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

W. D. Cotton and a cast of AIPS

Version 15 May 84

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
Gary Fickling - Systems and plotting
Eric Greisen - Head knocker, TVs and AIPS system integration
Kerry Hilldrup - Unix implementation
Fred Schwab - Mathematics
Craig Walker - Gadfly and VLBI
Don Wells - Hardware implementations and image processing techniques
Gustaaf van Moorsal - Spectral line

CHAPTER 5 CATALOGUES

CHAPTER 6 DISK FILES

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.

- "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 refered 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".

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 currently dominate the astronomical computing community but 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 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 standalone 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 file. AIPS then initiates the requested task and suspends itself indefinitely. The task reads the disk file and depending on the value of a logical "hidden" adverb (DOWAIT in AIPS and RQUICK in the task) may resume AIPS. The task then does the requested operation and before stopping resumes AIPS if this was not done previously. The task sends AIPS a return code when AIPS is resumed.

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 catalogued 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 may be 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 two basic types of files and two types of I/O to access them. The main data files which are assumed to contain the bulk of the data are read in a double buffered mode with large blocks being transferred. The extension files are read by single buffered transfers of 512 bytes. 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. More details about the I/O routines can be found in the chapter on I/O.

1.5 STYLE

1.5.1 Precursor Comments

The main point of this exercise is to make routines comprehensible to programmers who read them. Comment statements are the most powerful tool for this purpose. "Prologue" comments are placed immediately following the PROGRAM, SUBROUTINE, or FUNCTION statement. These comments 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. Prologue 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 prologue 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)*VEC0(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 = ATAN2 (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      CONTINUE
      Z = Z ** (B-A)
```

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

1.5.4 Statement Numbers

The use of GO TO statements is the cause of most logic errors in programming. Unfortunately, FORTRAN offers us no alternative. 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.

1.5.5 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
```

1.5.6 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 overlayed on the "smaller" machines.

1.5.7 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 characers 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 66, except for the addition of INCLUDE, ENCODE, and DECODE statements and the use of a minimum number of local assembly language in 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 FORTRAN IV PLUS or FORTRAN 77 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 ' <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.

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 prologue comments, 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, LOGICAL*2, REAL*4, and REAL*8 in any order. We prohibit DIMENSION, INTEGER, REAL, DOUBLE PRECISION, INTEGER*3, INTEGER*4, LOGICAL*1, LOGICAL*4, REAL*6, and CHARACTER statements and any use of these statements for data initialization. COMPLEX*8 and COMPLEX*16 are allowed.

2. Common statements: COMMON. We prohibit unlabeled common and use of the COMMON statement to give the types and dimensions of variables.

3. Equivalence statements: EQUIVALENCE.

4. Data initialization statements: DATA. We prohibit the use of DATA statements to initialize variables in commons. 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 'accc.INC' , where '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. Since the statement order is fixed, an include text file may contain statements of only one of the above types. For example,

```
      INCLUDE 'DBWT.INC'
      INCLUDE 'CBWT.INC'
```

causes the text

```
C                                              Include DBWT
      INTEGER*2 BWTLUN,BWTIND,BWTREC,BWTDAT(256)
      LOGICAL*2 WASERR
C                                              End DBWT
C                                              Include CBWT
      COMMON /BWTCH/ BWTLUN,BWTIND,BWTREC,WASERR,BWTDAT
C                                              End CBWT
```

to be inserted.

1.6.4 Variable Declaration

The programmer is urged 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 below, 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.

1.7 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.

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 value 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 existance 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.

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 ot 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 bi-monthly 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 Gary Fickling 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, catalogues and fills a new uv data file.
- CANDY This task creates, catalogues 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, cataloguing, 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. 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 are 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 catalogue
   - create and catalogue output file
2. Process data (SENDUV in FUDGE, SENDMA in TAFFY)
3. Convert output file to integer form if requested (OUTMA, TAFFY only)
4. write history (FUGHIS in FUDGE, TAFHIS, called from OUTMA in TAFFY)
5. Shut down (DIE)
   - unmark catalogue 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 catalogue 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,
     *   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. Ex:
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       Unit 1 is the line printer
C
C       If IRET .GT. 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 catalogue 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=-1.0D0.  This is to allow for the completion of
C  pending operations, i.e. preparation of HIstory cards.  Data
C  returned is ignored.  If valid data is to be returned then SENDUV
C  should be modified.
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  catalogue after the last call to DIDDLE.
C
C  Inputs:
C  NUMVIS      R*8  Visibility number, -1.0=> 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 which
```

```
C                    there is data, the julian day corresponding to
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 IA1 < IA2
C  RPARM(*)     I*2  Random parameter array which includes U,V,W etc
C                    but also any other random parameters.
C  VIS(3,*)     R*4  Visibilities in order real, imaginary, weight (Jy)
C                    Nonpositive weight means the data is flagged.
C
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  Volumn 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 of epoch CAT4(K4EPO) of phase center.
C                    (Deg.)
C  DEC          R*8  Declination of epoch CAT4(K4EPO) of phase center.
C                    (deg)
C  FREQ         R*8  Frequency of observation (Hz)
C  NRPARM       I*2  # random parameters.
C  NCOR         I*2  # stokes parameters.
C  CATBLK(256)I*2  Catalogue header record. See the chapter on
C                    catalogues for 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. NB U,V,W,
C                    time and 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  Catalogue 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 current value is

UMA0:[AIPS.15MAY84.APL.PGM].

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 catalogued file. DIDDLE can defer returning the next row; this allows the use of scrolling buffer. TAFFY can handle multidimensional, 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.

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 catalogue header record for the output file. For many purposes the only modifications needed to NEWHED are to modify the values in DATA statements from the default values supplied. The beginning portion of NEWHED is reproduced below.

```
      SUBROUTINE NEWHED (IRET)
C-------------------------------------------------------------------
C   NEWHED is a routine in which the user performs several operations
C   associated with beginning the task.  For many purposes simply
C   changing some of the values in the DATA statments will be all that
C   is necessary.  The following functions are/can be performed
C   in NEWHED:
C       1) Modifying the catalogue header block to represent the
C   output file.  The MINIMUM modifications required here are those
C   required to define the size of the output file; ie.
C      CATBLK(K2DIM)    = the number of axes,
C      CATBLK(K2NAX+i)  = the dimension of each axis, and
C      CATBLK(K2BPX)    => 1 = integer*2, 2 = real*4 pixel values.
C   Other changes can be made either here or in DIDDLE; the
C   catalogue block will be updated when the history file is
C   written.
C       2) Checking the input image and/or input parameters.
C   For example, if a given first axis type such as
C   Frequency/Velocity is required this should be checked.  The
C   routine currently does this and all that is required to
C   implement this is to modify the DATA statments.
C   A returned value of IRET .NE. 0 will cause the task to terminate.
C   A message to the user via MSGWRT about the reason for the
C   termination would be friendly.  This can be done by encoding
C   the message into MSGTXT, setting IRET to a non-zero value
C   and issuing a GO TO 990.
C       3) 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 statment.
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, BADD(10)
      INTEGER*2 NCODE, NTYPES, IOFF, IERR, INDXI, INC, INDEX, ITYPE,
     *   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, BADD
      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./
C                                          Set desired output pixel type
C                                          0 = same as input,
C                                          1=I*2, 2=R*4;
      DATA ITYPE /0/
```

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 occurence 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 variable
C   FBLANK.
C       DIDDLE may accumulate a scrolling buffer by returning a negative
C   value of IRET.  This tells the calling routine to defer writting the
C   next row.  If rows are deferred then and 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 writting 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   Unit 1 is the line printer
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   catalogue 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  Catalogue header block
```

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.

If the output file from TAFFY is to be in the form of scaled integers, the temporary results are kept in a scratch file. 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 souce area; this is usually assigned the logical name "APLPGM:" whose current value is UMA0:[AIPS.15MAY84.APL.PGM].

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 to change 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 catalogue header record are given in the chapter on catalogues.

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 catalogue header record (NEWHED)
   - create and catalogue output file
   - Enter antenna information (In UVFIL only)

3. Read/generate data (GETUV in UVFIL, MAKMAP in CANDY)

4. Convert output file to integer format if requested (CANDY only in OUTMA)

5. Write history (and antenna file) (FILHIS in UVFIL, CANHIS in CANDY)

6. Shut down (DIE)
   - Unmark catalogue file statuses
   - Restart AIPS if not done previously

2.2.1 UVFIL

UVFIL creates, catalogues 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 catalogue header block and to enter information about the antennas. The beginning portion of this routine follows:

```
      SUBROUTINE NEWHED (IRET)
C-------------------------------------------------------------------
C   NEWHED is a routine in which the catalogue header is constructed.
C   Necessary values can be read in in the areas markes "USER CODE
C   GOES HERE".
C
C   NOTE: the AIPS convention for the coordinate reference value
C   for the STOKES axis is that 1,2,3,4 represent I, Q, U, V
C   stokes' parameters and -1,-2,-3,-4 represent RR, LL, RL and
C   LR correlator values.  Currently set for R and L polarization
C   ie Ref. value = -1 and increment = -1.
C
C   The MINIMUM information required here is that
C   required to define the size of the output file; ie.
C      CATBLK(K2GCN) = Pseudo 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   catalogue 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 AIPS manual vol. 2 for details.
C
C    Input:
C     CATBLK(256)     I*2   Output catalogue header, also CAT4, CAT8
C                           The OUTNAME, OUTCLASS, OUTSEQ are entered
C                           elsewhere.
C    Output:
C     CATBLK(256)     I*2   Modified output catalogue header.
C     IRET            I*2   Return error code, 0=>OK, otherwise abort.
C    Also the antenna informtion can be filled into a common.
C-------------------------------------------------------------------
      INTEGER*2 CATBLK(256), SEQOUT, I, NAXIS, NRAN, ANTSYM(30),
     *          N0, N1, N2, N8, N256, NCHAN, NPOLN,
     *          DISKO, JBUFSZ, IERR, NANT, NDIM(7), INDEX, INC,
     *          ISTAR
      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), IATUTC,
     *          CRPIX(7), CRINC(7), UNITS(2), OP4TO8, BANDW,
     *          TELE(2), OBSR(2), INSTR(2), BLANK(2),
     *          UT1UTC, OBSDAT(2)
      REAL*8    CAT8(64), ANTLOC(3,30), GST0, CRVAL(7), XCOUNT
      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'
C                                        Antenna info common
```

```
      COMMON /ANTS/ ANTLOC, GST0, IATUTC, UT1UTC, ANTNAM, NANT,
     *   ANTSYM
      COMMON /BUFRS/ BUFFER, JBUFSZ
      COMMON /INPARM/ INFILE, IN2FIL,
     *   NAMOUT, CLAOUT, XSOUT, XDISO,
     *   APARM, BPARM,
     *   SEQOUT, DISKO
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE  (CATBLK, CAT4, CAT8)
      DATA N0, N1, N2, N8, N256 /0,1,2,8,256/,  BLANK /2*'    '/
      DATA OP8TO4 /'8TO4'/, 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  ','    '/
```

The user supplied routine FIDDLE returns visibility records which are written into the catalogued 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       Unit 1 is the line printer
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
C       See the precursor comments for UVPGET in the chapter on
C  the catalogues for a description of the contents of
C  COMMON /UVHDR/ which allows easy access to much of the information
C  from the catalogue header (CATBLK) and which describes the order
C  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=-1.0D0.  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 that 1 and 5
C  will probably also work.)
C
C       The current contents of CATBLK will be written back to the
C  catalogue after the last call to FIDDLE.
C
C  Inputs:
C  NUMVIS      R*8  Visibility number, -1.0=> 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 of epoch CAT4(K4EPO) of phase center.
C                   (Deg.)
C  DEC         R*8  Declination of epoch CAT4(K4EPO) of phase center.
C                   (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  Catalogue header record. See the catalogue chapter
C                   for more 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  Catalogue 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 nonstandard program souce area; this is usually assigned the logical name "NOTPGM:" whose current value is UMA0:[AIPS.15MAY84.NOTST.PGM].

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 give an example of reading a formatted disk file using Fortran 77.

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

```
      SUBROUTINE NEWHED (IRET)
C-------------------------------------------------------------------
C   NEWHED is a routine in which the user performs several operations
C   associated with beginning the task.  For many purposes simply
C   changing some of the values in the DATA statments will be all that
C   is necessary.  The following functions are/can be preformed
C   in NEWHED:
C       1) Creating the catalogue header block to represent the
C   output file.  The MINIMUM information required here is that
C   required to define the size of the output file; ie.
C      CATBLK(K2DIM)= the number of axes,
C      CATBLK(K2NAX+i) = the dimension of each axis, and
C      CATBLK(K2BPX) => 1 = integer*2, 2 = real*4 pixel values.
C   Other changes can be made either here or in MAKMAP; the
C   catalogue block will be updated when the history file is
C   written.
```

```
C       2) Setting default values of some of the input parameters
C   As currently set the default OPCODE is the first value in the
C   array CODES which is set in a data statment.

C    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
      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, ITYPE
      REAL*4    CODES(10), UNITS(2,10), ATYPES(2,7),
     *   BLANK(2), TEMP, FCHARS(3)
C***********************************************************************
C   SAMPLE CODE
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                                          Set desired output pixel type
C                                          1=I*2, 2=R*4
      DATA ITYPE /1/
```

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 occurence of
C   blanking.  CATBLK(K2NAX) values per call are expected returned.
C  NOTE: blanked values are denoted by the value of the common variable
C  FBLANK
C
C       Up to 10 history entries can be written by using 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       Unit 1 is the line printer
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       AIPS I/O  LUN 16 is open and not available to MAKMAP.
C  FORTRAN unit numbers greater than 50 will probably not get the
C  AIPS routines confused.  (Any unit numbers other that 1 and 5
C  will probably also work.)
C
C       The current contents of CATBLK will be written back to the
C  catalogue 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  Catalogue header block
C
```

Pixel blanking is supported thru 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 nonstandard program souce area; this is usually assigned the logical name "NOTPGM:" whose current value is UMA0:[AIPS.15MAY84.NOTST.PGM].

2.3 MODIFIYING 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 (and UNIX ?) versions of AIPS 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 VAX/VMS DEBUGGER IN AIPS.

The symbolic debugger in VAX/VMS systems is a very powerful tool for debugging AIPS tasks. 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. These procedures are @COMLNK for non-AP standard tasks, @NCOMLNK for non-AP nonstandard tasks, @APCLNK for standard AP tasks, and @NAPCLNK for nonstandard AP tasks.

- Use the verb WAITTASK after starting a task with the debugger on. This keeps AIPS and the debugger from trying to talk to you at the same time and will resume AIPS when the task quits for any reason.

- 'WATCHPOINT' doesn't work in AIPS programs. If a WATCHPOINT is set, all AIPS I/O routines will fail.

- When specifying a routine, type "SET SCOPE routine\routine" or give the SET SCOPE command twice; the debugger doesn't think that you are serious if you only do it once.

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 catalogue (MAPOPN, CATDIR, CATIO)
   - create and catalogue output file (MCREAT, UVCREA)
   - create scratch files (SNCRC)

3. Process data

4. Write history (HISCOP, HIADD, HICLOS)

5. Shut down (DIETSK, DIE)
   - destroy scratch files
   - unmark catalogue 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. The task must read these values from the disk file. After AIPS has started up a task it, suspends itself indefinitely. 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).

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

Each catalogued 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 thru 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 chapter on device I/O and ZTTYIO in particular.

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 catalogued disk file. Two

others will create and catalogue 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. A number of skeleton task 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 is 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 many compilers allow the Fortran 66 conventions. As a result, the AIPS system 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.

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. 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 string to another

- CHCOMP compares two packed character strings.
- CHFILL fills a 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. The AIPS convention is to assume that all integers are the smallest supported on the machine. In the standard versions of our routines integers are explicitly declared to be INTEGER*2. If this type of integer is not supported on a given machine then a preprocessor is required to convert INTEGER*2 to INTEGER. When possible, it is best to tell the compiler that all undeclared and literal values are INTEGER*2.

The limitation to 16 bit integers causes problems in a number of cases. To take care of these problems we use 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.

- 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

A problem related to the use of integers is the default type of call arguments. This problem occurs on compilers which support two lengths of integers, but the default for undeclared and literal integers is the long integer. In this case, if the call statement includes an expression, the result of the expression will be a long integer whereas the routine will expect a short integer.

To avoid the problems resulting from expressions and literal values in call arguments, we advocate avoiding 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 memnonic. For example, most tasks whose principle 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. 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          2.0    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)
------------------------------------------------------------------
PRTTP
Type:  Task  (interactive only)
Use:   To print on the line printer 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.
------------------------------------------------------------------
```

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.

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 sould 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, preceeded 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 In The Task (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.

3.5 RESTARTING AIPS

When AIPS starts a task it suspends itself indefinately. 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'
```

On CVAX:: "INCS:" is currently logically assigned the value of UMA0:[AIPS.15MAY84.INC]. For development and testing purposes INCLUDE files may be kept in directories other than the one specified by INCS: for instance on a VAX one might use:

```
INCLUDE 'UMA0:[WDC]DUVZ.INC'
```

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) * 22

+ (# 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 catagory at any time.

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 standalone utility program SETPAR. The constants kept in this common are described in the chapter on disk I/O.

3.7.2 Catalogue Pointer Common

The catalogue 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 catalogue 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 catalogue 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 catalogue 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 catalogue 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 catalogued files and create several scratch files. The status of the catalogued files are marked 'READ' or 'WRIT' in the catalogue 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 catalogue files and destroy scratch files under any circumstances in which it controls its death.

Many tasks acomplish 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. SCRFIL contains the physical names of the scratch files and SCRVOL contains the disk numbers of the scratch files.

NCFILE tells how many catalogue files are marked, FVOL contains the disk numbers of the catalogued files marked, FCNO contains the catalogue slot numbers of the marked files, and FRW contains flags for each of the marked catalogue 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 and MXSFDK are used by the scratch file creation routine SNCRC. RQUICK is also carryed 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, catalogue file status words will be cleared and scratch file 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
```

an 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 wildcard 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 default 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 catalogue (or catalogues) 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 wildcards; "*" 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 default and wildcard are fully supported by the standard catalogue 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. Wildcards are supported in the output name; basically a wildcard 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 wildcard 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 catalogued file may (and usually does) have auxillary files containing information related to the catalogued 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 files (type 'AN') should be copied and for images any CLEAN components files (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 initialized the history common.
- HISCOP creates and catalogues 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.
- 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 an integer or real array dimensioned to be at least 72 bytes and calling HIADD. An example:

```
      INTEGER*2  CARD(36)
      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 percursor comments for these routines can be found in AIPS manual volumn 3.

3.9.2 Extension Files (EXTCOP)

A simple copy of any or all extension files of a given type may be performed with a single call to the utility routine EXTCOP. Certain extension file types are excluded from being copied by EXTCOP, these being history files (type 'HI') and plot files (type 'PL'). If the new and old files are physically the same files, then EXTCOP makes no changes and simply returns. A description of the call sequence is given at the end of this chapter.

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 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 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 electrostaic 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.

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 is 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 a packed character string. An example of the use of ZTTYIO is the following:

```
      INTEGER*2  N1, N72
      INTEGER*2  TYYLUN, TYYIND, IRET, LINE(36)
      LOGICAL*2  T,F
      REAL*4     READ, WRITE
                    .
                    .
      DATA N1, N72 /1,72/,   TTYLUN, TTYIND /5,0/
      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. 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 SNCRC. SNCRC 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 SNCRC can be found at the end of this chapter.

An example of the use of SNCRC is the following:

```
      INTEGER*2 SIZE(2), SYM, ISCR, IRET, NX, NY, NP(2), BP, N2
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DDCH.INC'
           .
           .
      INCLUDE 'INCS:CFIL.INC'
      INCLUDE 'INCS:CDCH.INC'
           .
           .
      DATA SYM  /'XX'/, 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
C                                        SIZE is a pseudo I*4.
      CALL MAPSIZ (N2, NP, BP, SIZE)
```

```
C                                          Create scratch file of type
C                                          'XX'
      CALL SNCRC (SIZE, SYM, ISCR, IRET)
C                                          Test for errors...
```

In the above example, the scratch file created will be number ISCR in the /CFILES/ common. Some routines, such as the FFT routine DKSFFT, accept these numbers directly. These scratch files can be opened as follows:

```
      CALL ZOPEN (LUN, IND, SCRVOL(ISCR), SCRFIL(1,ISCR), T, T, T,
     *    IRET)
```

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

3.12 TERMINATING THE PROGRAM

Most tasks create scratch files or open catalogued files which have status words marked in the catalogue directory. These scratch files should always be destroyed by the end of the program, and the catalogue 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 catalogued files. If this common is used, the shutdown routine DIE will take care of deleting all scratch files, unmarking catalogue 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 are 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.

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 On A VAX[B

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 hierarchial directory structure with logical names for each of the subdirectories. An example of the directory structure is:

CVAX::UMA0:[AIPS.15MAY84.APL.ZSUB.VMS]

where:
CVAX:: denotes the DECNET node name (optional)
UMA0: is the name of the disk drive
[AIPS. is the main AIPS directory
.15MAY84. is the directory for the current release
.APL. indicates "standard" non-array processor code
.ZSUB. indicates the "Z" subroutine sub directory
.VMS] indicates the VMS subroutine library.

The logical name for the directory described above is APLVMS:

The steps in installing a task on a VAX/VMS system are then:

1. Set the logical assignments. This is usually done with a command procedure which will be described below.

2. Create or put the source and help files in the correct directories. A later section will describe the use of directories.

3. Compile and link edit the task with the AIPS libraries. The compile and link edit procedures will be described in a later section.

3.14.1.1 Logical Assignments - The logical assignments are usually set with the command procedure CDNEW. Programmers in the same VMS group as AIPS can invoke this by:

```
$@NEW:CDNEW
```

Programmers in other groups need to install this procedure in their

directory. The listing of this procedure follows:

```
$! CDNEW
$!-------------------------------------------------------------
$! CDNEW  sets the various process logical assignments required.
$! It also causes the default protection to go to Owner and
$! Group having RWED.
$! This is a sample procedure.  Each programmer should have a copy
$! of this procedure in his own area.
$!-------------------------------------------------------------
$ ASS UMA0:[AIPS.15MAY84] NEW:
$ @NEW:ASSIGNP
$ EXIT
```

The directory given in the above example should be changed to the current value for your system. Check with the AIPS system manager for details.

3.14.1.2 Where To Put The Files (AIPS Directories). - The proper AIPS directory depends of the type of file involved. The following partial list tells which directory by logical name corresponds to different types of files.

File Type	Logical name
INCLUDE file	INCS:
HELP file	HLPFIL:
Documentation	DOCTXT:
"Standard" subroutines	APLSUB:
"Nonstandard subroutines	NOTSUB:
"Standard" programs without array processor	APLPGM:
"Nonstandard" programs without array processor	NOTPGM:
"Standard" programs using array processor	APLAPG:
"Nonstandard" programs using array processor	NOTAPG:
VMS "Z" routines	APLVMS:
Execute module	LOAD:

3.14.1.3 Where To Put The Files (Programmers Directory) - For many purposes, it is convienent to leave a task in the programmers 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.

3.14.1.4 Compile And Link Edit Procedures - There are a number of command procedures to compile and link edit programs. These procedures use logical names for the different libraries, so the procedure CDNEW must be run before the compile and link edit procedures. These procedures are kept in the directory obtained with the logical name NEW:. Programmers using their own directories will need to copy the relevant procedures to their own area and to change the directories for source code or execute modules as needed. The compile and link edit procedures are used as in this example:

```
$@COMLNK "task name"
```

The following list describes the use of the major compile and link edit routines.

Compile subroutine and enter into library

Subroutine type	Procedure
Standard (not AP)	COMRPL
Nonstandard (not AP)	NCOMRPL
FPS AP routines	FCOMRPL
Pseudo AP routines	PCOMRPL

Compile and link edit task

Task type	Procedure
Standard (no AP)	COMLNK
Nonstandard (no AP)	NCOMLNK
Standard (AP)	APCLNK
Nonstandard (AP)	NAPCLNK
AIPS or utility pgm	ACOMLNK

3.15 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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statments.
- Exxx.INC. These contain Fortran EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statments.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible 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 CDCD.INC

```
C                                                          Include CDCH
      COMMON /DCHCOM/ NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR,
     *    CSIZPR, NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NBITWD,
     *    NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, XPRDMM, XTKDMM,
     *    NTVACC, NTKACC, UCTSIZ, BYTFLP, SYSNAM, VERNAM, USELIM,
     *    IFILIT, RLSNAM
      COMMON /FTABCM/ DEVTAB, FTAB
C                                                          End CDCH.
```

3.15.2 CFIL.INC

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

3.15.3 CMSG.INC

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

3.15.4 CUVH.INC

```
C                                                               Include CUVH
      COMMON /UVHDR/ FREQ, RA, DEC, SOURCE, NVIS, ILOCU,
     *   ILOCV, ILOCW, ILOCT, ILOCB, JLOCC, JLOCS, JLOCF,
     *   JLOCR, JLOCD, INCS, INCF, ICOR0, NRPARM, LREC, NCOR, ISORT
C                                                               End CUVH
```

3.15.5 DDCH.INC

```
C                                                               Include DDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *   CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *   NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *   NTKACC, UCTSIZ, BYTFLP, USELIM, IFILIT,
     *   DEVTAB(50), FTAB(1)
C                                                               End DDCH.
```

3.15.6 DFIL.INC

```
C                                                               Include DFIL
      REAL*4    SCRFIL(6,20)
      INTEGER*2 NSCR, SCRVOL(20), IBAD(10), LUNS(10), MXSFDK(10)
      INTEGER*2 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, NACOUN,
     *   MSGREC, MSGKIL
      REAL*4    MSGTXT(20)
C                                                               End DMSG.
```

3.15.8 DUVH.INC

```
C                                                               Include DUVH
      INTEGER*2 NVIS(2), ILOCU, ILOCV, ILOCW, ILOCT, ILOCB,
     *   JLOCC, JLOCS, JLOCF, JLOCR, JLOCD, NRPARM, LREC,
     *   NCOR, ISORT, INCS, INCF, ICOR0
      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)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *   CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *   NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *   NTKACC, UCTSIZ, BYTFLP, IFILIT,
     *   USELIM, DEVTAB(50)
C                                                               End IDCH.
```

3.16 ROUTINES

3.16.1 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
```

3.16.2 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
```

3.16.3 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
```

3.16.4 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.
```

3.16.5 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
```

3.16.6 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
```

3.16.7 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
```

3.16.8 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
```

3.16.9 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
```

3.16.10 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
```

3.16.11 DIE - does the housekeeping necessary for an orderly death of the task, primarily clearing catalogue 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 catalogue are communicated by COMMON /CFILES/

```
NCFILE    I*2  Number of files marked in catalogue.
FVOL(50)  I*2  Volumn 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 volumn numbers
SCRFIL(6,20) R*4 Scratch file physical names.
```

3.16.12 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
```

3.16.13 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   Catalogue slot number for old file
 CNONEW      I*2   Catalogue slot number for new file
 CATNEW(256) I*2   Catalogue 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.
```

3.16.14 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
```

3.16.15 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
```

3.16.16 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
```

3.16.17 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
```

3.16.18 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  Catalogue slot number of old history file.
        CNONEW          I*2  Catalogue slot number of new history file.
In/Out: CATBLK(256)     I*2  Catalogue 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.
```

3.16.19 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.
```

3.16.20 PSFORM - prepares a string patterm 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
```

3.16.21 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
```

3.16.22 SNCRC - creates a scratch file, in doing so it attempts to put the file on a disk drive other than the one on which it put the last file.

```
   SNCRC (SIZE, SYM, ISCR, IRET)
Inputs: SIZE   I*2(2)   File size in bytes as Pseudo I*4
        SYM    I*2      File type (2 characters)
Output: ISCR   I*2      CFILES scratch file number.
                        used (on output).
        IRET   I*2      Error code:  0 => ok
                                     1 => disk space unavailable
                                     2 => other
If IRET > 0, file has not been created.
SNCRC uses the common in the  INCLUDE CFIL.INC:
       NSCR     I*2       Number of scratch files already.
       IBADD    I*2(10)   Disk drives to avoid.
       MXSFDK   I*2(10)   Next number for scratch file to use
                          on each disk.  Is one filled when
                          called with NSCR .le. 0.
       SCRFIL   R*4(6,20) Scratch file names.
       SCRVOL   I*2(20)   Scratch file volumns
```

3.16.23 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 + (ICOR-IABS (ICOR0)) * INCS

```
UVPGET (IERR)
Inputs: From common /MAPHDR/
   CATBLK(256)    I*2  Catalogue 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
   JLO          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.
   INCS           I*2  Increment in data for stokes (see above)
   INCF           I*2  Increment in data for freq. (see above)
   ICOR0          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(2)      P 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
```

3.16.24 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 hardcoded 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.
```

3.16.25 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
```

3.16.26 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
```

3.16.27 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(*)   I/O buffer
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 statments 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 statments 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 statments control the execution of other statments, eg. 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.

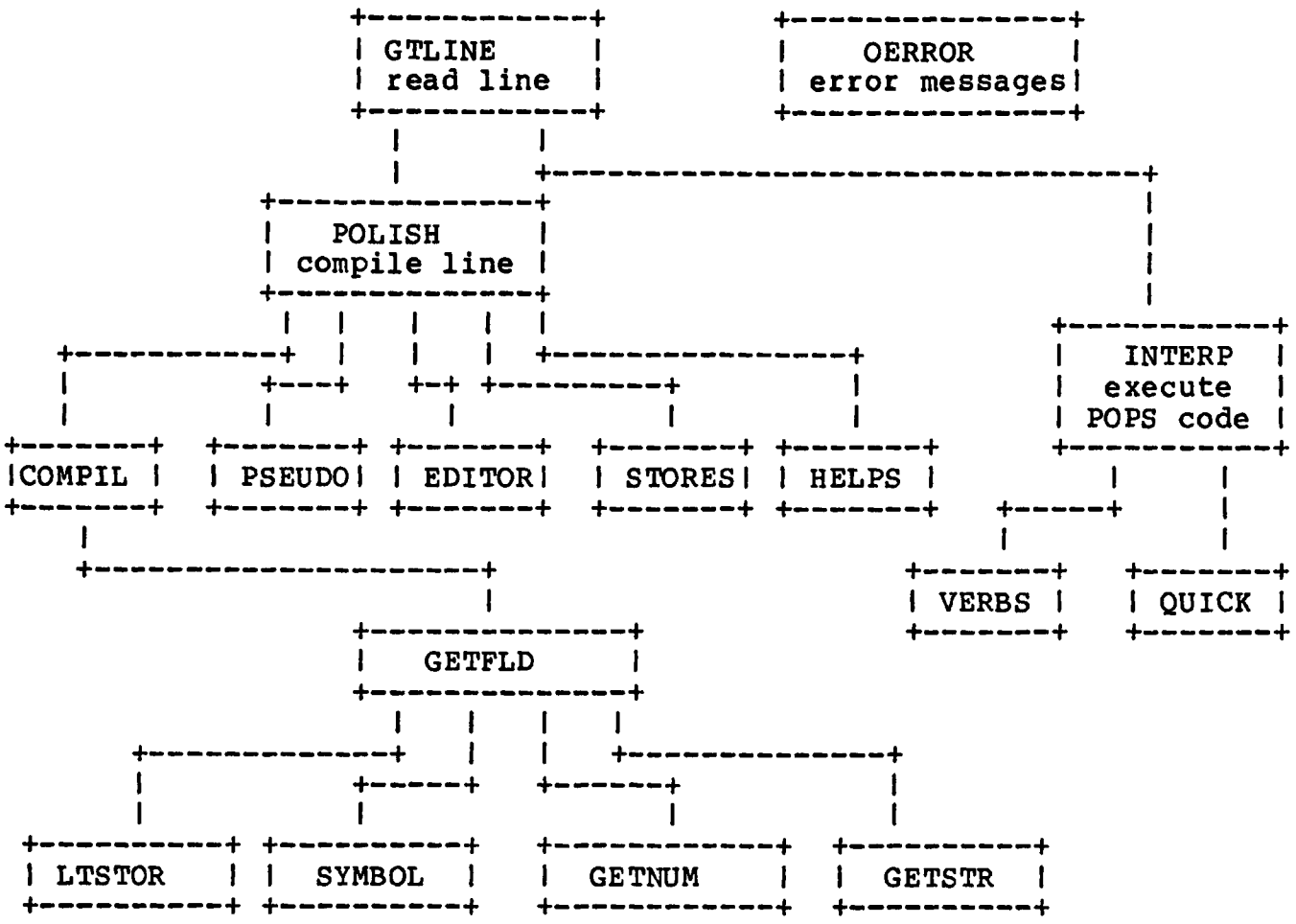
6. Prepare batch file. AIPS can run in a batch mode. To do this, the user enters and/or edits a list of commands in a batch file for later execution. This can be done either in the normal AIPS or a special batch version of AIPS named BATER.

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 statments, 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 heirarchial. 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 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, ie. 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 7380
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 7381.

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 postion 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 changable 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
SP0	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 a character 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 (eg. scalar, character string, verb etc.) and TAG is a label for the particular symbolic (eg. 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 and add its name to PNAME. If GTLINE determines that an error has occured 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.

| 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 statment, 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 great than the precedence of the operator currently being scanned. This takes place until the top operator on the B stack has precede e 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 statment "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 empying 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 contigious 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 statment 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 the 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, four temporary verbs are available, T1VERB, T2VERB, T3VERB and T4VERB (verb numbers 900-903) These are accessable 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 (ACOMRPL), and recompile AIPS and relink it. Once verbs are tested they should be

moved to a more permanant AU routine.

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.

4.5 INSTALLING NEW ADVERBS

New, temporary, adverbs can be created in an executing AIPS task by SCALAR, ARRAY or STRING statments 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 amoung the other adverbs in CAPL.INC as in the POPSDAT file. The adverb value will be kept in this variable and is therefore directly available to verbs.

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 extent (number of positions) of the first dimension of the array of character strings. This is normally 1 as there are only scalar character string adverbs at the moment. The next integer field is blank and the next REAL (F7.0) field is the number of characters in the string.

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
ERASE	5	5	79	
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
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-GO          2    4   120                             \ AU2
SPY           3    4   121
C-WAITTASK            122
C-ABORTASK    8    4   123
C-TPUT        4    4   124
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
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	
TVWEDGE	7	4	250	\ AU5C
IMWEDGE	7	4	251	
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	
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	
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	
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	

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		
T1VERB	6	4	900		\ AUT
T2VERB	6	4	901		
T3VERB	6	4	902		
T4VERB	6	4	903		
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
SOURCE	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	20.00
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		0.00	

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	

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 = ' '
DOPOS = 1 ;  DOMAX = 1 ;  DOWIDTH = 1 ;
*
PROC TSTDUM
SCALAR X, Y, I , J , DELTAX , DELTAY
FINISH
*
PROC ABS(X); IF X>=0 THEN RETURN(X); ELSE RETURN(-X); END
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 QEXIT; PRIO=22; EXIT; 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,
     *    SOURCE, 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, XRASHF, XDCSHF,
     *    XFLDSZ, XPHAT, XGNERR, XTMSMO, DOOUTP, DOCNCT, DONEW, DOCONF,
     *    DOALPH, ERRORA, GRNAME, GRADDR, GRPHON, SLOTAD,
     *         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/  C
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, SP0, 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(7390)
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, SOURCE(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)
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(20), XYRATO, DOINVR, DOCENT, ZXRATO, SKEW, DOCONT
      REAL*4  DOVECT, ICUT, PCUT, DIST, IMSIZE(2), 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, XRASHF(16), XDCSHF(16), XFLDSZ(2,16),
     *    XPHAT, XGNERR(30), XTMSMO(30), DOOUTP, DOCNCT, DONEW,
     *    DOCONF, DOALPH, ERRORA, SLOTAD,
     *          STRA1, STRB1, STRC1, STRA2(2), STRB2(2), STRC2(2),
     *          STRA3(3), STRB3(3), STRC3(3), 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(10752), KXORG
      REAL*4 C(5376)
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, SP0, 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

CATALOGUES

5.1 OVERVIEW

AIPS keeps a catalogue with a directory which contains an entry for each data file and its associated extension files. The catalogue 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. The intent of this chapter is to describe the contents of the catalogue header and to describe the use of the routines that access the catalogue header record.

The information in the catalogue header record is patterned after the FITS format tape header, although it is not nearly as flexible. The catalogue header describes the order and amount of data, its format, scaling information for scaled integer files, 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. Pixels locations must be evenly spaced along each axis, although a proper redefination 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 catalogue 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 catalogue header record and modifies it to describe the data in the new file.

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

5.2 PUBLIC AND PRIVATE CATALOGUES.

AIPS catalogues may be either public, ie. all files on a given disk are in the same catalogue, or private, ie. each user has a separate catalogue on each disk. The standalone 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 catalogue routines.

5.3 FILE NAMES

AIPS data files, especially catalogued 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 the given by disk number, file name, file class, file sequence number, file physical type, user number, and for extension files, the version number. These are 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 catalogue using various default and wildcard conventions.

2. Disk and catalogue number. Just as the AIPS user frequently uses the disk and catalogue numbers to specify files using the verb GETNAME, programs usually keep track of catalogued files by means of the disk and catalogue numbers, file types, and version numbers for extension files. (Scratch files are sometimes specified by their order numbers in the /CFILES/ common.)

3. Physical name. The host operating system needs a name for the file for its own catalogue. 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 catalogue numbers, the file type and version, and the user number for systems with private catalogues. These physical names may be up to 24 characters long.

   An example from a VAX system with private catalogues is "DA0n:ttdcccvv.uuu" ; where n is the zero relative disk drive number, DA0n: is a logical variable which is assigned

to a directory, tt is a two character file type (eg. 'MA'), d is the one relative disk drive number, ccc is the catalogue slot number, vv is the version (01 for "MA" and "UV" files), and uuu is the users number in hexidecimal notation.

5.4 DATA CATALOGUE

5.4.1 Structure Of The Catalogue Header Record

The catalogue header block is a fixed format data structure 512 bytes long (one byte is defined in AIPS as half a short integer). The catalogue header block contains double and single precision floating point numbers, integers (both short and Pseudo I*4), and character strings. The catalogue 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 catalogue 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" refer 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 minumum
336	4	R*4	K4BLK= 85	Value of indeterminate pixel (real maps only)
340	4	I*2(2)	K2GCN=171	Number of random par. groups given as a Pseudo-I*4 number. 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	Number of pixels on each axis
		(K2CTPN= 7)		
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	Image name
		(K4IMNO= 1)		Character offset in packed string
380	6	C*6	K4IMC= 93	Image class
		(K4IMCO=13)		Character offset in packed string
386	2	C*2	K4PTY= 93	Map physical type (i.e. 'MA','UV')
		(K4PTYO=19)		Character offset in packed string
388	2	I*2	K2IMU=195	Image user ID number
390	2	I*2	K2NIT=196	# 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	Names of subsidiary file types
		(K2EXTN=10)		(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 <u>after</u> 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 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 catalogue header block is obtained from a common named /MAPHDR/. Many AIPS utility routines get the catalogue header record from this common, so it is a good place to store it.

5.4.1.1 Image Files - Images consist of a single multidimensional (up to 7), rectangular array of pixel values. The structure of this array is defined by the catalogue header record which contains the number of dimensions (K2DIM), the number of pixels on each axis (K2NAX) and the format of the data (K2BPX). If the data is in the form of scaled integers, the scaling parameters 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 pixels 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 axis and is the rotation of the coordinate system from north toward east.

Since there is no explicit provision made in the catalogue 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 definations 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.

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.

5.4.1.2 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 catalogue header record by the pseudo integer*4 value pointed to by K2GCN. 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 (ie. u, v, w, time, baseline). The data array is described by the catalogue 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 an 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 catalogue 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.

5.4.2 Routines To Access The Data Catalogue

5.4.2.1 MAPOPN And MAPCLS - There are a number of utility routines to access the catalogue header record. In many cases, most of the catalogue operations can be taken care of by the pair of routines MAPOPN and MAPCLS. MAPOPN will locate the correct catalogue entry from a given Name, class, disk, sequence and physical type following all default and wildcard conventions. MAPOPN then reads the catalogue header record, opens the main data file and marks the catalogue 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 catalogue header record if requested and clear the catalogue status word for the file. A description of the call sequence of MAPOPN and MAPCLS is described at the end of this chapter.

5.4.2.2 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 catalogue directory. The routine CATDIR is the basic method of accessing the catalogue directory. This routine will find the desired file given the name, class, etc. following the usual default and wildcard conventions. CATDIR returns the disk number and catalogue slot number. Given a disk number and catalogue slot number CATIO can read or write a catalogue header record and/or change the status word. Detailed descriptions of CATDIR and CATIO can be found at the end of this chapter.

5.4.3 Routines To Interpret The Catalogue Header

There are a number of specialized routines which obtain information from the catalogue 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.

5.4.4 Catalogue Status

The AIPS catalogue directory keeps a status word for each catalogued 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 catalogue. 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 catalogue 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 catalogue header block.

5.5 IMAGE CATALOGUE

5.5.1 Overview

The image catalogue 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 catalogue which performs the same functions for graphics devices (e.g. TEK4012 storage screens).

There is one image catalogue file for each television device whose physical name corresponds to ICl0000n, 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 cataloguing up to N subimages, plus a (N-1)/51+1 block directory. The directory immediately precedes the catalogue blocks for each image plane. For each TV graphics overlay plane there is one catalogue 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 catalogue 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 catalogue blocks themselves are essentially duplicates of the map catalogue blocks except that scaling information replaces the extension file index of the map catalogue.

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

Directory Block (Grey-scale image)

OFFSET	LENGTH	TYPE	DESCRIPTION
0	2	I*2	Sequence number of last sub-image catalogued 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
.	.		
.	.		
.	.		

Catalogue Block for each image or subimage:

Most of the Image Catalogue block is identical to the map CAtalog block of the source of the image. (See section on CA 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	Catalogue 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 catalogue 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 catalogue. The following is a short description of each; detailed descriptions of the call sequences is given at the end of this chapter.

- ICINIT clears the Image Catalogue for a given plane.
- ICOVER asks if there are any overlapped images in each quadrant visible.
- ICWRIT adds a new block to the catalogue.
- ICREAD 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 Catalogue 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 pixels location. In a similar fashion, most spectral observations are done with evenly spaced frequency channels which results in a non linear relation between the velocity of a channel and the channel number. AIPS memo no. 27 describes in great detail the approach AIPS uses to these problems. Much of the following sections is taken from this memo.

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. 26 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 catalogue 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 Catalogue Information - In addition to the normal axis coordinate information carried in the catalogue header, described previously in this chapter, the catalogue 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 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 aperature 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.

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 initilizes the /LOCATI/ common based on the current catalogue 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.

