UNIX.BELL.SYS3.MASC
!                                       "Bell System V UNIX"
APLSYS5         APL.DEV.UNIX.BELL.SYS5
!                                       "Bell Version 7 UNIX"
APLV7           APL.DEV.UNIX.BELL.V7
!                                       "Amdahl's Version 7 UNIX (UTS)"
APLUTS          APL.DEV.UNIX.BELL.V7.UTS
!                                       "Berkeley UNIX"
APLBERK         APL.DEV.UNIX.BERK
!                                       "Berkeley 4.1 UNIX"
APL4PT1         APL.DEV.UNIX.BERK.4PT1
!                                       "VAX Berkeley 4.1 UNIX"
APL1VAX         APL.DEV.UNIX.BERK.4PT1.VAX
!                                       "Berkeley 4.2 UNIX"
APL4PT2         APL.DEV.UNIX.BERK.4PT2
!                                       "Alliant Berkeley 4.2 UNIX"
```

```
APLALLN          APL.DEV.UNIX.BERK.4PT2.ALLN
!                                          "Convex Berkeley 4.2 UNIX"
APLCVEX          APL.DEV.UNIX.BERK.4PT2.CVEX
!                                          "NRAO-VLA Convex local"
APLVLAC1         APL.DEV.UNIX.BERK.4PT2.VLAC1
!                                          "NRAO-CV Convex local"
APLNRAO1         APL.DEV.UNIX.BERK.4PT2.NRAO1
!                                          "Sun Berkeley 4.2 UNIX"
APLSUN           APL.DEV.UNIX.BERK.4PT2.SUN
!                                          "VAX Berkeley 4.2 UNIX"
APL2VAX          APL.DEV.UNIX.BERK.4PT2.VAX
!                                          "Generic VMS"
APLVMS           APL.DEV.VMS
!
! "-------------------------------------------------------------------"
! " Documentation areas                                               "
! "-------------------------------------------------------------------"
!
DOC              DOC
!                                          "Gripes"
DOCGRIP          DOC.GRIP
!                                          "Public"
DOCPUBL          DOC.PUBL
!                                          "Text"
DOCTXT           DOC.TEXT
!                                          "AIPS user lists"
DOCWHO           DOC.WHO
!
! "-------------------------------------------------------------------"
! " 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
```

```
!                                                    "Alliant INCLUDEs"
INCALN          INC.NOTST.ALLN
!                                                    "Cray COS INCLUDEs"
INCCOS          INC.NOTST.COS
!                                                    "Convex INCLUDEs"
INCVEX          INC.NOTST.CVEX
!                                                    "Modcomp INCLUDEs"
INCMC4          INC.NOTST.MC4
!                                                    "UNIX INCLUDEs"
INCUNIX         INC.NOTST.UNIX
!                                                    "VMS INCLUDEs"
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 COS pseudo AP"
QCOS            Q.DEV.PSAP.COS
!                                       "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
!                                       "Cray COS programs"
QPGCOS          Q.PGM.NOTST.COS
!                                       "Modcomp programs"
QPGMC4          Q.PGM.NOTST.MC4
!                                       "UNIX programs"
QPGUNIX         Q.PGM.NOTST.UNIX
!                                       "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
!                                       "Cray COS programs"
QYPGCOS         QY.PGM.NOTST.COS
!                                       "Modcomp programs"
QYPGMC4         QY.PGM.NOTST.MC4
!                                       "UNIX programs"
QYPGUNIX        QY.PGM.NOTST.UNIX
!                                       "VMS programs"
QYPGVMS         QY.PGM.NOTST.VMS
!
! "-------------------------------------------------------------"
```

```
! " System RUN file area - useful procedures for everyone                "
! "-------------------------------------------------------------------"
!
RUNSYS          RUN
!
! "-------------------------------------------------------------------"
! " System manager areas                                               "
! "-------------------------------------------------------------------"
!
!                                          "Generic"
SYSTEM          SYSTEM
!                                          "Generic UNIX"
SYSUNIX         SYSTEM.UNIX
!                                          "Alliant UNIX"
SYSALLN         SYSTEM.UNIX.ALLN
!                                          "Convex UNIX"
SYSCVEX         SYSTEM.UNIX.CVEX
!                                          "NRAO-VLA Convex local"
SYSVLAC1        SYSTEM.UNIX.CVEX.VLAC1
!                                          "NRAO-CV Convex local"
SYSNRAO1        SYSTEM.UNIX.CVEX.NRAO1
!                                          "Sun UNIX"
SYSSUN          SYSTEM.UNIX.SUN
!                                          "Local UNIX"
SYSLOCAL        SYSTEM.UNIX.LOCAL
!                                          "UNIX installation"
INSUNIX         SYSTEM.UNIX.INSTALL
!                                          "Generic VMS"
SYSVMS          SYSTEM.VMS
!                                          "NRAO-CVAX Local VMS"
SYSCVAX         SYSTEM.VMS.LOCAL
!                                          "VMS installation"
INSVMS          SYSTEM.VMS.INSTALL
!                                          "Generic COS"
SYSCOS          SYSTEM.COS
!                                          "Pittsburgh Supercomputer Center"
SYSPSC          SYSTEM.COS.PSC
!                                          "COS installation"
INSCOS          SYSTEM.COS.INSTALL
!                                          "Generic Modcomp"
SYSMC4          SYSTEM.MC4
!
! "-------------------------------------------------------------------"
! " Y-routine areas                                                    "
! "-------------------------------------------------------------------"
!
!                                          "Generic"
YGEN            Y.DEV
!                                          "DeAnza"
YDEA            Y.DEV.DEA
!                                          "III 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
!                                          "Stubbed"
YSTUB           Y.DEV.STUB
!                                          "Comtal Vision 1/20"
YV20            Y.DEV.V20
!
! "-------------------------------------------------------------"
! " Programs that reference Y-routines                          "
! "-------------------------------------------------------------"
!
!                                          "Standard programs"
YPGM            Y.PGM
!                                          "Non-standard programs"
YPGNOT          Y.PGM.NOTST
!                                          "Cray COS programs"
YPGCOS          Y.PGM.NOTST.COS
!                                          "Modcomp programs"
YPGMC4          Y.PGM.NOTST.MC4
!                                          "UNIX programs"
YPGUNIX         Y.PGM.NOTST.UNIX
!                                          "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.