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.
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 latitide
		= 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

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 as provided in the AIPS catalogue 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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statments.
- Exxx.INC. These contain Fortran EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statments.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible 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,
     *   K2PTPN, K2CTPN, K2EXTN,
     *   K2GCN, K2PCN, K2DIM, K2NAX, K2BPX, K2INH,
     *   K2IMS, K2IMU,  K2NIT, K2TYP, K2ALT, K2EXT, K2VER,
     *   I4RAN, I2VOL, I2CNO, I2WIN, I2DEP, I2COR,
     *   I2TRA, I2PLT, I2OTH
C                                                          End CHDR.
```

5.7.2 CLOC.INC

```
C                                                               Include CLOC
      COMMON /LOCATI/ RPVAL, COND2R, AXDENU, 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, TVDUMS, TVZOOM,
     *    TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC, TYPMOV
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 K2PTPN, K2CTPN, K2EXTN
      INTEGER*2 K2GCN, K2PCN, K2DIM, K2NAX, K2BPX, K2INH,
     *    K2IMS, K2IMU, K2NIT, K2TYP, K2ALT, K2EXT, K2VER
      INTEGER*2 I4RAN, I2VOL, I2CNO, I2WIN, I2DEP, I2COR,
     *    I2TRA, I2PLT, I2OTH
C                                                               End DHDR.
```

5.7.5 DLOC.INC

```
C                                                              Include DLOC
      REAL*8    RPVAL(4), COND2R, AXDENU
      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, TVDUMS(7),
     *   TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *   TVSPLM, TVSPLC, TYPMOV(16)
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	Catalogue 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 catalogue 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: 'SRCH' high seq # (if SEQ 0), return defaults 'SRNH' high seq # (if SEQ 0), NOT return defaults 'SRCN' next match, return defaults 'SRNN' next match, NOT return defaults '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 catalogue 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 catalogue file
               4 =>  CLOSE blocked by non-REST status
               5 =>  end of catalogue 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 catalogue.

```
    CATIO (OP, IVOL, CNO, CATBLK, STAT, BUFF, IERR)
Inputs:  OP      R*4       'READ' => get block into CATBLK
                           'WRIT' => put CATBLK onto disk catalogue
                           'UPDT' => as WRIT but for use when the
                                     calling program has previously
                                     set the status to WRITE
         IVOL    I*2       Disk volume containing catalogue (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 catalogue file
                              2 => input parameter error
                              3 => cannot read catalogue 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 ICINIT - Initialize image catalog for plane IPLANE.

```
      SUBROUTINE ICINIT (IPLANE, BUFF)

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

5.8.5 ICOVER - checks to see if there are partially replaced images in any of the TV planes currently visible by quadrant. Currently this routine is in the AIPSUB: area.

```
      ICOVER (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
```

5.8.6 ICWRIT - Write image catalog block in ICTBL into image catalog.

```
      ICWRIT (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
```

5.8.7 ICREAD - Read image catalog block into ICTBL.

```
      ICREAD (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
```

5.8.8 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)

  Inputs: XPIX   R*4   X pixel position
          YVAL   R*8   Y coordinate value
  Output: XVAL   R*8   X coordinate value
  Common: /LOCATI/ position parameters must have been set
           up by SETLOC
```

5.8.9 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)

  Inputs: YPIX   R*4   Y pixel position
          XVAL   R*8   X coordinate value
  Output: YVAL   R*8   Y coordinate value
  Common: /LOCATI/ position parameters must have been set
           up by SETLOC
```

5.8.10 MAPCLS - closes a map file and clears the catalogue 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    Catalogue slot number of file
     LUN      I*2    Logical unit # used for file
     IND      I*2    FTAB pointer for LUN
     CATBLK   I*2(256) New catalogue header which can optionally
                        be written into header if OP=WRIT or INIT
                        Dummy arguement if OP=READ
     CATUP    L*2    If TRUE write CATBLK into catalogue,
                        ignored if OP = READ
  Outputs:
     IERR      I*2   0 = O.K.
                     1 = CATDIR couldnt access catalogue
                     5 = illegal OP code
```

5.8.11 MAPOPN - opens a map file marking the catalogue 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	Catalogue slot containing map
CATBLK(256)	I*2	Buffer containing current catalogue 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 = Catalogue i/o error 5 = Illegal OP code 6 = Can't open file
Buffer:		
WBUFF(256)	I*2	Working buffer for CATIO and CATDIR

5.8.12 ROTFND - finds the map rotation angle from a given catalogue block

```
    ROTFND (CAT4, ROT, IERR)
```

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

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

```
      SETLOC (DEPTH)

   Inputs: DEPTH   I*2(5)   Position of map plane axes 3 - 7
   Common: /MAPHDR/ catalogue block (not modified)
           /LOCATI/ position parms - created here
```

5.8.14 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
```

5.8.15 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 catalogue header record. 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) and stokes parameter (ICOR) is given by :

```
     NRPARM+(CHAN-1)*INCF+(ICOR-IABS (ICOR0))*INCS

      SUBROUTINE UVPGET (IERR)
   Inputs: From common /MAPHDR/
      CATBLK(256)    I*2  Catalogue 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
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.	
INCS	I*2	Increment in data for stokes (see above)	
INCF	I*2	Increment in data for freq. (see above)	
ICOR0	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(2)	P 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 Stokes' parameters	
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	

5.8.16 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)
```

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

5.8.17 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)
```

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)

COMMON Inputs:
/LOCATI/ position parms deduced from the map header by subroutine SETLOC

Units are as in the mapheader: degrees for position coordinates.

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 catalogued 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 auxilliary 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 Volumn 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 incore 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 (EXTIO) returns the data in an array which allows easy implementation of data structures.

This chapter discusses the various aspects of disk files, creating, destroying, reading, writing etc. The cataloguing 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 MANAGMENT

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

  The routine MCREAT creates and catalogues an image file using the description of the file contained in a catalogue 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 catalogue header record is described in detail in another chapter.

- UVCREA

  The routine UVCREA creates and catalogues a uv data file using the description of the file contained in the catalogue header record passed to UVCREA in the common /MAPHDR/. The catalogue header record must be sufficiently complete to determine the name, class, etc and size of the required file.

- SNCRC

  The routine SNCRC 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. Use of SNCRC is described in more detail in the chapter describing tasks.

- EXTINI

  The creation of most extension files is hidden from the casual programmer in the create/open/initialize routine EXTINI. EXTINI will be discussed in more detail in the section in this chapter on I/O to extension files.

- 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 catalogue the file created.

6.3.2 Example Using ZCREAT

The use of ZCREAT is demonstrated in the following:

```
      INTEGER*2 NBYTE(2), SYM, IRET, NX, NY, NP(2), BP, N2
      LOGICAL*2 MAP
      REAL*4    R8TOP4, 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 = catalogue slot number
C                                           (arbitrary for
C                                           uncatalogued files).
C                                       IVER = extension file
C                                           version number.
C                                           1 for main catalogued
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 catalogued, then a catalogue header record should be constructed and call made to

CATDIR and CATIO before the call to ZCREAT to get the catalogue 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 catalogue 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 catalogue 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 catalogue 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 catalogued 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 uncatalogue the file destroyed. CATDIR should be used to uncatalogue a catalogue 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 EXTIO 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, catalogue slot number, version number and user ID number. The file type of image files is 'MA' and of uv data files is 'UV'. The disk number and catalogue slot number for catalogued 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 SNCRC, EXTINI 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 EXTINI 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 transfered 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 bookeekping. 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.

- EXTINI opens and initializes an extension file, will create and catalogue the extension file if necessary.

- EXTIO does random access mixed reads and writes to extension files. EXTIO deals with one logical record at a time in an array which can be used as a data structure. EXTIO takes care of file expansion and other bookkeeping chores. Requires initialization by EXTINI.

- MAPOPN finds a catalogued image or uv data file in the catalogue, opens it and returns the catalogue header and marks the catalogue status.

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

- MDISK does double buffered I/O for image files; requires initialization by MINIT.

- UVINIT initializes I/O for uv data files; can specify a starting visibility record number.

- UVDISK does double buffered I/O for uv data files; requires initialization by UVINIT.

- MAPCLS closes a catalogued image or uv data file, updates the catalogue header block if requested and clears the catalogue 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, ie. 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). VAX does Fortran opens, Modcomp allocates these at task build time.
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. Modcomp does not use this value.
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	Catalogue files.
16 - 25	Map (image or uv data) files.
26	Graphics files
27 - 30	General (non-map) disk files.
31 - 32	Magnetic tape drives.

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.

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	# 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
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
XPRDMM	R*4	Printer points per millimeter
XTKDMM	R*4	Graphics points per millimeter
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 catalogue size (0=>public)
BYTFLP	I*2	Byte flip, 0=none, 1=bytes, 2=words
SYSNAM	R*4(5)	System name (20 char)
VERNAM	R*4	Version ID (4 char)
USELIM	I*2	Maximum user number
IFILIT	I*2	Spare
RLSNAM	R*4(2)	Release name (8 characters)
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 catalogue header block, the first axis going the fastest. The pixels may be one of several types, but in practice, they are either scaled short integers or floating point values. Blanking of pixels is allowed by use of a special value (magic value blanking) specified by the header. For more information about the catalogue header and the typical axes used see the chapter on the catalogue.

Image files are stored on the disk with each row beginning on a block boundry. 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 boundries.

6.4.4.1 Opening Image Files - The simplest way to find, open and close a catalogued image file is with the routines MAPOPN and MAPCLS. These routines and the alternate ways to find an image in the catalogue are discussed in the chapter on the catalogue 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 = catalogue slot number
C                                            (arbitrary for
C                                            uncatalogued files).
C                                        IVER = extension file
C                                            version number.
C                                            1 for main catalogued
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 MINIT And MDISK - Once the image file is opened, I/O is normally initialized by a call to MINIT, I/O is done by calls to MDISK with a final call to MDISK to flush the buffer if necessary. MINIT sets up the bookkeeping for one plane of an image at a time; if multiple planes are to be read, multiple calls to MINIT must be made. A rectangular window in a given plane can be specified to MINIT, and it can be instructed to read or write the rows in reverse order by reversing the values of WIN(2) and WIN(4). A subimage cannot be specified for write.

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

6.4.4.3 Multi-plane Images (COMOFF) - If the image has more than two dimensions, planes parallel to the first plane can be accessed using the block offset argument to MINIT. The subroutine COMOFF can be used to compute the block offset. The block offset is a pseudo I*4 number whose value for the first plane is (1,0). COMOFF returns a value which is to be added to the block offset for the first plane. An example of the use of COMOFF to compute the block offset:

```
      INTEGER*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 COMOFF (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 MINIT to get the
C                                        specified plane.
```

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

6.4.4.4 Example Of MINIT And MDISK - In the following is an example in which two files are read, the pixel values are added and a third file is written.

```
      SUBROUTINE FLADD (NX, NY, ISCR1, ISCR2, ISCR3, IERR)
C-----------------------------------------------------------------------
C   FLADD adds the values in the scratch files in the /CFILES/ common
C   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
      LOGICAL*2 T,F
      REAL*4 READ, WRITE, FINI
      REAL*4 BUFF1(1), BUFF2(1), BUFF3(1)
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CFIL.INC'
C                                        Buffer common from rest of
C                                        program.
      COMMON /BUFRS/ BUFF1, BUFF2, BUFF3, BUFSZ1, BUFSZ2, BUFSZ3
      DATA T, F /.TRUE.,.FALSE./, READ, WRITE, FINI
     *   /'READ','WRIT','FINI'/, BO /1,0/, N1, N2, N8 /1,2,8/,
C                                        Use LUNs 16, 17, 18
     *   WIN /4*0/,  LUN1, LUN2, LUN3 /16,17,18/
C-----------------------------------------------------------------------
```

```
C                                          Set bytes per pixel (floating)
      BP = 2 * NWDPFP
C                                          Open and init ISCR1
      CALL ZOPEN (LUN1, FIND1, SCRFIL(ISCR1),
     *   SCRFIL(1,ISCR), 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 MINIT (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 ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), SCRFIL(1,ISCR2),
     *   T, F, T, IERR)
C                                          Check for error
      IF (IERR.EQ.0) GO TO 30
         ENCODE (80,1000,MSGTXT) IERR, READ, N2
         GO TO 990
 30   CALL MINIT (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 ZOPEN (LUN3, FIND3, SCRVOL(ISCR3), SCRFIL(1,ISCR3),
     *   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 MINIT (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 MDISK (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 MDISK (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 MDISK (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 MDISK (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 occured - 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 MINSK And MSKIP - There are some operations such as transposing images in which it is convenient to read every n th row of an image. The pair of routines MINSK and MSKIP will do this operation. Descriptions of these routines can be found at the end of this chapter.

6.4.5 Image File Manipulation Routines

There are a number of AIPS utility routines available to operate on files. Many of these involve copying data from catalogue files to scratch files or vice versa with or without various format conversions. An important member of this class is the FFT routine. Details of the call sequences to these routines are given at the end of this chapter.

- CONVRT (convert) will convert a R*4 map into an I*2 map or an I*2 map into an R*4 map depending upon the type of the input map.

- DSKFFT is a disk based, two dimensional FFT.

- MSCALE will read from a floating point file, rescale the values to correspond to the maximum and minimum, and write these scaled values to an integer format map file.

- MSCALF is like MSCALE but will handle blanked pixels.

- MSCALI will copy the values in an integer file to a floating point map file.

- PLNGET reads a selected portion of a selected plane from a catalogued 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 REAL*4 scratch file image into a catalogued image (either I*2 or R*4).

6.4.6 Uv Data Files

6.4.6.1 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 and a five day offset is added to the time parameter for each subarray to further distinguish between subarrays.

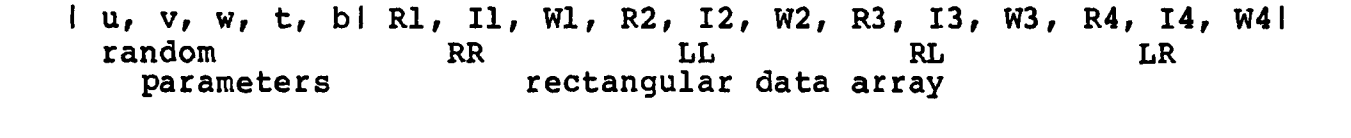
6.4.6.2 Visibility Record Structure - AIPS uv data is organized in the data file in the same way that similar data is organized of a FITS format tape. Each logical record consists of all data on a given baseline for a given integration period; that is all polarizations and frequencies are contained in 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 catalogue 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 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 provides a place to store the values for these parameters.

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



The symbols in the above are:

- u = u coordinate in wavelengths at the reference frequency
- v = u coordinate
- w = w coordinate
- t = time in days since reference data given in antenna file for this subarray. The time is 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.

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 may be changed in a file; 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 catalogue 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 most 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.

6.4.6.3 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 catalogue 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) and stokes parameter (ICOR) is given by :

```
NRPARM + (CHAN-1) * INCF + (ICOR-IABS (ICOR0)) * INCS
```

6.4.6.4 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 polarizarion (RCP, LCP). They may be requested to work on multiple frequency channels.

- 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.

6.4.6.5 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 MINIT and MDISK. One signifigant difference between UVDISK and MDISK is that UVDISK can be requested to process multiple logical records in a single call. 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 is that, unlike MINIT, UVINIT returns the buffer pointer for the first call so the output buffer can be written into before the first call to UVDISK.

UVINIT sets up the bookkeeping for UVDISK which does double buffered (if possible) quick return I/O. UVDISK will run much more efficiently if on disk the requested 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 non-tape 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 tape I/O, data is always written with the block size specified to UVINIT one I/O operation per call. For disk 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 MDISK with opcode 'FINI'.

The input/output argument to UVDISK, NIO, can be very useful for controling 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.

6.4.6.6 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  P   I*4  Number of visibility records
C    LREC      I*2  logical record length.
C    NRPARM    I*2  number of random parameters.
C    ICOR0     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    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   Inputs from common /MAPHDR/
C    CATBLK(256)  I*2  Catalogue 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, BO(2), BP, NFREQ, NSTOKE,
     *   WIN(4), NX, NY, BUFSZ1, BUFSZ2, LUN1, LUN2, I, IV, IFQ, IST,
     *   VO(2), NIOIN, NIOUT, CATBLK(256), INDEX, JCOR0
      LOGICAL*2 T,F
      REAL*4 READ, WRITE, FLUSH
      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                                          Catalogue header block common
```

```
      COMMON /MAPHDR/ CATBLK
      DATA T, F /.TRUE.,.FALSE./, READ, WRITE, FLUSH,  VO /0,0/,
     *   /'READ','WRIT','FLSH'/, BO /1,0/, N1, N2, N8 /1,2,8/,
     *   WIN /4*0/
C-----------------------------------------------------------------------
C                                          Set bytes per pixel (floating)
      BP = 2 * NWDPFP
C                                          Take absolute value of first
C                                          Stokes' value.
      JCOR0 = IABS (ICOR0)
C                                          Find dimension of freq
C                                          and stokes axes.
      NFREQ = CATBLK(K2NAX+JLOCF)
      NSTOKE = CATBLK(K2NAX+JLOCS)
C                                           Open and init ISCR1
      CALL ZOPEN (LUN1, FIND1, SCRFIL(ISCR1),
     *   SCRFIL(1,ISCR), 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                                          Read 8 record at a call.
 10   NIOIN = 8
      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 ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), SCRFIL(1,ISCR2),
     *   T, F, T, IERR)
C                                           Check for error
      IF (IERR.EQ.0) GO TO 30
         ENCODE (80,1000,MSGTXT) IERR, READ, N2
         GO TO 990
 30   NIOUT = 8
      CALL UVINIT (WRITE, LUN2, FIND2, NVIS, VO, LREC, NIOUT,
     *   BUFSZ2, BUFF2, BO, BP, BIND2, IERR)
     *   BP, BO, IERR)
C                                           Check for error
      IF (IERR.EQ.0) GO TO 60
         ENCODE (80,1010,MSGTXT) IERR, WRITE, N2
         GO TO 990
C                                          Loop thru 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 thru records
         DO 100 IV = 1,NIOIN
C                                          Loop through frequency
```

```
            DO 90 IFQ = 1,NFREQ
C                                           Loop through stokes' axes
               DO 80 IST = 1,NSTOKE
C                                           Compute pointer in the
C                                           buffer to imag. part
                  INDEX = NRPARM + (IFQ-1) * INCF +
     *               (IST-JCOR0) * 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
  100       CONTINUE
C                                            Write output
         CALL UVDISK (WRITE, LUN2, FIND2, BUFF2, BIND2, NIOUT, IERR)
C                                            Check for error
         IF (IERR.EQ.0) GO TO 110
            ENCODE (80,1060,MSGTXT) IERR, WRITE
            GO TO 990
C                                           Loop back for more data
  110    GO TO 60
C                                           Finished, flush buffer.
C                                           No more output records.
  120  NIOUT = 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 volumn 2 for most types of extension files.

The routines EXTINI and EXTIO make I/O to extension files much simpler than the image and uv data routines. A single call to EXTINI will create an extension file if necessary, catalogue 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 statment or through the use of a common. EXTIO will automatically expand the file when necessary in increments of the number of records specified to EXTINI.

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.

An example of the use of EXTINI and EXTIO is demonstrated in the following example. In this example an antenna file (type 'AN') is read and an offset (XOFF, YOFF, znd ZOFF) is added to the coordinates for each antenna. This example uses the INCLUDEs DANT.INC and CANT.INC which use a common to form a data structure to hold the antenna records. Details of the structure of the antenna file records are found in the AIPS manual volumn 2.

```
      INTEGER*2 AN, VOL, CNO, IANT, CATBLK, LUN, FIND, LRECL, BP, NREC,
     *   BUFFER(512), VER, NANT
      REAL*4    READ, WRITE, CLOSE
      REAL*8    XOFF, YOFF, ZOFF
      INCLUDE 'INCS:DANT.INC'
C                                        INCLUDE filled in for this
C                                        example
C                                                        Include DANT
       INTEGER*2 NOSTA,MNTSTA
       REAL*4  STANAM(2),STAXOF,ANTSP1(18),POLTYA,POLAA,
     *   AMPA,POLA1,POLA2,POLA3,ANTSP2(7),POLTYB,POLAB,AMPB,POLB1,
     *   POLB2,POLB3,ANTSP3(7),BPFRA,BPFRB
       REAL*8 STABX,STABY,STABZ,CLKIFA,LOIFA,CLKIFB,LOIFB
C                                                        End DANT
                .
                .
      INCLUDE 'INCS:CANT.INC'
C                                        INCLUDE filled in for this
C                                        example
C                                                        Include CANT
```

```
      COMMON /ANTCOM/ STANAM, STABX, STABY, STABZ, NOSTA, MNTSTA,
     *   STAXOF, ANTSP1, CLKIFA, LOIFA, BPFRA, POLTYA, POLAA, AMPA,
     *   POLA1, POLA2, POLA3, ANTSP2, CLKIFB, LOIFB, BPFRB, POLTYB,
     *   POLAB, AMPB, POLB1, POLB2, POLB3, ANTSP3
C                                                              End CANT
      COMMON /MAPHDR/ CATBLK
      DATA AN /'AN'/, READ, WRITE, CLOSE /'READ','WRIT','CLOS'/
                .
                .
C                                        Setup for EXTINI
      NREC = 1
C                                        EXTINI will fill in LRECL
C                                        and BP if initially 0.
      LRECL = 0
      BP = 0
C                                        Use first AN file.
      VER = 1
C                                        Open 'WRIT' for mixed reads
C                                        and writes.
      CALL EXTINI (WRITE, AN, VOL, CNO, VER, CATBLK, LUN, FIND,
     *   LRECL, BP, NREC, BUFFER, IERR)
C                                        Check for error
      IF (IERR.NE.0) GO TO 999
C                                        Get number of records from
C                                        word 4 of the file header.
C                                        (this is the no. of antennas)
      NANT = BUFFER(4)
C                                        Loop through records.
      DO 100 IANT = 1,NANT
C                                        read record.
         CALL EXTIO (READ, LUN, FIND, IANT, STANAM, BUFFER, IERR)
C                                         Check for error
         IF (IERR.NE.0) GO TO 999
C                                        Offset antenna positions.
         STABX = STABX + XOFF
         STABY = STABY + YOFF
         STABZ = STABZ + ZOFF
C                                        Write record back
         CALL EXTIO (WRITE, LUN, FIND, IANT, STANAM, BUFFER, IERR)
C                                         Check for error
         IF (IERR.NE.0) GO TO 999
 100     CONTINUE
C                                        Close file.
      CALL EXTIO (CLOSE, LUN, FIND, IANT, STANAM, BUFFER, IERR)
C                                         Check for error
      IF (IERR.NE.0) GO TO 999
```

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 that 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. These routines can access both disk and tape drives.

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 bookeeping information(the first always contains the value of the LUN). Examples of the use of the FTAB are found in MINIT, MDISK, MINSK, MSKIP, UVINIT and UVDISK which use ZMIO and ZWAIT. Descriptions of the way these routines use the FTAB are to be found at the end of this chapter. A description of the call arguments to ZMIO and ZFIO 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. ZFIO can be used on "map" (image and uv data files) but it is much less efficient than ZMIO and ZWAIT. Routines using ZFIO can use the 16 words in the FTAB allocated to the file as was described for routines using ZMIO.

For non-disk transfers, the number of bytes transfered 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 EXTINI and EXTIO.

6.6 ROUTINES

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

```
   COMOFF (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
```

6.6.2 CONVRT - Changes an I*2 file into a R*4 file or vice versa.

```
   CONVRT (ILUN, ILUN2, ISLOT, IVOL, IOSIZ, IOBLK, ROSIZ,
  *   ROBLK, IERR)
Inputs:
   ILUN    I*2  LUN for closed map file.
   ILUN2   I*2  LUN for a scratch map file.
   ISLOT   I*2  Catalogue slot number for map.
   IVOL    I*2  Disk volume number of map.
In/Out:
   IOSIZ   I*2  Size in 'bytes' (one half integer words) of IOBLK.
   IOBLK   I*2(IOSIZ/2)  Scratch integer I/O buffer.
   ROSIZ   I*2  Size of output buffer in 'bytes'
   ROBLK   R*4(ROSIZE/(2*NWDPFP))  Scratch I/O buffer.
   COMMON /MAPHDR/  Current map header.  The header is updated to
                    reflect changes made by this program.
Output:
   IERR    I*2  Error code. 1=warning, could not destroy old map.
                2=error converting map. 3=map left uncatalogued.
                4=map not real or integer. Map unchanged.
```

6.6.3 DSKFFT - a disk based, two dimensional FFT. The data are stored by rows. To save an extra transposition, the input array is assumed to have been written in transposed order and in the "center at the corners" convention for the uv to sky transform.

```
   DSKFFT (NR, NC, IDIR, HERM, LI, LW, LO, LENBUF, NBUF,
  *   APSIZ, 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. (forward = sky=>uv, reverse = uv=>sky)
                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 .ge. 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. (may be LI if
                LW isn't, may NOT be LW)
    LENBUF  I*2 Buffer length in bytes.  LENBUF must be a power of
                two, and the buffer must be long enough to hold a
                row of the input array and to hold a row of the
                output array.
                The buffers are passed in COMMON /BUFRS/BUF(*).
    NBUF    I*2 Number of buffers: either 1 or 2.
    APSIZ   R*4 Size of AP main data memory in words.
   Output:
    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.
NOTE: DSKFFT also uses Commons /BUFRS/ and /CFILES/
```

6.6.4 EXTINI - creates/opens an extension file. If a file is created it is catalogued 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 (eg. 'CC')
    VOL             I*2   Volumn number
    CNO             I*2   Catalogue slot number
    VER             I*2   Version number: (<= 0 => write a new one,
                          read the latest one)
    CATBLK(256)     I*2   Catalogue block of catalogued 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 extent
                          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	Catalogue 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.

Useage notes:
For sequential access, EXTINI leaves pointers for EXTIO such that if IRNO .le. 0 reads will begin at the start of the file and writes will begin after the last previous record.
File should be marked 'WRIT' in the catalogue 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	Volumn 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 files. 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 GETlVS - gets and reformats uv data. Returns one stokes' type per frequency channel. Requires setup by SETlVS.

```
      GETlVS (MODE, MVIS, JADR, JINC, SFACT, ALLWT, STOKES,
     *   DATA, WT, VIS, IRET)
```

Inputs:		
MODE	I*2	Operation number (see SETlVS). 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 SETlVS.
JINC	I*2	Increment between vis.
SFACT(2)	R*4	Factors set by SETlVS.
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 cards 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.)

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

Inputs:		
KEYS(N)	R*8	array of parameter names (packed characters)
VALUES(N)	R*8	array to receive values or defaults
N	I*2	number of parameters (dimension of keys and values)
ENDMRK	R*8	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 ENDMARK, ignore any keywords.
LUN	I*2	LUN to read from (used in call to ZTOPEN)
FIND	I*2	FTAB pointer for input. (from ZTOPEN)
Outputs:		
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*2(2)    Pseudo-I*4 file size in bytes
```

6.6.10 MAPCLS - closes a map file and clears the catalogue 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    Catalogue slot number of file
      LUN      I*2    Logical unit # used for file
      IND      I*2    FTAB pointer for LUN
      CATBLK   I*2(256) New catalogue header which can optionally
                         be written into header if OP=WRIT or INIT
                         Dummy arguement if OP=READ
      CATUP    L*2       If TRUE write CATBLK into catalogue,
                         ignored if OP = READ
   Outputs:
      IERR      I*2   0 = O.K.
                      1 = CATDIR couldn't access catalogue
                      5 = illegal OP code
```

6.6.11 MAPOPN - opens a map file marking the catalogue 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   Catalogue slot containing map
  CATBLK(256)I*2     Buffer containing current catalogue 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 = Catalogue 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 catalogues an image data file based on a catalogue 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      Catalogue slot number
   IERR         I*2      Error code; 0 => o.k.
                                     1 => couldnt create,no room
                                     2 => no create, duplicate name
                                     3 => no room in catalogue
                                     4 => I/O problem on catalogue
                                     5 => Other Create errors
Common: (in/out)
   CATBLK       I*2(256) Catalogue block (via common MAPHDR)
   CATB4        R*4(128) Catalogue block (equivalenced to CATBLK)
```

The file created will be catalogued 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 catalogue 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 catalogue 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 MDISK - reads or writes image data to/from disks and other devices.

```
   MDISK (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

MDISK sets array index to the start of the next line wanted.
NOTE: the line sequence is set by the WIN parameter in MINIT,
if the vaules 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.
MINIT MUST be called before MDISK.
```

6.6.15 MINIT - initialized the I/O tables for MDISK.

```
   MINIT (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 COMOFF + 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
```

MINIT 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: i.e. 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 transfered per I/O op
          2 =>  No. of major times a buffer has been acessed
          3 =>  No. of major lines remaining on disk
          4 =>  Output index for first pixel in window
          5 =>  No. pixels to increment for next major line
          6 =>  Which buffer to use for I/O; -1 => single buffer
          7 =>  Block offset in file for next operation (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 MINSK - initializes the I/O tables for the "scatter read" I/O routine MSKIP.

```
      SUBROUTINE MINSK (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.
                  MSKIP must be called this number of times to c
                  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 assigments:
             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 MSKIP is called)
            10 = bytes / call
            11 = buffer flag, -1= single, 1=>current buffer is 1
                 2=>current buffer=2 (buffer already read)
            12 = buffer size in pixels (1/2 for double buffering)
            13 = NROW (the number of rows to read)
            14 = BTYOFF the byte offset when double buffering.
```

6.6.17 MSCALE - will read from a floating point file, rescale the values to correspond to a max and min and write these scaled values to an integer format map file. The two files must be open before this routine is called.

```
     MSCALE (XMIN, XMAX, NAX, INP, IBLC, ITRC, ISLUN, ISIND,
    *   ISBSIZ, XBLK, IDLUN, IDIND, IDBSIZ, IDBLK, SCALEF, OFFSET,
    *   IERR)
  In/out:   XMIN     R*4 actual minimum value of floating pt values.
            XMAX     R*4 actual maximum value.
                         Input: used to determine scaling & must
                         encompass full data range.  If subimaging
                         is done, output will be actual in subim.
  Inputs:   NAX      I*2 the number of axes
            INP      I*2(7) # points on each axis (input file)
            IBLC     I*2(7) start point for input axes:
                            IBLC(1) = 0  => use full array
```

	ITRC	I*2(7) end point on each axis of input array
	ISLUN	I*2 the logical unit number of the source file.
	ISIND	I*2 the FTAB pointer for the source file.
	ISBSIZ	I*2 the buffer size in bytes for the source file
	XBLK	R*4(ISBSIZ/4) the source file I/O buffer.
	IDLUN	I*2 the logical unit number for the I*2 destination file.
	IDIND	I*2 the FTAB pointer for the destination file.
	IDBSIZ	I*2 buffer size in bytes for the destination fil
	IDBLK	I*2(IDBSIZ/2) destination file I/O buffer.
Outputs:	SCALEF	R*8 the scale factor used to scale the dest file
	OFFSET	R*8 offset used in scaling destination file.
	IERR	I*2 error indicator. 0 = no error.

6.6.18 MSCALF - will read from a floating point file, rescale the values to correspond to a max and min and write these scaled values to an integer format map file. The two files must be open before this routine is called. MSCALF is similar to MSCALE except that blanking capability is included. FBLANK is the value of the undefined pixel in the floating point scratch array. This pixel is set to -32768 in the integer format file.

```
      MSCALF (XMIN, XMAX, NAX, INP, IBLC, ITRC, ISLUN, ISIND,
     *    ISBSIZ, XBLK, IDLUN, IDIND, IDBSIZ, IDBLK, FBLANK, SCALEF,
     *    OFFSET, IERR)
```

In/Out:	XMIN	R*4 actual minimum value of floating pt values.
	XMAX	R*4 actual maximum value. Input: used to determine scaling & must encompass full data range. If subimaging is done, output will be actual in subim.
Inputs:	NAX	I*2 the number of axes
	INP	I*2(7) # points on each axis (input file)
	IBLC	I*2(7) start point for input axes: IBLC(1) = 0 => use full array
	ITRC	I*2(7) end point on each axis of input array
	ISLUN	I*2 the logical unit number of the source file.
	ISIND	I*2 the FTAB pointer for the source file.
	ISBSIZ	I*2 the buffer size in bytes for the source file
	XBLK	R*4(ISBSIZ/4) the source file I/O buffer.
	IDLUN	I*2 the logical unit number for the I*2 destination file.
	IDIND	I*2 the FTAB pointer for the destination file.
	IDBSIZ	I*2 buffer size in bytes for the destination fil
	IDBLK	I*2(IDBSIZ/2) destination file I/O buffer.
	FBLANK	R*4 the value of the floating point blank pixel
Outputs:	SCALEF	R*8 the scale factor used to scale the dest file
	OFFSET	R*8 offset used in scaling destination file.
	IERR	I*2 error indicator. 0 = no error.

6.6.19 MSCALI - will read from an integer file, and write these values to a floating point map file. The two files must be open before this routine is called.

```
      MSCALI (SCALEF, OFFSET, NAX, INP, IBLC, ITRC, ISLUN,
     *   ISIND, ISBSIZ, ISBLK, IDLUN, IDIND, IDBSIZ, XBLK, FBLANK,
     *   XMIN, XMAX, IERR)
   Inputs:   SCALEF   R*4 SCALEF * Pixel value + OFFSET = actual
                      pixel value.
             OFFSET   R*4 See OFFSET.
             NAX      I*2 the number of axes
             INP      I*2(7) # points on each axis (input file)
             IBLC     I*2(7) start point for input axes:
                             IBLC(1) = 0  => use full array
             ITRC     I*2(7) end point on each axis of input array
             ISLUN    I*2 the logical unit number of the source file.
             ISIND    I*2 the FTAB pointer for the source file.
             ISBSIZ   I*2 the buffer size in bytes for the source file
             ISBLK    I*2(ISBSIZ/2) the source file I/O buffer.
             IDLUN    I*2 the logical unit number for the I*2
                      destination file.
             IDIND    I*2 the FTAB pointer for the destination file.
             IDBSIZ   I*2 buffer size in bytes for the destination fil
             XBLK     R*4(IDBSIZ/4) destination file I/O buffer.
             FBLANK   R*4 the value of the floating point blank pixel
   In/Out:   XMIN     R*4 Min map value: output changed only if do
                          subarray
             XMAX     R*4 Max map value: ditto
   Outputs:  IERR     I*2 error indicator.
                          0 = no error.
```

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

```
      SUBROUTINE MSKIP (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.

     MINSK MUST be called before MSKIP.
```

6.6.21 PLNGET - reads a selected portion of a selected plane from a catalogued 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 of 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 catalogue 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.

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

6.6.22 PLNPUT - writes a subregion of a REAL*4 scratch file image into a catalogued 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 catalogue 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 catalogue
                        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 catalogue it must be updated
        by a call to CATIO etc. before calling this routine.
```

6.6.23 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.

```
      SETVIS (MODE, NCH, 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.
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.

6.6.24 SETlVS - 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 GETlVS. There is also a check to make sure the desired data is available. Calls to GETlVS will reformat the data. Needs values set by UVPGET.

SETlVS (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.

6.6.25 UVCREA - creates and catalogues a uv data file using the catalogue 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      Catalogue slot number
   IERR       I*2      Error code; 0 => o.k.
                                   1 => couldnt create,no room
                                   2 => no create, duplicate name
                                   3 => no room in catalogue
                                   4 => I/O problem on catalogue
                                   5 => Other Create errors
COMMON: /MAPHDR/ catalogue block used a lot, final seq # on output
```

6.6.26 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 tape I/O data is always written with the block size specified to UVINIT; one I/O operation per call. For disk 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 MDISK with opcode 'FINI'.

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

```
    UVDISK (OP, LUN, FIND, BUFFER, NIO, BIND, IERR)
```

Inputs:

OP	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 visibilites 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.

6.6.27 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 non tape 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	P	I*4	Total number of visibilities to read. NVIS+VISOFF must be no greater than the total number in the file.
VISOFF	P	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. Determines block size for tape I/O
BUFSZ		I*2	Size in bytes of the buffer. If 32767 given, 32768 is assumed.

```
   BUFFER( )  I*2  Buffer
   BO       P 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.
                 For double buffer read, 1 more I/O will have
                 been done than indicated.
          4-5  = Block offset for next I/O.
            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 - 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 catalogue header record. 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) and stokes parameter (ICOR) is given by :

```
     NRPARM+(CHAN-1)*INCF+(ICOR-IABS (ICOR0))*INCS

      SUBROUTINE UVPGET (IERR)
   Inputs: From common /MAPHDR/
      CATBLK(256)    I*2  Catalogue 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
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.
INCS		I*2	Increment in data for stokes (see above)
INCF		I*2	Increment in data for freq. (see above)
ICOR0		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(2)	P	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.

```
      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 - All types of AIPS data files may be compressed by the subroutine ZCMPRS. The file must be open before the call to ZCMPRS. Note: it is dangerous to compress files written by EXTIO 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 EXTINI in the section on extension file I/O in this chapter.

```
   ZCMPRS (IVOL, PNAME, ISIZE, LSIZE, LUN, IERR)
Inputs:  IVOL   I*2      volume number
         PNAME  I*2(12)  physical file name
         ISIZE  I*2(2)   original size bytes pseudo I*4
         LSIZE  I*2(2)   desired final size bytes pseudo I*4
         LUN    I*2      logical unit number under which file is
                         open
Outputs: IERR   I*2      error code: 0 => ok
                                     1 => input data error
                                     2 => compress error FMGR
```

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.

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

```
   ZCREAT (IVOL, PHNAME, NBYTE, MAP, IERR)
Inputs:
   IVOL     I*2       Disk drive unit number.
   PHNAME   R*4(6)    Physical file name given by ZPHFIL.(char.)
                      left justified, padded with blanks.
   NBYTE P  I*4       Size of the file in bytes. Will be rounded to
                      next higher 512 byte block boundary.
                      NOTE: this is a "Pseudo I*4".
   MAP      L*2       .TRUE. if map file, .FALSE. otherwise.
Outputs:
   IERR     I*2       Error return code. I*2.  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 512 bytes (a byte being defined as half of a short integer). NREC gives the random access record number (in units of 512 bytes). For non-disk devices, NREC contains the number of bytes.

```
      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*2  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 -> I/O error
                                   4 -> end of file
                                   5 -> begin of medium
                                   6 -> end of medium
```

6.6.35 ZMIO - a low level random access, large record, double buffered device I/O.

```
      ZMIO (OP, LUN, FIND, BLKNO, NBYTES, BUFF, IBUFF, IERR)
   Inputs:
      OP       R*4  Operation - 'READ', 'WRIT'. ASCII - 4 characters.
      LUN      I*2  Logical unit number of a previously opened map.
      FIND     I*2  Pointer to FTAB returned by ZOPEN.
      BLKNO P  I*2  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 for which LUN > NDEVT. Tape units are assigned an I/O channel and given an FTAB entry for double buffering.

```
      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 - construct a physical file name in PNAM from ITYPE, IVOL, NSEQ, and IVER. New version designed either for public data files or user specific files. This routine contains the logical assignment list for Graphics devices and is thus site dependent as well as machine dependent.

```
EXAMPLE: If ITYPE='MA', IVOL=8, NSEQ=321, IVER=99, NLUSER=762 then
            PNAME='DA07:MA832199;1' for public data or
            PNAME='DA07:MA832199.762;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  Number of the disk volume to be used.
   NSEQ     I*2  User supplied sequence number. 000-999.
                 This is the catalogue slot number for catalogued
                 files.
   IVER     I*2  User suppplied version number. 00-255.
                 Always 1 for map or uv data files.
Outputs:
   PNAM  R*4(6)  >= 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 (NOT USED ON VAX).
            PNAME     R*4(6) disk-file type.  Only type ('HE' ect)
                      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 not used with VAX.
   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

```
      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
			7 => wait service error

CHAPTER 7

DEVICES

7.1 OVERVIEW

Programs in the AIPS system occasionally need to talk to peripheral devices. This chapter discusses such devices other than disk drives, TV displays, array processors, and plotters which are covered elsewhere. Many of the same routines used for disk I/O are also used for I/O to other devices but their use may be modified to suit the physical properties of the particular device. The details of the call sequence for the relevant routines discussed in this chapter are given at the end of the chapter.

7.2 TAPE DRIVES

Tapes are used in AIPS primarily for long term storage of data, images or text files. The principle differences in the AIPS system between use of tape and disk is that tapes, by their physical nature, are sequential access devices and the physical block size of data depends on the program writting the tape. In addition, AIPS batch jobs are forbidden to talk to tape drives.

The usual problems of Fortran I/O apply to tapes, i.e. it is not predictable from one machine and/or operating system to another. For this reason standard AIPS programs do not use Fortran I/O for tapes. Also, some versions of Fortran cannot read or write some file structures such as those containing variable length, blocked, unspanned records.

Since AIPS tasks work directly from I/O buffers a program using tape must take account of the details of the way data is written on tape. One exception to this is writing variable length, blocked, but unspanned records; such records may be assembled and written using the AIPS utility routine VBOUT.

7.2.1 Opening Tape Files

Tape files are opened using ZOPEN in a way similar to disk files. Details about ZOPEN and examples of its use can be found in the chapter on disk I/O. However, to tell the AIPS routines that the file is on a tape drive and to specify which tape drive, the LUN and file name are different from those used for disk files. The LUN for tape files must be 31 or 32. When constructing the name of the file using ZPFIL use 'MT' as the file type and the (one relative) tape drive number as the volumn number, the rest of the values sent to ZPHFIL are ignored by ZOPEN and are arbitrary.

7.2.2 Positioning Tapes

Once the file has been opened in AIPS the tape may be positioned, mounted or dismounted, or EOFs may be written using ZTAPE. NOTE: mounting and dismounting are generally done only by the AIPS program itself. Details of the call sequence to ZTAPE are given at the end of this chapter. The following list gives the opcodes recognized by ZTAPE.

1. 'ADVF' = advance file marks
2. 'ADVR' = advance records
3. 'BAKF' = backspace file marks.
4. 'BAKR' = backspace records.
5. 'DMNT' = dismount tape.
6. 'MONT' = mount tape.
7. 'REWI' = rewind the tape on unit LUN
8. 'WEOF' = write end of file on unit LUN: writes 4 EOFs, positions tape after first one
9. 'MEOF' = write 4 EOF marks on tape, position tape before the first one

7.2.3 I/O To Tape Files

The same routines to write to disk files can be used to talk to tape files although several call arguments have altered meanings for tape files.

7.2.3.1 MINIT/MDISK And UVINIT/UVDISK - Double buffered I/O can be done using MINIT/MDISK and UVINIT/UVDISK. For these pairs of routines the primary difference between their use on disk and tape is that the physical blocks on the tape are: 1) a single logical record of an image (a row, or the first dimension) if written using MINIT/MDISK or 2) the number of logical records (visibilities) requested in a single call (NPIO) to UVINIT. Since these routines know or care little about the internal structure of the data read or written, in practice, any format records can be processed.

7.2.3.2 ZFIO - Single buffered I/O can be done using ZFIO but the input variable used for to block number becomes the byte count for the transfer.

7.2.3.3 VBOUT - The utility routine VBOUT will collect variable length records and block them, unspanned, into IBM format physical blocks up to 4008 bytes long. The tape must be opened with ZOPEN as a non-map file. The principle use of this routine is to write VLA "EXPORT" format tapes. Details of the call sequence as well as other important useage notes are found at teh end of this chapter.

7.2.4 Tape Data Structure

In order to make efficient use of tape storage a number of logical records may be grouped into a single physical record. In general these logical records may be fixed or variable length and may or may not span physical blocks. In addition, logical records may be formatted (text, usually ASCII) or binary. Such details need to be determined before attempting to read or write such files.

Fixed length logical records are packed into physical records as defined by the record size and block length. Since the order and size of these records is well defined there is no need for additional control information.

For variable length logical records, control bytes are added to the record to determine the boundaries of logical records. Unfortunately, the details of the of variable length record structure varies from computer to computer and from operating system to operating system. If you wish to read or write one of these tapes you have to find the details of the formats for the machines in question.

7.3 GRAPHICS DISPLAYS

The graphics devices currently supported in AIPS fall into three categories: TV display devices such as the IIS, hardcopy devices such as the Versatec printer/plotter, and interactive graphics terminals such as the Tektronix 4012. This section deals with the Tektronics type graphics terminals. The other devices are discussed in the chapter on plotting.

A graphics terminal can be used in two major modes: as a temporary display device, or as an interactive graphics device. When used as a temporary display device, a task will read graphics commands from a plot file, convert these device independent commands to the form needed by the device, and finally write to the device. The AIPS task that does this is TKPL. A programmer wishing to write a task to intepret a plot file for another type of graphics terminal, would start with TKPL and convert the routines TKDVEC, TKCHAR, and TKCLR to send the proper commands to the device.

When using a graphics terminal in the interactive mode, the programmer probably will go straight from the data file to the graphics terminal without going through a plot file. In general, an interactive task or verb will open the display device, display the data, activate the cursor, read the cursor position in the absolute device coordinates, convert these coordinates into more useful units, and then perform some useful function with the converted units, such as display them.

7.3.1 Opening The Graphics Terminal.

The graphics terminal is opened as a non map file using ZOPEN. AIPS logical unit 7 is reserved for this device type, and should be used in the call to ZOPEN. When constructing the device name with ZPHFIL, a device type of 'TK' must be used. A volume number of 1 and zero values for the other auguments should be used to remain consistent with other tasks. On the VAX, each AIPS is assigned a graphics terminal on start up according to a set of logical names. Thus, ZOPEN on the VAX ignores everything in the name except TK.

7.3.2 Writing To The Graphics Terminal

Before writing to the graphics terminal, the programmer must set some values in common. Common INCS:CTVC.INC can be initialized by calling routine YTVCIN. Most values in this common are for the TV display, but array MAXXTK contains the maximum X and Y values in device units (for the Tektronix 4012, these values range from 1 to 1024 for X and 1 to 780 for Y). In common INCS:CTKS.INC, the graphics buffer size, TKSIZE, should be set to 20. The current position in use in the buffer, TKPOS, should be set to zero. Scale factors SCALEX and SCALEY and offsets RX0 and RY0 must be calculated and assigned. If a subroutine is told to scale a value then the X value in absolute device units will be equal to

SCALEX * value_input_for_X + RX0.

Usually the first thing a programmer will want to do when writing to the terminal is to clear the screen. This can be done with subroutine TKCLR.

Setting the beginning of the line (sometimes called drawing a dark vector) and drawing lines from the current position to a new position (a bright vector) are done with routine TEKVEC. TEKVEC is given an X and Y position and a control code which tells it if it should draw a dark vector or a bright vector, and if it should consider X and Y to be in absolute device units or if the values should be scaled. TEKVEC will automatically truncate vectors that run off the plot and write the buffer when it fills up.

Characters can be written to a Tectronix terminal by calling routine TKCHAR. TKCHAR allows the programmer to write characters either horizontally or vertically. TKCHAR uses the hardware character generator in the Tektronics, so the characters only come in one size. Choosing the starting position of the characters involves a combination of TEKVEC and TKCHAR. First, a vector position on the plot is chosen by calling TEKVEC with the 'dark vector' option. Then an offset from the vector position in character sizes is chosen by use of the DCX and DCY parameters in TKCHAR. Programmers who need a character generator can find one in task PRTPL that can be adapted to a graphics terminal.

Before closing the graphics terminal, the programmer should write any remaining buffers to the graphics device by calling TEKFLS.

7.3.3 Activating And Reading The Cursor.

Subroutine TKCURS will activate the cursor on the Tektronix 4012 and wait for a response from the 4012 keyboard. After the user positions the cursor and presses any key, the cursor will disappear and TKCURS will return the last coordinate position in absolute Tektronix units. The programmer will probably have to convert this position into plot coordinates by using information in the image catalog.

7.3.4 Updating The Image Catalog

The image catalog should be updated when an image is written to the graphics terminal. This is essential when one task (or verb) writes an image to the device, and another task (or verb) needs information about the plot on the screen. For example, task TKPL can be used to display a contour map on the terminal, and verb TKPOS can be used to print map coordinate values at selected positions on the plot. The TKPOS uses information in the image header to convert an absolute Tectronix cursor position into the map axis units such as RA and DEC. The routines ICINIT and ICWRIT can be used to set up the image catalog for the graphics terminal. See the chapter on

catalogues for a detailed description of the image catalog and the example below for making a minimum image catalog entry.

7.3.5 An Example

This example code shows how to open the graphics terminal, clear the screen, draw a box, and write some text in the center of the box. Opening the map, getting parameters from AIPS, etc., are not shown. In a non-trivial example, calculating the scaling parameters and updating the image catalog would be much more involved.

```
      INTEGER*2 TK, N0, N1, ITKLUN, ITKIND, IERR, TKSIZE, TKPOS,
     *   IPOS, IDRAW, NCHAR, IHORZ, IPLANE, BUFFER(256), VOL, CNO,
     *   CATBLK(256), LINE(40)
      LOGICAL*2 T,F
      REAL*4    DEVNAM(6), BLCX, BLCY, TRCX, TRCY, CENTER, DCX, DCY
          ...
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DTVC.INC'
      INCLUDE 'INCS:DTKS.INC'
      INCLUDE 'INCS:CHDR.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CTVC.INC'
      INCLUDE 'INCS:CTKS.INC'

      .
      .
      .
C                                        Open the Tektronix device.
      ITKLUN = 7
      CALL ZPHFIL (TK, N1, N0, N0, DEVNAM, IERR)
      IF (IERR.NE.0) GO TO 900
      CALL ZOPEN (ITKLUN, ITKIND, N1, DEVNAM, F, T, T, IERR)
      IF (IERR.NE.0) GO TO 900
C                                        Set variables in common.
      CALL YTVCIN
      TKSIZE = 20
      TKPOS = 0
C                                        Make screen be 100 by 100
C                                        units.
      SCALEX = MAXXTK(1) / 100.0
      SCALEY = MAXXTK(2) / 100.0
      RX0 = 0.0
      RY0 = 0.0
C                                        Clear screen.
      CALL TKCLR (ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                        Set corners
      BLCX = 25.0
      BLCY = 25.0
      TRCX = 75.0
      TRCY = 75.0
C                                        1 is the code for scale
```

```
C                                       X and Y and position vector.
      IPOS = 1
C                                       2 is the code for scale X and
C                                       Y and draw vector.
      IDRAW = 2
C                                       Draw a box.
      CALL TEKVEC (BLCX, BLCY, IPOS, ITKIND, IERR)
      CALL TEKVEC (BLCX, TRCY, IDRAW, ITKIND, IERR)
      CALL TEKVEC (TRCX, TRCY, IDRAW, ITKIND, IERR)
      CALL TEKVEC (TRCX, BLCY, IDRAW, ITKIND, IERR)
      CALL TEKVEC (BLCX, BLCY, IDRAW, ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                       Write some characters in
C                                       the center of the box.
      NCHAR = 14
      ENCODE (NCHAR, 1000, LINE)
C                                       Position at center.
      CENTER = 50.0
      CALL TEKVEC (CENTER, CENTER, IPOS, ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                       Compute offset to start
C                                       writing characters.
      DCX = - NCHAR / 2.0
      DCY = - 0.5
      IHORZ = 0
C                                       Write message
      CALL TKCHAR (NCHAR, IHORZ, DCX, DCY, LINE, ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                       Write any remaining buffer to
C                                       screen.
      CALL TEKFLS (ITKIND, IERR)
C                                       Update image catalog although
C                                       for this example plot has no
C                                       relation to map.
C                                       Calculate image plane.  These
C                                       values are found in common
C                                       set up in CDCH.INC.
      IPLANE = NGRAY + NGRAPH + NTKDEV
      CALL ICINIT (IPLANE, BUFFER)
C                                       CATBLK, VOL and CNO were
C                                       found when map was opened.
      CATBLK(I2VOL) = VOL
      CATBLK(I2CNO) = CNO
C                                       Set plot type to MISC
      CATBLK(I2PLT) = 1
      CALL ICWRIT (IPLANE, NO, CATBLK, BUFFER, IERR)
C                                       Close graphics terminal.
      CALL ZCLOSE (ITKLUN, ITKIND, IERR)
      .
      .
      .
 1000 FORMAT ('This is a test')
```

7.4 INCLUDES

7.4.1 CTKS.INC

```
C                                                          Include CTKS
      COMMON /TKSPCL/ TKBUFF, SCALEX, SCALEY, RX0, RY0, RXL, RYL,
     *   TKPOS, TKSIZE
C                                                          End CTKS
```

7.4.2 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, TVDUMS, TVZOOM,
     *   TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC, TYPMOV,
     *   YBUFF
C                                                          End CTVC
```

7.4.3 DTKS.INC

```
C                                                          Include DTKS
      REAL*4    TKBUFF(20), SCALEX, SCALEY, RX0, RY0, RXL, RYL
      INTEGER*2 TKPOS, TKSIZE
C                                                          End DTKS
```

7.4.4 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, TVDUMS(7),
     *   TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *   TVSPLM, TVSPLC, TYPMOV(16), YBUFF(168)
C                                                          End DTVC
```

7.5 ROUTINES

7.5.1 ICINIT - Initializes image catalog for plane IPLANE

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

7.5.2 ICWRIT - writes image catalog block in ICTBL into image catalog.

```
   ICWRIT (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
```

7.5.3 MDISK - reads or writes image data to/from disks and other devices.

```
      MDISK (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
```

MDISK sets array index to the start of the next line wanted.
NOTE: the line sequence is set by the WIN parameter in MINIT, if the vaules 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.

MINIT MUST be called before MDISK.

7.5.4 MINIT - initialized the I/O tables for MDISK.

```
      MINIT (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 COMOFF + 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
```

MINIT 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: i.e. 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 transfered per I/O op
          2 =>  No. of major times a buffer has been acessed
          3 =>  No. of major lines remaining on disk
          4 =>  Output index for first pixel in window
          5 =>  No. pixels to increment for next major line
          6 =>  Which buffer to use for I/O; -1 => single buffer
          7 =>  Block offset in file for next operation (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)
```

7.5.5 TEKFLS - writes the output buffer TKBUFF to the TEKTRONIX 4012.

```
    TEKFLS (FIND,IERR)
Inputs:
  FIND     I*2 FTAB position assigned to TEK 4012.
Outputs:
  IERR     I*2 error flag. 0=ok,  .GT. 1=write error from ZFIO
```

7.5.6 TEKVEC - puts control characters, and X and Y coordinates into the TEKTRONIX output buffer.

```
    TEKVEC (XX, YY, IN, FIND, IERR)
Inputs:
  XX    I*2 X coordinate value.
  YY    I*2 Y coordinate value.
  IN    I*2 control value:
            1 = Scale XX and YY and precede coordinates
                by 'write dark vector' control character
            2 = Scale XX and YY, put in buffer; will write
                bright vector.
            3 = XX and YY are not scaled, 'write dark
                vector' control character is put into
                the buffer.
            4 = no scale, write bright vector.
  FIND    I*2 FTAB position of TEKTRONIX device.
Output:
  IERR    I*2 error code, 0=ok, 1=write error.
Common variables modified:
  TKBUFF
  TKPOS
  RXL, RYL
```

7.5.7 TKCHAR - writes characters to a TEKTRONIX 4012.

```
    TKCHAR (INCHAR, IANGL, DCX, DCY, TEXT, ITFIND, IERR)
Inputs:
  INCHAR   I*2 number of characters.
  IANGL    I*2 0=horizontal, other = vertical.
  DCX      R*4 X distance in characters from current position.
  DCY      R*4 Y distance in characters from current position.
  TEXT     R*4(??) packed characters.
  ITFIND   R*4 FTAB index of open TEK.
Outputs:
  IERR     I*2 error indicator. 0 = ok.
```

7.5.8 TKCLR - will clear the screen for a Tektronix 401n.

```
    TKCLR (DEVFND, IERR)
Inputs:
  DEVFND  I*2   FTAB index of an open device.
Output:
  IERR    I*2   Error code from the last I/O routine. 0=ok.
```

7.5.9 TKCURS - activates the cursor on the TEKTRONIX 4012 and waits for a response from the 4012 keyboard. After the response the cursor will disappear and TEKCUR will return the coordinate positions. The TEKTRONIX must have opened (by ZOPEN) before this routine is called.

```
    TKCURS (IFIND, IOBLK, IX, IY, IERR)
Inputs:
  IFIND    I*2 index into FTAB for open TEKTRONIX device.
  IOBLK    I*2(256) I/O block for TEKTRONIX device.
Outputs:
  IX       I*2 x cursor position.
  IY       I*2 y cursor position.
  IERR     I*2 0=ok, 1=TEK write error. 2=TEK read error.
```

WARNING: This routine assumes a normal interface to a TEK 401n. Thus it may not work on all CPUs.

7.5.10 TKDVEC - converts TEK4012 vectors to actual commands to the TK buffer. Positions are assumed to be in bounds.

```
    TKDVEC (IN, X, Y, FIND, IERR)
Inputs:
  IN   I*2   1 => dark vector, 2 => bright vector
  X    I*2   X coordinate value.
  Y    I*2   Y coordinate value.
  FIND I*2   FTAB position of TEKTRONIX device.
```

Outputs:
  IERR I*2 error code, 0=ok, 1=write error.
Common variables modified:
  TKBUFF
  TKPOS

7.5.11 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 tape I/O data is always written with the block size specified to UVINIT; one I/O operation per call. For disk 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 MDISK with opcode 'FINI'.

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

UVDISK (OP, LUN, FIND, BUFFER, NIO, BIND, IERR)

Inputs:		
OP	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 visibilites 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.

7.5.12 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 non tape 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	P	I*4	Total number of visibilities to read. NVIS+VISOFF must be no greater than the total number in the file.
VISOFF	P	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. Determines block size for tape I/O
BUFSZ		I*2	Size in bytes of the buffer. If 32767 given, 32768 is assumed.
BUFFER( )		I*2	Buffer
BO	P	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.
               For double buffer read, 1 more I/O will have
               been done than indicated.
        4-5  = Block offset for next I/O.
          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)
```

7.5.13 VBOUT - VBOUT writes variable blocked records of I*2 data to tape. Maximum block size on the tape is 4008 bytes. Tape must be opened (non-map) before first call. For overlaid programs COMMON /VBCOM/ should be kept in a segment which is core-resident from the first call to the last call to VBOUT. A call with N = 0 will cause all data remaining in the buffer to be written. Character data must be in ASCII two characters per integer as local integers: ie call ZCLC8 followed by ZI16IL on such data before calling VBOUT.

```
VBOUT (N, IDATA, LUN, NUM, IERR)

Inputs:
   N      I*2 Number of I*2 words in array IDATA
              If N = 0 the buffer is flushed.
   IDATA  I*2 Array containing data to be written.
   LUN    I*2 LUN of tape to be written on.
   NUM    I*2 The record number to be written, must be 1 on
              the first and only the first record in a file.
Output:
   IERR   I*2  Return error code 0 => OK
                             1 => LSERCH error (tape not open)
                             2 => ZFIO error.
```

7.5.14 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.

7.5.15 ZOPEN - opens logical files, performing full open on disk files for which LUN > NDEVT. Tape units are assigned an I/O channel and given an FTAB entry for double buffering.

```
   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
```

7.5.16 ZPHFIL - construct a physical file name in PNAM from ITYPE, IVOL, NSEQ, and IVER. New version designed either for public data files or user specific files. This routine contains the logical assignment list for Graphics devices and is thus site dependent as well as machine dependent.

```
EXAMPLE: If ITYPE='MA', IVOL=8, NSEQ=321, IVER=99, NLUSER=762 then
            PNAME='DA07:MA832199;1' for public data or
            PNAME='DA07:MA832199.762;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  Number of the disk volume to be used.
   NSEQ     I*2  User supplied sequence number. 000-999.
                 This is the catalogue slot number for catalogued
                 files.
   IVER     I*2  User suppplied version number. 00-255.
                 Always 1 for map or uv data files.
Outputs:
   PNAM  R*4(6)  >= 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.

7.5.17 ZTAPE - Performs standard tape manipulating functions.

```
      ZTAPE (OP, LUN, FIND, COUNT, IERR)

   Inputs:
      OP   R*4  Operation to be performed. 4 characters ASCII.
                    'ADVF' = advance file marks
                    'ADVR' = advance records
                    'BAKF' = backspace file marks.
                    'BAKR' = backspace records.
                    'DMNT' = dismount tape. Works for VMS 3.0 & later.
                    'MONT' = mount tape. Works for VMS 3.0 and later.
                    'REWI' = rewind the tape on unit LUN
                    'WEOF' = write end of file on unit LUN: writes 4
                             EOFs, positions tape after first one
                    'MEOF' = write 4 EOF marks on tape, position tape
                             before the first one
      LUN  I*2  logical unit number
      FIND I*2  FTAB pointer.  Drive number for MOUNT/DISMOUNT.
      COUNT I*2 Number of records or file marks to skip. On MOUNT
                this value is the density.
     Outputs:
      IERR  I*2  Error return: 0 => ok
                               1 = File not open
                               2 = Input specification error.
                               3 = I/O error.
                               4 = End Of File
                               5 = Beginning Of Medium
                               6 = End Of Medium
```

CHAPTER 8

WAWA ("EASY") I/O

8.1 OVERVIEW

We have created a fairly coherent set of routines which attempt to hide most of the nasty details mentioned in the previous sections. 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.

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 file the programmer wants open.

2. All the parameters needed to specify a catalogued 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 catalogued 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. I/O to "map" type files is always in R*4 format as seen by the user. On input automatic scaling from I*2 occurs. A separate routine can be used to find the min/max of an output file, but it may be more convenient for the programmer to accumulate these as his algorithm progresses. A separate subroutine may be called to convert output R*4 maps to I*2.

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 catalogued 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 (they 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 = TSKNAM|NPOPS CLASS = 'SCRTCH' 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 catalogued 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. MAPFIX - Convert a catalogued R*4 map to a catalogued I*2 map.
15. MAPMAX - Find MAX & MIN of an R*4 map and enter into catalog.
16. GETHDR - Retrieve catalog header for an open catalogued file.
17. HDRINF - Retrieve specified items from map header.
18. TSKBEn - Combination of IOSETn and some task startup chores.
19. 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 required 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 EXTINI and EXTIO 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 catalogue slot numbers of the relevant files from the /WAWAIO/ common variable FILTAB(POCAT,) (file must be open to do so).

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 chapters of disk and 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'      Catalogue 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'      Catalogue 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 ine 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 Catalogue 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 catalogue 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 catalogue 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 overlayed 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 'ZFT5.INC'      File table space
        INCLUDE 'IBUn.INC'      WaWa buffer/table sizes
        INCLUDE 'IITB.INC'      WaWa I/O common
        INCLUDE 'IDCH.INC'      System parms
        INCLUDE 'DHDR.INC'      Header pointers
        INCLUDE 'DMSG.INC'      Messages, POPS #, ...
        INCLUDE 'DCAT.INC'      Catalogue header
Commons:
        INCLUDE 'CBUF.INC'
        INCLUDE 'CITB.INC'
        INCLUDE 'CDCH.INC'
        INCLUDE 'CHDR.INC'
        INCLUDE 'CMSG.INC'
        INCLUDE 'CCAT.INC'
Equivalences:
        INCLUDE 'EBUF.INC'
        INCLUDE '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 => Catalogue operation error
11 => Catalogue status error
12 => Map not in catalogue
13 => EXT file not in catalogue
14 => No room in header/catalogue
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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statments.
- Exxx.INC. These contain Fortran EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statments.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible 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 CBUF.INC

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

8.7.9 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.
```

8.7.10 CCAT.INC

```
C
      COMMON /MAPHDR/ CAT2
C
```

Include CCAT

End CCAT.

8.7.11 EBUF.INC

```
C
      EQUIVALENCE (RBUF(1), IBUF(1))
C
```

Include EBUF

End EBUF.

8.7.12 ECAT.INC

```
C
      EQUIVALENCE (CAT2(1), CAT4(1), CAT8(1))
C
```

Include ECAT

End ECAT.

8.7.13 ZFT5.INC

```
C
      INTEGER*2 FTAB(310)
C
```

Include ZFT5

End ZFT5.

8.8 DETAILED DESCRIPTIONS OF THE SUBROUTINES.

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

```
CLENUP
   no arguements
```

8.8.2 FILCLS - Close a file and clean up any I/O pending to it.

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

8.8.3 FILCR - Create a non-map of "file" type file associated with the catalogued 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)
   NBLOCK    I*2     Number of 512-byte blocks requested
   VER       R*4     Version of newly created file
```

8.8.4 FILDES - Destroy a catalogued 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 FILOPN - Find a catalogued or extension file in catalogue, open file and associate it with the LUN.

```
FILOPN (LUN, NAMS, EXT, TYPE, VER, ERROR)
 Inputs:
    LUN       I*2     logical unit number
    NAMS(9)   R*4     namestring specifying catalogue
    EXT       L*2     Desired file is an extension of a catalogued
                      file?
    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)
```

8.8.7 GETHDR - Fetch the header block of a catalogued, 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
```

8.8.8 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
```

8.8.9 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

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

```
MAPCR (NAMS, HDR, ERROR)
Inputs:
   NAMS(9)   R*4     Specified catalog entry
   HDR(256)  I*2     Catalog block specifying enough
                     information to determine file size:
                     specifically # of axes and # of pixels
                     on each axis.
```

8.8.11 MAPFIX - Convert a catalogued (including scratch) R*4 map to a catlogued I*2 map. If MAX & MIN are filled in in the R*4 header, they they will be used to determine scaling. If not, or if they are incorrect and cause an overflow, a new Max and Min will be determined and entered in header.

```
MAPFIX (NAMIN, NAMOUT, ERROR)
Inputs:
   NAMIN(9) R*4      Input catalog string
   NAMEOUT(9) R*4    Output catalog string
```

8.8.12 MAPIO - Transfer one line of data between core area DATA and a disk map-type file. On READ, data are converted from I*2 to R*4 if necessary and 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".

On WRIT data output is unscaled R*4, only. When you start writing MAX and MIN in the header will be marked as "INDE" or indefinite. If you want an I*2 map, you should make an R*4 scratch map and then call MAPFIX. If Max and Min are still indefinite at this time MAPFIX will figure them out with an extra pass through the map. You can also set them yourself in the map (catalogued on disk) header and save some time. 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
   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 WIN 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 multidimensional 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 catalogued files (i.e. no associated files) to simplify call sequence.

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

8.8.17 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
RPARM(*)	R*4	Array to receive passed parameters

8.8.18 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 is true 0 => ok, > 0 => troubles

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

UNSCR
no arguments

CHAPTER 9

USING THE TV DISPLAY

9.1 OVERVIEW

The most useful implementations of the AIPS system include one or more computer peripheral devices capable of displaying images with multiple levels of gray and/or color. We refer to such devices as TV displays since most are implemented via large binary memories and standard television monitors. The main program AIPS and some tasks (e.g. BLANK) use the TV display as an interactive input, as well as display, device. Other tasks (e.g. UVMAP, MX, APCLN) use the TV display simply to show the stages of the data processing. All use of the TV is optional and the AIPS system will run without such a device. The number of TV displays in the local system is parameterized (under control of the stand alone program SETPAR) and all programs are told which TV display (if any) is assigned to the current user.

9.1.1 Why Use (or Not Use) The TV Display?

There are numerous reasons to use the TV display in writing AIPS routines. Gray scale images provide a more realistic view of image data allowing the eye to integrate over noisy regions and to separate closely spaced features. Contour images require much more elaborate software to generate and they make unreasonably definitive assertions about the intensity levels. The TV may be used to display intermediate results which are never stored on disk. And the TV may be used to interact with the user in a very wide variety of ways. Current interactive usages include modification of the black and white transfer function, modification of pseudo coloring, selection of features of interest, selection of subimage corners, dynamic, multi-image displays, and communication to the task of simple information. The last is used primarily to tell iterative tasks that they may stop at the current iteration.

Despite these desirable features, an AIPS programmer should not put the TV in a task unless it is truly useful. A TV option requires some, potentially considerable extra coding effort and, during execution, some significant extra real and CPU time. Many TV devices also require a high rate of I/O in order to load an image and, especially, to interact with the user. If an algorithm is based on the TV display, then it will not be available at those AIPS sites which do not have one. Although TV displays can function as graphics

devices, many of them are very slow in that mode. Finally, tasks which use the TV will interfere with the interactive AIPS user's other uses of the display by replacing current images in the TV memory or modifying the zoom, scroll, transfer functions, et al.

9.1.2 The AIPS Model Of A TV Display Device.

As AIPS was being designed, it was realized that there was already a wide variety of TV display devices on the market and that the market would not hold still. The NRAO initially purchased two International Imaging Systems (IIS) Model 70/E displays. However, that company changed rapidly to Model 70/F and now sells only a Model 75. Our initial choice undoubtedly colors our image of what a TV display device does and how it does it. Nonetheless, we have attempted to design the code to be very general and to account for the range of options available on individual models of display and for the range of different manufacturers.

We regard the TV display as being a computer peripheral device which accepts the basic I/O operations of open, close, initialize, read, and write. Special Z routines are provided in AIPS since we do not assume that these I/O operations are identical, for all TVs and host operating systems, to those for disks, tapes, or Fortran devices. We assume that the TV display may be subdivided logically into a variety of sub-units which control the various functions of the display. Special libraries of subroutines, subdivided by model of TV display, are provided for communicating to these sub-units. These subroutines are called "Y routines" because all of them have names beginning with the letter Y. The NRAO has, at this time, developed the Y routines for IIS Models 70/E and 70/F. In addition, we store, distribute, and attempt to maintain sets of Y routines developed by other institutions for other models of displays. At the moment, we have Y routines for DeAnza, developed by Walter Jaffe at the STScI, and for IIS Model 75, developed by IIS.

AIPS also uses, at both the Y and non-Y programming levels, a TV device parameter common. This common is initialized by a Y routine (YTVCIN) and is maintained via a disk file and a stand alone program (SETTVP). The common contains both fundamental parameters (i.e. number of memories, display size, maximum intensity, maximum zoom, etc.) and parameters describing the current state of the TV (i.e. which planes are on, current zoom and scroll, etc.).

In order to provide the full functionality of the basic AIPS verbs and tasks, a TV display device needs to contain the following sub-units. Note, these subunits are logical devices. They may be implemented as control registers in the device or in numerous other fashions. It is only necessary that the Y routines impose on the device a control that forces it to this general structure.

1. IMAGE MEMORIES: These are one or more memories n bits deep which hold the gray-scale images to be displayed. All n bits of the image contribute to the display. The memory is assumed to have a fixed number of pixels on each axis and to be addressable at the

individual pixel level. The addresses are assumed to be one-relative and to begin at the lower left of the display. The number of bits, the dimensions of each axis, and the number of memories are parameters inside AIPS. It is also assumed that each memory may be turned on and off in each of the three colors individually.

2. GRAPHICS MEMORIES: These are one or more memories each 1 bit deep used to display graphical information such as axis labels or line drawings on top of the gray-scale images. It is assumed that the overlay planes have the same number of pixels on each axis as the image memories and that each overlay plane may be enabled or disabled individually. It is nice to be able to assign unique colors to each of the overlay planes. AIPS will want to use four overlay planes, but all standard programs will work more or less normally with only one. The number of graphics memories is a parameter.

3. CURSOR AND BUTTONS: The cursor is some form of marker which may be enabled or disabled and which is under the positional control of some mechanical device (e.g. trackball, joy stick, thumb wheels). The position of the cursor on the TV screen may be read at any time it is enabled. The "buttons" are some device to signal conditions to the programs such as "this is the desired position" or "time to quit". AIPS assumes that there are four such buttons returning to the program a value between 0 and 15. Simultaneity of more than one button is never used, however.

4. LOOK UP TABLES: These are tables of numbers which convert the stored n-bit image intensities to the desired display intensities. AIPS assumes that there is one n-bit in, n-bit out look up table ("LUT") for each color of each image memory. AIPS also assumes that there is a second set of three look-up tables, called the output function memory ("OFM"), which converts the sums of all enabled memories to the final displayed intensities. In practise, AIPS uses the individual channel LUTs for black and white enhancement (most of the time) and the OFM for pseudo-color enhancement. There are algorithms, such as TVHUEINT, which utilize the full capability of the two sets of look-up tables. Arrays inside AIPS are likely to be dimensioned for 8-bit image planes and a 10-bit OFM. (These assumptions probably should be generalized in time.)

5. SCROLL: It is assumed that each image memory may be displayed on the TV screen shifted along both axes by varying amounts. AIPS assumes that each memory may be scrolled independently and that the graphics memories may be scrolled together independent of the image memories. The minimum increments of scroll along each axis are parameters. Note that AIPS does not make heavy use of scroll except for the TVROAM display and, of course, TVSCROLL.

6. SPLIT SCREEN: It is assumed that the screen may be divided into quadrants and different image channels enabled in each quadrant. There is a control parameter specifying the degree to which the local TV display has this capability. AIPS currently uses split screen primarily in the TVROAM display, but also uses it during

image enhancement in the channel blink routines.

7. ZOOM: AIPS assumes that the display of an image may be blown up about any pixel by automatic pixel replication by integer powers of two without affect on the images stored in the image memories. The highest power of two available is a parameter. Zoom is important to the TVMOVIE algorithm and is used in many of the image enhancement routines.

The most important TV operations of AIPS could be implemented on a TV device having one image memory, appropriate LUTs, and a cursor with buttons. Additional image memories, graphics memories, an OFM, scroll, split screen, and zoom are needed primarily for less important aspects of the basic operations and for some interesting, but esoteric operations.

There are several other sub-units in the IIS Model 70 which are supported by the Y routines in that sub-library. They include an input function memory (translates input to the TV from the host and from the ALU), a histogram generator, a feedback arithmetic logic unit, shift and min/max registers, and the like. Although there are no standard routines in AIPS which use these units, there are two new nonstandard tasks for histogram equilization which make some use of them. The special Y routines used by these two tasks will be described below, but they should not (yet) be required for other kinds of TV devices - if they are even possible on them.

9.2 FUNDAMENTALS OF THE CODING

9.2.1 The Parameter Commons And Their Maintenance

All application routines must open the TV device via a call to TVOPEN and close it via a call to TVCLOS. TVOPEN opens a disk file called ID10000n with exclusive use requested, where n is the number of the assigned TV device. From the first record of this file, it reads a 256-word record containing parameters which describe the structure and current status of the assigned TV device. The parameters are stored in a common called /TVCHAR/ which is obtained by including DTVC.INC and CTVC.INC. TVCLOS puts back to the disk the time variable portions of this common and then closes the file. In this way, several users/programs may share the TV in sequence and all will know the current status information. The disk file may be initialized and the individual parameters set by using the stand alone program SETTVP. The parameters are important to the correct functioning of the local TV device and must be set and maintained carefully.

The fixed portion of /TVCHAR/, namely that portion not written by TVCLOS, includes the parameters:

NGRAY	The number of n-bit image memories.
NGRAPH	The number of 1-bit graphics overlay memories.
NIMAGE	The number of images which may be stored simultaneously in a gray-scale image plane (affects the image catalogue mostly).

MAXXTV(2)	The number of pixels in the X and Y directions.
MAXINT	The highest gray-scale intensity = 2 ** n - 1.
SCXINC	The minimum increment in scroll in the X direction.
SCYINC	The minimum increment in scroll in the Y direction.
MXZOOM	The highest power of two for zooming.
NTVHDR	The number of integer words in the TV I/O header (probably no longer used).
CSIZTV(2)	The size of characters in pixels in the X, Y directions.
GRPHIC	The bit pattern representing the set of graphics overlay memories (normally -32768).
ALLONE	The bit pattern representing all bits on (-1).
MAXXTK(2)	The number of pixels in the X, Y directions on the TEK graphics device.
CSIZTK(2)	The size of characters on the TEK graphics device in pixels in the X, Y directions.
TYPSPL	Type of split screen: 0 none, 1 vertical division only, 2 horizontal division only, 3 either, 4 both.
TVALUS	Number of TV arithmetic logic units.
TVXMOD	Mode for loading TV in X direction: 0 none, 1 ok in AIPS order (to right), 2 ok in reverse direction.
TVYMOD	Mode for loading TV in Y direction: 0 none, 1 ok in AIPS order (to top), 2 ok in reverse direction.
TVDUMS(7)	Spare room

The time variable portion of the /TVCHAR/ common is:

TVZOOM(3)	Current zoom: power of two, X, Y zoom center
TVSCRX(16)	Current X scroll for 15 image planes and graphics.
TVSCRY(16)	Current Y scroll for 15 image planes and graphics.
TVLIMG(4)	Bit pattern for which images are on by quadrant: quadrants are numbered CCW from top right and the lsb is for gray plane one and NGRAY+NGRAPH bits are used.
TVSPLT(2)	Current split screen position in X, Y.
TVSPLM	10 * (number planes in X) + (number planes in Y) in Roam mode.
TVSPLC	Roam mode: digits imply which channels in which order.
TYPMOV(16)	Movie loop code: 2 * (magnification power of two) + 8 * (number frames remaining). Add 1 if this is the first plane of the movie.
YBUFF(168)	Machine dependent parameters.

There is a second TV include which controls I/O, but is little used elsewhere. It is obtained by including DTVD.INC and CTVD.INC and contains:

TVLUN	LUN of open TV device.
TVIND	Position of TV device in FTAB for I/O.
TVLUN2	LUN of open TV parameter disk.
TVIND2	Position of parameter disk in FTAB.
TVBFNO	Not used (map style I/O no longer supported).
TVMAP	Not used.

9.2.2 The I/O Routines

Four basic I/O operations for TV devices are supported: open, close, I/O reset ("master clear"), and data transfer (read/write). The actual Z subroutines which perform these operations are both TV device and host operating system specific. The subroutines are stored in the subdirectory appropriate for the host operating system with names reflecting the TV device type. To insure that the correct Z routines are link edited, a layer of Y routines is interposed between these Z routines and all other non-Y AIPS routines. No non-Y subroutine or program should call these Z routines. These Z subroutines have names of the form ZMMMOO, where MMM is the TV model (i.e. M70 for IIS Models 70 and 75, DEA for DeAnza) and OO is the type of operation (OP for open, CL for close, MC for I/O reset, and XF for data transfer).

Note that the four Z routines may have TV device specific call sequences. The current implementations are

Z...OP :

ZM70OP (LUN, IND, IERR)
ZDEAOP (LUN, IND, IERR)
Performs the needed channel assignment and opens a non-map entry in the FTAB. The DeAnza version also calls ZDEAXF ('DAT ',...) to initialize the I/O.

Z...CL :

ZM70CL (LUN, IND, IERR),
ZDEACL (LUN, IND, IERR),
Performs a simple close (deassign) via a call to ZCLOSE and clears the FTAB entry. The DeAnza version calls ZDEAXF ('DET ',...) to perform a deallocation before calling ZCLOSE.

Z...MC :

ZM70MC (FTAB(channel)) - Vax version
Performs a "rewind" QIO operation causing the IIS to reset its I/O status.
ZM70MC - Modcomp version
Performs a "home" I/O operation causing the IIS to reset its I/O status.
ZDEAMC
Null subroutine.

Z...XF :

ZM70XF (OPER, NBYTES, HEADER, BUFFER, IERR)
Writes an eight-word command HEADER to the IIS after preparing the checksum word of the header. Then reads from or writes to the IIS NBYTES of BUFFER. Issues a master clear on error.
ZDEAXF (OPER, BUFFER, NBYTES, EP1, EP2, WAIT, IERR)
"Calls to ZDEAXF map one to one to calls to IP8 routines in the DeAnza IP8500 level 0 software package." Does requested I/O operation using opcode definitions contained in IP8IOF.MAR (supplied by DeAnza, not NRAO).

9.2.3 The Y Routines:

The Y routines may be divided into three groups which we call levels 0 through 2. Level 0 routines do not perform I/O to the TV device. Instead, they prepare data to be fed to lower level Y routines and/or handle common parameters and various conversions. It has been found that this level of Y routine often needs little alteration from one model of TV to the next. Level 1 routines do call Z...XF to perform I/O to the TV device. They may be called by both Y and non-Y routines and hence must be implemented for all TV devices. Level 2 routines also perform I/O in general, but are only called by Y routines. Hence, these do not have to be implemented for all TV devices. The reader should note that the division of Y routines into these three levels is not quite so clear as the above description would indicate. For one, some level 2 routines may have to graduate to level 1 as new application code is developed. For another, some of the level 0 routines are actually TV independent as coded for the IIS Models 70 and 75. They are called Y routines simply to allow more efficient, level 0 or 1 implementations for other TV devices.

On normal AIPS systems, the Y subroutines are stored in subdirectories separated by type of TV device. On our Vax, the subdirectories are [AIPS.reldate.APL.YSUB.xxx] with logical names APLxxx: where xxx is IIS for IIS Model 70, M75 for IIS Model 75, and DEA for DeAnza and where reldate is the date of the current AIPS release. The compile procedures select the value of xxx appropriate to the local TV device and write the object code into the link editor library in the [AIPS.reldate.APL] area. This library is then used for link editing all programs and tasks. The careful reader will note that this method does not allow for more than one kind of TV device on a given host computer. To date, we have been able to get away with this deficiency. In the future, we may have to improve the AIPS start-up procedure and the task activation subroutine (ZACTV8) so that the number of the assigned TV device is used to determine from which library the executable modules are taken.

The following sections provide a brief overview of the current Y routines. The precursor comments of most of the Y routines are reproduced near the end of this chapter.

9.2.3.1 Level 0

- YCHRW writes characters into an image or graphics plane. The M70 version is TV independent and uses a 7 x 9 pixel area per character. The backround intensity is set to 1 for multi-bit channels and 0 for graphics.
- YCNECT writes a line segment in an image or graphics plane at a specified intensity. The M70 version is TV independent.
- YCUCOR converts cursor positions and obtains the corresponding image header. It is a specialized version of YCURSE to avoid any TV I/O and to do the image catalog work.

M70 and DeAnza versions are identical.

- YCURSE enables/disables cursor and cursor blink and reads cursor position and buttons value. The main complications come from corrections for zoom and scroll. The IIS Model 70/E is tricky, the Model 70/F and DeAnza are easier.

- YGRAPH enables/disables graphics overlay planes by altering the graphics color look up tables. A non-essential nicety is the use of complimentary colors when two or more graphics planes are enabled at the same pixel.

- YLNCLR computes a piecewise linear OFM with gamma correction. Called a Y routine solely because of the use of a 10-bit OFM.

- YSLECT enables/disables gray and graphics channels setting the proper values into TVLIMG.

- YTVCIN provides initial values for the TV characteristics commons.

- YZERO clears a gray or graphics memory by the fastest possible method.

- YTVCLS close the TV device. Actually is just an interface to the appropriate Z...CL subroutine.

- YTVMC reset the TV I/O status. Actually is just an interface to the appropriate Z...MC subroutine.

- YTVOPN Open the TV device. Actually is just an interface to the appropriate Z...OP subroutine.

9.2.3.2 Level 1

- YCRCTL reads/writes the cursor/trackball control register including position, enable/disable on each axis, blink control.

- YIMGIO reads/writes a line of image data from/to a gary-scale or graphics plane. It will perform buffer swaps if needed to get the desired angle and bit-level corrections when graphics planes are read. This is the most heavily used Y routine.

- YINIT initializes all subunits of the TV, clears the TV memories, resets the image catalog, and resets status parameters in common.

- YLUT reads/writes the full channel-level lookup table for one or more image channels and colors.

- YOFM reads/writes the full OFM lookup table for one or more colors.

- YSCROL writes the scroll control registers for one or more channels.

- YSPLIT reads/writes the split screen control registers. This is the actual control of the split screen center and of which channel(s) are enabled/disabled in each quadrant.

- YZOOMC writes the zoom control registers giving magnification and zoom center.

9.2.3.3 Level 2

9.2.3.3.1 IIS Models 70 And 75

- YALUCT reads/writes the IIS arithmetic logic unit control registers. No actual function is performed until a feedback operation is done via YFDBCK. This routine is very IIS specific and we doubt that its functions can be implemented on other TVs.

- YCONST reads/writes the constant "biases" which are added to the sums of the individual enabled channels before the signals are sent to the OFM.

- YFDBCK causes a feedback operation to occur. The ALU does its thing with one or more channels and returns an 8 or 16 bit result to one or two channels. A magic bit causes the function to be a simple zeroing of a channel.

- YGGRAM reads/writes the lookup table used for graphics planes.

- YGRAFE reads/writes the graphics control register which assigns a graphics plane as the "blotch" plane and another as the "status" plane. No use is made of this.

- YGYHDR prepares a basic I/O control header for writing/reading image data to/from the IIS.

- YIFM reads/writes a portion of the input function memory. This lookup table can be used in writing data to the TV memory and in the feedback operation. AIPS does not do the former and only one non-standard task does the latter.

- YMAGIC (Model 75 only) initializes graphics, zoom, and scroll subunits (called by YINIT only).

- YMKHDR prepares a basic I/O control header for the IIS.
- YMNMAX reads the min and max output from the sum of all enabled gray-scale planes for each color.
- YRHIST reads a portion of the histogram of the output of the OFM for a selected color. The IIS can do this on the fly if properly equipped.
- YSHIFT reads/writes the shift registers which shift the 13-bit output of the sum of all enabled channels before the data get to the OFM.
- YSTCUR reads/writes the IIS cursor array. This 64 x 64 bit array provides a wide choice of patterns for the display "cursor". AIPS uses only a simple plus sign with a blank pixel at the center.

9.2.3.3.2 DeAnza

- YGGRAM reads/writes the lookup table used for the graphics planes.
- YLOWON finds lowest channel number in a channel mask.
- YMKCUR creates and loads the cursor pattern memory with a specified shape. Only the AIPS plus sign is implemented.
- YTCOMP performs logical tests on parameter values. It is used to minimize I/O to the DeAnza control registers.
- YDEA.INC Include file giving parameter definitions to specify positions in YBUFF which correspond to the various registers in a DeAnza TV device.

9.3 CURRENT APPLICATIONS

This section is devoted to a generally brief overview of the current application code. Primarily it will be used simply to point out which routines do what, with some comment on the methods. This should suffice as an introductory guide to the code for applications programmers wishing to include the TV display in their programs. In a couple of cases, some of the actual code will be reproduced in order to clarify the use of the various service routines. The precursor remarks for some of the most commonly used, non-Y service routines are reproduced at the end of this chapter.