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. ,index UPDAT

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 few differences from older releases of AIPS. First, there are a larger number of directories. This means that programmers need to know more 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.

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

A.5.1.1 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.

A.5.1.2 AIPS 'Version' 'Option'

This procedure starts up a given version of AIPS. On CVAX, '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

A.5.2.1 COMRPL 'SubroutineSpec' 'Option'

This routine will 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 (NOI4, NOOPTIMIZE, DEBUG, WARNINGS=DECLARATIONS, STANDARD=(NOSYNTAX,SOURCE_FORM) with 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
```

A.5.2.2 COMLNK 'ProgramSpec' 'Option'

This procedure will 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.

A.5.2.3 COMTST 'ProgramSpec' 'Option'

This is a version of COMLNK designed for 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 COMTST 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".

A.5.2.4 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

A.5.3.1 VERSION 'Version'

This command will set the default version 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.

A.5.3.2 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-NLAO:/OUTPUT-FORK.LOG"
```

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

```
$ FORK COMLNK IMLOD
```

A.5.3.3 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. Programmers on CVAX 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 make our experimental version of GRDAT and DSKFFT link with MX, 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.

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):

```
$ FORTRAN/NOI4/DEBUG/NOOPTI GRDAT
$ FORTRAN/NOI4/DEBUG/NOOPTI 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 Check Out System

Programmers at NRAO must use the checkout procedure to change AIPS code. 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 task that is still checked out of NEW cannot be checked out of TST, or vice versa.

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. This is because a concentrated (although not necessarily concerted) effort has been made to make them as similar as possible.

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, as edited for UNIX, appears in section A.2.3 above. It can be used to assign the appropriate mnemonics and/or to create a complete directory tree. It is very much like the VMS version, except that all intermediate directories are also defined. This is necessary if the file is to be used to create a complete directory tree. It could certainly be argued that we should not need to maintain separate VMS and UNIX versions of this file.

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:

```
$LIBR/AIPSUB/SUBLIB:$AIPSUB
```

APL subroutine source code search paths and object libraries:

Standard routines

```
$LIBR/APLSUB/SUBLIB:$APLSUB
```

Non-standard routines

```
$LIBR/APLNOT/SUBLIB:$APLNUNIX
$LIBR/APLNOT/SUBLIB:$APLNOT
```

Z-routines

```
$LIBR/APLALLN/SUBLIB:---Your local Z-routine directory goes here---
$LIBR/APLALLN/SUBLIB:$APLALLN---For example---
$LIBR/APLALLN/SUBLIB:$APL4PT2---For example---
$LIBR/APLALLN/SUBLIB:$APLBERK---For example---
$LIBR/APLALLN/SUBLIB:$APLUNIX
$LIBR/APLALLN/SUBLIB:$APLGEN
```

Q subroutine source code search paths and object libraries:

Standard routines

```
$LIBR/QSUB/SUBLIB:$QSUB
```

Non-standard routines

```
$LIBR/QNOT/SUBLIB:$QNOT
```

Q-routines

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

Y subroutine source code search paths and object libraries:

Standard routines

```
$LIBR/YSUB/SUBLIB:$YSUB
```

Non-standard routines

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

Y-routines

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

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

AIPGUNIX => UNIX specific stand alone programs

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

AIPPGM => Standard stand alone programs

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

APL-task source code search paths and link libraries:

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

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

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

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

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

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

Q-task source code search paths and link libraries:

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

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

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

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

Y-task source code search paths and link libraries:

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

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

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

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

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

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

QY-task source code search paths and link libraries:

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