9.3.1 Status Setting

By "status setting", we mean initializing the TV device, clearing memory channels, enabling and disabling portions of the display, and the like. Many of the applications which involve loading images to the TV display will zero the relevant memories (via YZERO) and clear the corresponding portions of the image catalog (via ICINIT) before carrying out their primary functions. However, the simplest examples of status setting are those performed by various AIPS verbs. The subroutine AU5 performs the verbs TVINIT (via YINIT), TVCLEAR (as follows), GRCLEAR (like TVCLEAR without the MOVIST call), TVON, TVOFF, GRON, GROFF (via calls to YSLECT), TV3COLOR (use YSLECT to turn off all channels, then YSLECT to turn on channels 1 through 3 in red, green, blue, resp.), and CURBLINK (via YCURSE).

The verb TVCLEAR is coded as follows. The channel number is picked up as an integer, the decimal code is converted to a bit pattern (via DECBIT), the movie status parameters are reset (via MOVIST), and then a loop over all selected gray planes is done to zero the memory (via YZERO) and clear the image catalogue (via ICINIT).

```
C                                       Open TV device
      CALL TVOPEN (CATBLK, JERR)
      IF (JERR.EQ.0) GO TO 50
         POTERR = 101
         GO TO 980
....
 200  ICHAN = ABS(TVCHAN) + EPS
C                                       convert to channel bit mask
      CALL DECBIT (NGRAY, ICHAN, ICHAN, ITEMP)
C                                       clear movie parameters
      CALL MOVIST (ONCODE(2), ICHAN, N0, N0, N0, IERR)
      DO 210 IP = 1,NGRAY
C                                       is plane requested
         IF (IAND(ICHAN,N2**(IP-1)).EQ.0) GO TO 210

C                                       clear image catalogue
            CALL ICINIT (IP, INBUF)
C                                       clear TV memory
            CALL YZERO (IP, JERR)
            IF (JERR.NE.0) GO TO 975
 210     CONTINUE
```

```
      GO TO 900
......
C                                          normal TV close
 900  CALL TVCLOS (CATBLK, JERR)
      GO TO 999
```

9.3.2 Load Images, Label

Images are loaded to the TV by a wide variety of tasks (e.g. APCLN, TVPL, BLANK) and by several verbs (TVLOD, TVROAM, TVMOVIE). TVLOD will be illustrated in this subsection and the others mentioned in later subsections.

The full code from subroutine AU5A for TVLOD, except the declarations, formats, error branches, and the like, is reproduced below. It begins by opening the TV control file and device (via TVOPEN). It moves the user adverbs to local variables to avoid changing their (global) values and opens the map file (via MAPOPN). It converts the user's PIXRANGE adverb using standard defaults (via RNGSET) and fills in some of the image catalogue parameters in the header. It sets the window parameters (via TVWIND), selects a single gray scale memory plane (via DECBIT), and clears the movie parameters (via MOVIST). Finally, it finishes up the image catalogue parameters, puts the header in the image catalogue, and reads, scales, and loads the image to the TV memory (all via TVLOAD). Afterwards, it closes the map file (via MAPCLS) and the TV device and disk file (via TVCLOS).

```
      INTEGER*2 N0, N1, N6, N7, N12, N6176, MA
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:CHDR.INC'
      DATA MA /'MA'/, N0, N1, N6, N7, N12, N6176 /0,1,6,7,12,6176/
C                                          open TV
      CALL TVOPEN (INBUF, IERR)
      IF (IERR.NE.0) GO TO 980
C                                          Map open junk: TVLOD, TVROAM
      IF (BRANCH.GT.2) GO TO 20
C                                          adverbs -> local variables
C                                          Adverbs used:
C                                          TVCHAN = tv channel
C                                          INNAM  = File name
C                                          INCLS  = File class
C                                          INSEQ  = File sequence number
C                                          INDSK  = Disk number
C                                          USERID = User ID number
C                                          TVBLCO = TV bottom left corner
C                                          TVTRCO = TV top right corner
C                                          TVXINC = TV x pixel increment
C                                          TVYINX = TV y pixel increment
C                                          PXRANG = Range of pixel values
C                                          TVCORN = BLC on TV screen for
C                                                   image
         ICHAN = IROUND(TVCHAN)
         IVOL = INDSK + EPS
```

```
      USID = ABS(USERID) + EPS
      SEQNO = INSEQ + EPS
      IF (USID.EQ.0) USID = NLUSER
      IF (USID.EQ.MAGIC) USID = 0
      CALL CHCOPY (N12, N1, INNAM, N1, SNAME)
      CALL CHCOPY (N6, N1, INCLS, N1, SCLAS)
      CALL RCOPY (N7, TVBLCO, LBLC)
      CALL RCOPY (N7, TVTRCO, LTRC)
      INC(1) = TVXINC + EPS
      INC(2) = TVYINC + EPS
      IMA = MA

C                                          open map file
      CALL MAPOPN (READ, IVOL, SNAME, SCLAS, SEQNO, IMA, USID,
     *    DLUN, DIND, CNO, CATBLK, INBUF, IERR)
      POTERR = 33
      IF (IERR.GT.1) GO TO 975
C                                          CATBLK, CT4, CT8 equivalenced
C                                          Image cat: fill in some
C                                          set image scaling too
      CALL RNGSET (PXRANG, CT4(K4DMX), CT4(K4DMN), CT8(K8BSC),
     *    CT8(K8BZE), CT4(I4RAN))
      CATBLK(I2VOL) = IVOL
      CATBLK(I2CNO) = CNO
      CALL CHCOPY (N2, N1, FUNTYP, N1, CATBLK(I2TRA))
      ITVC(1) = TVCORN(1) + EPS
      ITVC(2) = TVCORN(2) + EPS
      POTERR = 49
C                                          TVLOD
C                                          load one image plane
C                                          set windows
 100  TYPE = -1
      CALL TVWIND (TYPE, INC, LBLC, LTRC, ICHAN, ITVC, IWIN, IERR)
      IF (IERR.NE.0) GO TO 970
C                                          convert channel number
 110  CALL DECBIT (NGRAY, ICHAN, ICHAN, I)
      ICHAN = I
      CALL DECBIT (NGRAY, ICHAN, ICHAN, I)
C                                          clear movie parameters
      CALL MOVIST (OFF, ICHAN, N0, N0, N0, IERR)
C                                          do the TV load, img catlg
      CALL TVLOAD (DLUN, DIND, I, INC, ITVC, IWIN, N6176, IERR)
      IF (IERR.EQ.0) POTERR = 0
      GO TO 970
...
C                                          Close down ops
 970  CALL MAPCLS (READ, IVOL, CNO, DLUN, DIND, CATBLK, F, INBUF,
     *   IERR)
C
 975  CALL TVCLOS (INBUF, IERR)
```

The verbs TVWEDGE, IMWEDGE, and IMERASE load step wedge or pure zero images to the TV. They occur in subroutine AU5C. This routine calls TVFIND and possibly TVWHER to determine which image is desired. It then computes a buffer of appropriate values calling ISCALE (as TVLOAD does). AU5C then does a lot to set an appropriate image

catalogue header and writes that to the catalogue via ICWRIT. Finally it loads the TV rows via calls to YIMGIO.

The image labeling verbs TVLABEL and TVWLABEL are implemented from subroutine AU5B. This routine calls TVFIND to determine which image is to be labeled and IAXIS1 to do the labeling. Subroutines IAXIS1 and ITICS are very similar to the standard axis labeling routines used to make plot files and to write directly to the TEK graphics device. Characters are written to a graphics memory with a black background by calls to IMANOT and lines are written to the graphics memory by calls to IMVECT. (See the precursor comments of these routines at the end of this chapter.)

9.3.3 UVMAP

UVMAP uses the TV display for a fairly simple purpose --- to show the pattern of sampled uv cells (after convolution of the data to the grid). In principle, the algorithm is simple: associate uv cells with TV pixels and display 0 on the TV when the uv cell is unsampled (0.00) and display MAXINT on the TV when the cell is sampled (not 0.0). Unfortunately, the uv grid may be larger than the TV display and the disk file contains the grid in transposed, quadrant-swapped order. The first problem is solved by decimation (examine only every n'th cell in X and m'th cell in Y. The quadrant swapping is solved by addressing the TV beginning in the middle and by starting in the middle of the buffer which is written to the TV. The transposition is solved by writing the rows of the file as columns on the TV. The subroutine in UVMAP which does this (UVDISP) uses the image writing mode parameters (TVYMOD and TVXMOD) to handle this correctly when possible and to leave the display in transposed order when not (i.e. TVYMOD = 0).

9.3.4 APCLN, VM, MX, Et Al.

Iterative map analysis programs can make good use of the TV display. The user may, for example, request that the CLEAN task (APCLN) display the residual map after each major cycle. APCLN does this, then turns on the cursor and waits up to 15 seconds for the user to push Button D to signify that sufficient iterations have been performed. Several tasks (currently MX, VM, APGS, REGLR) use code similar to that in APCLN for loading the image to the TV and requesting the user input. Given below is the TV subroutine from APCLN. Note that it uses the array processor to scale the data for YIMGIO. This is reasonable, but only for tasks which are already using the array processor for more important computations. The costs of opening and closing the AP device and performing the I/O to it make any improvement in computational speed marginal for computations such as these. Note also the scaling parameters used here. The lowest displayed intensity gets TV value 1.01 and the highest gets MAXINT+0.99 (after the 0.5 for rounding is added and before the integers are truncated by routine VFIX). This scaling is assumed (primarily by CURVALUE) for all linear transfer functions. TV value

zero is reserved for "blanked" (indefinite) pixels and should always be given zero intensity on the display (by the LUTs and OFMs).

```
      SUBROUTINE DISPTV (TVPASS)
C-----------------------------------------------------------------------
C   DISPTV displays the current residual map on the TV, showing
C   the region centered on BOX(1).
C   Inputs:  TVPASS  I*2   code: 0 => clear screen, else don't
C                                0,1 => don't question the user about
C                                quitting
C   Output:  TVPASS  I*2   code: 32700 => user wants to quit cleaning
C-----------------------------------------------------------------------
      INTEGER*2 TVPASS, JROW(1), WIN(4), MX(2), MY, FIND, BIND, IERR,
     *   ICH, CATBLK(256), S2H(256), IQ, IB, IBLANK, ZERO(2),
     *   ONE(2), TWO(2), THREE(2), I, FOUR(2)
      INTEGER*2 IWIN(4), OIND, IY
      INTEGER*2 N0, N1, N2, N3, N4, N5, N6, N256
      REAL*4    XN(4), XBUFF(1), REED, S4H(128), TD, RPOS(2), ON, OFF
      REAL*4    WRIT, FINI, XFLUX, PREFIX(2), TVLMAX, TVLMIN
      LOGICAL*2 MAP, EXCL, WAIT, LERR, F
      REAL*8    S8H(64)
      INCLUDE 'INCS:DCLN.INC'
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DTVC.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DTVD.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CCLN.INC'
      INCLUDE 'INCS:CFIL.INC'
      INCLUDE 'INCS:CTVC.INC'
      INCLUDE 'INCS:CHDR.INC'
      INCLUDE 'INCS:CTVD.INC'
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE (JROW(1), BUFF2(1)),    (BUFF1(1), XBUFF(1))
      EQUIVALENCE (S2H, S4H, S8H, BUFF1(513))
      DATA MAP, EXCL, WAIT / .TRUE., 2*.TRUE./
      DATA WRIT, REED, FINI, ON, OFF /'WRIT','READ','FINI','ONNN',
     *   'OFFF'/
      DATA N0, N1, N2, N3, N4, N5, N6, N256 /0,1,2,3,4,5,6,256/
      DATA F, IBLANK /.FALSE., ' '/
      DATA ZERO, ONE, TWO, THREE, FOUR /0,0, 1,0, 2,0, 3,0, 4,0/
C-----------------------------------------------------------------------
C                                          open TV
      ICH = 1
      CALL TVOPEN (BUFF1, IERR)
      IF (IERR.EQ.0) GO TO 10
         ENCODE (80,1000,MSGTXT) IERR
         CALL MSGWRT (N6)
         GO TO 999
C                                          clear TV on 1st iteration
 10   IF (TVPASS.NE.0) GO TO 20
         CALL YZERO (ICH, IERR)
         IF (IERR.EQ.0) GO TO 15
            ENCODE (80,1010,MSGTXT) IERR
            CALL MSGWRT (N6)
```

```
            GO TO 998
 15       CALL ICINIT (ICH, XBUFF)
C                                          set scaling parameters
C                                          try to keep previous scaling
C                                          unless real changed a lot
 20    IF (TVFMAX.GT.TVFMIN) GO TO 25
          TVFMAX = TVREMX
          TVFMIN = TVREMN
 25    IF (TVREMX.GT.TVFMAX) TVFMAX = TVREMX
       IF (TVREMN.LT.TVFMIN) TVFMIN = TVREMN
C                                          Change scaling if a factor of
C                                          10 needed.
       TVLMAX = TVFMAX - TVFMIN
       IF (0.1*TVLMAX.LE.TVREMX-TVREMN) GO TO 30
          TVFMIN = AMIN1 (0.1*TVFMIN, TVREMN)
          TVFMAX = AMAX1 (TVFMIN+0.1*TVLMAX, TVREMX)
          TVLMAX = TVFMAX - TVFMIN
C                                          scale from 0.51 to
C                                          MAXINT + 0.49
 30    XN(1) = TVFMIN
       XN(2) = TVFMAX
       XN(3) = (MAXINT - 0.02) / TVLMAX
       XN(4) = 0.51 - TVFMIN * XN(3)
       CALL APPUT (XN, ZERO, FOUR, N2)
C                                          Write scaling info
       XFLUX = TVLMAX
       CALL METSCA (XFLUX, PREFIX, LERR)
       TVLMIN = TVFMIN * XFLUX / TVLMAX
       TVLMAX = TVFMAX * XFLUX / TVLMAX
       ENCODE (80,1020,MSGTXT) TVLMIN, TVLMAX, PREFIX
       CALL MSGWRT (N1)
C                                          center window on box 1
       WIN(1) = (WINM(3,1) + WINM(1,1)) / 2 - MAXXTV(1) / 2 + 1
       WIN(1) = MAX0 (N1, WIN(1))
       WIN(2) = (WINM(4,1) + WINM(2,1)) / 2 - MAXXTV(2) / 2 + 1
       WIN(2) = MAX0 (N1, WIN(2))
       WIN(3) = (WINM(3,1) + WINM(1,1)) / 2 + MAXXTV(1) / 2
       WIN(3) = MIN0 (NX, WIN(3))
       WIN(4) = (WINM(3,1) + WINM(1,1)) / 2 + MAXXTV(2) / 2
       WIN(4) = MIN0 (NY, WIN(4))
       DO 70 I = 1,2
          IWIN(I) = (MAXXTV(I) - WIN(I+2) + WIN(I) + 1)/2
          IF (IWIN(I).GE.1) GO TO 50
             IWIN(I) = 1
             WIN(I)  = (WIN(I+2) + WIN(I) - MAXXTV(I) + 1)/2
             GO TO 60
 50       IWIN(I+2) = IWIN(I) + WIN(I+2) - WIN(I)
          IF (IWIN(I+2).LE.MAXXTV(I)) GO TO 70
 60       IWIN(I+2) = MAXXTV(I)
          WIN(I+2) = WIN(I) + IWIN(I+2) - IWIN(I)
 70    CONTINUE
C                                          Prepare to read map.
       CALL ZOPEN (LUNRES, FIND, RESVOL, RESFIL, MAP, EXCL, WAIT, IERR)
       CALL MINIT (REED, LUNRES, FIND, NX, NY, WIN, XBUFF, BUFSZ1,
      *   BPRES, BORES, IERR)
       MX(1) = WIN(3) - WIN(1) + 1
```

```
      MX(2) = 0
      MY =  WIN(4) - WIN(2)+ 1
C                                          loop, passing map to TV.
      DO 100 I = 1,MY
         IY = I + IWIN(2) - 1
         CALL MDISK (REED, LUNRES, FIND, XBUFF, BIND, IERR)
         IF (IERR.NE.0) GO TO 110
C                                          row to AP
         CALL APPUT (XBUFF(BIND), FOUR, MX, N2)
         CALL APWD
C                                          clip at max, min
         CALL VCLIP (FOUR, N1, ZERO, ONE, FOUR, N1, MX)
C                                          scale, add constant
         CALL VSMSA (FOUR, N1, TWO, THREE, FOUR, N1, MX)
C                                          to integer (rounded)
         CALL VFIX (FOUR, N1, FOUR, N1, MX)
         CALL APWR
C                                          row back to core
         CALL APGET (JROW, FOUR, MX, N1)
         CALL APWD
C                                          Send row to TV.
         CALL YIMGIO (WRIT, ICH, IWIN, IY, N0, MX, JROW, IERR)
         IF (IERR.NE.0) GO TO 110
 100     CONTINUE
 110  CALL ZCLOSE (LUNRES, FIND, IERR)
C                                          Release the AP
      CALL BPRLSE
C                                          Image catalog
      CALL COPY (N256, CATBLK, S2H)
C                                          depth = 1 for 2-D image
      CALL FILL (N5, N1, S2H(I2DEP))
C                                          TV corners
      CALL COPY (N4, IWIN, S2H(I2COR))
C                                          image corners
      CALL COPY (N4, WIN, S2H(I2WIN))
C                                          scaling
      S2H(I2TRA) = IBLANK
      S8H(K8BSC) = 1.0D0
      S8H(K8BZE) = 0.0D0
      S4H(I4RAN) = TVFMIN
      S4H(I4RAN+1) = TVFMAX
      S4H(K4DMN) = TVREMN
      S4H(K4DMX) = TVREMX
C                                          => not disk file map
      S2H(I2VOL) = 0
      S2H(I2CNO) = 0
C                                          write to image catalog
      CALL ICWRIT (ICH, IWIN, S2H, XBUFF, IERR)
      IF (IERR.EQ.0) GO TO 120
         ENCODE (80,1110,MSGTXT)
         CALL MSGWRT (N6)
C                                          Ask user to quit?
 120  IF (TVPASS.LT.2) GO TO 998
         ENCODE (80,1120,MSGTXT)
         CALL MSGWRT (N1)
         ENCODE (80,1121,MSGTXT)
```

```
         CALL MSGWRT (N1)
         RPOS(1) = MAXXTV(1)/2.0
         RPOS(2) = MAXXTV(2)/2.0
         TD = 0.2
         CALL YCURSE (ON, F, F, RPOS, IQ, IB, IERR)
         IF (IERR.NE.0) GO TO 998
         DO 130 I = 1,75
            CALL ZDELAY (TD, IERR)
            CALL YCURSE (REED, F, F, RPOS, IQ, IB, IERR)
            IF (IB.GT.7) GO TO 140
            IF (IB.GT.0) GO TO 135
            IF (IERR.NE.0) GO TO 135
 130        CONTINUE
 135     ENCODE (80,1135,MSGTXT)
         CALL MSGWRT (N1)
         GO TO 150

C                                          Wants to quit
 140     TVPASS = 32700
         ENCODE (80,1140,MSGTXT)
         CALL MSGWRT (N3)
C                                          Off cursor
 150     CALL YCURSE (OFF, F, F, RPOS, IQ, IB, IERR)
 998  CALL TVCLOS (BUFF1, IERR)
C
 999  RETURN
C-------------------------------------------------------------------
 1000 FORMAT ('CANT OPEN TV IER=',I6)
 1010 FORMAT ('IMCLEAR ERROR =',I6)
 1020 FORMAT ('TVDISP: DISPLAY RANGE =',2F8.3,1X,A4,A1,'JY')
 1110 FORMAT ('CAN''T UPDATE IMAGE CATALOG  IER=',I6)
 1120 FORMAT ('HIT BUTTON D WITHIN 15 SECONDS TO STOP CLEANING NOW')
 1121 FORMAT ('HIT BUTTONS A, B, OR C TO CONTINUE SOONER')
 1135 FORMAT ('CONTINUING')
 1140 FORMAT ('TV BUTTON D HIT: HAVE DONE ENOUGH I GUESS')
      END
```

9.3.5 Plot Files (TVPL)

Plots in AIPS are usually produced as device independent plot files (see the chapter on plotting). The task which interprets such files and writes on the TV display is called TVPL. It will scale line drawings to fill the TV screen or, at the user's option, plot them at the original pixel scaling (converted to TV pixels). Grey-scale plot files are always done at pixel scaling. The character and vector portions of the plot are written to one of the graphics planes (chosen by the user) via subroutines IMVECT and IMCHAR. Grey-scale records, if any, are written via YIMGIO to the user-specified grey-scale memory. TVPL also updates the image catalogue as needed.

9.3.6 Transfer Function Modification, Zooming

Subroutine AU6A carries out the verbs OFFTRAN, TVTRAN, TVLUT, and TVMLUT which perform modifications on the black and white (or single color) LUTs of the specified gray-scale memories. OFFTRAN simply writes a linear, 0 through MAXINT array to the LUTs via YLUT. TVTRAN is implemented by the subroutine IENHNS which is also used by other verbs and tasks (e.g. TVFIDDLE, BLANK, TVMOVIE, TVBLINK). IENHNS allows a linear LUT with the cursor position controlling the slope and intercept and buttons allowing a switch in the sign of the slope and a continually updated plot of the LUT. TVLUT and TVMLUT allow the user to plot his own LUT function on a graphics plane with the cursor and the buttons. They both use the subroutine GRLUTS.

Subroutine AU6 implements the verbs OFFPSEUD, OFFZOOM, and OFFSCROL to clear the OFM, the zoom setting, and the scroll(s). It also implements interactive setting of the zoom factor and center (verb TVZOOM), of individual channel scrolls (TVSCROLL), and of the pseudo-color OFM (TVPSEUDO). OFFPSEUD simply sends a linear OFM to all colors via YOFM; OFFZOOM sends a 0 zoom factor via YZOOMC, and OFFSCROL sends a 0 scroll via YSCROL. TVZOOM makes considerable use of YCURSE and YZOOMC, while TVPSEUDO uses YCURSE and alternately IMLCLR (RGB color triangle), IMPCLR (circle in hue), and IMCCLR (color contours). AU6 also implements a much more complicated enhancement algorithm in which one gray-scale channel is used to set the intensity and another to set the hue. This algorithm requires the TV to have both LUTs for each channel and an OFM for the sum of the enabled channels. A log function is put in the LUTs and an exponential in the OFM which carries out the required multiplication of the two signals. Subroutines HIENH and HILUT actually carry out most of the algorithm including interactive enhancements (via an algorithm similar to IENHNS) and switching of the roles of the two channels.

One of the most commonly used image enhancement routines is TVFIDL. It is called by the verb TVFIDDLE via subroutine AU6C and task BLANK. It is a deliberately limited interactive routine designed to provide easy to use enhancement in black and white (via IENHNS) or pseudocolor (via IMCCLR with a single type of color contour). A simple zoom procedure is also provided. During image enhancement the cursor position controls slope and intercept and during zoom the cursor position controls zoom center. Button A (value 1) alternately selects color and black and white enhancement, button B/C increments/decrements the zoom and selects zoom mode. As in all interactive algorithms, button D (values >= 8) terminates the function.

The algorithm for TVSCROLL is a good example to present in detail since the action required when the cursor moves is quite simple. The most important thing to notice below is the routine DLINTR. This routine tests the output of YCURSE to see if anything has changed. If not, it delays the program by some period of time which increases slowly as the time since the last change increases. Without this algorithm, the tight loop on reading the TV cursor is capable of jamming the CPU and I/O channels especially when the user does not move the cursor.

```
C                                              open TV device
      CALL TVOPEN (BUFFER, IERR)
C                                              get start time
      CALL ZTIME (ITW)
      IF (IERR.EQ.0) GO TO 10
         POTERR = 101
         GO TO 980
......
C                                              TVSCROL
C                                              user instructions
 500  ENCODE (80,1500,MSGTXT)
      CALL MSGWRT (N1)
      ENCODE (80,1505,MSGTXT)
      CALL MSGWRT (N1)
C                                              find channel(s) to scroll
C                                              scroll graphics too ?
      IC = ABS(TVCHAN) + EPS
      CALL DECBIT (NGRAY, IC, IC, J)
      IF (ABS(GRCHAN).GT.EPS) IC = IOR (IC, GRPHIC)
      IF (IC.NE.0) GO TO 505
         IC = MOD (TVLIMG(1), N2**NGRAY)
         IF (IC.NE.TVLIMG(1)) IC = IOR (IC, GRPHIC)
 505  IX = 0
      IY = 0
      RPOS(1) = MAXXTV(1)/2
      RPOS(2) = MAXXTV(2)/2
C                                              turn on cursor
      CALL YCURSE (ON, F, F, RPOS, QUAD, IBUT, IERR)
      IF (IERR.NE.0) GO TO 900
C                                              force scroll
 510  CALL YSCROL (IC, IX, IY, T, IERR)
      IF (IERR.NE.0) GO TO 900
         PPOS(1) = RPOS(1)
         PPOS(2) = RPOS(2)
C                                              read until cursor moves
 520     CALL YCURSE (READ, F, F, RPOS, QUAD, IBUT, IERR)
         IF (IERR.NE.0) GO TO 900
C                                              test for change
         CALL DLINTR (RPOS, IBUT, F, QUAD, PPOS, ITW, DOIT)
         IF (.NOT.DOIT) GO TO 520
C                                              cursor moved, change scroll
            IX = RPOS(1) - MAXXTV(1)/2
            IY = RPOS(2) - MAXXTV(2)/2
C                                              any button => done
            IF (IBUT.EQ.0) GO TO 510
               POTERR = 0
               GO TO 900
......
C                                              close down
C                                              cursor off, TV closed
 900  IF (BRANCH.GE.4) CALL YCURSE (OFF, F, F, RPOS, QUAD, IBUT, JERR)
 910  CALL TVCLOS (BUFFER, JERR)
```

9.3.7 Object Location, Window Setting

Subroutine AU5 performs the verbs TVPOS, IMXY, IMPOS (see below), and TVNAME (via TVFIND) as well as a variety of status setting verbs. IMPOS is implemented as follows. It calls TVWHER to find the cursor position indicated by the user. Then it checks all enabled memories via ICREAD to see if there is an image displayed at that pixel position. Finally, it calls MP2SKY to set up the coordinate commons and get the primary positions and goes through some other messy stuff to display the results to the user.

```
      CALL TVOPEN (CATBLK, JERR)
      IF (JERR.EQ.0) GO TO 50
         POTERR = 101
         GO TO 980
......
C                                          IMPOS
C                                          read cursor to get position
 600  CALL TVWHER (IQUAD, RPOS, IBUT, JERR)
      IF (JERR.NE.0) GO TO 975
C                                          image pix -> map pixel pos
 625  IX = RPOS(1) + EPS
      IY = RPOS(2) + EPS
C                                          Find lowest plane with x,y
      IN2 = NGRAY + NGRAPH
      DO 630 IP = 1,IN2
C                                          skip off channels
         IF (IAND (TVLIMG(IQUAD), N2**(IP - N1)).EQ.0) GO TO 630
C                                          get img cat block
         CALL ICREAD (IP, IX, IY, CATBLK, IERR)
C                                          loop if x,y not in image
         IF (IERR.EQ.N1) GO TO 630
         IF (IERR.EQ.0)  GO TO 650
         GO TO 975
 630     CONTINUE
C                                          x,y not in on image
      ENCODE (80,1630,MSGTXT) IX, IY
      CALL MSGWRT (N6)
      GO TO 900
C                                          image -> map positions
 650  CALL IMA2MP (RPOS, RPOS)
      ENCODE (80,1650,MSGTXT) RPOS
      CALL MSGWRT (N5)
C                                          map -> sky positions
 660  CONTINUE
         CALL MP2SKY (RPOS, SKYPOS)
C                                          3rd axis pairs w 1st or 2nd
         IF ((AXTYP.EQ.2) .OR. (AXTYP.EQ.3)) CALL AXSTRN (CTYP(1,3),
     *      SKYPOS(3), KLOCA, NCHLAB(1), SAXLAB(1,1))
C                                          Primary axes
C                                          Tell user results via MSGWRT.
         ENCODE (80,1660,MSGTXT)
         ICH = 8
         DO 665 I = 1,2
            CALL AXSTRN (CTYP(1,I), SKYPOS(I), I-N1, ILEN, RSTR)
            CALL CHPACK (ILEN, RSTR, ICH, MSGTXT)
```

```
            ICH = ICH + ILEN
            CALL CHFILL (N2, RBLANK, ICH, MSGTXT)
            ICH = ICH + 2
 665        CONTINUE
         ILEN = 81 - ICH
         CALL CHFILL (ILEN, RBLANK, ICH, MSGTXT)
         CALL MSGWRT (N5)
C                                          Secondary axes values
         IF ((NCHLAB(1).LE.0) .AND. (NCHLAB(2).LE.0)) GO TO 900
            ICH = 8
            DO 670 I = 1,2
               IF (NCHLAB(I).LE.0) GO TO 670
                  CALL CHPACK (NCHLAB(I), SAXLAB(1,I), ICH, MSGTXT)
                  ICH = ICH + NCHLAB(I)
                  CALL CHFILL (N2, RBLANK, ICH, MSGTXT)
                  ICH = ICH + 2
 670           CONTINUE
            ILEN = 81 - ICH
            CALL CHFILL (ILEN, RBLANK, ICH, MSGTXT)
            CALL MSGWRT (N5)
            GO TO 900
......
C                                          normal TV close
 900  CALL TVCLOS (CATBLK, JERR)
      GO TO 999
```

The interactive window setting verbs TVWIN, TVBOX, TVSLICE, and REBOX are initiated from subroutine AU5C and performed primarily by subroutine GRBOXS. This routine is another instance of interactivity via YCURSE and line drawing via IMVECT. It uses YCUCOR at the end to obtain the image catalogue header and thence, to correct the cursor positions to map pixel locations.

CURVALUE is an interactive verb which displays on a TV graphics channel the position and image value of the pixel currently under the TV cursor. It is implemented by subroutine AU6B. The image values are read from the original map files on disk, if possible, using MAPOPN, MINIT, and MDISK. However, the intensities of step wedges and temporary images (i.e. intermediate residual maps displayed by APCLN) are read from the TV memory via YIMGIO. The routine makes extensive use of IMCHAR and, although too long to reproduce here, is an interesting example of AIPS image plus TV coding.

9.3.8 Blotch Setting, Use

A "blotch" is a region within an image over which some action is to be performed. Pixels outside the blotch are ignored or have some alternative action performed on them. At present, AIPS has two functions which generate and use blotches: the verb TVSTAT which returns image statistics within the blotch area and the task BLANK which blanks out all pixels outside the blotch. In both, the user uses the TV cursor to set the vertices of one or more polygonal areas and the routines draw lines on a graphics plane between the vertices. When the user is done, the routines fill in the blotch areas on the TV graphics and then read and act on the map file. Subroutine AU6D implements TVSTAT for whatever image is visible on the TV, obtaining the polygons through subroutine GRPOLY. AU6D itself does the data reading, determination of whether a pixel is inside or outside the blotch, and the computation and display of the image statistics. Task BLANK uses internal subroutines BLNKTV and BLKTVF to display the image (via TVLOAD), allow transfer modification (via TVFIDL), to obtain the polygons (BLKTVF), and to use them to blank the output image (BLNKTV). The subroutine BLTFIL does the filling of the polygons on the TV graphics screen for both TVSTAT and BLANK.

9.3.9 Roam

Roam is mode of display which requires multiple gray-scale memories and the capability to do split screen and scroll. Adjacent portions of the image are loaded into separate image memories. Then the screen is split horizontally and/or vertically and the appropriate memories are enabled in each quadrant each with scroll. This allows the user to view a screen-size portion of a rather larger image. By shifting the scroll and split point interactively, the user may select which portion is viewed. Roam is implemented in AIPS from the subroutine AU5A. This routine loads the image to the TV memories in a manner similar to TVLOD (above). However, it uses TVWIND to determine a much more complicated window and must itself play with windows further before calling TVLOAD. The interactive portion of the Roam is carried out by AU5A calling subroutine TVROAM. That routine can handle images of up to 1 x 4, 4 x 1, or 2 x 2 planes and uses YCURSE for interactive input, YSCROL to set the scroll (identical for all planes), and YSPLIT to set the split point and enable the appropriate channels. A zoom option is also available.

9.3.10 Movie, Blink

The verb TVMOVIE is a very interesting algorithm implemented via subroutines AU5D and TVMOVI. A movie is a method of displaying a 3-dimensional image as a time sequence of 2-dimensional planes. Each gray-scale TV memory is subdivided into a 2 x 2, 4 x 4, or 8 x 8 matrix of images of consecutive planes of the cube. During the display phase, the zoom factor is set to 2, 4, or 8, respectively, so that only one plane is visible at a time. The zoom center is moved from frame to frame at a user controlled rate to simulate a movie. Subroutine AU5D determines which zoom factor and windows to use, zeros the gray-scale memories, loads the planes to the TV (via TVLOAD), transfers the LUT of the first TV memory to the other TV memories, draws border lines around each plane (via IMVECT), annotates each plane with the 3rd coordinate axis value, and puts a small pointer in the image as well. TVMOVI executes an interactive alogorithm in which the cursor controls the frame rate and the buttons allow a single frame at a time mode and interactive enhancement of the LUTs (via IENHNS) or the OFM (via IMCCLR). The verb REMOVIE is also done by AU5D and TVMOVI using the stored parameters which describe how the movie was loaded to the TV memories (parameter TYPMOV in the /TVCHAR/ common).

The subroutines AU6A and TVBLNK implement the verbs TVBLINK and TVMBLINK. Blinking is simply enabling one gray-scale memory for a while, then disabling it and enabling another for a second period of time, then disabling the second channel and re-enabling the first, and so on. These two verbs allow manual as well as timed switching between the two planes and transfer function modification via the subroutine IENHNS (see above).

9.3.11 Non-standard Tasks

There are a number of tasks in AIPS which are seriously non-standard in their coding and in their use of various devices. Among these are several which use the TV display. We will list them here briefly. Programmers should not use these tasks as models of how to code in AIPS and should not assume that they can even be made to run on non-VMS, non-IIS systems.

- IMLHS uses up to 3 maps to create a false color image on the TV. It uses the first map to modulate the brightness of the image, the 2nd to modulate the hue and the 3rd to modulate the saturation. If any of the images are omitted the corresponding parameter is set to a constant. (Note: verb TVHUEINT is standard and does a similar function with two images.)

- TVHLD loads up to 13-bit image to two TV memories and performs an interactive histogram equilization of the display. Can feed the result back to a 3rd TV memory. This task uses YRHIST, YALUCT, YFDBCK, YIFM, and the dual-channel mode of the IIS and will be hard to implement on TV display devices other than the IIS.

- TVHXF does an interactive histogram equilization of the image which is currently displayed. This task uses YRHIST which is currently IIS specific. However, a TV-independent (but SLOW) YRHIST can be coded is someone wishes to do the work.

- TVSLV loads an image, prepared by tasks TVCUB and TVSLD, to the TV. The image is a 3-dimensional representation of a data cube.

- UVDIS attempts to take an FFT of an image and display the complex results on the TV as intensity and color-encoded phase.

9.4 INCLUDES

9.4.1 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, TVDUMS(7),
     *   TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *   TVSPLM, TVSPLC, TYPMOV(16), YBUFF(168)
C                                                       End DTVC
```

9.4.2 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, TVDUMS, TVZOOM,
     *   TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC, TYPMOV,
     *   YBUFF
C                                                       End CTVC
```

9.4.3 DTVD.INC

```
C                                                       Include DTVD
      INTEGER*2 TVLUN, TVIND, TVLUN2, TVIND2, TVBFNO
      LOGICAL*2 TVMAP
C                                                       End DTVD
```

9.4.4 CTVD.INC

```
C                                                          Include CTVD
      COMMON /TVDEV/ TVLUN, TVIND, TVLUN2, TVIND2, TVBFNO, TVMAP
C                                                          End CTVD
```

9.5 Y-ROUTINE PRECURSOR REMARKS:

9.5.1 Level 0

9.5.1.1 YCHRW - writes characters into image planes of the TV. The format is 5 by 7 with one blank all around: net 7 in X by 9 in Y This version will work on all TVs which allow horizontal writing to the right. It is a Y routine to allow for hardware character generators on some TVs.

```
      YCHRW (CHAN, X, Y, COUNT, STRING, SCRTCH, IERR)

   Inputs: CHAN    I*2      channel select  (1 to NGRAY + NGRAPH)
           X       I*2      X position lower left corner first char.
           Y       I*2      Y position lower left corner first char.
           COUNT   I*2      number of characters in STRING
           STRING  R*4      character string
   Output: SCRTCH  I*2(>)   scratch buffer (dim = 14*count+8 < 1031)
           IERR    I*2      error code of Z...XF:0 - ok
                                                 2 - input error
```

9.5.1.2 YCNECT - writes a line segment on the TV. This version will work on all TVs. It is called a Y routine to allow the use of hardware vector generators on those TVs equiped with them.

```
      YCNECT (X1, Y1, X2, Y2, IC, BUFFER, IERR)

    Inputs: X1      I*2       start X position
            Y1      I*2       start Y position
            X2      I*2       end X position
            Y2      I*2       end Y position
            IC      I*2       Channel (1 to NGRAY+NGRAPH)
            BUFFER  I*2(512)  BUFFER(1 - 512) contains desired
                              intensity (size here for I2S)
    Output: IERR    I*2        error code : 0 => ok
```

9.5.1.3 YCUCOR - takes a cursor position (corrected for zoom, but not scroll) corrects it for scroll, determines the quadrant of the TV, and gets the corresponding image header in common /MAPHDR/ and returns the image coordinates.

```
      YCUCOR (RPOS, QUAD, CORN, IERR)

   Inputs: RPOS    R*4(2)  X,Y screen pos before  zoom & scroll
   Output: QUAD    I*2     TV quadrant to use for scrolls
                           Out: if in=-1, no scroll, else find
                           quadrant (needs real TV pos)
           CORN    R*4(7)  Image coordinates (pixels)
           IERR    I*2     error code of Z...XF : 0 - ok
                                                  2 - input error
```

9.5.1.4 YCURSE - reads cursor positions and controls the blink and visibility of the TV cursor.

```
      YCURSE (OP, WAIT, CORR, RPOS, QUAD, EVTMOD, IERR)

   Inputs: OP      R*4     'READ' read cursor position
                           'ONNN' place cursor at RPOS & leave on
                           'OFFF' turn cursor off
                           'BLNK' reverse sense of cursor blink
           WAIT    L*2     wait for event & return RPOS & EVTMOD
                           (done on all OPs)
           CORR    L*2     T => correct RPOS for zoom & scroll
   In/Out: RPOS    R*4(2)  X,Y screen pos before  zoom & scroll
           QUAD    I*2     TV quadrant to use for scrolls
                           In: if <1 >4, no scroll
                           Out: if in=-1, no scroll, else find
                           quadrant (needs real TV pos)
   Output: EVTMOD  I*2     event # (0 none, 1-7 low buttons,
                           8-15 the "quit" button)
           IERR    I*2     error code of Z...XF : 0 - ok
                                                  2 - input error
```

9.5.1.5 YGRAPH - is used to turn graphics overlay planes on and off by altering the graphics color look up table. The color pattern is:

CHAN = 1	insert	yellow	drawing plots
2	insert	green+.05 red	axis labels
3	insert	blue + 0.6 green + red	blotch
4	insert	black	label backgrounds
5-7	add	nothing	null channels
8	insert	purple	cursor

```
   YGRAPH (OP, CHAN, SCRTCH, IERR)

Inputs: OP      R*4       'ONNN' or 'OFFF'
        CHAN    I*2       channel number (1 - 8)
Output: SCRTCH  I*2(256)  scratch buffer
        IERR    I*2       error code of Z...XF: 0 => ok
                                                2 => input error
```

9.5.1.6 YLNCLR - computes a piecewise linear OFM and writes it to the TV. If NEND(NPOINT) is 256 (512) then the OFM is repeated 4 (2) times.

```
   YLNCLR (COLOR, NPOINT, NEND, SLOPE, OFFSET, GAMMA, BUFFER, IERR)

Inputs: COLOR   I*2          color bit mask: RGB = 421
        NPOINT  I*2          # of segments
        NEND    I*2          end points of segments
        SLOPE   R*4(NPOINT)  slopes of segments
        OFFSET  R*4(NPOINT)  offsets of segments
        GAMMA   R*4          power applied to colors (1 /gamma)
Output: BUFFER  I*2(1024)    scratch buffer
        IERR    I*2          error code of Z...XF : 0 - ok
Form is C = (i-1)*SLOPE + OFFSET with 0 <= C <= 1.0.
```

9.5.1.7 YSLECT - enables and disables gray and graphics planes.

```
   YSLECT (OP, CHAN, COLOR, BUFFER, IERR)

Inputs: OP      R*4       'ONNN' or 'OFFF'
        CHAN    I*2       channel number ( 1 to NGRAY+NGRAPH)
        COLOR   I*2       0 - all,   1,2,3 = R,G,B, resp.
Output: BUFFER  I*2(256)  scratch buffer (for graphics only)
        IERR    I*2       error code of Z...XF: 0 - ok
                                                2 - input err
YSLECT sets TVLIMG in the TV device parms common /TVDEV/
```

9.5.1.8 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.
NOTE: These are default values only. They are reset to the current true values by a call to TVOPEN.
NOTE: YTVCIN resets the common values of TVZOOM and TVscroll, but does not call the TV routines to force these to be true. A separate call to YINIT or YZOOMC and YSCROL is needed.

```
YTVCIN
(no arguments)
```

9.5.1.9 YZERO - fills an TV memory plane with zeros the fast way. Note: this is equivalent to YINIT, but avoids linking with all the routines called by the main parts of YINIT.

```
YZERO (CHAN, IERR)

Inputs:  CHAN   I*2   channel # ( 1 - NGRAY+NGRAPH), 0 => all
Outputs: IERR   I*2   error code of Z...XF: 0 - ok
                                            2 - input error
```

9.5.1.10 YTVCLS - closes TV device associated with LUN removing any EXCLusive use state and clears up the FTAB.

```
YTVCLS (LUN, IND, IERR)

Inputs:  LUN    logical unit number
         IND    pointer into FTAB
Output:  IERR   error code: 0 -> no error
                            1 -> Deaccess or Deassign error
                            2 -> file already closed in FTAB
                            3 -> both errors
                            4 -> erroneous LUN
```

9.5.1.11 YTVMC - issues a "master clear" to the TV. This resets the TV I/O system (if necessary) to expect a command record next. YTVMC gets all needed parameters from the TV device common. The TV must already be open.

```
YTVMC
(no arguments)
```

9.5.1.12 YTVOPN - performs a system "OPEN" on the TV device. It is a Y routine in order to call the appropriate Z routine only.

```
YTVOPN (LUN, IND, IERR)

Inputs:  LUN    I*2   Logical unit number to use
Output:  IND    I*2   Pointer to FTAB entry for open device
         IERR   I*2   Error code: 0 => ok
                      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
```

9.5.2 Level 1

9.5.2.1 YCRCTL - reads/writes the cursor/trackball control register of TV.

```
YCRCTL (OP, ON, X, Y, LINKX, LINKY, RBLINK, BUTTON,
*   VRTRTC, IERR)

Inputs:  OP      R*4   'READ' from TV or 'WRIT' to TV
         VRTRTC  L*2   T => do on vertical retrace only
In/Out:  ON      L*2   T => cursor visible, F => off
         X       I*2   X position cursor center (1-512, 1 => LHS)
         Y       I*2   Y position cursor center (1-512, 1 => bot)
         LINKX   L*2   T => trackball moves cursor in X
         LINKY   L*2   T => trackball moves cursor in Y
         RBLINK  I*2   rate of cursor blink: 0-3 no-fast blink
Output:  BUTTON  I*2   button value (0 - 15)
         IERR    I*2   error code of Z...XF : 0 => ok
                                              2 => input error
```

9.5.2.2 YIMGIO - reads/writes a line of image data to the TV screen. For graphics overlay planes, the data are solely 0's and 1's in the least significant bit of IMAGE after a READ. For WRIT, all bits of each word should be equal (i.e. all 1's or all 0's for graphics). NOTE***** on WRIT, the buffer may be altered by this routine for some IANGLs.

```
      YIMGIO (OP, CHAN, X, Y, IANGL, NPIX, IMAGE, IERR)

   Inputs: OP      R*4          'READ' from TV or 'WRIT' to TV
           CHAN    I*2          channel number (1 to NGRAY+NGRAPH)
           X       I*2          start pixel position
           Y       I*2          end pixel position
           IANGL   I*2          = 0 => horizontal (to right)
                                = 1 => vertical (up the screen)
                                = 2 => horizontal (to left)
                                = 3 => vertical (down the screen)
           NPIX    I*2          number of pixels
   In/Out: IMAGE   I*2(NPIX)    data (only no header)
   Output: IERR    I*2          error code of Z...XF - 0 => ok
                                                       2 => input err
```

9.5.2.3 YINIT - initializes the TV subunits: doing everything.

```
      YINIT (SCRTCH, IERR)

   Output: SCRTCH  I*2(1024)  scratch buffer (can be 256 for CHAN &
                              1 for ZERO & REST)
           IERR    I*2        error code of Z...XF - 0 => ok
                                                     2 => input error
```

9.5.2.4 YLUT - reads/writes full channel look up tables to TV.

```
      YLUT (OP, CHANNL, COLOR, VRTRTC, LUT, IERR)

   Inputs: OP      R*4        'READ' from TV, 'WRIT' to TV
           CHANNL  I*2        channel select bit mask
           COLOR   I*2        color select bit mask (RGB <-> 421)
           VRTRTC  L*2        T => do it only during vertical retrace
   In/Out: LUT     I*2(256)   look up table (ls 9 bits used)
   Out:    IERR    I*2        error code of Z...XF : 0 => ok
```

9.5.2.5 YOFM - reads/writes full OFM look up tables to TV.

```
      YOFM (OP, COLOR, VRTRTC, OFM, IERR)

   Inputs: OP      R*4        'READ' from TV, 'WRIT' to TV
           COLOR   I*2        color select bit mask (RGB <-> 421)
           VRTRTC  L*2        T => do it only during vertical retrace
   In/Out: OFM     I*2(1024)   look up table (ls 10 bits used)
   Out:    IERR    I*2        error code of Z...XF : 0 => ok
```

9.5.2.6 YSCROL - writes the scroll registers on the TV.

```
      YSCROL (CHANNL, SCROLX, SCROLY, VRTRTC, IERR)

   Inputs: CHANNL  I*2   bit map channel select
           VRTRTC  L*2   T => do it on vertical retrace only
   In/Out: SCROLX  I*2   amount of X scroll (>0 to right)
           SCROLY  I*2   amount of Y scroll (>0 upwards)
   Output: IERR    I*2   error from Z...XF : 0 => ok
   YSCROL updates the scroll variables in /TVDEV/ common
```

9.5.2.7 YSPLIT - reads/writes the look up table/ split screen control registers of the TV.
Quadrants are numbered CCW from top right.

```
      YSPLIT (OP, XSPLT , YSPLT , RCHANS, GCHANS, BCHANS,
     *   VRTRTC, IERR)

   Inputs: OP      R*4     'READ' from TV, 'WRIT' to TV
           VRTRTC  L*2     T => do on vertical retrace only
   In/Out: XSPLT   I*2     X position of split (1-512, 1 => LHS)
           YSPLT   I*2     Y position of split (1-512, 1 => bot)
           RCHANS  I*2(4)  chan select bit mask 4 quadrants : red
           GCHANS  I*2(4)  chan select bit mask 4 quadrants : green
           BCHANS  I*2(4)  chan select bit mask 4 quadrants : blue
   Output: IERR    I*2     error code of Z...XF: 0 => ok
                                                 2 => input error
```

9.5.2.8 YZOOMC - writes (ONLY!) the zoom control registers of the TV.

```
   YZOOMC (MAG, XZOOM, YZOOM, VRTRTC, IERR)

Inputs: MAG     I*2   0-3 for magnification 1,2,4,8 times, resp.
        XZOOM   I*2   X center of expansion (1-512, 1 => LHS)
        YZOOM   I*2   Y center of expansion (1-512, 1 => bot)
Output: IERR    I*2   error code of Z...XF: 0 -> ok
                                            2 -> input error
YZOOMC updates the /TVDEV/ common TVZOOM parameter
```

9.5.3 Level 2 (Used As Level 1 In Non-standard Tasks)

9.5.3.1 YALUCT - reads / writes the TV arithmetic logic unit control registers. The actual feedback-ALU computation is performed only upon a call to YFDBCK.

```
   YALUCT (OP, ARMODE, BFUNC, NFUNC, CONSTS, OUTSEL,
  *   EXTOFM, ESHIFT, SHIFT, CARYIN, CARRY, EQUAL, IERR)

Inputs: OP      R*4      'READ' from TV or 'WRIT' to TV
In/Out: ARMODE  L*2      T => arithmetic mode F => logic mode
        BFUNC   I*2      function number (1-16) in blotch
        NFUNC   I*2      function number (1-16) outside blotch
        CONSTS  I*2(8)   constant array (may select as ALU output)
        OUTSEL  I*2(8)   lookup table selects output based on carry
                         (lsb), equal, ROI (msb) input. values -
                         0 - 7 : constants 1 - 8
                         8     : accumulator channel pair
                         9     : selected OFM
                         10    : ALU
                         11    : external
        EXTOFM  L*2      T => extend sign of OFM on input to ALU
        ESHIFT  L*2      T => extend sign of ALU output if SHIFT
        SHIFT   L*2      T => right shift ALU output
        CARYIN  L*2      T => add one to arithmetic results
Output: CARRY   L*2      T => carry condition occurred in frame
        EQUAL   L*2      T => equal condition occurred in frame
        IERR    I*2      error code of Z...XF : 0 - ok
                                                2 - input error
```

9.5.3.2 YFDBCK - sends a feedback command to the TV.

```
      YFDBCK (COLOR, CHANNL, BITPL, PIXOFF, BYPIFM, EXTERN,
     *   ZERO, ACCUM, ADDWRT, IERR)

   Inputs:  COLOR   I*2  bit map of color to be fedback (RGB = 4,2,1)
            CHANNL  I*2  bit map of channels to receive feedback
            BITPL   I*2  bit map of bit planes to receive feedback
            PIXOFF  I*2  offset fedback image to left by 0 - 1 pixels
    NOTE: I2S literature claims only 1 bit here not the three
          that their software (NOT this routine) uses.
            BYPIFM  L*2  F => image goes thru IFM lookup before store
            EXTERN  L*2  T => image from external input (iedigitizer)
            ZERO    L*2  T => feed back all zeros
            ACCUM   L*2  T => use 16-bit accumulator mode
                         then CHANNL must give even-odd pair lsbyte
                         goes to even (lower) # channel
            ADDWRT  L*2  T => additive write  F => replace old data
   Outputs: IERR    I*2  error code of Z...XF: 0 -> ok
                                               2 -> input error
```

9.5.3.3 YGYHDR - builds an TV header to write image data. The actual I/O must be done by calls to Z...XF.

```
      YGYHDR (OP, NPIXEL, XINIT, YINIT, IANGLE, CHANNL,
     *   PLANES, PACKED, BYPIFM, BYTE, ADDWRT, ACCUM, VRTRTC, HEADER,
     *   IERR)

   Inputs: OP      R*4     'READ' from TV or 'WRIT' to TV
           NPIXEL  I*2     number of pixel values to I/O
           XINIT   I*2     first pixel X coordinate (1-512, 1 -> LHS)
           YINIT   I*2     first pixel Y coordinate (1-512, 1 -> bot)
           IANGLE  I*2     (0 => data I/O horizontal to right, 1 =>
                           up, 2 => to left, 3 => down)
           CHANNL  I*2     channel select bit mask
           PLANES  I*2     bit plane select bit mask
           PACKED  L*2     T => 2 values/word, F => 1 value/word
           BYPIFM  L*2     F => IFM lookup applied to data (write)
           BYTE    L*2     T => 8 values/byte (needs XINIT = 8*n+1)
           ADDWRT  L*2     T => OR data with present memory contents
           ACCUM   L*2     T => use 16-bit accumulator mode
           VRTRTC  L*2     T => do it only during vertical retrace
   Output: HEADER  I*2(8)  header to be sent to TV
           IERR    I*2     error code of 0 => ok
                                         2 => input error
```

9.5.3.4 YIFM - reads/writes a section of TV input function memory. This look up table takes 13 bits in and gives 8 bits out.

```
      YIFM (OP, START, COUNT, PACK, VRTRTC, IFM, IERR)

   Inputs: OP      R*4      'READ' from TV or 'WRIT' to TV
           START   I*2      start address of IFM ( 1 - 8192)
           COUNT   I*2      # elements of IFM to transfer (1-8192)
           PACK    L*2      T => 2 values/word, F => 1 value/word
           VRTRTC  L*2      T => do it only on vertical retrace
   In/Out: IFM     I*2(>)   function values (0-255)
   Output: IERR    I*2      error code of Z...XF: 0 - ok
                                                  2 - input error
```

9.5.3.5 YRHIST - reads the histogram of the output of a selected OFM of the TV.
**** Warning: the results are 18-bit integers stored in a standard AIPS pseudo I*4 order (ls 16 bits in first word).

```
      YRHIST (MODE, COLOR, INITI, NINT, HISTOG, IERR)

   Inputs: MODE    I*2          selects area to histogram: 0 blotch,
                                1 not blotch, 2 all, 3 external bltch
           COLOR   I*2          bit map of single color (RGB - 4,2,1)
           INITI   I*2          first intensity to histo (1 - 1024)
           NINT    I*2          # values to get
   Output: HISTOG  I*2(2*NINT)  histogram
           IERR    I*2          error code of Z...XF : 0 => ok
                                                       2 => input err
```

9.5.4 Selected Applications Subroutines

9.5.4.1 TVOPEN - opens the TV, passing pointers through common /TVDEV/.

```
      TVOPEN (BUF, IERR)

   OUTPUTS: BUF   I*2(256)   Scratch buffer
            IERR  I*2        Error return from ZOPEN
                             = 10 TV unavailable to this version
```

9.5.4.2 TVCLOS - closes the TV device and the TV status disk file, updating the information on the disk.

```
      TVCLOS (BUF, IERR)

   Outputs: BUF    I*2(256)   Scratch buffer
            IERR   I*2        Error code : 0 => ok
                              else as returned by ZFIO
```

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

```
      TVFIND (MAXPL, TYPE, IPL, UNIQUE, CATBLK, SCRTCH,
     *   IERR)

   Inputs: MAXPL    I*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 => I/O error in image catalog
                                   3 => TV error
```

9.5.4.4 TVWIND - sets windows for normal and split screen TV loads.

```
      TVWIND (TYPE, PXINC, BLC, TRC, ICHAN, ITVC, IWIN,
     *   IERR)

   In/out : TYPE    I*2        In: <0 -> 1 plane, other -> split method
                               Out: 0 -> 1 plane, other = 10 * (#planes
                                    in X) + (# planes in Y)
            PXINC   I*2(2)     X, Y increments
            BLC     R*4(7)     User requested bot left corner
            TRC     R*4(7)     User requested top right corner
            ICHAN   I*2        User requested TV chan (decimal form)
            ITVC    I*2(4)     IN: first 2 user req. TVCORN
                               Out: full "pseudo-TV" corners
   Output:  IWIN    I*2(4)     Window into map
            IERR    I*2        error code: 0 -> ok, else fatal
   Common: /MAPHDR/ CATBLK     image header used extensively, the
                               depth array is set here
```

9.5.4.5 TVLOAD - loads a map from an already opened map file to one TV memory plane. TVLOAD puts TV and map windows in the image header and writes it in the image catalog. It assumes that the other parts of the image header are already filled in (and uses them) and that the windows are all computed.

```
      TVLOAD (LUN, IND, IPL, PXINC, IMAWIN, WIN, BUFSZ,
     *   IERR)

   Inputs:
      LUN       I*2     Logical unit # of map file
      IND       I*2     FTAB pointer for map file
      IPL       I*2     Channel to load
      PXINC     I*2(2)  Increment in x,y between included pixels
      IMAWIN    I*2(4)  TV corners: BLC x,y TRC x,y
      WIN       I*2(4)  Map window: ""
      BUFSZ     I*2     Buffer size in bytes
   Outputs
      IERR      I*2     Error code: 0 => ok
                                    1 => input errors
                                    2 => MINIT errors
                                    3 => MDISK errors
   Commons: /MAPHDR/ CATBLK  image header
            /IMBUF / BUFF    work space for I/O
```

9.5.4.6 TVFIDL - does an interactive run with button A selecting alternately TVTRANSF and TVPSEUDO (color contour type 2 only), button B incrementing the zoom and C decrementing the zoom.

```
      TVFIDL (ICHAN, NLEVS, IERR)

   Inputs: ICHAN     I*2     Selected gray-scale channel
           NLEVS     I*2     Number of gray levels (usually MAXINT+1)
   OUTPUT: IERR      I*2     Error code: 0 -> ok
                             else set by Z...XF
```

9.5.4.7 IMANOT - is used to annotate an image by writing the string into the lettering plane (usually graphics plane 2) and, if possible writing a block of ones NEDGE pixels wider than the string into graphics plane 4 to force a black background.

```
      IMANOT (OP, X, Y, IANGL, CENTER, COUNT, STRING,
     *   SCRTCH, IERR)
```

Inputs:	OP	R*4	'ONNN' enables the 2 graphics planes
			'OFFF' disables the 2 planes
			'INIT' zeros and enables the 2 planes
			'WRIT' writes strings to the planes
	X	I*2	X position of string
	Y	I*2	Y position of string
	IANGL	I*2	0 - horizontal, 3 - vertical (DOWN)
	CENTER	I*2	0 - XY are lower left first character
			1 - XY are center of string
			2 - XY are top right of last character
	COUNT	I*2	number of characters in STRING
	STRING	R*4()	character string
Output:	SCRTCH	I*2(>)	scratch buffer (>256, 14*count)
	IERR	I*2	error code of Z...XF : 0 - ok
			2 - input error

9.5.4.8 IMCHAR - causes characters to appear on the TV by calling IMCHRW.

```
      IMCHAR (CHAN, X, Y, IANGL, CENTER, COUNT, STRING,
     *   SCRTCH, IERR)
```

Inputs:	CHAN	I*2	channel number (1 - NGRAY+NGRAPH)
	X	I*2	X position of string
	Y	I*2	Y position of string
	IANGL	I*2	0 - horizontal (to right), 3 - vertical (down) ONLY ones supported.
	CENTER	I*2	0 - XY are lower left of first character
			1 - XY are center of string
			2 - XY are upper right of last character
	COUNT	I*2	number of characters in STRING
	STRING	R*4()	character string to go to TV
Output:	SCRTCH	I*2(>)	scratch buffer (14*count)
	IERR	I*2	error code of Z...XF: 0 - ok
			2 - input error

9.5.4.9 IMVECT - writes a connected sequence of line segments on a TV channel calling YCNECT

```
IMVECT (OP, CHAN, COUNT, XDATA, YDATA, SCRTCH, IERR)

Inputs: OP      R*4          'ONNN' line of ones (max intensity)
                             'OFFF' line of zeros (min intensity)
        CHAN    I*2          channel number (1 to NGRAY+NGRAPH)
        COUNT   I*2          number of X,Y pairs ( > 1)
        XDATA   I*2(COUNT)       X coordinates X1,X2,...
        YDATA   I*2(COUNT)       Y coordinates Y1,Y2,...
Output: SCRTCH  I*2(512)         scratch buffer
        IERR    I*2          error code of Z...XF - 0 => ok
                                                    2 => input error
```

9.5.4.10 IENHNS - performs an interactive linear enhancement of TV LUTs. X cursor => intercept, Y cursor => slope, high button => quit

```
IENHNS (ICHAN, ICOLOR, ITYPE, RPOS, BUFFER, IERR)

Inputs: ICHAN   I*2          channel select bit mask
        ICOLOR  I*2          color select bit mask
In/Out: ITYPE   I*2          on in: 1 => do plot, A, B switch plot
                                       C switch sign of slope
                                    2 => no plot, A, B return
                                       C switch sign of slope
                                    3 => no plot, return any button
                             on out - button value
        RPOS    R*4(2)       Cursor position: initial -> final
Output: BUFFER  I*2(>768)    scratch buffer
        IERR    I*2          error code of Z...XF: 0 => ok
```

9.5.4.11 DLINTR - is called by interactive routines to delay the task when nothing is happening (i.e. the user is thinking or out to lunch.) It also prevents cursor wrap around.

```
DLINTR (RP, IEV, DOCOR, QUAD, PP, IT, DOIT)

Inputs: IEV   I*2        not = 0  => event has occurred
        DOCOR L*2        Scroll correction parameter for YCURSE
        QUAD  I*2        quadrant parameter for YCURSE
In/out: RP    R*4(2)     cursor position read (fixed on wraps)
        PP    R*4(2)     previous cursor position
        IT    I*2(3)     time of last action
Output: DOIT  L*2        T => something has happened.
```

9.5.4.12 RNGSET - calculates range parameters for displaying a map using the IRANGE adverb supplied by POPS plus scaling information derived from the map header.

```
RNGSET (IR, MMAX, MMIN, BSC, BZE, RANG)

INPUTS:
   IR(2)    R*4   Range values specified by user
   MMAX     R*4   Map maximum value from header
   MMIN     R*4   Map minimum value
   BSC      R*8   Map scaling factor from header
   BZE      R*8   Map zero offset from header
OUTPUTS:
   RANG(2)  R*4   Output range values calculated using
                     defaults and map scaling
```

9.5.4.13 DECBIT - translates a decimal based channel number into a binary channel no. e.g. 1453 => 2**0 + 2**3 + 2**4 + 2**2 A maximum of nine channels are addressable (6 at a time)

```
DECBIT (NMAX, ICHAN, IPL, LOW)

INPUTS:
   NMAX        I*2     Maximum allowed channel number
   ICHAN       I*2     Input channel decimal number
OUTPUTS:
   IPL         I*2     Binary channel # pattern
   LOW         I*2     Lowest of specified channels
```

9.5.4.14 MOVIST - sets and resets the movie status parameters in the TV common.

```
MOVIST (OP, ICHAN, NFR, NFRPCH, MAG, IERR)

Inputs: OP       R*4   'ONNN' when turning on a movie
                       'OFFF' when clearing channel(s)
        ICHAN    I*2   Bit pattern of channels involved (OFFF)
                       Actual first channel number (1-NGRAY, ONNN)
        NFR      I*2   Number of frames in movie total (ONNN)
        NFRPCH   I*2   Number of frames per TV channel (ONNN)
        MAG      I*2   Magnification number (0 - 3, ONNN)
Output: IERR     I*2   Error = 2 => bad input, else ok
```

CHAPTER 10

PLOTTING

10.1 OVERVIEW

Plotting in AIPS is usually a two step process. First a task or a verb creates an AIPS "plot file" which consists of plot device independant "commands" that tell a device how to draw the plot. As of the time this chapter was written, this file is always an extension file associated with a cataloged file. However, the plot file could itself be a cataloged file. The second step in obtaining a plot is to run a task to read the plot file and write it to a specific device, such as a TV, or a hardcopy plotter. This two step method greatly reduces the number of plot programs that must be written and maintained. For instance, if a new graphics device is added to the system then only one new program that reads the plot file and writes to the new device is needed. All the other plotting programs work with no modification. Another advantage is that a plot file may exist for an extended period of time, thus allowing plots to be written to different devices, and copies to be generated at later times without duplicating the calculations needed in making the plot.

There are exceptions to the two step process. For example, slices of map files can be plotted directly on the Tektronics 4012. This is done to simplify matters in interactive situations such as gaussian fitting of slices.

AIPS contains some very powerful routines for plotting in an variety of coordinate systems in use in astronomy. The complexity of these routines is commensuate to their power. Fortunately, a set of plot program templates exist to provide a starting point. These routines are described in a latter section in this chapter.

10.2 PLOT FILES

10.2.1 General Comments.

Plot files are a generalized representation of a graphics display. They contain scaling information and commands for drawing lines, pixels, and characters, and a command for putting miscellaneous information in the image catalog. The image catalog is used by programs that must know details about an image currently displayed on the graphics device in order to allow user interaction with the device. For example a program may want to read a cursor position and translate it to the coordinate system of the image displayed on the graphics device.

The records in plot files do not include a record length value. This means that it is inconvenient to invent new types of records (i.e. new opcodes) or to add new values on to the end of records of existing types because all of the programs must be changed. On the other hand, the rigid format definitions aided in debugging the code several years ago and continue to assure the integrity of I/O systems (AIPS device plotting programs refuse to proceed if they encounter an unknown opcode). So far, the increased flexibility supplied by length values seems not to have been absolutely required in AIPS.

The character drawing record includes neither a size value nor an angle value. This is because character plotting capabilities are device dependent. Orientations are either vertical or horizontal (and not backwards) and the position offsets for plotting character strings are specified in units of the device character size, permitting the device plotting program to position strings nicely no matter what size it chooses to use. It also follows that most plots produced by AIPS have only one size of character. One AIPS application program (PROFL) draws its own characters by using the line drawing commands in order to plot characters with arbitrary size, orientation, and even perspective.

10.2.2 Structure Of A Plot File

The first physical record (256 words) in the plot file contains information about the task which created the file. It is not logically part of the "plot file", but is there to provide documentation of the file's origins. This record is ignored by the programs that actually do the plotting. The primary use of this information is by the the verb EXTLIST that lists all the plot files associated with a cataloged file. When new types of plots are added to AIPS, an experienced programmer should update the verb EXTLIST (found in subroutine AU8A) to list useful things about the plot. Otherwise the verb will print a line telling the user that he has a plot file of type UNKNOWN. A novice AIPS programmer should leave this code alone.

The contents of the first physical record are task-dependent and have the form:

FIELD	TYPE	DESCRIPTION
1.	I*2(3)	Task name (2 chars / word)
2.	I*2(6)	Date/time of file creation YYYY,MM,DD,HH,MM,SS
3.	I*2	Number of words of task parameter data
4.	R*4(*)	Task parameter block as transmitted from AIPS (preferably with defaults replaced by the values used).

The rest of the plot file contains a generalized representation of a graphics display. This representation is in the form of scaling information and commands for drawing lines, pixels, and characters and a command for putting miscellaneous information in the image catalog.

The lowest level plot file I/O routines read and write 256 word blocks. The applications programmer will be concerned with routines that read and write logical records.

The logical records are of 6 types and vary in length. With the exception of the 'draw pixels' record, logical records do not cross the block boundaries. Unused space at the end of a block consists of integer zeros. All values in the plot file are I*2 variables or ASCII characters. This aids in exporting plot files to other computers via tape. Unfortunately, this also limits the values that can be stored in the plot file, thus forcing us to use a scaling factor and offset for some plots to prevent integer overflow. The scaling factor and offset are not in the plot file. This causes problems for interactive tasks that read positions from a graphics device and then try to convert them to the original coordinates. These interactive tasks must make do with information from the map header and data from the "miscellaneous information" record.

Plot files have names of the format PLdsssvv, where d is the disk volume number, sss is the Catalog slot number of the associated map, and vv is the version number.

10.2.3 Types Of Plot File Logical Records

10.2.3.1 Initialize Plot Record. - The first logical record in a plot file must be of this type.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equal to 1 for this record type).
2.	I*2	User number.
3.	I*2(3)	Date: yyyy, mm, dd
4.	I*2	Type of plot: 1 = miscellaneous 2 = contour 3 = grey scale 4 = 3D profile 5 = slice 6 = contour plus polarization lines 7 = histogram

10.2.3.2 Initialize For Line Drawing Record. - This record provides scaling information needed for a plot. The plot consists of a 'plot window' in which all lines are drawn and a border (defined in terms of character size) in which labeling may be written. The second record in a plot file must be of this type.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equal to 2 for this record type).
2.	I*2	X Y ratio * 100. The Ratio between units on the X axis and units on the Y axis (X / Y). For example if the decrement between pixels in the X direction on a map is twice the decrement in the Y direction the X Y ratio can be set to 2 to provide proper scaling. Some programs may ignore this field. For example IISPL when writing grey scale plots to the IIS.
3.	I*2	Scale factor (currently 16383 in most applications). This number is used in scaling graph positions before they are written to disk. BLC values in field 4 are represented on disk by zero and TRC values are represented by integers equal to the scale factor).
4.	I*2(4)	The bottom left hand corner X and Y values and the top right hand X and Y values respectively in the plot window (in pixels).
5.	I*2(4)	1000 * the fractional part of a pixel allowed to occur outside the (integer) range of BLC and TRC (field 4 above) in line drawing commands
6.	I*2(4)	10 * the number of character positions outside the plot window on the left, bottom, right, and top respectively
7.	I*2(5)	Location of the X Y plane on axes 3,4,5,6,7. This field is valid only for plots associated with a map.

10.2.3.3 Initialize For Grey Scale Record. - This record if needed must follow the 'init for line drawing' record. This record allows proper interpretation of pixels for raster type display devices. Programs that write to line drawing type devices (e.g. the TEKTRONIX 4012) ignore this record.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 3 for this record type).
2.	I*2	Lowest allowed pixel intensity.
3.	I*2	Highest allowed pixel intensity.
4.	I*2	Number of pixels on the X axis.
5.	I*2	Number of pixels on the Y axis.

10.2.3.4 Position Record. - This record tells a device where to start drawing a line, row/column of pixels or character string.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 4 for this record type).
2.	I*2	scaled x position i.e. a value of 0 represents the

		BLC values defined in the 'init for line drawing' record, and a value equal to the scale factor represents the TRC value.
3.	I*2	Scaled Y position.

10.2.3.5 Draw Vector Record. - This record tells a device to draw a line from the current position to the final position specified by this record.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 5 for this record type).
2.	I*2	Scaled final X position.
3.	I*2	Scaled final Y position.

10.2.3.6 Write Character String Record. - This record tells a device to write a character string starting at the current position.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 6 for this record type).
2.	I*2	Number of characters.
3.	I*2	Angle code: 0 = write characters horizontally. 1 = write characters vertically.
4.	I*2	X offset from current position in characters * 100
5.	I*2	Y offset from current position in characters * 100 (net position refers to lower left corner of 1st char)
6.	I*2(n)	ASCII characters (n = INT((field2 + 1) / 2))

10.2.3.7 Write Pixels Record. - This record tells a raster type device to write an n-tuple of pixel values starting at the current position. Programs that write to line drawing type devices ignore records of this type.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 7 for this record type).
2.	I*2	Number of pixel values.
3.	I*2	Angle code: 0 = write pixels horizontally. 1 = write pixels vertically (up).
4.	I*2	X offset in characters * 100.
5.	I*2	Y offset in characters * 100.
6.	I*2(n)	n (equal to field 2) pixel values.

10.2.3.8 Write Misc. Info To Image Catalog Record. - This record tells the programs that write to interactive devices (TEKPL, IISPL) to put up to 20 words of miscellaneous information in the image catalog starting at word I2TRA + 2. This information is interpreted by routines such as AU9A (TKSKYPOS, TKMAPPOS, ect.). Routines that write to non-interactive graphics devices (PRTPL) ignore this record.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 8 for this record type).
2.	I*2	Number of words of information.
3.	I*2(n)	Miscellaneous info (n=value of field 2).

10.2.3.9 End Of Plot Record. - This record marks the end of a plot file.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 32767 for this record type).

10.3 PLOT PARAFORM TASKS

10.3.1 Introduction

Three paraform tasks (PFPL1, PFPL2 and PFPL3) are available in AIPS for developing plot tasks that read a map and create a plot file to be associated with the map. These tasks use the standard AIPS defaults for adverb values such as IMNAME, BLC, TRC, XYRATIO, PIXRANGE, etc., and work for both integer and floating point maps. The programs are heavily commented and modular.

The three tasks correspond to the three types of plots that can be found in AIPS. The first type is a plot of an X Y plane of the map or a subimage of the map. In this case the X and Y axis of the plot are the same as the X and Y axis of the map. Examples of this type are produced by tasks CNTR and GREYS. A second type of plot is when the X axis of the plot is a slice of the X and Y axis of the map and the Y axis of the plot is some other value such as intensity. Task SL2PL will create a plot of this type from a slice of a map. The third type of plot is when the axis of the plot has no real relation to the map axis. An example of this type of plot is the histogram produced by task IMEAN.

The structure of all three paraform tasks are very similar. The major differences are in subroutine PLINIT (the subroutine that initializes the commons used in labeling the plot), PLLABL (this routine does the actual labeling), and in the example plots in subroutine PLTTOR. The adverbs received from AIPS also differ slightly. The tasks will be discussed individually in a following section, but first we will describe the general structure of all three programs. The tasks perform the following steps:

1. Open a map file corresponding to the users inputs from AIPS.

2. Create an extension file of type PL (plot) to be associated with the map file. The header of the map file will be updated to include this new extension file.

3. Write the plot file records to draw the borders and labels of the plot. The programmer can customize this section of the program by changing data statements and assignment statements

in the main program.

4. Write the rest of the plot file records to the plot file. This is done by subroutine PLTTOR. The programmer will have to modify the code in PLTTOR for his needs.

5. Do the necessary clean up functions, write end of plot records, close all files, etc.

10.3.2 Getting Started

The first step is choosing a new name and making copies, using the new name, of the source code file and the help file. On the Vax, one should copy files NOTPGM:PFPLn.FOR, and HLPFIL:PFPLn.HLP ("n" stands for 1, 2 or 3) to a user directory and work with the program there. Useful information on running a task from a user's directory, and on compiling and linking tasks and on modifying skeleton tasks can be found in other chapters of this manual.

When a task is renamed, some source code must be changed. The first line of the program

```
PROGRAM PFPLn
```

and the data statement

```
DATA PRGNAM /'PF','PL','n '/
```

should be changed to use the new name. The name in the HELP file should also be changed.

Next, the programmer should compile and link the task in his directory and try running it from AIPS by using adverb VERSION. This will assure the programmer that the task does work, and also demonstrate the current output of the task.

10.3.3 Labeling The Plot

The labeling of the plot takes place in two subroutines called by subroutine PLTTOR. PLINIT will set a number of variables in common that give the labeling routines and the plot drawing routines information about the corners of the plot, the types of the axes, the type of labeling, the size of the plot borders in characters, and other details.

Subroutine PLLABL uses the information provided by PLINIT to actually write the commands in the plot file to draw the labels, borders, and tic marks.

The programmer can customize the labeling somewhat without changing either PLINIT or PLLABL by setting values in an array PCODE, and changing data statements in the main program.

Optional text can be printed at the bottom of a plot by setting values NTEXT (number of lines of text), and TEXT (an array containing the actual text lines). These values are currently set in data statements in the main program. The programmer can choose to set NTEXT to zero to suppress all of the lines. If the programmer wishes to use more than two lines, then the second dimension of array TEXT must be changed in all the routines in which TEXT is declared.

See the section on the individual programs for details on setting PCODES.

10.3.4 Plotting

Plotting consists of reading the map, collecting the data, and then drawing lines or writing grey scale pixels. All of these steps are usually done in subroutine PLTTOR. Reading a map is usually done with routine GETROW (see below). Setting a starting point of a line is usually done with routine PLPOS. Setting the end point of a line is done with PLVEC. Grey scale pixels are written with subroutine PLGRY.

10.3.5 Map I/O

This program does not use the Easy I/O (WaWa) package, but instead uses the standard AIPS I/O package grouped into a few subroutines. This approach attempts to make life a little easier by hiding a few of the messy details, but not to eliminate the flexibility of the standard I/O by hiding it under a complex system. These routines use the "copy mode" approach to I/O in that data is read into a large buffer and then copied with scaling from the large I/O buffer to a smaller buffer when a row is needed. This is less efficient than using the bare AIPS I/O routines but frees the programmer from having to deal with indexes into the large array, and handling both floating and integer maps in the upper level program.

There are four I/O routines in this program, MAKNAM (fills in a real array with all the data items that go into specifying a map), INTMIO (initializes the I/O routines to read or write a cataloged map), REIMIO (initializes counters for reading a different subimage or making another pass through a map opened by INTMIO) and GETROW (reads a row of a map, and converts the values to floating point numbers, if necessary). MAKNAM and INTMIO are used in straight forward ways to open the map. The programmer can usually ignore these two routines unless a second map must be opened. If the program must make more than one pass through the data REIMIO can be used to reset all of the counters. REIMIO assumes that the map is already opened in INTMIO and that a second pass is being made through the data. This routine can NOT be used to read different subimages from the same map at the same

time. GETROW must be used (usually in subroutine PLTTOR) to read data from the map, one row at a time.

The I/O routines in this program use a common named MAPHDR. This name was chosen to interface with several of the plotting routines which expect this common to have the map header as the first 256 words. Besides the map header, this common contains an array, IMSTUF, which has several data items of interest. IMSTUF(9) is of particular interest since it contains the number of data values (pixels) in each row of the map. This number is usually the upper limit of a loop which operates on each element in the map row. A description of all the elements of IMSTUF are listed in the following table:

1. AIPS I/O Logical unit number

2. FTAB index

3. Integer (1) or real (2) flag.

4. Blanked value for integers 0=no blanking.

5. Catalog slot of image.

6. Size of I/O buffer in bytes.

7. Disk volume number of image.

8. Number of dimensions in image.

9. Number of values read per row of image.

10-16. Number of values along all 7 axes

17-30. Window in BLC TRC pairs along all 7 axes.

31-36. Current position on last six axes.

37 1 if read forward -1 if backward read on 2nd axis.

Minor modifications in the I/O routines could be made to produce routines for reading UV data, but this has not yet been done.

10.3.6 Cleaning Up

Some of the adverbs passed from AIPS may not be used for some types of plots. The programmer can make things easier for the AIPS user by removing them from the help file. The programmer must then remove them from the common /INPARM/, which can be found in the main program and in several of the subroutines. The variable NPARMS is initialized in an assignment statement in the main program. This must be changed to correspond to the new number of floating point numbers received from AIPS.

10.3.7 The Three Paraform Plot Tasks

10.3.7.1 PFPL1 - This task should be used when developing a plotting task in which the X and Y axis of the plot are the same as the X and Y axis of the map.

Much of the labeling is controlled by values of array PCODE. The values for the elements of PCODE are summarized in the following table.

If PCODES(1) equals

- 1 then the plot axis consists of an unlabeled rectangular border.
- 2 then draw a rectangular border plus the title and text at the bottom.
- 3 then draw a rectangular border, labels, and border tick marks indicating absolute coordinates (r.a., decl., etc.).
- 4 then draw a rectangular border, labels, and border tick marks indicating coordinates relative to the coordinates of the image reference pixel (units usually in arc seconds).
- 5 draw border, labels, and border tick marks indicating coordinates relative to the center of the subimage plotted (units usually in arc seconds).
- 6 draw border, labels, and border tick marks indicating image pixel numbers.

If PCODES(2) equals

- 0 then label the X axis with the X axis value found in the map header.
- other then label the X axis using variable XUNIT which is set in a data statement in the main program.

If PCODES(3) equals

- 0 then label the Y axis with the Y axis value found in the map header.
- other then label the Y axis using variable YUNIT which is set in a data statement in the main program.

If PCODES(4) equals

- 0 then use the "standard" title consisting of map name, source name, and frequency.

other then use the title given in data statement for variable TITLE in the main program.

If PCODES(5) equals

0 then no grey scale pixels are to be written for the plot.

other then grey scale pixels with a range given by PIXRNG (these values are usually passed from AIPS in adverb PIXRANGE) can be written to the plot. This code value causes an 'init for grey scale' record to be written to the plot file.

Usually a task will let the AIPS user choose the value of PCODES(1) by setting adverb LTYPE, e.g., PCODES(1) is set to LTYPE after the task gets this adverb value from AIPS.

When using PLPOS and PLVEC the positions for this type of plot are given in pixels.

The unmodified version of PFPL1 contains code in PLTTOR to read the map, and draw a grey scale plot. The user should remove this example found between comment lines "** Plot specific code" and "** End plot specific code" and insert the code for his own application.

10.3.7.2 PFPL2 - This task should be used when developing a plotting task in which the X axis of the plot is a slice of some plane of the map, and the Y axis is some other value such as intensity. The PCODE usage is described below.

PCODES(1) equals

The label type of the X axis. The codes are the same as for PFPL1.

If PCODES(2) equals

0 then label the X axis with the units determined by the "standard" slice labeling algorithm.

other then label the X axis using variable XUNIT which is set in a data statement in the main program.

If PCODES(3) equals

0 then label the Y axis with the units found in the map header for the map intensity.

other then label the Y axis using variable YUNIT which is set in a data statement in the main program.

If PCODES(4) equals

0	then use the "standard" title consisting of map name, source name, and frequency.
other	then use the title given in data statement for variable TITLE in the main program.

If PCODES(5) equals

0	then use the "standard" slice message at the bottom of the plot. This message will give the center of the slice. This message occurs above the message found in TEXT as described above.
other	then do not print the "standard slice message"

The example program in PFPL2 will plot a slice of the X Y plane. The user should remove the example found between comment lines "** Plot specific code" and "** End plot specific code" and insert the code for his own application. This example uses no interpolation (it uses the value of the nearest pixel) and is NOT adequate for a production program. See the code in task SLICE for a good set of interpolation routines and a "rolling buffer" scheme.

10.3.7.3 PFPL3 - This task should be used when developing a plotting task in which the X and Y axis have no relation to the map X and Y axis. The plot could be of a function, a histogram of some values, or a table.

The only PCODES value used are PCODES(4) and PCODES(5). If PCODES(4) is 0 then the program plots the "standard" title line. Otherwise, it uses whatever string is in variable TITLE. If PCODES(5) is not zero then this signals the existence of grey scale pixels. The program automatically uses whatever strings are in variables XUNIT and YUNIT to label the units for X and Y. Thus, the programmer will have to edit the data statements for these variables in the main program, or fill them in by some other means.

The example program in the unmodified version of PFPL3 will plot a simple histogram of map intensities. The subroutine PLTTOR reads the map to determine the histogram values and the range of the Y axis (number of pixels). Then the standard initializing routine (PLINIT) and labeling routine (PLLABL) are called. Finally the histogram is plotted. The programmer must remove the two sections of example code found between two sets of comment lines "** Plot specific code" and "** End plot specific code" and insert the code for his own application.

10.3.8 Routines

10.3.8.1 PLEND - Do some plotting cleanup functions. Write "end of plot" record, close plot file, check for vectors that were off the plot.

```
      PLEND (IOBLK, ISTAT)
In/Out:
  IOBLK  I*2(256)  Work I/O buffer
  ISTAT  I*2       0=successful completion, other=dies unnaturally.
```

10.3.8.2 PLPOS - This routine will put a 'position vector' command in an AIPS plot file.

```
      PLPOS (X, Y, IERR)
Inputs:
  X      R*4  X value.
  Y      R*4  Y value.
  COMMON /PLTCOM/
Output:
  IERR   I*2  Error code.  0 means OK.
```

10.3.8.3 PLVEC - This routine will put a 'draw vector' command in an AIPS plot file.

```
      PLVEC (X, Y, IERR)
Inputs:
  X      R*4  X value.
  Y      R*4  Y value.
  COMMON /PLTCOM/
Output:
  IERR   I*2  Error code.  0 means OK.
```

10.3.8.4 PLMAKE - This routine will create and open a plot file, put it in the map header and write the first record into the plot file.

```
      PLMAKE (NP, RPARM, IERR)
Inputs:
  NP     I*2      Number of floating point words in parameter list
                  received from AIPS.
  RPARM  R*4(NP)  AIPS parameters.
Output:
  IERR   I*2      Error code. two digit, first digit indi-
                  cates subroutine: 1: MAPOPN, 2: MADDEX,
                  3: ZPHFIL, 4: GINIT, second digit indi-
                  cates error code of that subroutine.
```

10.3.8.5 PLGRY - This routine will put draw grey scale commands in the plot file.

```
      PLGRY (IANGLE, NVAL, VALUES, IERR)
Inputs:
  IANGLE I*2     Angle code. 0 = horizontal, 1 = vertical.
  NVAL   I*2     The number of grey scale pixel values.
  VALUES R*4(?)  Grey scale values.
Output:
  IERR   I*2     Error code. 0=ok.
```

10.3.8.6 MAKNAM - This routine will construct a WaWa I/O name string, given the values that make up the thing.

```
      MAKNAM (INAME, INCLAS, SEQ, VOL, TYPE, USER, NAMSTR)
Inputs:
  INAME   R*4(3)  file name
  INCLAS  R*4(2)  file class
  SEQ     R*4     file sequence number.
  VOL     R*4     file disk volume.
  TYPE    R*4     file type.
  USER    R*4     file user number.
Output:
  NAMSTR  R*4(9)  "Name string" in the tradition of WaWa I/O.
                  NAME(1:3) name, NAME(4:5) class, NAME(6) seq,
                  NAME(7) volume, NAME(8) type, NAME(9) user.
```

10.3.8.7 INTMIO - This routine will open a map file, set values in common for use with close down routine DIE and set up two arrays containing all the values and counters needed by reading and writing routines compatible with this one.

```
      INTMIO (ILUN, ACCESS, NAME, BLC, TRC, IBSIZE, IHD, IMSTUF, DSCAL, IERR)
Inputs:
  ILUN    I*2     Logical unit number to use for the map file.
  ACCESS  R*4     'READ' or 'WRITE' status to mark catalog.
  NAME    R*4(9)  "Name string" in the tradition of WaWa I/O.
                  NAME(1:3) name, NAME(4:5) class, NAME(6) seq,
                  NAME(7) volume, NAME(8) type, NAME(9) user.
  BLC     R*4(7)  Bottom left corner of map.
  TRC     R*4(7)  Top right corner of map
  IBSIZE  I*2     Size of I/O buffer in INTEGER*2 values.
Outputs:
  COMMON /CFILES/ Values updated so that subroutine DIE will
                  close this file.
  IHD     I*2(256) Map header.
  IMSTUF  I*2(37)  I/O pointers and stuff that are needed by other
                   I/O routines compatible with this one. They are:
                   1. LUN
                   2. FTAB index
                   3. integer (1) or real (2) flag.
                   4. Blanked value for integers 0=no blanking.
```

5. Catalog slot of image.
6. Size of I/O buffer in bytes of all things.
7. Volume number of image.
8. Number of dimensions in image.
9. Number of values read per row of image.
10-16. Number of values along all 7 axis
17-30. Window in BLC TRC pairs along all 7 axis.
31-36. Current position on last six axis.
37 1 if read fwd -1 is backwrd read on 2nd axis.