```
$LIBR/QNOT/SUBLIB:$QYPGUNIX
$LIBR/APLNOT/SUBLIB:$QYPGUNIX
```

```
$LIBR/QSUB/SUBLIB:$QYPGUNIX
$LIBR/QVEX/SUBLIB---For example---:$QYPGUNIX
$LIBR/YSUB/SUBLIB:$QYPGUNIX
$LIBR/YSTUB/SUBLIB---For example---:$QYPGUNIX
$LIBR/APLSUB/SUBLIB:$QYPGUNIX
$LIBR/APLALLN/SUBLIB---For example---:$QYPGUNIX
$LIBR/APLSUB/SUBLIB:$QYPGUNIX

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

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

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

$LIBR/QSUB/SUBLIB:$QYPGM
$LIBR/QVEX/SUBLIB---For example---:$QYPGM
$LIBR/YSUB/SUBLIB:$QYPGM
$LIBR/YSTUB/SUBLIB---For example---:$QYPGM
$LIBR/APLSUB/SUBLIB:$QYPGM
$LIBR/APLALLN/SUBLIB---For example---:$QYPGM
$LIBR/APLSUB/SUBLIB:$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

A.7.1.1 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 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.SH

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.

A.7.1.2 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.                                             "
: "-------------------------------------------------------------------"
```

A.7.1.3 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.                                               "
: "-------------------------------------------------------------------"
```

A.7.1.4 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.                                              "
: "-------------------------------------------------------------------"
```

A.7.1.5 Executing AIPS Tasks Under A Debugger

Unlike VMS, where tasks linked with debug will automatically execute in debug mode, under UNIX the initiation of debug mode is a manual process. That is, the programmer's favorite debugger is invoked by name with the target executable module as one of its arguments. For AIPS tasks, the trick is to define the environment variable DBUGR as the full pathname to the desired debugger. This can usually be determined by the UNIX command "which". Under the C-shell, for example, the following line will define DBUGR as the pathname to the debugger "dbx":

```
setenv DBUGR `which dbx`
```

Under the Bourne or Korn shell, the syntax for the same would be the following:

```
DBUGR=`which dbx`; export DBUGR
```

The programmer must remember to do this prior to starting up AIPS. Thereafter, any task execution starts up under the control of the specified debugger. AIPS will automatically "wait" until the user exits the debugger.

A.7.2 Compiling And Linking

A.7.2.1 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
```

A.7.2.2 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 ia 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.                                              "
: "------------------------------------------------------------------"
```

A.7.2.3 COMTST

Use COMLNK.

A.7.2.4 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.7.3 Miscellaneous Routines

A.7.3.1 VERSION 'Version'

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

A.7.3.2 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 &
```

A.7.3.3 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.7.4 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/YM7O/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/YM7O/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.7.5 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. Each time the preprocessor is invoked on a Fortran module, $INCS is redefined as the official include file search path for the host implementation. The intent is to preserve the definition of $INCS as inviolate, despite any efforts to the contrary. However, it is possible to define your own environment variable for use with these INCLUDE statements. It's just that the logical name "INCS" is reserved. If a logical other than INCS is

used, the preprocessor will issue a warning message, but will otherwise cheerfully process the file. It is also possible to use INCLUDE statements that contain no logical, in which case the preprocessor will simply look for the file in the current working directory. In any case, if the preprocessor cannot locate the specified file, it will abort with an error message. The preprocessor also inserts the full pathname of the file that was included as a comment just before the included source text.

A.7.6 Executing Private Versions Of Tasks

The requirements for executing private versions of tasks are somewhat different from the VMS implementation. A version of the task HELP file (i.e., ".HLP" file) must reside in the same directory as the executable module, just as under VMS, but that's where the similarity ends. The stumbling block here is that case distinction for the values assigned to string adverbs in AIPS has been discontinued. Someday this may be reinstated, but until then, all such string values are converted to upper case. This makes it very difficult to specify pathnames that contain lower case characters. Therefore, the kludge solution for string adverbs like VERSION, INFILE and OUTFILE is to use predefined environment variables (i.e., defined prior to starting up AIPS) whose definitions are the pathnames to files that cannot otherwise be represented because they contain lower case characters. The names chosen for these environment variables must, of course, be entirely upper case. Deep in the bowels of the UNIX Z-routines, these environment variables are translated in order to extract the required pathname.

Therefore, in order to execute a private version of a task, you must first define an upper case environment variable as the pathname of the directory in which both the desired executable module and its corresponding ".HLP" file reside. These filenames must, of course, also be entirely upper case. When you start up AIPS, you can then specify the name of the environment variable (without the $ prefix) as the value of the VERSION adverb. For example, if we want to execute a private version of the task FOO, and FOO.EXE and FOO.HLP reside in /aippgmr/khilldru/tasklib, then we must define an environment variable, for instance, MYVAR as:

```
setenv MYVAR /aippgmr/khilldru/tasklib   (C shell)
```

or

```
MYVAR=/aippgmr/khilldru/tasklib          (Bourne or Korn shell)
export MYVAR
```

Then, in AIPS, after we set VERSION='MYVAR', we can type INPUTS FOO, HELP FOO, EXPLAIN FOO or GO FOO, and AIPS should find the necessary files.

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.

INDEX