DSCAL	R*8(2)	Scale factors to use with this image.
IERR	I*2	Error code. 0=ok.

10.3.8.8 REIMIO - This routine will reinitialize the counters in IMSTUF for reading another subimage of a map opened and set up with INTMIO. All IMSTUF values that can be found in the header are re-initialized even if they are not changed by the standard routines.

REIMIO (BLC, TRC, IBSIZE, IHD, IMSTUF, DSCAL, IERR)

Inputs:

BLC	R*4(7)	Bottom left corner of map.
TRC	R*4(7)	Top right corner of map
IBSIZE	I*2	Size of I/O buffer in INTEGER*2 values.
IHD	I*2(256)	Map header.
IMSTUF(1)	I*2	LUN
IMSTUF(2)	I*2	FTAB index
IMSTUF(7)	I*2	Volume number of image.
IMSTUF(5)	I*2	Catalog slot of image.
IMSTUF(6)	I*2	Size of I/O buffer in bytes of all things.

Outputs:

IMSTUF(3)	I*2	Integer (1) or real (2) flag.
IMSTUF(4)	I*2	Blanked value for integers 0=no blanking.
IMSTUF(8)	I*2	Number of dimensions in image.
IMSTUF(9)	I*2	Number of values read per row of image.
IMSTUF(10-16)		Number of values along all 7 axis
IMSTUF(17-30)		Window in BLC TRC pairs along all 7 axis.
IMSTUF(31-36)		Current position on last six axis.
IMSTUF(37)	I*2	1 if read fwd -1 is bckwrd read on 2nd axis.
DSCAL	R*8(2)	Scale factors to use with this image.
IERR	I*2	Error code. 0=ok.

10.3.8.9 GETROW - This routine will read a row of an image file that has been opened with and initialized with INTMIO. The routine will copy the row from the I/O buffer to the user buffer, converting integer values to floating point, if necessary.

GETROW (IMSTUF, DSCAL, IOBLK, ROW, EOF, IERR)

Inputs:

IMSTUF	I*2(37)	I/O pointers, LUNs, counters and such. They are set in INTMIO.
DSCAL	R*8(2)	Actual value = DSCAL(1) * disk value + DSCAL(2)

In/Out:

IOBLK	I*2(?)	I/O buffer.

Outputs:

ROW	R*4(?)	Scaled output row of image.
EOF	L*2	TRUE means last row specified in INTMIO by the BLC, TRC arguments has been read.
IERR	I*2	Error code, 0=ok, others from MDISK.

CHAPTER 11

USING THE ARRAY PROCESSORS

11.1 OVERVIEW

Many of the more important of the AIPS tasks do a great deal of computation while the cpu of the host computer of most AIPS systems is rather slow. In order to make AIPS tasks on these computers run at intresting speeds we use hardware arithmetic units called Array Processors. These array processors (or APs) have their own memory and high speed, pipelined arithmetic hardware enabling them to run much faster than the host for certain specialized operations. Since not all computers running AIPS will have array processors attached there is a library of Fortran routines which emulate the functions of the array processor; these routines and a common in the host memory constitute the "pseudo-array processor". This chapter will describe the use AIPS makes of array processors and explain how to use APs. At the end of this chapter is a list of the major AP routines with detailed comments on the call sequence.

11.1.1 Why Use The Array Processor?

The principle reason for using an array processor is speed. The design of most array processors optimizes its performance for repetitive arithmetic operations making it mush faster at vector arithmetic than the host CPU. Since most APs operate asynchronously from the host CPU they constitute a co-processor which increases the capacity of the system.

A second advantage of using an array processor is that it contains its local memory. On systems with limited physical memory or address space this can be an important consideration. It will be possible in the near future to get array processors with many megawords of local memory. Such large memories will allow the use of more efficient methods of processing data.

11.1.2 When To Use And Not To Use The AP.

The array processor is most efficient at very repetitive operations such as doing FFTs and multiplying large vectors. Its efficiency is greatly degraded for non-repetitive operations or operations requiring a great number of decisions based on the results of computations. In fact, some array processors have very limited capability to make decisions based on the results of computations.

Since the APs have their own program and data memory, the AP instructions and the data must be transfered to and the results transfered from the AP. These I/O operations may cost more cpu time than the amount saved by using the array processor.

As a general rule, use of the AP is more efficient than the CPU when multiple or complex (such as FFTs) operations which are highly reptitious are going to be done on relatively large amounts of data (thousands of words or more). In other cases using the AP will probably not help much and will keep other processes from using this valuable resource.

11.2 THE AIPS MODEL OF AN ARRAY PROCESSOR

The model of an array processor used is colored strongly by our use of Floating Point Systems FPS AP-120B array processors. However, expressed in general terms, this model can be emulated on other real or virtual (pseudo) array processors. It should be noted that use of the APs requires vectorized programming, hence, implementation on super computers or other vector machines should be relatively efficient. The following describes the fundamental features of the AIPS model of array processors.

- AIPS currently uses APs essentially as vector arithmetic units. That is, data is sent into the AP, some (usually vector) operation is done, and the reults is returned to the host CPU. The principle difficulty in the implementation of AIPS on other array or vector processors is that our concept of a vector operation is rather more general than that of most computing hardware manufactures. Many of the more complex of the AIPS operations are better described as pipelined scalar operations. In the AIPS useage, most high level control and use of disk storage is done in the host CPU and only arithmetic operations are done in the AP.

- AIPS considers the AP to be a device which can be assigned via BPINIT and deassigned via BPRLSE. Basically, this means that data will not dissappear from the task's assigned AP data memory.

- An AP should have a relatively large local data memory. The size of the AP data memory is obtained from a common set by ZDCHIN which reads it from a disk file. The value in this disk file can be modified by the AIPS utility program SETPAR. In the case of pseudo (virtual) AP's, this memory is

physically in the host CPU. A similar implementation could be done for an AP with significantly less capacity than an FPS AP-120B.

- In addition to data memory, the AP is assumed to have an array of 16 integer registers (SPAD) which can be read from the host CPU. These are used to communicate the addresses of maxima, minima, etc. This capability is not extensively used.

- AIPS assums that the array processor is programmable in that functions are used which are not now or likely ever to be in a standard library. If the AP is not programmable or is otherwise incapable of emulating one of the AIPS functions, then these functions must be performed in the host CPU and hidden from the AIPS routines.

- Communication with the AP by AIPS is via Fortran call statements which specify the data in the AP memory and other control information, transfer data between the AP and host CPU, or synchronize the operation of the AP and host CPU.

- Data in the AP memory is specified by a base address and an increment. In current implementations these addresses are absolute but this is not assumed. The calling process is assumed to have absolute control over an address space beginning at address 0 and extending to the address indicated in the device characteristic common (include CDCH.INC) as (1024KAPWRD). Word addressing only is used.

- Many of the most crucial functions used by AIPS routines depend on data dependent address generation and logic flow. As mentioned above, implementation of AIPS on an array processor without this capability is possible but much of its speed will be lost.

- AIPS assumes that the AP can handle either integer or real data values (with the same word size). Complex values consist of a pair of real values in adjacent locations, the first being the real part and the second being the imaginary part.

11.3 HOW TO USE THE ARRAY PROCESSOR

Since the array processors used by AIPS have their own program and data memories the instructions must be loaded in to the AP and data sent to, and results returned from the AP. Since the AP runs asynchronously from the host cpu there most also be ways to synchronize the operations. Then general operations are given in the following list with the name of the subroutine AIPS uses for the given operation (we use the Floating Point Systems conventions):

1. Assign / Initialize the AP. (BPINIT)
2. Transfer data to the AP. (APPUT)
3. Wait for transfer to complete. (APWD, APWAIT)
4. Load and execute the AP program. (many)
5. Wait for computations to finish. (APWR, APWAIT)
6. Transfer data back to host cpu. (APGET)
7. Wait for transfer to complete. (APWD, APWAIT)
8. Release AP. (BPRLSE)

11.3.1 AP Data Addresses

The AIPS convention for specifying data in the AP memory, which follows the Floating Point Systems (FPS) conventions, is to specify data by the zero relative memory address of the first element in an array, the memory address increment between the elements of an array, and the number of elements in the array. On FPS APs the memory address is an absolute address but in implementations on other APs the address may be a relative address but this should be hidden from the programmer.

11.3.1.1 Pseudo I*4 Addresses - The FPS array processors require unsigned 16 bit addresses and the address space is limited to 64 kwords. To accomodate both the FPS requirements and to allow for future implementation of APs with larger memory size all arguments to AP routines (except BPINIT and the host array names in APPUT and APGET) are pseudo I*4. Thus, addresses up to 31 bits are allowed and the first integer of the pseudo I*4 array is an unsigned short integer. The AIPS utility routines for pseudo I*4 numbers are ZR8P4 which converts between REAL*8 and pseudo I*4 and ZMATH4 which does pseudo I*4 arithmetic. The call sequences for these routines are given at the end of this chapter.

11.3.1.2 Array Processor Memory Size - Since different array processors will have different memory sizes the memory size of the AP is carried in the Device Characteristics Common which is obtained by the includes DDCH.INC and CDCH.INC. The size of the AP is in the I*2 value KAPWRD as the multiple of 1024 words of AP data memory. Any operation with the AP should check that enough data memory is available and if possible scale the operation to make full use of the available memory.

11.3.2 Assigning The AP

The array processor is assigned to the calling task using the AIPS routine BPINIT. BPINIT incorporates the AIPS priority system and provides for smooth use of the AP for batch tasks. The AIPS AP priority scheme is to give tasks with lower Pops numbers (the number at the end of the task name when it is running) higher priority. This is done by keeping a list of AP tasks in BPINIT. When a task asks for an AP, BPINIT then checks to see if any AP tasks with a lower pops number are running; if so then BPINIT suspends the task for a short period and then checks again. The number of times a task goes through the check - suspend loop before asking for the AP at the next opportunity is proportional to its Pops number.

BPINIT also sets values in common /BPROLC/ (includes DBPR.INC and CBPR.INC) which control the AP roller subroutine BPROLL. The text of these includes is shown at the end of this chapter and the use of the values are described in the detailed description of BPROLL given at the end of this chapter.

On some systems batch AIPS tasks present more of a problem. AIPS batch tasks are usually run at lower priority than interactive tasks so they may grab the AP and then not get enough cpu cycles to finish that AP operation for a very long time. To avoid this problem, BPINIT increases the priority of the batch task to that of an interactive task while it has the AP.

BPRLSE is used to deassign the AP. BPRLSE also lowers the priority of batch tasks after the AP is released.

In the interests of a smooth and friendly system for users it is important not to hog the AP for long periods of time. The priority system should then work to give lower Pops numbered AIPS users a larger fraction of the time if they need the AP. A task should in general not keep the AP tied up for more than 5 to 10 minutes at a time, less if that is practical. For tasks which may need to keep the same data in the AP for long periods of time, such as tasks which compute models based on CLEAN components, there is an AP roller subroutine BPROLL.

BPROLL determines if it is time to roll out the AP based on values set by BPINIT, will create a scratch file (using the /CFILES/ system), copy the specified contents of the AP memory to a scratch file, release the AP, wait a short period of time, re-assign the AP and load the previous contents back into the AP memory. Details of

the call sequence to BPROLL are found at the end of this chapter. IMPORTANT NOTE: BPROLL (and APROLL) work properly only for floating point data. Integer values rolled will not be restored correctly.

11.3.3 Data Transfers To And From The AP.

The fundamental routines for getting data to and from the Array Processor memory are APPUT and APGET; details of the call sequences can be found at the end of this chapter. In addition, for image-like data there is the routine APIO.

APIO transfers image-like data between disk files and the array processor. The file open and close and initialization logic are all contained in this routine. Information about the file and the the desired properties of the I/O are passed to APIO in the array FLIST. APIO can access either catalogued 'MA' type files or scratch files using the /CFILES/ common system. APIO can handle arbitrary row lengths. This is done by breaking up the logical records if they are larger than 16384 bytes or the buffer size.

NOTE: it is important that data read with APIO either have a logical record length of 16384 bytes or less or have been written by APIO with the same buffer size; this may be a problem for catalogued files if the row length is greater that 4096 for real format data or 8192 for scaled integers. The problem is that APIO will break up logical records if they are longer than 16384 bytes or the buffer size and MDISK may leave blank space on the disk if the shorter logical record does not fill a disk sector. For this reason it is good to use a buffer size of 16384 bytes or greater when reading or writing catalogued files with APIO. It is IMPORTANT to always use the same size buffer when accessing a given file.

Useage notes for APIO:

1. Opening the file.

   If APIO determines that the file is not open it will do so. The file can be either a catalogued file or a scratch file using the /CFILES/ common system. If the catalogue slot number given in FLIST is 0 or less the file is assumed to be a scratch file. File open assumes that the file type is 'MA' (if catalogued), file is opened patiently without exclusive use.

2. Initialization.

   APIO initializes the I/O using the values in FLIST when it opens the file. It may be initialized again at any time using OPCODE 'INIT'. Also switching between 'READ' and 'WRIT' will force flushing the buffer ('WRIT') and initialization. Any initialization when the current operation is 'WRIT' will cause the buffer to be flushed.

3. Closing the file.

The file may be closed with a call with opcode 'CLOS'. If the file is being written and a 'CLOS' call is issued, APIO will flush the buffer. This means that if APIO is being used to write to a disk it MUST be called with OPCODE='CLOS','READ', or 'INIT' to flush the buffer. NOTE: All pending AP operations MUST be complete before calling APIO with opcode 'CLOS'.

4. AP timing calls.

APIO calls APWD before getting data from or sending data to the AP but does not call APWR. The calling routine should call APWR as appropriate.

More details about the call arguments are found at the end of this chapter and an example of the use of APIO is given in a later section.

11.3.4 Loading And Executing AP Programs.

Loading and executing AP programs is done in a single call to the relevant routine. The call argument also includes the specification of the data, location of the output array, and any processing flags. A list of the AP routines currently supported in AIPS is found at the end of this chapter. If the function desired is not available then it is possible to write it for the AP.

11.3.5 Timing Calls

Since array processors normally run asynchronously from the host CPU timing calls are necessary. The subroutine calls basically suspend the operation of the calling program until the specified AP operation is completed. FPS claims that data transfers and computations (not involving the same AP memory) may be overlapped; however, the results of doing this are erratic and this practice should be avoided. On occasion there appear to be timing problems whose symptoms are erratic and very wrong results which go away when apparently unnecessary timing calls are added; such as calls to APWR between calls to computation routines.

We use three (FPS) timing calls:

- APWD suspends the calling program until data transfers to or from the AP are complete.
- APWR suspends the calling program until the AP completes all computations.
- APWAIT suspends the calling program until all data transfers and computations are complete.

11.3.6 Writing AP Routines.

If the current library of AP routines does not contain the desired function there are two possibilities for coding the function: 1) microcoding the routine or 2) using the Vector Functor Chainer (or equivalent on non-FPS APs) to combine existing functions to create the desired function. If either of these is chosen the programmer should also write the corresponding pseudo-AP routines if the task is likely to have general use.

In order to use microcode or Vector Function Chainer (VFC) routines the following steps must be performed:

1. Compile VFC (or other high level language routines) to assembly (microcode) language. For FPS code this is done by the FPS routine VFC.

2. Assemble microcode into machine code. For FPS code this is done using APAL.

3. Link edit microcode routines together to make an executable module. For FPS code this is done using APLINK. APLINK creates a Fortran or host assembly language routine with the executable module in a data statment.

4. Compile/assemble the Fortran/assembly language module and put in the appropriate subroutine link edit library.

It is beyond the scope of this manual to describe the use of the FPS or other AP software, the reader is referred to the appropriate manual provided by the AP vendor.

11.3.6.1 Microcoding Routines. -

It is beyond the scope of this manual to give details about microcoding for array processors, see the AP manuals for these details. The general principles of efficient microcoding are that several of the hardware units, address computation, floating add, floating multiply, and memory access, may be given instructions in a given cycle. In addition, the floating point hardware is pipelined. That is, even though it takes several cycles for an operation, it is broken up into several, single cycle steps and a new operation can be initiated each cycle.

This architecture allows for very efficient loops. The loop may be broken into several sections and one section from each of several passes through the loop may be processed in parallel. Efficient coding of loops may become very complicated but careful coding may speed up the process by a factor of several. The source code for NRAO written microcode is kept in the file FPSSUB:WDC.AP.

11.3.6.2 Vector Function Chainer. - The principle purpose of the Vector Function Chainer is to combine a number of microcoded routines into a single AP call. This can greatly reduce the overhead of the host cpu talking to the AP; and, if the individual AP operations are relatively numerous and short chaining routines can make a dramatic improvement in the speed of the overall process.

The Vector Function Chainer uses source code that looks vaguely like Fortran but has very limited capabilities and essentially no access to the data memory. Hopefully, in the future there will be efficient Fortran compilers for APs. (FPS has such a compiler for the 120B but NRAO doesn't have a copy).

11.3.7 FFTs

One of the more common operations using the array processor is the Fast Fourier Transform (FFT). We have adopted the FPS convention for real-to-complex FFTs in packing the imaginary part of the last complex value into the real part of the first value in the array. This is allowed because the imaginary part of the first value and the real part of the last value are always zero. This convention allows the use of the same AP memory or disk space for the input and output arrays from a real-to-complex FFT.

We also adopt the convention for FFTs that the second half of a one dimensional array come first and that the center is N/2+1 where N is the number of elements in the array (always a power of two). In two dimensions this means basically that the center of the array is at the corners with the first element of an NX x NY array being (NX/2+1,NY/2+1). An exception to this is that the AIPS two dimensional FFT routine DSKFFT expects the normal order when transforming from the sky plane to the aperature plane (reverse transform).

The AIPS utility routine DSKFFT will FFT a two dimensional array kept in a /CFILES/ system scratch file. Real-to-complex, complex-to-real, or full complex transforms can be done in either direction. For real-to-complex or complex-to-real transforms the maximum and minimum values in the output array and real-to-complex transforms can return either complex, the real part of the result, or the amplitude of the result. Details of the call sequence for DSKFFT are given at the end of this chapter.

The FFT routines require REAL format data without blanking in an array which is a power of two on a side. In addition, the center of an image in a catalogued file may not be in the required (NX/2+1,NY/2+1) position which will produce a phase ramp in the transformed array. Two AIPS utility routines are useful in this case 1) PEAKFN which finds the location of the peak of an image near the center (say of a dirty beam) and 2) PLNGET which will subimage a catalogued file, float scaled integer input, zero fill the excess, and rotate the center of the image. Detailed descriptions of these routines are given at the end of this chapter.

11.4 PSEUDO-ARRAY PROCESSOR

Since not all systems have array processors and many AIPS systems are running on VAXes which have very large address spaces and virtual memory, there is a set of Fortran and assembly language routines which emulate the functions of an array processor, ie. the "pseudo-array processor". The pseudo-AP consists of a Common, obtained by the INCLUDEs DAPC.INC, CAPC.INC, and EAPC.INC, which serves as the AP data memory and a set of routines which operate on data in this common. There are pseudo-AP routines duplicating all of the functions of the true array processor so that a task is simple linked with the appropriate library to use either a true or the pseudo-AP. Listings of the pseudo-AP includes appear at the end of this chapter.

Since pseudo-AP routines need to address "memory" locations with addresses larger than 32767 true INTEGER*4 values are used in the pseudo-AP routines. The AIPS conversion routine ZP4I4 converts from pseudo I*4 to true I*4; details of the call sequence to ZP4I4 are given at the end of this chapter. In addition, since Fortran required one relative indexing whereas the AP addressing is zero relative, pseudo AP routines must add 1 to addresses.

11.5 EXAMPLE OF THE USE OF THE AP

In the following example of the use of the array processor, the elements of two scratch files containing arrays N x M using the /CFILES/ system (numbers ISCRA and ISCRB) are added and returned to the file ISCRC. This makes very inefficient use of the AP but illustrates the basic features. This example also illustrates use of APIO.

```
      SUBROUTINE FILADD (ISCRA, ISCRB, ISCRC, N, M, IRET)
C-----------------------------------------------------------------
C   FILADD adds two REAL N x M arrays in the /CFILES/ scratch files
C   ISCRA and ISCRB and writes the result in scratch file ISCRC.
C    Inputs:
C     ISCRA      I*2  /CFILES/ scratch file number of first input file.
C     ISCRB      I*2  /CFILES/ scratch file number of second input file.
C     ISCRC      I*2  /CFILES/ scratch file number of output file.
C     N          I*2  Length of a row in the array
C     M          I*2  Number of rows in the array.
C    Output:
C     IRET       I*2  Return error code 0=>OK, otherwise APIO error
C                     code.
C-----------------------------------------------------------------
      INTEGER*2 N, M, APLOCA(2), APLOCB(2), APLOCC(2), INCR,
     *   FLIST(22,3), TWO(2), LEN(2), PLUS, KAP, LOOP, IRET,
     *   ISCRA, ISCRB, ISCRC,
     *   N0, N22
      REAL*4 BUFF1(4096), BUFF2(4096), BUFF3(4096), READ, WRITE, CLOSE
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:CDCH.INC'
      DATA READ, WRITE, CLOSE, PLUS /'READ','WRIT','CLOS','PL'/
      DATA N0, N22 /0,22/,  TWO /2,0/
```

```
C-------------------------------------------------------------------------
C                                          Setup for APIO
      CALL FILL (N22, N0, FLIST)
C                                          Pixel type = floating
      FLIST(4,1) = 0
C                                          Size of array
      FLIST(5,1) = N
      FLIST(6,1) = M
C                                          Buffer size (4096 reals)
      FLIST(13,1) = 4096 * 2 * NWDPFP
C                                          Copy for other files
      CALL COPY (N22, FLIST(1,1), FLIST(1,2))
      CALL COPY (N22, FLIST(1,1), FLIST(1,3))
C                                          Set LUNs
      FLIST(1,1) = 16
      FLIST(1,2) = 17
      FLIST(1,3) = 18
C                                          Set /CFILES/ file numbers
      FLIST(2,1) = ISCRA
      FLIST(2,2) = ISCRB
      FLIST(2,3) = ISCRC
C                                          Set AP pointers,
      APLOCA(1) = 0
      APLOCA(2) = 0
      LEN(1) = N
      LEN(2) = 0
C                                          Address for B file
      CALL ZMATH4 (APLOCA, PLUS, LEN, APLOCB)
C                                          Address for C file
      CALL ZMATH4 (APLOCB, PLUS, LEN, APLOCC)
C                                          Increment = 1
      INCR(1) = 1
      INCR(2) = 0
C                                          Grab AP
      CALL BPINIT (N0, N0, KAP)
C                                          Start loop.
      DO 100 LOOP = 1, M
C                                          File A to AP
         CALL APIO (READ, FLIST(1,1), APLOCA, BUFF1, IRET)
C                                          Check for error
         IF (IRET.NE.0) GO TO 999
C                                          File B to AP
         CALL APIO (READ, FLIST(1,2), APLOCB, BUFF2, IRET)
C                                          Check for error
         IF (IRET.NE.0) GO TO 999
C                                          Wait for data transfer
         CALL APWD
C                                          Add
         CALL VADD (APLOCA, INCR, APLOCB, INCR, APLOCC, INCR, LEN)
C                                          Wait for opertaion to finish
         CALL APWR
C                                          Write result to disk.
         CALL APIO (WRITE, FLIST(1,3), APLOCC, BUFF3, IRET)
C                                          Check for error
         IF (IRET.NE.0) GO TO 999
```

```
 100     CONTINUE
C                                          Release the AP
      CALL BPRLSE
C                                          Close files.
      CALL APIO (CLOSE, FLIST(1,1), APLOCA, BUFF1, IRET)
C                                          Check for error
      IF (IRET.NE.0) GO TO 999
      CALL APIO (CLOSE, FLIST(1,2), APLOCB, BUFF2, IRET)
C                                          Check for error
      IF (IRET.NE.0) GO TO 999
      CALL APIO (CLOSE, FLIST(1,3), APLOCC, BUFF3, IRET)
 999  RETURN
C-------------------------------------------------------------------------
      END
```

11.6 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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statments.
- Exxx.INC. These contain Fortran EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statments.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible 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.

11.6.1 CAPC.INC

```
C                                                          Include CAPC
      COMMON /APFAKE/ RWORK, APCORE
      COMMON /SPF/ SPAD
C                                                          End CAPC
```

11.6.2 CBPR.INC

```
C                                                          Include CBPR
      COMMON /BPROLC/ XTLAST, DELTIM, DELAY, TRUEAP
C                                                          End CBPR
```

11.6.3 CDCD.INC

```
C                                                               Include CDCH
      COMMON /DCHCOM/ NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR,
     *   CSIZPR, NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NBITWD,
     *   NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, XPRDMM, XTKDMM,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, SYSNAM, VERNAM, USELIM,
     *   IFILIT, RLSNAM
      COMMON /FTABCM/ DEVTAB, FTAB
C                                                               End CDCH.
```

11.6.4 DAPC.INC

```
C                                                               Include DAPC
      REAL*4    APCORE(1), RWORK(4096)
      INTEGER*4 APCORI(1), IWORK(4069), SPAD(16)
      COMPLEX   CWORK(2048)
C                                                               End DAPC
```

11.6.5 DBPR.INC

```
C                                                               Include DBPR
      REAL*4    DELAY
      REAL*8    XTLAST, DELTIM
      LOGICAL*2 TRUEAP
C                                                               End DBPR
```

11.6.6 DDCH.INC

```
C                                                               Include DDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *   CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *   NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *   NTKACC, UCTSIZ, BYTFLP, USELIM, IFILIT,
     *   DEVTAB(50), FTAB(1)
C                                                               End DDCH.
```

11.6.7 EAPC.INC

```
C                                                                Include EAPC
      EQUIVALENCE (APCORE, APCORI), (RWORK, IWORK, CWORK)
C                                                                End EAPC
```

11.6.8 IDCH.INC

```
C                                                                Include IDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *   CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *   NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *   NTKACC, UCTSIZ, BYTFLP, IFILIT,
     *   USELIM, DEVTAB(50)
C                                                                End IDCH.
```

11.7 ROUTINES

11.7.1 Utility Routines

11.7.1.1 APIO - transfers image-like data between disk files and the array processor. The file open and close and initialization logic are all contained in this routine. Information about the file and the the desired properties of the I/O are contained in the array FLIST. APIO can access either catalogued 'MA' type files or scratch files using the /CFILES/ common system.

```
APIO (OPCODE, FLIST, APLOC, BUFFER, IRET)
```

Inputs:

OPCODE R*4 Code for the desired operation.
- 'INIT' forces the initialization of the I/O.
- 'READ' reads a logical record from the disk and sends it to the specified AP location.
- 'WRIT' Gets data from the AP and writes it to disk.
- 'CLOS' Closes the file and flushes the buffer if necessary.

FLIST(22) I*2 An array containing information about the file and the I/O. Parts are to be filled in by the calling routine and are for use by APIO.
- 1 = LUN, must be filled in,
- 2 = disk number for catalogues files or /CFILES/ number for scratch files.
- 3 = catalogue slot number for catalogued files, .LE. 0 indicates that the file is a scratch file.
- 4 = pixel type. 0=>floating, 1=scaled integer.
- 5 = Length of a logical record (row) in pixels.
- 6 = Number of rows in a plane.
- 7,8 = P I*4 value to be added to (1,0) for the block offset.
- 9-12 = the window desired in the image, 0's=> all of image. The logical records must fit in the buffer and be smaller than 16384 bytes to subimage rows. Reversing the order of FLIST(10) and FLIST(12) will cause the rows to be accessed in the reverse order.
- 13 = Buffer size in bytes. 32767 => 32768.

Used by APIO:
- 14 = FTAB pointer
- 15 = Number of MDISK calls per logical record.
- 16 = Current OPCODE,
  - 0 = none, INIT on next call
  - 1 = READ
  - 2 = WRITE
- 17-18 = actual length of logical row as Pseudo I*4
- 19-22 = Spare.

APLOC P I*4 Base address in AP for data.

BUFFER(*) R*4 Working buffer.

```
Output:
   IRET        I*2 Return code, 0 => OK or
                                1 = Bad OPCODE,
                                2 = Attempt to window too large
                                    a file.
                                3 = Buffer too small (<NBPS bytes)
                                MDISK error codes + 10, or
                                MINIT error codes + 20, or
                                ZOPEN error codes + 30.
```

11.7.1.2 BPROLL - checks if it is time to roll the AP as determined by values bet by BPINIT, copies the first NWORDs of AP main data memory to a scratch file, gives up the AP, does a task delay for DELAY, goes back into the AP queue and loads the scratch file back into the AP. If NWORD .le. 0 then the AP is not rolled but the AP is given up and the task goes back into the AP queue.
NOTE: APROLL is called by BPROLL and uses common /CFILES/ for the scratch file. A scratch file of "type" 'AR' created by APROLL and then destroyed by BPROLL after use.
NOTE: LUN 8 is used for I/O and a AIPS "map" I/O slot is opened if the AP memory is actually rolled.
IMPORTANT NOTE: BPROLL (and APROLL) work properly only for floating point data. Integer values rolled will not be restored correctly.

```
   BPROLL (NWORD, BUFFER, BUFSZ, IRET)

Inputs:
 NWORD      P I*4  Number of words of AP memory to save.
                   If .le. 0 the contents of the AP memory are not
                   saved.
 BUFFER(*)    R*4  Work buffer.
 BUFSZ        I*2  Size of BUFFER in bytes.
Inputs from COMMON /BPROLC/  (set by BPINIT)
 TRUEAP       L*2  True if a real AP (to be rolled)
 XTLAST       R*8  Real time AP assigned (min).
 DELTIM       R*8  Time interval between rolls (min).
 DELAY        R*4  Time to delay task (seconds).
Outputs:
 IRET         I*2  Return error code, 0=>OK
                                      2 => couldn't reload AP.
```

11.7.1.3 DSKFFT - a disk based, two dimensional FFT. The data are stored by rows. To save an extra transposition, the input array is assumed to have been written in transposed order and in the "center at the corners" convention for the uv to sky transform.

```
   DSKFFT (NR, NC, IDIR, HERM, LI, LW, LO, LENBUF, NBUF,
  *   APSIZ, 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. (forward = sky=>uv, reverse = uv=>sky) 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 .ge. 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. (may be LI if LW isn't, may NOT be LW)
LENBUF	I*2	Buffer length in bytes. LENBUF must be a power of two, and the buffer must be long enough to hold a row of the input array and to hold a row of the output array. The buffers are passed in COMMON /BUFRS/BUF(*).
NBUF	I*2	Number of buffers: either 1 or 2.
APSIZ	R*4	Size of AP main data memory in words.
Output:		
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.

NOTE: DSKFFT also uses Commons /BUFRS/ and /CFILES/

11.7.1.4 PEAKFN - searches a region around the center of an image to locate the pixel location of the maximum. Will handle data cubes and either integer or floating images.

```
PEAKFN (LUN, VOL, CNO, IDEPTH, CATBLK, IBUFF,
*   BUFFER, JBUFSZ, PEAKX, PEAKY, IRET)
```

Inputs:		
LUN	I*2	Logical unit number to use.
VOL	I*2	Disk on which image resides.
CNO	I*2	Catalog slot number of image.
IDEPTH(5)	I*2	Depth in image of desired plane.
CATBLK(256)	I*2	Catalog header block for image.
IBUFF(*)	I*2	Integer work buffer.
BUFFER(*)	R*4	Real work buffer should be physically the same as IBUFF.
JBUFSZ	I*2	Size of the IBUFF/BUFFER in bytes
Output:		
PEAKX	R*4	X coordinate of peak pixel location.
PEAKY	R*4	Y coordinate of peak pixel location.

IRET	I*2	Return code, 0=> OK, otherwise error.

11.7.1.5 PLNGET - reads a selected portion of a selected plane parallel to the front and writes it into a specified scratch file. The output file will be zero padded and a shift of the center may be specified. Output file is REAL*4 but the input may be either INTEGER*2 of 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 are given, 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 catalogue 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.

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

11.7.1.6 ZP4I4 - Converts Pseudo I*4 integer to true I*4. (Currently only converts first unsigned 16 bits to I*4)

```
  ZP4I4 (P4, I4)
Input:
  P4  I*2(2)   pseudo I*4 value
Output:
  I4  I*4      I*4 value
```

11.7.2 Array Processor Routines

The names and functions of the general purpose AP routines are given in the following brief list. A number of specialized routines for CLEANing, gridding uv data and model computations have been omitted.

- APGET (HOST, AP, N, TYPE) Transfer data from AP to host
- APGSP (I, NREG) Reads the value of an SPAD register (FPS and pseudo)
- APPUT (HOST, AP, N, TYPE) Transfer data from host to AP.
- APRFT (UDATA, UFT, UPH0, NFT, NDATA) Computes real, inverse Fourier transform from arbitrarily spaced data.
- APWAIT (no arguments) Suspends host until all transfers and computations are complete.
- APWD (no arguments) Suspends host until all transfers of data are complete.
- APWR (no arguments) Suspends host until all computations are complete.
- BOXSUM (A, I, NB, C, J, N) Does a boxcar sum on a vector.
- BPINIT (I1, I2, I3) Assigns and initializes AP.
- BPRLSE (no arguments) Releases the AP
- CFFT (C, N, F) Complex FFT.
- CRVMUL (A, I, B, J, C, K, N) Complex - real vector multiply.
- CSQTRN (CORNER, SIZE, ROW) In-place transpose of square complex matrix.
- CVCMUL (A, I, B, C, J, N) Multiplies a complex scalar times the complex conjugate of a complex vector producing a real vector.

- CVCONJ (A, I, C, K, N) Take complex conjugate of complex vector.
- CVEXP (A, I, C, K, N) Complex vector exponentiation.
- CVJADD (A, I, B, J, C, K, N) Adds a complex vector to the complex conjugate of another complex vector.
- CVMAGS (A, I, C, K, N) Complex vector magnitude squared.
- CVMMAX (A, I, C, N) Finds the maximum square modulus of a complex vector.
- CVMOV (A, I, C, K, N) Copy one complex vector to another.
- CVMUL (A, I, B, J, C, K, N, F) Multiply two complex vectors.
- CVSDIV (A, I, B, C, J, N) Divide a weighted complex vector by a complex scalar, weight is multiplied by the amplitude of the scalar.
- CVSMS (A, I, B, C, J, D, K, N ,FLAG) Subtract a real vecrot tiems a complex scalar from a complex vector.
- DIRADD (A, IA, B, N) Complex directed add.
- HIST (A, I, C, N, NB, AMAX, AMIN) Compute histogram of a vector.
- LVGT (A, I, B, J, C, K, N) Logical vector greater than.
- MAXMIN (A, I, MAX, MIN, N) Find maximum and minimum values in a vector.
- MAXV (A, I, C, N) Find maximum in an array.
- MINV (A, I, C, N) Find minimum in an array.
- MTRANS (A, I, C, K, MC, NC) Matrix transpose.
- PHSROT (A, I, B, J, PHAS0, DELPHS, N) Imposes a phase gradient on a complex vector.
- POLAR (A, I, C, K, N) Rectangular to polar conversion.
- RECT (A, I, C, K, N) Polar to rectangular conversion.
- RFFT (C, N, F) Real to complex or vice versa fast Fourier transform.
- SVE (A, I, C, N) Sum of vector elements.
- SVESQ (A, I, C ,N) Sum of the square of the elements of a vector.

- VABS (A, I, C, K, N) Vector absolute value.
- VADD (A, I, B, J, C, K, N) Vector add.
- VCLIP (A, I, B, C, D, L, N) Vector clip.
- VCLR (C, K, N) Vector clear.
- VCOS (A, I, C, K, N) Vector cosine.
- VDIV (A, I, B, J, C, K, N) Vector division.
- VEXP (A, I, C, K, N) Vector exponentiation.
- VFILL (A, C, K, N) Vector fill.
- VFIX (A, I, C, K, N) Vector real to integer.
- VFLT (A, I, C, K, N) Vector integer to real.
- VIDIV (A, I, D1, D2, B, J, N) Divide a vector by the product of two scalar integers.
- VLN (A, I, C, K, N) Vector natural logarithm.
- VMA (A, I, B, J, C, K, D, L, N) Vector multiply and add.
- VMOV (A, I, C, K, N) Copy one vector to another.
- VMUL (A, I, B, J, C, K, N) Vector multiply.
- VNEG (A, I, C, K, N) Take negative of a vector.
- VRVRS (C, K, N) Reverse a vector.
- VSADD (A, I, B, C, K, N) Vector scalar add.
- VSIN (A, I, C, K, N) Vector sine.
- VSMA (A, I, B, C, K, D, L, N) Vector scalar multiply and add.
- VSMAFX (A, I, B, C, D, L, N) Vector scalar multiply, add and fix.
- VSMSA (A, I, B, C, D, L, N) Vector scalar multiply, scalar add.
- VSMUL (A, I, B, C, K, N) Vector scalar multiply.
- VSQ (A, I, C, K, N) Vector square.
- VSQRT (A, I, C, K, N) Vector square root.
- VSUB (A, I, B, J, C, K, N) Subtract two vectors.

- VSWAP (A, I, C, K, N) Swap two vectors.

- VTRANS (M, N, IAD, LV) Transpose a row stored M x N array of row vectors of length LV.

11.7.3 AP Routine Call Sequences

A note should be made about the conventions used in the description of the routines. Data addresses are normally denoted by A, B, C, or D and their increments (stride) by I, J, K, L and an element count by N. In the descriptions of the routines, many of the values in AP memory are referred by the name given to the variable giving the address, e.g., A(mI) is used to denote the value in memory location A + m*I. All input variables are pseudo I*4 unless otherwise marked.

11.7.3.1 APGET - Transfer data from AP memory to host core.

```
APGET (HOST, AP, N, TYPE)

Inputs:
  AP     P I*4    Target area in AP; 0-relative, increment=1
  N       I*2     Number of elements
  TYPE    I*2     Data type:
                  0   data is I*4 in host
                  1   data is I*2 in host
                  2   data is R*4 in host
Output:
  HOST(*) R*4/I*2 Data array in "host"
```

11.7.3.2 APGSP - Read contents of SPAD register. FPS and Pseudo AP only.

```
APGSP (I, NREG)

Inputs:
  NREG  I*2  SPAD register number desired
Outputs:
  I     I*2  Contents of the SPAD register.
```

11.7.3.3 APPUT - Transfer data from host memory to AP memory.

```
APPUT (HOST, AP, N, TYPE)
```

Inputs:

AP	P I*4	Target area in AP; 0 - relative, increment=1.
N	I*2	Number of elements
TYPE	I*2	Data type: 0 data is I*4 in host 1 data is I*2 in host 2 data is R*4 in host

Output:

HOST(*)	R*4/	Data array in "host"

11.7.3.4 APRFT - Computes a real, inverse fourier transform from arbitarily but uniformly spaced data.

```
APRFT (UDATA, UFT, UPH0, NFT, NDATA)
```

Inputs:

DATA	AP base address of input data.
FT	AP base address of output F. T.
PH0	AP base address of phase information for F. T. 0=COS((TWOPI/(NG*NFT))*(1-ICENT)(1-BIAS)) 1=SIN((TWOPI/(NG*NFT))*(1-ICENT)(1-BIAS)) 2=COS((TWOPI/(NG*NFT))*(1-ICENT)) 3=SIN((TWOPI/(NG*NFT))*(1-ICENT)) 4=COS((TWOPI/(NG*NFT))*(1-BIAS)) 5=SIN((TWOPI/(NG*NFT))*(1-BIAS)) 6=COS((TWOPI/(NG*NFT))) 7=SIN((TWOPI/(NG*NFT))) ICENT = center pixel of grid BIAS = center of data array (1 rel) NG = No. tabulated points per cell.
NFT	Number of FT points
NDATA	Number of data points.

11.7.3.5 APWAIT - Suspend host task until all AP I/O and computations are complete.

```
APWAIT
```

11.7.3.6 APWD - Suspend host task until all AP I/O is complete.

```
APWD
```

11.7.3.7 APWR - Suspend host task until all AP computations are complete.

```
APWR
```

11.7.3.8 BOXSUM - Do a boxcar sum on a vector; values at the ends of the vector are the sum of the values within one boxcar length of the ends.

```
BOXSUM (A, I, NB, C, J, N)
```

Inputs:

A	input vector address
I	input vector increment
NB	boxcar width
C	output vector address; output vector should not overlap input
J	output increment
N	number of elements

11.7.3.9 BPINIT - Implements AIPS AP priority for true AP, increases the task priority for AIPS batch tasks using a true AP and assigns an AP.

```
BPINIT (I1, I2, I3)
```

Inputs:

I1	I*2	Dummy
I2	I*2	Dummy

Outputs:

I3	I*2	AP number (Neg. to indicate virtual AP, ie. not to be rolled.

11.7.3.10 BPRLSE - Releases the AP, lowers task priority for AIPS batch tasks using a true AP.

```
BPRLSE
```

11.7.3.11 CFFT - Do an in-place complex fast Fourier tasnsform.

```
CFFT (C, N, F)
```

Inputs:

C	Base address (0-rel) of complex array to transform
N	Number of points in array (must be power of two.
F	Transform direction; 1 -> Forward -1 -> Backward

11.7.3.12 CRVMUL - Multiply the elements of a complex vector by the elements of a real vector.

```
C(mK)+iC(mK+1) = (A(mI)*B(mJ)) + i(A(mI+1)*B(mJ))
                                 m=0 to N-1
```

```
CRVMUL (A, I, B, J, C, K, N)
```

Inputs:

A	Source complex vector base address.
I	Increment of A
B	Source real vector base address
J	Increment of B
C	Destination vector base address
K	Increment of C
N	Element count

11.7.3.13 CSQTRN - Do an inplace transpose of square matrices of complex values.

```
CSQTRN (CORNER,SIZE,ROW)
```

Inputs:

CORNER	AP location of first corner of matrix encountered.
SIZE	Size (number of reals) of a row or column.
ROW	Number of locations in AP between beginnings of the rows.

11.7.3.14 CVCMUL - Multiply a scalar complex value times the complex conjugate of a vector producing a real vector.

```
C(K) = REAL(B)*A(K)+IMAG(B)*A(K+1)  K = 1,N

   CVCMUL (A,I,B,C,J,N)
```

Inputs:

A	Source complex vector base address.
I	Increment of A
B	Address of scalar (real part)
C	Destination real vector base address.
J	Increment of C
N	Element count (reals)

11.7.3.15 CVCONJ - Take complex conjugate of a vector.

```
C(k) = Re(A(k)) - i * Im(A(k)
                                   for k = 0,N-1
CVCONJ (A, I, C, K, N)
```

Inputs:

A	Source vector base address.
I	Increment of A
C	Destination vector base address
K	Increment of C
N	Element count

11.7.3.16 CVEXP - Exponentiate a complex vector.

```
C(mK) + iC(mK+1) = COS (A(mI)) + i SIN (A(mI))
                              m = 0 to N-1

CVEXP (A, I, C, K, N)
```

Inputs:

A	Source vector base address.
I	Increment of A
C	Destination vector base address
K	Increment of C
N	Element count

11.7.3.17 CVJADD - Add the elements of one complex vector to the complex conjugate of the elements of another complex vector.

$$C(k) = Re(A(k))+Re(B(k))+i\ (Im(A(k))-Im(B(k))) \quad \text{for } k = 0,N-1$$

CVJADD(A,I,B,J,C,K,N)

Inputs:

A	Source vector base address.
I	Increment of A
B	Source vector base address (conjugate)
J	Increment of B
C	Destination vector base address
K	Increment of C
N	Element count

11.7.3.18 CVMAGS - Square the magnitude of the elements of a complex vector.

$$C(mK) = A(mI)^{**}2+A(mI+1)^{**}2 \text{ for } m = 0,N-1$$

CVMAGS (A, I, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.19 CVMMAX - Find the maximum of the square modulus of a complex vector.

CVMMAX (A, I, C, N)

Inputs:

A	Source vector base address
I	Increment of A
C	Destination vector. 0 = MAX(A ** 2) (real) 1 = location of max (integer) SPAD(15) = index of max.

11.7.3.20 CVMOV - Copy one complex vector to another.

```
CVMOV (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.21 CVMUL - Multiply the elements of two complex vectors.

```
(C(mK)+iC(mK+1)) = (B(mJ)+iB(mJ+1)*(A(mI)+iA(mI+1)) if F=1
(C(mK)+iC(mK+1)) = (B(mJ)+iB(mJ+1)*(A(mI)-iA(mI+1)) if F=-1

CVMUL (A, I, B, J, C, K, N, F)
```

Inputs:

A	Source vector base address
I	A address increment
B	Source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count
F	Conjugate flag, 1 => normal complex multiply -1 => multiply with conj of A

11.7.3.22 CVSDIV - Divide the elements of a complex vector with weights by a complex scalar. The complex vector is expected to have data in the order real, imaginary, weight. The weight is multiplied by the amplitude of the complex scalar. This is used for AIPS uv data.

```
C(mJ) = (1./(B(1)**2+B(2)**2))*(A(mI)*B(1)+A(mI+1)*B(2))
C(mJ+1) = (1./B(1)**2+B(2)**2))*(A(mI+1)*B(1)-A(mI)*B(2))
C(mJ+2) = A(mI+2) * SQRT(B(1)**2+B(2)**2)  for m = 0, N-1

CVSDIV (A, I, B, C, J, N)
```

Inputs:

A	Source vector base address.
I	Increment of A
B	Source scalar address.
C	Destination vector base address
J	Increment of C
N	Element count

11.7.3.23 CVSMS - Subtract the elements of a real vector times the elements of a complex scalar from a complex vector, alternately i (SQRT(-1)) times the real vector times the complex scalar is subtracted from the complex vector. Since the element count is expected to be small the looping is not very efficient.

```
If FLAG > 0
D(mK) = A(mI) - B(1) * C(mJ)
D(mK+1) = A(mI+1) - B(2) * C(mJ)  for m=0, N-1

If FLAG < 0
D(mK) = A(mI) - i * B(1) * C(mJ)
D(mK+1) = A(mI+1) - i * B(2) * C(mJ)  for m=0, N-1

 CVSMS (A, I, B, C, J, D, K, N ,FLAG)
```

Inputs:

A	Source complex vector base address.
I	Increment of A
B	Source complex scalar address.
C	Source real vector base address
J	Increment of C
D	Destination complex vector base address
K	Increment of D
N	Element count
FLAG	Flag, if < 0 multiply complex scalar by i

11.7.3.24 DIRADD - Do a complex directed add.

```
B(A(IA*J)) = B(A(IA*J))+A(IA*J+1)  for J = 0,N-1
B(A(IA*J)+1) = B(A(IA*J)+1)+A(IA*J+2)

 DIRADD (A,IA,B,N)
```

Inputs:

A	Source vector base address 0 => address (integer) to be added to (address is zero relative) 1,2 => complex value (reals)
IA	Increment for A
B	Destination vector base address
N	Element count

11.7.3.25 HIST - Compute the histogram of a vector. Histogram element (NB-1)*(DATA-MIN)/(MAX-MIN) where DATA is the data value is incremented.

```
 HIST (A, I, C, N, NB, AMAX, AMIN)
```

Inputs:

A	Source vector base address.
I	A address increment.
C	Histogram base address Histogram must be cleared before first call.
N	Element count for A
NB	Number of bins in histogram
AMAX	Address of histogram maximum.
AMIN	Address of histogram minimum.

11.7.3.26 LVGT - Logical vector greater than.

```
C(mK) = 1.0   if A(mI)>B(mJ)
C(mK) = 0.0   if A(mI)=<B(mJ)   for m = 0,N-1

LVGT (A, I, B, J, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
B	Source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.27 MAXMIN - Search the given vector for maximum and minimum values.

```
MAXMIN (A, I, MAX, MIN, N)
```

Inputs:

A	Source vector base address
I	Increment of A
MAX	Location for maximum.
MIN	Location for minimum.
N	Element count.

11.7.3.28 MAXV - Find maximum value of a vector and address of the maximum.

```
MAXV (A, I, C, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination base address

```
          C(0) = Max (A(mI))  m = 0 to N-1
          C(1) = address. also in SPAD 15.
     N    Element count
```

11.7.3.29 MINV - Find minimum value of a vector and address of the minimum.

```
      MINV (A, I, C, N)
   Inputs:
      A    Source vector base address
      I    A address increment
      C    Destination  base address
           C(0) = Max (A(mI))  m = 0 to N-1
           C(1) = address. also in SPAD 15
      N    Element count
```

11.7.3.30 MTRANS - Transpose a matrix.

```
     C((p+qMC)K) = A((q+pNC)I)
                   p = 0 to MC-1
                   q = 0 to NC-1

      MTRANS (A, I, C, K, MC, NC)

   Inputs:
      A    Source matrix base address
      I    A address increment
      C    Destination  matrix base address
      K    C address increment
      MC   Number of columns of A
      NC   Numbers of rows of A
```

11.7.3.31 PHSROT - Add a phase gradient to a complex array.

```
  B(j) = A(j)*EXP(-i*(PHAS0+j*DELPHS)) for j = 0,N-1

      PHSROT (A, I, B, J, PHAS0, DELPHS, N)

  Inputs:
  A             Source vector base address.
  I             Increment of A
  B             Destination base address.
  J             Increment of B
  PHAS0         Address of complex unit vector with
                phase PHAS0
  DELPHS        Address of complex unit vector with
```

	phase DELPHS
N	Element count

11.7.3.32 POLAR - Rectangular to polar conversion.

```
C(mK) = SQRT (A(mI)**2 + A(mI+1)**2)
C(mK+1) = ARCTAN (A(mI+1) / A(mI)) for m = 0 to N-1

POLAR (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.33 RECT - Polar to rectangular vector conversion.

```
C(mK) = A(mI) * COS (A(mI+1))
C(mK+1) = A(mI) * SIN (A(mI+1)) for m = 0 to N-1

RECT (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.34 RFFT - Does an in-place real to complex forward of complex to rear inverse FFT.

```
RFFT(C, N, F)
```

Inputs:

C	Base address of source and destiantion vector
N	Real element count (power of 2)
F	flag, 1=>forward FFT, -1=> reverse FFT.

11.7.3.35 SVE - Sum the elements of a vector

C = SUM (A(mI)) m = 0 to N-1

SVE (A, I, C, N)

Inputs:

A	Source vector base address.
I	Increment of A
C	Destination scalar address
N	Element count

11.7.3.36 SVESQ - Sum the squares of the elements of a vector

C = SUM (A(mI) * A(mI)) for m=0 to N-1

SVESQ (A, I, C ,N)

Inputs:

A	Source vector base address.
I	Increment of A
C	Destination scalar address
N	Element count

11.7.3.37 VABS - Take the absolute value of the elements of a vector.

C(mK) = ABS (A(mI)) for m = 0 to N-1

VABS (A, I, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.38 VADD - Add the elements of two vectors.

C(mK) = A(mI) + B(mJ) for m = 0 to N-1

VADD (A, I, B, J, C, K, N)

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment

C Destination vector base address
K C address increment
N Element count

11.7.3.39 VCLIP - Limits the values in a vector to a specified range.

D(mL) = B if A(mI) < B
= A(mI) if B <= A(mI) < C
= C if C <= A(mI) for m = 0 to N-1

VCLIP (A, I, B, C, D, L, N)

Inputs:
A Source vector base address
I A address increment
B Address of lower limit
C Address of upper limit
D Destination vector base address
L D address increment
N Element count

11.7.3.40 VCLR - Fill a vector with zeroes.

C(mK) = 0 for m = 0 to N-1

VCLR (C, K, N)

Inputs:
C Destination vector base address
K C address increment
N Element count

11.7.3.41 VCOS - Take the cosine of elements in a vector.

C(mK) = COS (A(mI)) for m = 0 to N-1

VCOS (A, I, C, K, N)

Inputs:
A Source vector base address
I A address increment
C Destination vector base address
K C address increment
N Element count

11.7.3.42 VDIV - Divide the elements of two vectors

C(mK) = B(mJ) / A(mJ) for m = 0 to N-1

```
VDIV (A, I, B, J, C, K, N)
```

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.43 VEXP - Exponentiate the elements of a vector.

C(mK) = EXP (A(mI)) for m = 0 to N-1

```
VEXP (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.44 VFILL - Fill a vector with a constant.

C(mK) = A for m = 0 , N-1

```
VFILL (A, C, K, N)
```

Inputs:

A	Source scalar base address
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.45 VFIX - Convert the elements of a vector from real to integer.

C(mK) = FIX (A(mI)) for m = 0 to N-1

```
VFIX (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.46 VFLT - Convert the elements of a vector from integer to real.

C(mK) = FLOAT (A(mI)) for m = 0 to N-1

VFLT (A, I, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.47 VIDIV - Divide the given vector by the product of two integers.

B(mJ) = A(mI)/(D1*D2) for m = 0,N-1

VIDIV (A, I, D1, D2, B, J, N)

Inputs:

A	Source vector base address.
I	Increment for A
D1	first dividend. Actual value, not an address.
D2	Second dividend. Actual value, not an address.
B	Destination vector base address.
J	Increment for B
N	Element count.

11.7.3.48 VLN - Take the natural logrithm of the elements of a vector.

C(mK) = LOGe (A(mI)) for m=0 to N-1

VLN (A, I, C, K, N)

Inputs:

A	Source vector base address.

I	Increment of A
C	Destination vector base address
K	Increment of C
N	Element count

11.7.3.49 VMA - Multiply two vectors and adds a third.

D(mL) = (A(mI) * B(mJ)) + C(mK) for m = 0, N-1

VMA (A, I, B, J, C, K, D, L, N)

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Third source vector base address
K	C address increment
D	Destination vector base address
L	D address increment
N	Element count

11.7.3.50 VMOV - Copy the elements of one vector to another.

C(mK) = A(mI) for m = 0, N-1

VMOV (A, I, C, K, N)

Inputs:

A	Source vector base address.
I	Increment of A
C	Destination vector base address
K	Increment of C
N	Element count

11.7.3.51 VMUL - Multiply the elements of two vectors.

C(mK) = A(mJ) * B(mJ) for m = 0 to N-1

VMUL (A, I, B, J, C, K, N)

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Destination vector base address
K	C address increment

N Element count

11.7.3.52 VNEG - Take the negative of the elements of a vector.

C(mK) = - A(mI) for m = 0 to N-1

VNEG (A, I, C, K, N)

Inputs:
- A Source vector base address
- I A address increment
- C Destination vector base address
- K C address increment
- N Element count

11.7.3.53 VRVRS - Reverse the elements in a vector.

C(mK) = C((N-m)K) for m = 0, N-1

VRVRS (C, K, N)

Inputs:
- C Source and destination vector base address
- K C address increment
- N Element count

11.7.3.54 VSADD - Add a scalar to the elements of a vector

C(mK) = B + A(mI) for m = 0, N-1

VSADD (A, I, B, C, K, N)

Inputs:
- A Source vector base address
- I A address increment
- B Adding scalar address
- C Destination vector base address
- K C address increment
- N Element count

11.7.3.55 VSIN - Take the sine of the elements of a vector.

C(mK) = SIN (A(mI)) for m = 0,N-1 (A in radians)

VSIN (A, I, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.56 VSMA - Multiply the elements of a vector by a scalar and adds to the elements of another vector.

D(mL) = (A(mI) * B) + C(mK) for m = 0, N-1

VSMA (A, I, B, C, K, D, L, N)

Inputs:

A	First source vector base address
I	A address increment
B	Source scalar base address
C	Second source vector base address
K	C address increment
D	Destination vector base address
L	D address increment
N	Element count

11.7.3.57 VSMAFX - Multiply the elements of a vector by a scalar, add a scalar and round to an integer.

D(mL) = FIX (ROUND((A(mI)*B)+C)) for m = 0,N-1

VSMAFX (A, I, B, C, D, L, N)

Inputs:

A	Source vector base address
I	A address increment
B	Multiplying scalar address
C	Adding scalar address
D	Destination vector base address
L	D address increment
N	Element count

11.7.3.58 VSMSA - Multiply the elements of a vector by a scalar and add a second scalar.

D(mL) = (A(mI)*B)+C for m=0,N-1

VSMSA (A, I, B, C, D, L, N)

Inputs:

A	Source vector base address
I	A address increment
B	Multiplying scalar address
C	Adding scalar address
D	Destination vector base address
L	D address increment
N	Element count

11.7.3.59 VSMUL - Multiply the elements of a vector by a scalar.

C(mK) = A(mI) * B for m = 0, N-1

VSMUL (A, I, B, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
B	Multiplying scalar address
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.60 VSQ - Square the elements of a vector

C(mK) = A(mI)**2 for m = 0 to N-1

VSQ (A, I, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.61 VSQRT - Take the square root of the elements of a vector.

C(mK) = SQRT (A(mI)) for m = 0,N-1

VSQRT (A, I, C, K, N)

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.62 VSUB - Subtract the elements of two vectors.

C(mK) = B(mJ) - A(mI) for m = 0 to N-1

VSUB (A, I, B, J, C, K, N)

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count

11.7.3.63 VSWAP - Swap the elements of a vector.

A(mI) = C(mK) and C(mK) = A(mI) for m = 0, N-1

VSWAP (A, I, C, K, N)

Inputs:

A	First source/destination vector base address
I	A address increment
C	Second source/destination vector base address
K	C address increment
N	Element count

11.7.3.64 VTRANS - Transpose a (row-stored) M X N array of row vectors of length LV. The starting address is given by IAD. The algorithm works in place. It is adapted from Boothroyd's CACM ALG.#302. Other, probably better, algorithms, are CACM #'S 380 and 467, but they're not as simple to program.

VTRANS (M, N, IAD, LV)

Inputs:

M	First dimension of the vector array
N	Second dimension of the vector array
IAD	Base address of the array
LV	Length of the vectors.

11.7.3.64 VTRANS - Transpose a (row-stored) M X N array of row vectors of length LV. The starting address is given by IAD. The algorithm works in place. It is adapted from Boothroyd's CACM ALG.#302. Other, probably better, algorithms, are CACM #'S 380 and 467, but they're not as simple to program.

VTRANS (M, N, IAD, LV)

Inputs:

M	First dimension of the vector array
N	Second dimension of the vector array
IAD	Base address of the array
LV	Length of the vectors.

CHAPTER 12

THE Z ROUTINES

12.1 OVERVIEW

All of the functions that are required by AIPS programs that depend on the host computer or operating system are performed in routines whose names begin with a "Z" and are referred to as the "Z routines". In principle, all that is required to make AIPS work on a new machine is to develop a disk file structure and create a set of Z routines to interface AIPS programs to the operating system and the file structure. If routines other than "Z" (or "Y") routines are modified then they will have to be modified every time the AIPS system is updated. For this reason we recommend that no routines other than "Z" (or "Y") routines should be modified. This chapter will describe the functions of the upper layer of Z routines; in any implementation there will probably be additional lower level machine-dependent routines. Careful study of an existing implementation of AIPS is recommended before attempting a new installation

For purposes of discussion the Z routines will be divided up into a number of overlapping categories:

1. Data Manipulation Routines - These routines convert data formats from external (tape) integers and characters to local and vice versa, and move bits and bytes.

2. Disk I/O and File Manipulation - These routines create, destroy, expand, contract, open, close, read, and write disk files.

3. System Functions - These routines do various system functions such as starting and stoping processes, inquiring what processes are running, and inquiring how much space is available on a given disk drive.

4. Device I/O - These routines talk to the terminals, the tape drive, graphics devices, image displays, etc. This area overlaps heavily with the disk I/O area.

5. Directory and Text File Routines - These routines read the directories for, and contents of, text files.

6. Miscellaneous - There are a number of routines such as that which initializes the Device Characteristics Common which do not easily fit in one of the other catagories.

7. Television I/O routines. These routines are discussed in the chapter on televisions and are not discussed further here.

A detailed description of the call sequences to each of these routines and listings of the relevant INCLUDE files are at the end of this chapter.

12.1.1 Device Characteristics Common

Many of the parameters describing the host operating system and installation in AIPS programs are carried in the Device Characteristics Common which is obtained using the includes IDCH.INC, DDCH.INC and CDCH.INC. The text of these include files can be found at the end of this chapter.

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 standalone utility program SETPAR. The constants kept in this common, the values in DEVTAB, and the use of logical unit numbers are described in the chapter on disk I/O.

12.1.2 FTAB

The FTAB array in the device characteristics common is used to keep AIPS and system I/O tables. The FTAB has separate areas for the three different kinds of I/O: 1) device I/O to devices which do non need I/O tables, 2) non-map or regular I/O which is single buffered, nonwait-mode I/O and 3) map I/O which can be double buffered, wait mode I/O. The FTAB has space for 16 integer words for application routine use and space for one system I/O table for non-map files and two system I/O tables for map files. The size of the entries in FTAB for the different types of I/O are carried in the Device Characteristics Common. The type of the I/O (map or non-map) is declared by the calling routine to the file/device open routine ZOPEN.

The FTAB is divided up by ZDCHIN into three areas, one for each type of I/O. These areas are described in the following:

1. Non-FTAB I/O - This area has NTAB1 entries each NBTB1 bytes long. The first integer word in each entry is zero if that entry is not in use and the LUN of the corresponding device if the entry is in use.

2. FTAB "non-map" I/O - This area has NTAB2 entries each NBTB2 bytes long. The first 16 integer words in each entry are reserved for application routines; the first of these is zero if that entry is not in use and the LUN of the corresponding device if the entry is in use. Following these 16 integers is space for one copy of whatever system I/O table is required for the host system.

3. FTAB "map" I/O - This area has NTAB3 entries each NBTB3 bytes long. The first 16 integer words in each entry are reserved for application routines; the first of these is zero if that entry is not in use and the LUN of the corresponding device if the entry is in use. Following these 16 integers is space for two copies of whatever system I/O table is required for the host system.

Note that a byte is defined in this manual as half a short integer.

12.1.3 Disk Files

The AIPS system uses binary files for data and text files for source code and control information. The location and physical name of the various files depends very much on the host system and installation. The physical name of a file is derived by ZPHFIL and the location of a file is determined by ZPHFIL and ZOPEN (or ZTOPEN for text files).

12.1.3.1 Binary (data) Files - Binary files are divided into two types, "map" and "non-map" files corresponding to the two types of I/O. (It should be noted however that "non-map" I/O routines should work on "map" files.) Normally most AIPS binary files on a given disk are put in a single area or directory. Current implementations of AIPS use 8 characters for the basic physical name and 3 more if private catalogues are supported.

An example from a VAX system with private catalogues is "DA0n:ttdcccvv.uuu" ; where n is the zero relative disk drive number, DA0n: is a logical variable which is assigned to a directory, tt is a two character file type (eg. 'MA'), d is the one relative disk drive number, ccc is the catalogue slot number, vv is the version (01 for "MA" and "UV" files), and uuu is the users number in hexidecimal notation.

"Map" type files are files on which it should be possible to double buffer. It should be possible to contract "map" files but it is not necessary to expand "map" files so these files may be forced to be contigious on the disk. Contigious files are more efficient but they cause problems for users with large files. These files should be capable of random access with I/O beginning on a disk sector boundary.

"Non-map" files should be expandable and contractable. These files should be capable of random access with I/O beginning on a disk sector boundary.

12.1.3.2 Text Files - Text files are used primarily for storing source code and control information such as the RUN and HELP/INPUT files. Currently text files may be read but not written using AIPS routines. The source code routines are accessed primarily by AIPS managment routines such as the AIPS manual printing program.

Different types of text files are kept in different areas which have directories. The type of the text file is specified to ZPHFIL as one of several types; the directory may be further selected by the ZTOPEN argument VERSON which can specify the version (directory or area). The member (or file) name is specified to ZTOPEN and may contain up to eight characters. These types and the files kept in each area are described in the following:

- HE - These are the HELP files which specify which AIPS adverbs are to be sent to tasks and contain the primary user documentation.

- IN - Same as HE. This is a relic of older versions in which the HELP files and INPUTS files were distinct.

- RU - The RUN files usually contain instructions for the AIPS program. Other types of text files may appear in this area as input for AIPS tasks.

- DC - These are the programmer documentation files, primarily sections of the AIPS manual.

- SO - These are the "standard" source code routines. These routines are those which should conform to all AIPS coding standards.

- SR - This area contains source code which is used only by the AIPS program and standalone utility programs but not AIPS tasks.

- SI - This area contains the include files.

- SN - This area contains source code which has not been determined to meet all AIPS coding standards. Code in this area may give problems in a new installation.

- SF - This area contains the source code for the true array processor routines.

- SP - This area contains the source code for the pseudo array processor routines.

- SL - This area contains miscellaneous files.

12.2 DATA MANIPULATION ROUTINES

The internal form in which characters and integers are stored varies from computer to computer but a given FITS data tape should be able to be read on any AIPS system. Thus it is necessary to be able to convert between the external (FITS) formats to the internal formats. The format of data on FITS tape files is discudded in another chapter.

The following list gives the names and uses of the upper level data manipulation "Z" routines; in practical installations more Z routines will be required. Details of the call sequences of the upper lever, non TV Z routines are found at the end of this chapter.

- ZCLC8 - converts local characters to standard ASCII.

- ZC8CL - converts standard ASCII to the local characters.

- ZI16IL - converts standard 16 bit integers to the local short integer.

- ZI32IL - converts standard 32 bit integers to a pair of local short integers.

- ZI8L8 - converts 8 - bit unsigned binary numbers to bytes.

- ZILI16 - converts local short integers to external format 16 bit integers.

- ZP4I4 - converts pseudo I*4 to true I*4.

- ZR8P4 - converts between pseudo I*4 and REAL*8.

12.3 DISK I/O AND FILE MANIPULATION ROUTINES

This section describes the routines needed for manipulating disk data (binary) files. The physical names of disk data files are always constructed by ZPHFIL and these files are always opened by ZOPEN. There are separate routines for writing to the message file (ZMSGCL, ZMSGDK, and ZMSGOP) to avoid recursion when reporting an error message from one of the I/O routines.

A short description of the disk file routines are given in the following list; detailed descriptions of the call sequences are given at the end of the chapter.

- ZCLOSE - closes disk files or devices.
- ZCMPRS - contracts disk files.
- ZCREAT - creates disk files.
- ZDESTR - destroys disk files
- ZEXIST - determines if a given file exists.
- ZEXPND - expands "non-map" files.
- ZFIO - does "non-map" (single buffer) I/O to disk files and devices.
- ZMIO - does "map" (double buffer, wait mode) I/O to disk files and devices.
- ZMSGCL - closes the message file.
- ZMSGDK - writes to the message file.
- ZMSGOP - opens the message file.
- ZOPEN - opens disk files and devices.
- ZPHFIL - constructs physical file names.
- ZRENAM - changes the physical name of a file.

- ZWAIT - suspends the calling task until an I/O operation initiated by ZMIO is complete.

12.4 SYSTEM FUNCTIONS

There are a number of functions involving processes or system resources which must be done in a system dependent fashion. These include controlling processes (starting, killing, suspending and resuming) and determining the time, date, name of the current process, and the amount of CPU time used by the current task. Some of these may require special privileges.

The AIPS interactive program may start independent processes called tasks. These tasks do most of the computations. After AIPS has started a task it suspends itself indefinately; the newly started task then restarts AIPS either at the start or the end of its operation as specified by the user. There may be several tasks and the AIPS program running at a given time.

Communication between AIPS and tasks is rather primitive, AIPS writes a disk file with the control information which the task reads. When the task resumes AIPS it returns a completion error code.

The following list gives a short description of these routines; complete descriptions of the call sequence can be found at the end of this chapter.

- ZCPU - returns the amount of CPU time used by the current process.
- ZDATE - returns the current calender date.
- ZDELAY - delays the current task for a specified period.
- ZPRIO - raises or lowers a tasks priority.
- ZTACTQ - checks if a given process is active.
- ZTIME - returns the current time.
- ZTRSUM - resumes a specified task.
- ZACTV8 - activates a specified process.
- ZFREE - determines the amount of disk space available on each of the disks.

- ZSTAIP - restores the process to its normal state on the completion of an interactive AIPS process.
- ZSUSPN - suspends the executing task.
- ZTKILL - kills (aborts) a specified task.
- ZTQSPY - writes a list of the current AIPS processes running to the user monitor terminal and the message file.
- ZWHOMI - returns the name of the executing task.

12.5 DEVICE (NON-DISK) I/O ROUTINES

Many of the routines discussed in the disk I/O section will also work on other devices. There are a number of special functions required for non-disk devices. One example of these is the routine to talk to a terminal; some operating systems don't allow Fortran I/O to a terminal so this I/O is done through the routine ZTTYIO.

The following list gives a short description of these routines; complete descriptions of the call sequence can be found at the end of this chapter.

- ZDOPRT - plots a bit map onto the plotter.
- ZENDPG - does a page eject on the line printer.
- ZTAPE - positions a tape and writes file marks.
- ZTKBUF - formats the output buffer for the graphics output device.
- ZTTYIO - reads and writes to the terminal.
- ZTVMC - clears the image display.
- ZPRMPT - does a prompted read from the terminal.

12.6 DIRECTORY AND TEXT FILE ROUTINES

Text files are used for source code and control information and have been discussed previously in this chapter. Currently text files may be read but not written from AIPS routines.

The following list briefly describes the function of the special routines for text files; detailed descriptions of the call sequences are found at the end of this chapter.

- ZTOPEN - opens a text file.
- ZTREAD - reads a text file.
- ZTCLOS - closes a text file.
- ZTXMAT - searches a directory for files whose names begin with a given character string.
- ZGTDIR - returns the directory for a text file area.

12.7 MISCELLANEOUS

Several Z routines don't naturally fit in any of the above categories. The following list gives a brief description of each; details of the call sequence and function are given at the end of this chapter.

- ZDCHIN - initializes the Device Characteristics Common.
- ZMATH4 - does pesudo I*4 arithmetic.
- ZTFILL - initializes the FTAB array in the Device Characteristics Common.
- ZKDUMP - dumps an array to the user message monitor and message file in a number of different formats.

12.8 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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statments.
- Exxx.INC. These contain Fortran EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statments.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible 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.

12.8.1 CDCD.INC

```
C                                                       Include CDCH
      COMMON /DCHCOM/ NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR,
     *   CSIZPR, NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NBITWD,
     *   NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, XPRDMM, XTKDMM,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, SYSNAM, VERNAM, USELIM,
     *   IFILIT, RLSNAM
      COMMON /FTABCM/ DEVTAB, FTAB
C                                                       End CDCH.
```

12.8.2 CMSG.INC

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

12.8.3 DDCH.INC

```
C                                                               Include DDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *   CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *   NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *   NTKACC, UCTSIZ, BYTFLP, USELIM, IFILIT,
     *   DEVTAB(50), FTAB(1)
C                                                               End DDCH.
```

12.8.4 DMSG.INC

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

12.8.5 IDCH.INC

```
C                                                               Include IDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *   NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *   CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *   NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *   NTKACC, UCTSIZ, BYTFLP, IFILIT,
     *   USELIM, DEVTAB(50)
C                                                               End IDCH.
```

12.9 ROUTINES

12.9.1 Data Manipulation

12.9.1.1 ZCLC8 - converts local characters in a buffer to standard 8-bit ASCII in another buffer - which may be the same buffer!!

```
ZCLC8 (NCHAR, INB, NP, OUTB)
```

Inputs:	NCHAR	I*2	Number of characters
	INB	R*4(*)	Input buffer in local chars: start at 1
	NP	I*2	Start index in output buffer in units of 8-bit chars, 1-relative
Output:	OUTB	R*4(*)	Output buffer

12.9.1.2 ZC8CL - extracts 8-bit ASCII standard characters from a buffer and stores them in the local character form. Must work even when INARR and OUTARR start at the same address.

```
ZC8CL (NCHAR,NP,INARR,OUTARR)
```

Inputs:	NCHAR	I*2	Number of characters to extract
	NP	I*2	Start position in input buffer in units of 8-bit characters
	INARR	R*4(*)	Input buffer
Output:	OUTARR	R*4(*)	Output buffer

12.9.1.3 ZI16IL - ZI16IL extracts 16-bit, 2's complement integers from a buffer and puts them into the local small integer form. Must work even when INB and OUTB have the same address.

```
ZI16IL (NVAL, NP, INB, OUTB)
```

Inputs:	NVAL	I*2	# values to extract
	NP	I*2	start position in input counting from 1 in units of 16-bit integers
	INB	I*2(*)	Input buffer
Output:	OUTB	I*2(NVAL)	Output buffer

12.9.1.4 ZI32IL - extracts 32-bit, 2's complement integers from a buffer and puts them into the local small integer form. Must work even when INB and OUTB have the same address. The IBM order must apply to the output: i.e. the most significant part of the 32-bit integer must be at a lower index in OUTB than the least significant part. They will be picked up into standard pseudo I*4 via IP(2) = OUTB(i), IP(1) = OUTB(i+1).

ZI32IL (NVAL, NP, INB, OUTB)

```
Inputs: NVAL    I*2       # values to extract
        NP      I*2       start position in input counting from 1
                          in units of 32-bit integers
        INB     I*2(*)    Input buffer
Output: OUTB    I*2(2*NVAL)  Output buffer
```

12.9.1.5 ZI8L8 - converts 8-bit unsigned binary numbers to "bytes" (one-half of a local small integer). Must work when input and output buffers are the same.

ZI8L8 (NVAL, NP, INB, OUTB)

```
Inputs: NVAL    I*2       # values
        NP      I*2       First value to get from INB counting from 1
                          in units of 8-bit numbers
        INB     I*2(*)    Input buffer
Output: OUTB    I*2(NVAL/2)   Output buffer
```

12.9.1.6 ZILI16 - converts a buffer of local small integers to a buffer of standard 16-bit, 2's complement integers.

ZILI16 (NINT, INB, NP, OUTB)

```
Inputs: NINT    I*2       Number of integers
        INB     I*2(*)    Input buffer: start at index 1
        NP      I*2       start point in output buffer 1-relative in
                          units of standard 16-bit integers
Outputs:OUTB    I*2(*)    Out buffer
```

12.9.1.7 ZP4I4 - Converts Pseudo I*4 integer to true I*4.

```
ZP4I4 (P4, I4)
 Input:
  P4  I*2(2)   pseudo I*4 value
 Output:
  I4  I*4      I*4 value
```

12.9.1.8 ZR8P4 - converts between pseudo I*4 and R*8.

```
      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
    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.
```

12.9.2 Disk I/O

12.9.2.1 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.

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

   Inputs:  IVOL     I*2  volume number
            PNAME    R*4(6)  physical file name
            ISIZE P  I*4  original size bytes
            LSIZE P  I*4  desired final size bytes
            LUN      I*2  logical unit number under which file is
                          open
   Outputs: IERR     I*2  error code: 0 => ok
                                      1 => input data error
                                      2 => compress error FMGR
```

12.9.2.2 ZCREAT - Creates a disk file.

```
      ZCREAT (IVOL,PHNAME,NBYTE,MAP,IERR)

   Inputs:
      IVOL     I*2      Disk drive unit number.
      PHNAME   R*4(6)   Physical file name given by ZPHFIL.
                        left justified, padded with blanks.
      NBYTE P  I*4      Size of the file in bytes. Will be rounded to
                        next higher 512 byte block boundary.
      MAP      L*2      True if map file.
   Outputs:
      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.

12.9.2.3 ZDESTR - Destroy 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.
Output:
  IERR   I*2      Completion code. 0=good.  1=failed.
                                   1=file not found
                                   2=failed
```

12.9.2.4 ZEXIST - determines if a file exists. If so the size of the file is returned.

```
ZEXIST (IVOL, PHNAME, ISIZE, SCRTCH, IERR)

 Inputs:
   IVOL    I*2      The disk volume to seach.  Not used on the VAX.
                    This information is found in PHNAME.
   PHNAME  R*4(6)   File name.
 Outputs:
   ISIZE   I*2      Size of the file in 512 byte blocks.
   SCRTCH  I*2(256) Scratch buffer.  Not used on the VAX.
   IERR    I*2      Error code 0 = file exists, 1=file not found,
                    2 = other.
```

12.9.2.5 ZEXPND - increases the size of a 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
```

12.9.2.6 ZFIO - reads or writes one logical record between core and device LUN. For disk devices, the record length is always 512 bytes (a byte being defined as half of a short integer). NREC gives the random access record number (in units of 512 bytes). For non-disk devices, NREC contains the number of bytes.

```
   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*2  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 -> I/O error
                             4 -> end of file
                             5 -> begin of medium
                             6 -> end of medium
```

12.9.2.7 ZMIO - a low level random access, large record, double buffered device I/O routine.

```
   ZMIO (OP, LUN, FIND, BLKNO, NBYTES, BUFF, IBUFF, IERR)
Inputs:
   OP      R*4  Operation - 'READ', 'WRIT'. ASCII - 4 characters.
   LUN     I*2  Logical unit number of a previously opened map.
   FIND    I*2  Pointer to FTAB returned by ZOPEN.
   BLKNO P I*2  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)
```

12.9.2.8 ZMSGCL - closes message file associated with LUN removing any exclusive use state and clears up the FTAB.

```
      ZMSGCL (LUN, FIND, IERR)

   Inputs:  LUN  I*2  logical unit number
   Output:  IERR I*2  error code: 0 -> no error
                                  1 -> Deaccess or Deassign error
                                  2 -> file already closed in FTAB
                                  3 -> both errors
                                  4 -> erroneous LUN
```

12.9.2.9 ZMSGDK - reads or writes one 512-byte logical record betwen core buffer BUF and disk unit LUN. Special version for message writing.

```
      ZMSGDK (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*2  record number in file: starts with 1
      BUF    I*2  (256) array to hold record
   Output:
     IERR    I*2  error code: 0 => ok
                              1 -> file not open
                              2 => input error
                              other -> I/O error
```

12.9.2.10 ZMSGOP - opens message files, performing full RMS open on disk files for which LUN > NDEVT.

```
      ZMSGOP (LUN, IND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)

   Inputs:
      LUN    I*2    Fortran Logical file number.
      IVOL   I*2    Disk volume containing file, 1,2,3,...
      PNAME  R*4(2) 8 Character physical file name,left justified
      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 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

12.9.2.11 ZOPEN - opens logical files, performing full open on disk files for which LUN > NDEVT. Tape units are assigned an I/O channel and given an FTAB entry for double buffering.

```
   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
```

12.9.2.12 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. This routine contains the logical assignment list for Graphics devices and is thus site dependent as well as machine dependent.

```
EXAMPLE: If ITYPE='MA', IVOL=8, NSEQ=321, IVER=99, NLUSER=762 then
            PNAME='DA07:MA832199;1' for public data or
            PNAME='DA07:MA832199.762;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  Number of the disk volume to be used.
   NSEQ     I*2  User supplied sequence number. 000-999.
   IVER     I*2  User suppplied version number. 00-255.
Outputs:
```

```
      PNAM  R*4(6) >= 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.
```

12.9.2.13 ZRENAM - Rename a disk file.

```
      ZRENAM (IVOL, NAME1, NAME2, IERR)

   Inputs:
      IVOL   I*2    Volume number (1 relative).
      NAME1  R*4(6) Old file name. 24 char., left justified, padded
                    on the right with blanks, and packed.
      NAME2  R*4(6) New file name. like NAME1.
   Outputs:
      IERR   I*2    error code.
                    0 = successful completion
                    2 = old file not found
                    3 = volume not found or not ready
                    6 = new file name already exists in directory.
                    7 = other errors.
```

12.9.2.14 ZWAIT - waits until an I/O operation started by ZMIO is complete.

```
      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
                              7 => wait service error
```

12.9.3 System Functions

12.9.3.1 ZCPU - determines cumulative cpu usage in seconds for this process: i.e. each time a process calls ZCPU during an execution, TIME is larger.

```
      ZCPU (TIME, IOCNT)

   Output: TIME    R*4      Current CPU accumulation in seconds
           IOCNT P I*2      I/O count
```

12.9.3.2 ZDATE - returns local time of day.

```
      ZDATE (ID)

   Output: ID(1)   year since 0.
           ID(2)   month (1-12).
           ID(3)   day   (1-31).
```

12.9.3.3 ZDELAY - Cause a delay of time determined by the argument.

```
      ZDELAY (SECS, IERR)

   Input:
      SECS  R*4  Number of seconds to delay.
   Output:
      IERR  I*2  Error code. 0 = ok,  1 = error.
```

12.9.3.4 ZPRIO - changes the current program's machine priority between that of batch programs and that of interactive programs. This routine is used by tasks using true array processors.

```
      ZPRIO (OP, IERR)

   Inputs: OP    R*4    'UPPP' to inter., 'DOWN' to batch
           IERR  I*2    Error code: 0 => ok
                                    1 => bad OP
                                    2 => illegal request
                                    3 => other failures
```

12.9.3.5 ZTACTQ - determines if a specified task is active.

```
      ZTACTQ (NAME, ACTIVE, IERR)

   Inputs:  NAME      I*2(3)     actual task name.(2 char/integer)
   Output:  ACTIVE    L*2        T => task active.
            IERR      I*2        error number:
                                     0 => ok.
                                     1 => invalid task name.
```

12.9.3.6 ZTIME - returns the local time of day.

```
      ZTIME (IT)

   Output:  IT(1) I*2  hours  (0-23)
            IT(2) I*2  min    (0-59)
            IT(3) I*2  sec    (0-59)
```

12.9.3.7 ZTRSUM - resumes a hibernating task.

```
      ZTRSUM (CNAME, IERR)

   Inputs:    CNAME  I*2(3) task name.
   Outputs:   IERR   I*2    error code:
                                0 => ok
                                1 => invalid task name.
                                2 => insufficient privilege.
```

12.9.3.8 ZFREE - This routine will calculate the number of free 512 byte blocks that are available on the disks used for AIPS data and print the information on the screen. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZFREE (IERR)

   Inputs:
       From common /DCHCOM/
       NVOL   I*2  Number of AIPS disks.
   Output:
       IERR   I*2  0 = ok, 1=error in disk logical name.
```

12.9.3.9 ZSTAIP - performs any operations needed to normalize the local operating system at the conclusion of an interactive AIPS session. This routine is normally kept in the AIPS program library (not for tasks).

```
ZSTAIP (SCRTCH)

Outputs:  SCRTCH   I*2(256)    Scratch buffer
```

12.9.3.10 ZSUSPN - puts an executing process into hibernation. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
ZSUSPN (NPOPS, IERR)

Inputs:   NPOPS   I*2    pops number (not used here).
Output:   IERR    I*2    error no.; 0 => ok
```

12.9.3.11 ZTKILL - will delete the subprocess specified by NAME. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
ZTKILL (NAME, IERR)

Inputs:  NAME      I*2(3)     actual task name.(2 char/integer)
Output:  IERR      I*2        error number:
                                   0 => ok.
                                   1 => error.
```

12.9.3.12 ZTQSPY - obtains entire list of AIPS-originated tasks now running in system and prints info about them. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
ZTQSPY (TLIST)

Output:  TLIST   I*2(256)   Scratch buffer
```

12.9.3.13 ZWHOMI - determines the actual task name under which the present version of AIPS is running. It uses this information to set the value of NPOPS in the common /MSGCOM/. It then assigns the TV and TK devices setting NTVDEV and NTKDEV in common /DCHCOM/. It checks for remote entries at this stage and uses the true device numbers (set by ZDCHIN) to do the assignments. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZWHOMI (IERR)

   Output: IERR  I*2   error code:  0   ok.
                       1 => task is AIPS, but NPOPS illegal.
                       2 => task is not AIPS.
```

12.9.4 Non-disk I/O Routines

12.9.4.1 ZDOPRT - read a bit map such as produced by PRTDRW and convert it into a FORTRAN file that can be spooled to the printer-plotter as a plot.

```
      ZDOPRT (IVOL, IBMLUN, NCOPY, FILNAM, DESTRY, ISIZE,
     *   INBLK, IERR)

     Inputs:
        IVOL      I*2     volume no. of bit map disk (1 rel)
        IBMLUN    I*2     bit map logical unit number.
        NCOPY     I*2     Number of copies of the plot to make.
        FILNAM    R*4(6)  physical file name of bit map.
        DESTRY    L*2     destroy bit file when done?
        ISIZE     I*2     size of INBLK in words.
      In/Out:
        INBLK     I*2(*)  scratch buffer
     Outputs:
        IERR      I*2    error return code.
                           0 - good.
                          >0 - an error occurred.
```

12.9.4.2 ZENDPG - advances the line printer to avoid "burn-out" on electrostatic type printers.

```
      ZENDPG (LINE)

   Inputs:  LINE    I*2     # lines printed on page so far
```

12.9.4.3 ZTAPE - Performs standard tape manipulating functions.

```
      ZTAPE (OP, LUN, FIND, COUNT, IERR)

   Inputs:
       OP  R*4  Operation to be performed. 4 characters ASCII.
                    'ADVF' = advance file marks
                    'ADVR' = advance records
                    'BAKF' = backspace file marks.
                    'BAKR' = backspace records.
                    'DMNT' = dismount tape. Works for VMS 3.0 & later.
                    'MONT' = mount tape. Works for VMS 3.0 and later.
                    'REWI' = rewind the tape on unit LUN
                    'WEOF' = write end of file on unit LUN: writes 4
                             EOFs, positions tape after first one
                    'MEOF' = write 4 EOF marks on tape, position tape
                             before the first one
       LUN  I*2  logical unit number
       FIND I*2  FTAB pointer.  Drive number for MOUNT/DISMOUNT.
       COUNT I*2 Number of records or file marks to skip. On MOUNT
                 this value is the density.
     Outputs:
       IERR  I*2  Error return: 0 => ok
                         1 = File not open
                         2 = Input specification error.
                         3 = I/O error.
                         4 = End Of File
                         5 = Beginning Of Medium
                         6 = End Of Medium
```

12.9.4.4 ZTKBUF - puts the low order byte of IN into the proper byte of the TEKTRONIX output buffer (DRBUFF). The "Z" is to allow other conversions as required locally.

```
      ZTKBUF (IN, IT, FIND, IERR)

   Input:  IN      I*2  the low order byte of this word is put into
                        DRBUFF.
           IT      I*2 Type of data: 1 control, 2 position, 3 char
           FIND    I*2 FTAB position of TEK 4012 data.
   Output: IERR    I*2 error code. 0=ok, 1=write error.
   COMMON: TKPOS   Byte position in TKBUFF to place IN.
           TKBUFF  TEKTRONIX output buffer.
```

12.9.4.5 ZTTYIO - performs I/O to a terminal.

```
      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(*)  I/O buffer
   Output: IERR    I*2     Error code: 0 => ok
                                       1 => file not open
                                       2 => input parameter error
                                       3 => I/O error
                                       4 => end of file
```

12.9.4.6 ZTVMC - issues a "master clear" to the TV. This resets the TV I/O system (if necessary) to expect a command record next.

```
      ZTVMC

       No arguments
```

12.9.4.7 ZPRMPT - prompts user on CRT screen and reads a line. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZPRMPT (IPC,BUFF,IERR)

   INPUT:    IPC     I*2     prompt character.
   OUTPUT:   BUFF    I*2(40) line of user input.
             IERR    I*2     error code:   0 => ok.
                                           1 => read/write error.
```

12.9.5 Directory And Text File

12.9.5.1 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.
```

12.9.5.2 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 (NOT USED ON VAX).
            PNAME      R*4(6)  disk-file type.  Only type ('HE' ect)
                               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.
```

12.9.5.3 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.
```

12.9.5.4 ZTXMAT - opens the directory for a source file area and returns a list of member names whose first NCH characters match the first NCH characters of MNAME.

```
      ZTXMAT ( IVOL, PNAME, MNAME, NCH, VERSON, NAMES,
     *   NNAM, IERR)

   Inputs: IVOL     I*2       Volume number (not used in VAX version).
           PNAME    R*4(6)    File name: 24 packed chars
           MNAME    I*2(4)    Test member name
           NCH      I*2       Number of characters to compare
           VERSON   R*4(5)    Tells which dir to get names from.
```

```
Output: NAMES   I*2(4,64) Names which match NCH chars of MNAME
        NNAM    I*2       Number of names in NAMES
        IERR    I*2       Error code: 0 => ok
                          1 => none
                          2 => error in inputs or Open
                          3 => I/O error
```

12.9.5.5 ZGTDIR - gets alphabetized list of members of text files.

```
ZGTDIR (ITYPE, LNAME, HNAME, VERSON, NUM, NAMES, IERR)

Inputs:  ITYPE   I*2     type of file (HE, SO, etc).
         LNAME   I*2(4)  lowest name to include.
         HNAME   I*2(4)  include names lower than this one.
         VERSON  R*4(5)  Version. Set in AIPS as the adverb VERSION.
Output:  NUM     I*2      number of names found.
         NAMES   I*2(4,1000)  sorted file names.
         IERR    I*2      error code.
```

12.9.6 Miscellaneous

12.9.6.1 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 hardcoded parameter values and then resets some based on values on an alterable disk file.

```
ZDCHIN (NDEV, NDISK, NMAP, IOBLK)

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

12.9.6.2 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
```

12.9.6.3 ZKDUMP - dumps portions of an array in INTEGER*2, char*4, hex*2, and REAL*4: i.e. in as many forms as possible ZKDUMP is called a Z routine because the formats may not be acceptable on all machines. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
ZKDUMP (I1, I2, K, C)
```

Inputs:	I1	I*2	start subscript in integer array
	I2	I*2	end subscript in integer array
	K	I*2(*)	integer array
	C	R*4(*)	real array equivalenced to K

CHAPTER 13

FITS TAPES

13.1 OVERVIEW

The principle route for getting data and images into and out of AIPS is by FITS (Flexible Image Transport System) format tape files. FITS is an internationally adopted medium of exchange of astronomical data and allows easy interchange of data between observatories and image processing systems. FITS also has the advantage that it is a self-defining format and the actual bit pattern on the tape is independent of the machine on which the tape was written. The purpose of this chapter is to describe the general features of FITS and the details of the AIPS implementation of FITS. This chapter is not intended to be a rigorous description of the FITS standards.

The fundamental definition of the FITS system is given in Wells, Greisen, and Harten (1981), with an extension described in Greisen and Harten (1981). A proposed further extension is given in Harten, Grosbol, Tritton, Greisen and Wells, (1984). FITS has been adopted as the recommended medium of exchange of astronomical data by the IAU, the Working Group on Astronomical Software (WGAS) of the AAS, and WGLAS.

Because of the great flexibility of the FITS system, many of its features have been adopted for the internal data storage format in AIPS. See the chapter on the catalogue header for more details on the AIPS internal storage format.

There are three main portions of a FITS file 1) the main header, 2) the main data and 3) any number of records containing auxilary information. In addition, an extension of the original definition of the FITS structure allows storage of ungridded visibility data. Each of these is discussed in detail in the following sections.

13.2 PHILOSPHY

FITS is a philosophy as much as a data format. The underlying philosophy is to provide a standardized, simple, and flexible means to transport data between computers or image processing systems. FITS is standarized in the sense that any FITS reader should be able to read any FITS image, at least to the degree that the array read is of the correct dimension and pixel values have at least the correct relative

scaling. In addition, any FITS reader should be able to cope with any FITS format tape and at least skip over portions or ignore keywords that it doesn't understand.

The requirement of simplicity means that the implementation of FITS reading and writing be fairly straight forward on any computer used for astronomical image processing. Simple also implies that the structure of the file be self defining and to a large degree self documenting.

The main advantage of FITS is its flexibility. Due to the self defining nature of the files, a large range of data transport needs are fulfilled. The introduction of new keywords gives the ability to add new pieces of information as needed and the use of generalized extension files allows almost unlimited flexibility in the type of information to be stored. Thus FITS can grow with the needs of the Astronomical community.

The great flexibility of FITS is a potential weakness as well as a strength. There is a great temptation to proliferate keywords and new extension file types. This should be done with great caution. Since FITS is a worldwide medium of data exchange, there needs to be coordination of keywords and extension files to prevent duplication and inconsistencies in usage.

The most fundamental philosophical ideal of FITS is that no change in the system should render old tapes illegal or unreadable. This philosophy is reflected in the AIPS implementation of FITS in that all obselete implementations (e.g. old CLEAN component or antenna extension files) are trapped and processed in the most accurate manner possible.

13.3 IMAGE FILES

The most common form of astronomical information is the image and historically the first FITS tape files were for multidimensional images. The following sections describe FITS image files.

13.3.1 Overall Structure

The structure of a FITS image file consists of one or more records containing ASCII header information followed by one or more binary data records. (These may be followed by other records which are discussed in another section.)

All "logical" records on FITS tapes are 2880 8-bit bytes long with one record per tape block. (Larger blocking factors are being considered but have not yet been implemented.) The number of bits in a FITS record is an even multiple of words and bytes on any computer ever sold commercially. The definition of FITS allows standard ANSI labeled tapes but the AIPS implementation only writes unlabeled tapes. Labeled tapes may be read by AIPS by skipping header and trailer

records.

Each FITS header record contains 36 80-byte "card images" written in 7-bit ASCII (sign bit set to zero). These header records contain all the information necessary to read, and hopefully, label the image. In addition, other information including the processing history may be given.

Following the header records come the data records. These records contain the pixel values in one of several binary formats.

13.3.2 Header Records

Each "card image" in the header is in the form,

keyword = value / comment

Keywords should be no more than 8 characters long and the keyword = value should be readable by Fortran 77 list directed I/O. To accomodate more primitive systems, a fixed format is mandatory for the required keywords and suggested for the optional keywords. This fixed format is as follows:

- Keyword name beginning in column 1.
- "=" in column 9
- T or F (logical true or false) in column 30.
- Real part (integer or floating) right justified, ending in column 30.
- Imaginary part (integer or floating) right justified, ending in column 50.
- character string with a beginning "'" in column 11 and an ending "'" in or after column 20

The first keyword in a header must be SIMPLE and have a value of T (true) if the file conforms to FITS standards and an F (false) if it doesn't. (The ASCII string "SIMPLE = T" occupying the first 30 bytes of a file of 2880-byte records is the "signature" of FITS). The keywords and values must convey the size of the image and the number of bits per pixel value. Optionally, the coordinate system, scaling and other information may be given. In the AIPS implementation a considerable amount of information is given.

13.3.2.1 Keywords - The following keywords (data type) are required for ALL FITS files (for all time) in the order given.

1. SIMPLE (logical) says if the file conforms to FITS standards.

2. BITPIX (integer) is the number of bits used to represent the pixel value; 8 => 8 bit unsigned integers, 16 => 16 bit, twos complement signed integers, 32 => 32 bit, twos complement signed integers.

3. NAXIS (integer) is the number of axes in the array.

4. NAXIS1 (integer) is the number of pixels on the fastest varying axis.

5. up to NAXIS999 (integer) is the number of pixels on the 999 th fastest varying axis.

6. END , the last keyword must be END. The last header record should be blank filled past the END keyword.

AIPS routines can accept up to 7 dimensional images.

The following optional keywords were suggested by Wells et. al. (1981). Their order (between the required keywords and the END keyword) is arbitrary; in general, all of these keywords appear in an AIPS FITS header.

- BSCALE (floating) is the scale factor used to convert tape pixel values to true values (true = [tape BSCALE] + BZERO).

- BZERO (floating) is the offset applied to true pixel values (see BSCALE).

- BUNIT (character) gives the brightness units.

- BLANK (integer) is the tape pixel value assigned to undefined pixels.

- OBJECT (character) is the image name.

- DATE (character) is the date the file was written ('dd/mm/yy')

- DATE-OBS (character) is the date of data acquition ('dd/mm/yy').

- ORIGIN (character)is the tape writing institution.

- INSTRUME (character) is the data acquisition instrument.

- TELESCOP (character) is the data acquisition telescope.

- OBSERVER (character) is the observer name / identification.

- blank in col 1-8 (none) means columns 9 - 80 are a comment.
- COMMENT (none) means columns 9 - 80 are a comment.
- HISTORY (none) means columns 9 - 80 are a comment.
- CRVALn (floating) is the value of physical coordinate on axis n at the reference pixel.
- CRPIXn (floating) is the array location of reference pixel along axis n. CRPIX may be a fractional pixel and/or be outside of the limits of the array.
- CDELTn (floating) is the increment in physical coordinate along axis n as FORTRAN counter increases by 1.
- CTYPEn (character) is the type of physical coordinate on axis n.
- CROTAn (floating) is the rotation angle of actual axis n from stated coordinate type.
- DATAMAX (floating) is the maximum data value in file (after scaling).
- DATAMIN (floating) is the minimum data value in file.
- EPOCH (floating) is the epoch of coordinate system (years).

Of these keywords, all are well defined except the rotation; see the chapter on the catalogue header for more details on the current AIPS rotation conventions. AIPS routines can currently read up to 32768 header records each consisting of 36 card images.

13.3.2.2 History - In the AIPS implementation, the "HISTORY" cards contain the entries of the history file associated with the image. As they appear on the tape, these history entries are in the form:

HISTORY tsknam keyword1=value1, keyword2=value2 ... / comment

Where "tsknam" is the name of the task (or AIPS) making the entry and the keywords are the AIPS adverbs used. Thus these history records may be used to carry AIPS specific values which don't have official keywords. This feature is used, for example, to determine the default file name, class etc. when reading a file which was written on an AIPS system.

13.3.2.3 AIPS Nonstandard Image File Keywords - There are a number of keywords used by AIPS which are not standard.

- TABLES (integer) is the number of tables following the file. (now obsolete)
- DATE-MAP (character) is the date the map was made. ('dd/mm/yy')
- OBSRA (floating) is the Right ascension of the antenna and delay tracking position used for the observations.
- OBSDEC (floating) is the declination of the antenna and delay tracking position used for the observations.
- VELREF (floating) is the reference velocity.
- ALTRVAL (floating) is the value of the alternate (frequency/velocity) axis at the alternate reference pixel (ALTPIX).
- ALTRPIX (floating) is the alternate (frequency/velocity) reference pixel.
- RESTFREQ (floating) is the rest frequency of the spectral line being observed.
- XSHIFT (floating) is the offset of the phase center from the tangent point of the Right ascension after any rotation.
- YSHIFT(floating) is the offset of the phase center from the tangent point of the declination after any rotation.

A number of keywords which are specific to AIPS are hidden on HISTORY cards. These keywords are recognized if the first symbol in columns 10 - 17 is one of the following: 'AIPS', 'VLACV', or 'RANCID'.

- IMNAME (character) the name of the file in an AIPS (or RANCID) system used to generate the FITS tape.
- IMCLASS (character) the class of the AIPS file.
- IMSEQ (integer) the sequence number of the AIPS file.
- USERNO (integer) the AIPS user number.
- PRODUCT (integer) the type of CLEAN image. 1=>normal clean, 2=>components, 3=>residual, 4=>points.
- NITER (integer) the number of CLEAN components used for the image.

- BMAJ (floating) the major axis (FWHP) of the restoring beam. (degrees)

- BMIN (floating) the minor axis (FWHP) of the restoring beam.

- BPA (floating) the position angle (from north thru east) of the major axis of the restoring beam.

AIPS also recognizes, but does not write, the following non-standard keywords:

- OPHRAE11 (floating) an obscure number related to the Right ascension of the center on an image made on the VLA pipeline PDP11.

- OPHDCE11(floating) an obscure number related to the declination of the center on an image made on the VLA pipeline PDP11.

- MAPNAM11 (character) the name of the file on the VLA pipeline PDP11.

Any keywords which are not recognized by AIPS are written into the history file.

13.3.2.4 Coordinate Systems - The coordinate type and the system used for each type is given by the CTYPEn values. The character strings used for these values are identical to the strings used in the AIPS catalogue header record (CAT4(K4CTP+n-1)). The coordinate type is encoded into the first 4 characters of the coordinate type string (e.g. 'RA--' indicating Right ascension) and the system used is encoded into characters 5 - 8 (e.g. '-SIN' indicating a sine projection onto the sky). The coordinate systems and their symbolic names are described in detail in the chapter on the catalogue header and AIPS memo number 27. The coordinate system used to describe the polarization of an image needs careful attention.

The AIPS convention for projected geometries is to specify the tangent point of the projection as the reference pixel even though this need not correspond to an integer pixel and need not even be contained in the array given. The tangent point is the position on the sky where the plane on which the image is projected is tangent to the celestial sphere. For images derived from synthesis arrays, this is the position for which u, v, and w were computed. The reference pixel for a synthesis array beam image is the phase reference of the image; this should be the position of the peak of the beam (pixel value = 1.0).

The use of one rotation angle per axis cannot be used to define a general rotation of the axis system. Since the AIPS catalogue header uses the same convention, the same problems occur internally to AIPS. See the chapter on the AIPS catalogue header for a brief discussion of the conventions used in AIPS. The same conventions are used when

reading and writing FITS tapes.

13.3.2.5 Example Image Header - The following is an example of an image header written by AIPS (with most of the HISTORY entries removed).

```
0000000001111111111222222222233333333334444444444555555555566666666661
1234567890123456789012345678901234567890123456789012345678901234567890
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    4 /
NAXIS1  =                 2048 /
NAXIS2  =                 1024 /
NAXIS3  =                    1 /
NAXIS4  =                    1 /
OBJECT  = '3C405   '           / SOURCE NAME
TELESCOP= '        '           /
INSTRUME= '        '           /
OBSERVER= 'PERL    '           /
DATE-OBS= '27/10/82'           /OBSERVATION START DATE DD/MM/YY
DATE-MAP= '14/07/83'           /DATE OF LAST PROCESSING DD/MM/YY
BSCALE  =    7.04625720812E-05 /REAL = TAPE * BSCALE + BZERO
BZERO   =    2.18688869476E+00 /
BUNIT   = 'JY/BEAM '           /UNITS OF FLUX
EPOCH   =     1.950000000E+03 /EPOCH OF RA DEC
DATAMAX =     4.495524406E+00 /MAX PIXEL VALUE
DATAMIN =    -1.217470840E-01 /MIN PIXEL VALUE
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.99435165226E+02 /
CDELT1  =    -4.166666986E-05 /
CRPIX1  =     1.024000000E+03 /
CROTA1  =     0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.05961940065E+01 /
CDELT2  =     4.166666986E-05 /
CRPIX2  =     5.130000000E+03 /
CROTA2  =     0.000000000E+00 /
CTYPE3  = 'FREQ    '           /
CRVAL3  =    4.86635000000E+09 /
CDELT3  =     1.250000000E+07 /
CRPIX3  =     1.000000000E+00 /
CROTA3  =     0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =     1.000000000E+00 /
CRPIX4  =     1.000000000E+00 /
CROTA4  =     0.000000000E+00 /
HISTORY UVLOD /DATA BASE CREATED BY USER 76 AT 14-JUL-1983 10:17:08
HISTORY UVLOD OUTNAME='CYGA        ' OUTCLASS='XY    '
HISTORY UVLOD OUTSEQ=    1   OUTDISK=  3
        ...
ORIGIN  = 'AIPSNRAO VLA VAX3       '   /
DATE    = '19/08/83'                   / TAPE WRITTEN ON DD/MM/YY
HISTORY AIPS   IMNAME='CYGA        ' IMCLASS='IMAP  ' IMSEQ=   1   /
HISTORY AIPS   USERNO=   76           /
END
```

13.3.2.6 Units - The units for pixel values and coordinate systems should be SI units where appropriate (e.g. velocities in meters/sec); angles in degrees; pixel values in Jy, Jy/beam, magnitudes, or magnitudes/pixel.

13.3.3 Data Records

The data array starts at the beginning of the record following the last header record. The data occurs in the order defined by the header; in increasing pixel number with axis 1 the fastest varying and the last axis defined the slowest varying. Data is packed into the 2880 byte records with no gaps; that is, the first pixel of any given axis does not necessarily appear in the first word of a new record.

The bits in each word are in order of decreasing signifigance with the sign bit first. This convention means the PDP-11 and VAX machines will have to reverse the order of the bytes in 16 and 32 bit words before writing or after reading the tape. There are a number of AIPS utility routines for converting FITS tape data to the local convention; these are briefly described in the following list. Complete details of the call sequences etc. are given at the end of the chapter on the Z routines.

1. ZCLC8 converts local characters to standard 8-bit ASCII.

2. ZC8CL extracts 8-bit standard characters from a buffer and stores them in the local character form.

3. ZI16IL extracts 16-bit twos complement integers from a buffer and puts them in a local small integer array.

4. ZI32IL extracts 32-bit twos complement integers from a buffer and puts them in a local array of pseudo I*4 integers.

5. ZI8L8 converts 8-bit unsigned binary numbers to "bytes" (half of a local small integer).

6. ZILI16 converts a buffer of local small integers to a buffer of standard 16-bit, twos complement integers. ZR8P4 converts between pseudo I*4 and double precision (R*8).

13.4 RANDOM GROUP (UV DATA) FILES

The extension of the original FITS standards described by Greisen and Harten 1981 allows uv data to be written in FITS files. These files are called "Random group" FITS files. This extension is to allow multiple "images" i.e. rectangular data arrays each of which is arbitrarily located on some "axes". Thus each data array is preceeded by a number of "random" parameters which describe its location on axes on which it is not regularly gridded, e.g, u, v, w, time, and

baseline. The definition of what constitutes an "axis" is extremely vague. Currently AIPS FITS routines can accept up to 7 actual axes in the regular portion of a group and up to 20 random parameter words. The structure of a group is shown in the following.

| r1, r2, r3, ... rk | p11, p12, ... pmn |

where r1 ... rk are random parameters 1 thru k
p11 ... pmn are the pixel value in the order defined for image arrays. Two dimensions are used only for demonstration.

FITS image files are actually a subset of this more general structure but for historical reasons the random group FITS is treated as a special case of the image file. This has unfortunate consequences as will shortly become obvious. Most of the features of random group files are identical to image files and the discussion in the following section will concern the differences between image and random group FITS files.

13.4.1 Header Record

For obscure historical reasons, random group FITS files are declared to have zero pixels on the first axis; the first real axis is labeled axis 2 and so on. This will allow FITS image readers that don't know about random group files to do something reasonable, i.e. skip over the file. Thus a random group FITS file has one more axis described in the header than actually occurs in the data.

In addition to playing games with the axis numbers, random group FITS headers have the following required keywords (in any order):

1. GROUPS (logical) is true (T) if the data file is a random group FITS file.

2. PCOUNT (integer) is the number of random parameters preceding each data array.

3. GCOUNT (floating) is the number of groups in the file.

The random parameters may be labeled and scaled in a fashion similar to image axes and pixels. In addition, multiple word precision in some of the random parameters is allowed by giving multiple random parameters the same label. If several random parameters have the same name (PTYPE), their values should be summed after scaling. Labeling and scaling use the following optional keywords (arbitrary order):

- PTYPEn (character) is the label for the n-th random parameter. If several random parameters have the same value of PTYPEn they should be summed after scaling.

- PSCALn (floating) gives the scale factor for random parameter n. True_value = tape_value * PSCALn + PZEROn

- PZEROn (floating) gives the scaling offset for random parameter n.

A number of keywords which are specific to AIPS are hidden on HISTORY cards. These keywords are recognized if the first symbol in columns 10 - 17 is one of the following: 'AIPS', 'VLACV', or 'RANCID'.

- SORT ORDER (character) the order of the groups.

- WTSCAL (floating) an additional scaling factor for visibility weights.

13.4.2 Data Records

The binary data records are stored beginning in the first record following the last header record in much the same way that image files are stored; the beginning of a group does not necessarily correspond to the beginning of a record. The same pixel data types are allowed as for image files (note: the data type must be the same for all values both random parameters and the "data" array).

13.4.2.1 Weights And Flagging - Uv FITS files written by AIPS have as their first (real, i.e. second in the header) axis the 'COMPLEX' axis which is dimensioned 3. The values along this axis (coordinate values 1, 2, and 3) are real part (in Jy), imaginary part, and weight. A non positive weight indicates that the the visibility has been flagged. The scaling desired for the weight may be different for the real and imaginary parts so an additional scaling factor is stored in the header as a HISTORY entry as follows:

```
HISTORY AIPS WTSCAL = 2.76756756757E+01
                                / CMPLX WTS=WTSCAL*(TAPE*BSCALE+BZERO)
```

The use of WTSCAL allows the reader to recover the same values for the weights as the AIPS file which was used to generate the FITS file. If WTSCAL is ignored (or absent) the relative but not absolute scaling of the weights is preserved.

In addition to the form described above, AIPS will accept other forms of weighting/flagging data.

1. Magic value blanking. In this case the COMPLEX axis is dimensioned 2 (real and imaginary) and the header keyword BLANK is used to indicate undefined data values. Thus if either the real or imaginary parts are 'blanked' the data is assumed to be flagged (invalid).

2. Random parameter flagging. Data written on the VLA pipeline is in this format. The weights and flags are passed as random parameters. More on this later in the broadcast.

13.4.2.2 Antennas And Subarrays - If data from different arrays (or different VLA configurations) are combined, the physical identity of a given antenna may not be constant in a given data base. In order to identify the physical antennas involved in a given visibility record, AIPS uses a subarray number. The (subarray number - 1) * 0.01 is added to the baseline number to identify the subarray.

There is an antenna file or list for each subarray. The information about the antennas (e.g. locations etc.) is given in the antenna files. Currently AIPS writes these files as extension table files (described later) with the file version number corresponding to the subarray number.

AIPS will also recognize antenna locations given in the HISTORY cards. An example (from Greisen and Harten 1981) of this follows:

```
COMMENT  ANTENNA LOCATIONS IN NANOSECONDS:
HISTORY VLACV ANT  N= 2 X=  5470.525 Y=-14443.276 Z= -8061.210 ST='AW4'
HISTORY VLACV ANT  N= 4 X=  1667.280 Y= -4396.334 Z= -2452.399 ST='CW8'
HISTORY VLACV ANT  N= 5 X=    37.719 Y=   135.627 Z=   -50.585 ST='DE2'
HISTORY VLACV ANT  N= 6 X=  3353.710 Y= -8816.123 Z= -4910.700 ST='BW6'
HISTORY VLACV ANT  N= 7 X=   118.761 Y=   445.786 Z=  -170.397 ST='DE4'
HISTORY VLACV ANT  N= 9 X= 10924.708 Y=-28961.684 Z=-16194.042 ST='AW6'

COMMENT  FORMULA FOR BASELINES BETWEEN ANTENNA I AND J (I<J):
COMMENT  BASELINE(IJ) = LOCATION(I) - LOCATION(J)

COMMENT  FORMULA FOR UU, VV, WW :
COMMENT  UU = BX * SIN(HA) + BY * COS(HA)
COMMENT  VV = BZ * COS(DEC) + SIN(DEC) * (BY * SIN(HA) - BX * COS(HA))
COMMENT  WW = BZ * SIN(DEC) + COS(DEC) * (BX * COS(HA) - BY * SIN(HA))
           WHERE UU AND VV ARE THEN ROTATED TO THE EPOCH
```

The above example also defines the antenna geometry and u, v, and w terms used for VLA data.

13.4.2.3 Coordinates - The coordinate systems used to write FITS uv data tapes are very similar to the AIPS internal systems; the major difference being the use of 'DATE' (giving the Julian date) for time tagging the data rather than 'TIME1' (giving the time in days from the beginning of the experiment). See the uv data section of the disk I/O chapter for more details of the AIPS internal uv data coordinate

systems.

13.4.2.4 Sort Order - The ordering of visibility records is variable and may be changed by programs such as AIPS task UVSRT. The sort order is given as a two character code in the FITS header as in the following example:

```
HISTORY AIPS  SORT ORDER = 'XY'
```

Data sorted in AIPS has a two key sort order with the first key varying the slowest. The two keys are coded as characters given by the following table:

```
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
```

13.4.3 Typical VLA Record Structure

The following is a uv FITS header for continuum VLA data which demonstrates the use of multiple precision random parameters. Most of the HISTORY records are removed from this example. The header indicates that the data in this example is followed by two antenna files in the old AIPS tables format.

```
000000000111111111122222222223333333333444444444455555555556666666666
123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    6 /
NAXIS1  =                    0 /NO STANDARD IMAGE JUST GROUP
NAXIS2  =                    3 /
NAXIS3  =                    4 /
NAXIS4  =                    1 /
NAXIS5  =                    1 /
NAXIS6  =                    1 /
OBJECT  = '0923+350'           / SOURCE NAME
TELESCOP= '        '           /
INSTRUME= '        '           /
OBSERVER= 'COTT    '           /
DATE-OBS= '30/04/82'           /OBSERVATION START DATE DD/MM/YY
```

```
DATE-MAP= '11/10/83'           /DATE OF LAST PROCESSING DD/MM/YY
BSCALE  =    3.30987595420E-06 /REAL = TAPE * BSCALE + BZERO
BZERO   =    0.00000000000E+00 /
BUNIT   = 'JY      '           /UNITS OF FLUX
EPOCH   =      1.950000000E+03 /EPOCH OF RA DEC
OBSRA   =    1.40795415491E+02 /ANTENNA POINTING RA
OBSDEC  =    3.50133331865E+01 /ANTENNA POINTING DEC
TABLES  =                    2 /THIS IS THE ANTENNA FILE
CTYPE2  = 'COMPLEX '           /
CRVAL2  =    1.00000000000E+00 /
CDELT2  =      1.000000000E+00 /
CRPIX2  =      1.000000000E+00 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'STOKES  '           /
CRVAL3  =   -1.00000000000E+00 / STOKES AS RR, LL, RL, LR
CDELT3  =     -1.000000000E+00 /
CRPIX3  =      1.000000000E+00 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'FREQ    '           /
CRVAL4  =    4.88510000000E+09 /
CDELT4  =      5.000000000E+07 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
CTYPE5  = 'RA      '           /
CRVAL5  =    1.40795415491E+02 /
CDELT5  =      0.000000000E+00 /
CRPIX5  =      1.000000000E+00 /
CROTA5  =      0.000000000E+00 /
CTYPE6  = 'DEC     '           /
CRVAL6  =    3.50133331865E+01 /
CDELT6  =      0.000000000E+00 /
CRPIX6  =      1.000000000E+00 /
CROTA6  =      0.000000000E+00 /
GROUPS  =                    T /
GCOUNT  =                21389. /
PCOUNT  =                    7 /
PTYPE1  = 'UU-L    '           /
PSCAL1  =    2.56659543954E-09 /
PZERO1  =    0.00000000000E+00 /
PTYPE2  = 'VV-L    '           /
PSCAL2  =    3.46332811989E-09 /
PZERO2  =    0.00000000000E+00 /
PTYPE3  = 'WW-L    '           /
PSCAL3  =    2.33722136998E-09 /
PZERO3  =    0.00000000000E+00 /
PTYPE4  = 'BASELINE'           /
PSCAL4  =    1.00000000000E+00 /
PZERO4  =    0.00000000000E+00 /
PTYPE5  = 'BASELINE'           /
PSCAL5  =    1.00000000000E-02 /
PZERO5  =    0.00000000000E+00 /
PTYPE5  = 'DATE    '           /
PSCAL5  =    2.50000000000E-01 /
PZERO5  =    2.44508950000E+06 /
PTYPE7  = 'DATE    '           /
PSCAL7  =    1.52587890600E-05 /
```

```
PZERO7  =    0.00000000000E+00 /
         / WHERE BASELINE = 256*ANT1 + ANT2 + (ARRAY#-1)/100
HISTORY UVLOD RELEASE='15NOV83 ' /CREATED AT 11-OCT-1983 13:34:50
HISTORY UVLOD OUTNAME='0923+350  ' OUTCLASS='UVDATA'
HISTORY UVLOD OUTSEQ=    1   OUTDISK=  3
        ...
ORIGIN  = 'AIPSNRAO node CVAX   15NOV83'  /
DATE    = '11/10/83'                      / TAPE WRITTEN ON DD/MM/YY
HISTORY AIPS   IMNAME='0923+350   ' IMCLASS='XYAC  ' IMSEQ=   1   /
HISTORY AIPS   USERNO=   413           /
HISTORY AIPS   SORT ORDER = 'XY'
              / WHERE X MEANS DESC ABS(U)
              / WHERE Y MEANS DESC ABS(V)
HISTORY AIPS WTSCAL = 2.76756756757E+01 / CMPLX WTS=WTSCAL*(TAPE*BSCALE+BZER(
END
```

13.5 EXTENSION FILES

There is frequently auxilary information associated with an image or data set which needs to be saved in the same tape file. Examples of this in AIPS are the Antenna files and CLEAN component files. There is currently a draft proposal to the IAU (Harten et. al. 1984) defining a standard format for the invention of extension files to be written after the main data records (if any) and defining a "Tables" type extension file. The Tables extension files will be able to carry information which can be expressed in the form of a table. The following section will describe the proposed standards which are being incorporated into AIPS.

13.5.1 Standard Extension

The standard, generalized extension file is not a true tape file in the sense that it is separated by tape EOF marks, but is a number of records inside a FITS tape file which contains information of relevence to the file. Each standard extension "file" will have a header which is very similar to the main FITS header. This header consists of one or more 2880 8-bit byte "logical" records each containing 36 80-byte "card images" in the form:

keyword = value / comment

The extension file header begins in the first record following the last record of main data (if any) or the last record of the previous extension file. The format of the generalized extension "file" header is such that a given FITS reader can decide if it wants (or understands) a given extension file type and can skip over the extension file if the reader decides it doesn't.

Most of the standards concerning data types and bit orders for the main FITS data records also apply to extension files. One difference is that 8-bit pixel values can be used to indicate ASCII code.

The use of the generalized extension "files" requires the use of a single additional keyword in the main header:

- EXTEND (logical) if true (T) indicates that there may be extension files following the data records and if there are, that they conform to the generalized extension file header standards.

The required keywords in an extension file header record are, in order:

1. XTENSION (character) indicates the type of extension file, this must be the first keyword in the header.

2. BITPIX (integer) gives the number of bits per "pixel" value. The types defined for the main data records plus 8-bit ASCII are allowed.

3. NAXIS (integer) gives the number of "axes"; a value of zero is allowed which indicates that no data records follow the header.

4. NAXIS1 (integer) is the number of "pixels" along the first axis (if any).

5. NAXISn (integer) is the number of "pixels" along the last axis.

6. PCOUNT (integer) is the number of "random" parameters before each group. This is similar to the definition of random group main data records. The value may be zero.

7. GCOUNT (integer) is the number of groups of data defined as for the random group main data records. If an image-like file (e.g. a table file) is being written this will be 1.

8. END is always the last keyword in a header. The remainder of the record following the END keyword is blank filled.

There are three optional standard keywords for extension file header records. The order, between the required keywords and the END keyword, is arbitrary.

- EXTNAME (character) can be used to give a name to the extension file to distinguish it from other similar files. The name may have a hierarchical structure giving its relation to other files (e.g. "mapl.cleancomp")

- EXTVER (integer) is a version number which can be used with EXTNAME to identify a file.

- EXTLEVEL (integer) specifies the level of the extension file in a hierarchical structure. The default value for EXTLEVEL should be 1.

The number of bits in an extension file (excluding the header) should be given by the formula:

NBITS = BITPIX * GCOUNT * (PCOUNT + NAXIS1 * NAXIS2 * ... * NAXISn)

The number of data records following the header record are then given by:

NRECORDS = INT ((NBITS + 23039) / 2304O)

It is important that the above formulas accurately predict the number of data records in an extension "file" so that readers can skip over these "files". The data begins in the first record following the last record of the header.

Extreme caution must be exercized when inventing new types of extension files. In particular, duplication of types or several types with the same function must be avoided. This means that when a new extension file type is invented, it should be as general as possible so that it may be used for other similar problems.

13.5.2 Tables Extension

A very common type of extension file is one containing data that can be expressed in the form of a table. That is, a number of entries which are all identical in form. A general, self defining table extension file type is proposed by Harten et. al. (1984). The following sections describe the proposed format.

The table extension file uses ASCII records to carry the tabular information. Each table entry will contain a fixed number of entries (although the number can vary between different extension files). For each entry is given 1) a label (optional), 2) the beginning column, 3) an undefined value (optional) , 4) a Fortran format to decode the entry, 5) scaling and offset information (optional), 6) the units (optional).

13.5.2.1 Tables Header Record - The keywords for tables extension file headers are given in the following:

- XTENSION (character) is required to be the first keyword and has a value 'TABLE ' for table extension files.

- BITPIX (integer) is a required keyword which must have a value of 8 indicating printable ASCII characters.

- NAXIS (integer) is a required keyword which must have a value of 2 for tables extension files.

- NAXIS1 (integer) is a required keyword which given the number of characters in a table entry.

- NAXIS2 (integer) is a required keyword which gives the number of entries in the table. A value of 1 is allowed.

- PCOUNT (integer) is a required keyword which must have the value of 0 for tables extension files.

- GCOUNT (integer) is a required keyword which must have the value of 1 for tables extension files.

- TFIELDS (integer) is a required keyword which must follow the GCOUNT keyword. TFIELDS gives the number of fields in each table entry.

- EXTNAME (character) is the name of the table.

- EXTVER (integer) is the version number of the table.

- EXTLEVEL (integer) is the hierarchical level number of the table, 1 is recommended. (optional)

- TBCOLnnn (integer) the pixel number of the first character in the nnn th field .

- TFORMnnn (character) the Fortran format of field nnn (I,A,E,D)

- TTYPEnnnn (character) the label for field nnn. (optional, order arbitrary)

- TUNITnnn (character) the physical units of field nnn. (optional, order arbitrary)

- TSCALnnn (floating) the scale factor for field nnn. True_value = tape_value * TSCAL + TZERO. Note: TSCALnnn and TZEROnnn are not relevant to A-format fields. Default value is 1.0 (optional, order arbitrary)

- TZEROnnn (floating) the offset for field nnn. (See TSCALnnn.) Default value is 0.0 (optional, order arbitrary)

- TNULLnnn (character) the (tape) value of an undefined value. Note: an exact left-justified match to the field width as specified by TFORMnnn is required. (optional, order arbitrary)

- AUTHOR (character) the name of the author or creator of the table. (optional, order arbitrary)
- REFERENC (character) the reference for the table. (optional, order arbitrary)
- END must always be the last keyword and the remainder of the record must be blank filled.

The TFORMnnn keywords should specify the width of the field and are of the form Iww, Aww, Eww.dd, or Dww.dd (integers, characters, single precision and double precision). If -0 is ever to be distinguished from +0 (e.g. degrees of declination) the sign field should be declared to be a separate character field.

13.5.2.2 Table Data Records - The table file data records begin with the next record following the last header record and each contains 2880 ASCII characters in the order defined by the header. Table entries do not necessarily begin at the beginning of a new record. The last record should be blank filled past the end of the valid data.

13.5.2.3 Example Table Header And Data - The first two lines of numbers are only present to show card columns and are not part of the extension file.

```
         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
XTENSION= 'TABLE   '           / EXTENSION TYPE
BITPIX  =                     8 / PRINTABLE ASCII CODES
NAXIS   =                     2 / TABLE IS A MATRIX
NAXIS1  =                    60 / WIDTH OF TABLE IN CHARACTERS
NAXIS2  =                   449 / NUMBER OF ENTRIES IN TABLE
PCOUNT  =                     0 / RANDOM PARAMETER COUNT
GCOUNT  =                     1 / GROUP COUNT
TFIELDS =                     3 / NUMBER OF FIELDS IN EACH ROW
EXTNAME = 'AIPS CC '            / AIPS CLEAN COMPONENTS
EXTVER  =                     1 / VERSION NUMBER OF TABLE
TBCOL1  =                     1 / STARTING CHAR. POS. OF FIELD N
TFORM1  = 'E15.6   '            / FORTRAN FORMAT OF FIELD N
TTYPE1  = 'FLUX    '            / TYPE (HEADING) OF FIELD N
TUNIT1  = 'JY      '            / PHYSICAL UNITS OF FIELD N
TSCAL1  =                   1.0 / SCALE FACTOR FOR FIELD N
TZERO1  =                   0.0 / ZERO POINT FOR FIELD N
TBCOL2  =                    17 / STARTING CHAR. POS. OF FIELD N
TFORM2  = 'E15.6   '            / FORTRAN FORMAT OF FIELD N
TTYPE2  = 'DELTAX  '            / TYPE (HEADING) OF FIELD N
TUNIT2  = 'DEGREES '            / PHYSICAL UNITS OF FIELD N
TSCAL2  =                   1.0 / SCALE FACTOR FOR FIELD N
```

```
TZERO2  =                    0.0 / ZERO POINT FOR FIELD N
TBCOL3  =                     33 / STARTING CHAR. POS. OF FIELD N
TFORM3  = 'E15.6   '             / FORTRAN FORMAT OF FIELD N
TTYPE3  = 'DELTAY  '             / TYPE (HEADING) OF FIELD N
TUNIT3  = 'DEGREES '             / PHYSICAL UNITS OF FIELD N
TSCAL3  =                    1.0 / SCALE FACTOR FOR FIELD N
TZERO3  =                    0.0 / ZERO POINT FOR FIELD N
END
```

The rest of the header block is blank filled. The data cards start on the next block boundary.

```
  0.183387E+00   -0.138889E-03    0.694444E-04
  0.146710E+00   -0.138889E-03    0.694444E-04
  0.117368E+00   -0.138889E-03    0.694444E-04
  0.938941E-01   -0.138889E-03    0.694444E-04
  0.183387E+00   -0.138889E-03    0.694444E-04

                      .
                      .
                      .
```

13.5.3 Older AIPS Tables

Prior to the (presumed) establishment of the standard tables extension files, AIPS had it own tables file format and a large number of tapes have been written with these tables. These old tables were encoded in ASCII and could have any number of columns in the table. However, all values in the table had to be of the same data type and written with the same format. AIPS FITS readers will continue to recognize and deal with these obsolete tables indefinitely. The following sections describe these tables.

13.5.3.1 General Form Of Header - The presence of the old format AIPS tables is indicated in the main header by the presence of the integer keyword TABLES which gives the number of tables following the data records. Each table has a header record in a manner similar to the now standard extension file header but with different keywords. The header contains the following keywords:

1. TABNAME (character) gives the name of the file.
2. TABVER (integer) gives the version number of the file.
3. TABCOUNT (integer) gives the number of entries in the table.
4. TABWIDTH (integer) gives the number of values per table entry
5. TABCARDS (integer) gives the number of values per card image.
6. TTYPEn (character) gives a label for the n th column.
7. NUMTYPE (character) gives the data type used for internal storage (I*2, R*4, R*8)
8. FORMAT (character) gives the format for the table elements.
9. END is the last keyword.

13.5.3.2 Data Records - The data records consist of floating point values encoded in ASCII in 36 80-byte card images per record in a free field format. The values are encoded TABCARDS values per 80 byte card image.

13.5.3.3 CC Files - The details of the AIPS old CLEAN component (CC) table file are illustrated in the following example of a header. Component positions are given in degrees from the tangent point (reference pixel) of the image in the projected and rotated plane (i.e. not true RA and dec). Component flux densities are in Janskys. CLEAN components are stored 2 per card image written as 6E13.5.

```
TABNAME = 'AIPS  CC'                    / AIPS CLEAN COMPONENTS
TABVER  =                    1          / VERSION NUMBER
TABCOUNT=                  100          / # LOGICAL RECORDS IN TABLE
TABWIDTH=                    3          / # VALUES PER LOGICAL RECORD
TABCARDS=                    6          / # VALUES PER CARD IMAGE
TTYPE1  = 'DELTAX  '                    / COLUMN 1 LABEL
TTYPE2  = 'DELTAY  '                    / COLUMN 2 LABEL
TTYPE3  = 'FLUX(JY)'                    / COLUMN 3 LABEL
NUMTYPE = 'R*4     '                    / OUR INTERNAL STORAGE SIZE
FORMAT  = 'E13.5   '                    / FORMAT ACTUALLY USED HERE
END
```

13.5.3.4 AN Files - The details of the AIPS old antenna table file are illustrated in the following example of a header. Antenna positions are given in seconds (light travel time)

```
TABNAME = 'AIPS  AN'                    /ANTENNA IDS, LOCATIONS
TABVER  =                    1          /VERSION NUMBER
TABCOUNT=                   28          / # LOGICAL RECORDS IN TABLE
TABWIDTH=                    5          / # VALUES PER LOGICAL RECORD
TABCARDS=                    5          / # VALUES PER CARD IMAGE
TTYPE1  = 'AN NO.  '                    / COLUMN 1 LABEL
TTYPE2  = 'STATION '                    / COLUMN 2 LABEL
TTYPE3  = 'LX      '                    / COLUMN 3 LABEL
TTYPE4  = 'LY      '                    / COLUMN 4 LABEL
TTYPE5  = 'LZ      '                    / COLUMN 5 LABEL
END
```

13.6 AIPS FITS INCLUDES

There are several AIPS INCLUDEs which contain tables of KEYWORD names, data types and pointers to the AIPS catalogue header. Each of the sets consists of a declaration include (Dnnn.inc), an EQUIVALENCE include (Ennn.inc) and a DATA include (Vnnn.inc). These includes can be used directly by routines such as FPARSE. The basic components of these includes is shown below:

- AWORD (R*4) - this array contains the recognized keywords, two R*4 words per keyword with four characters per R*4 word. This array can be sent to GETCRD as the list of keywords.

- NCT (I*2) - this gives the number of required keyword names in CWORD which is equivalences at the beginning of AWORD.

- NKT (I*2) - this given the number of optional keywords names in KWORD which is equivalenced into AWORD after CWORD.

- ATYPE (I*2) - this array gives the data types corresponding to keywords in AWORD. 1=>logical variable, 2=>numerical value, and 3=>string.

- APOINT (I*2) - this array contains pointers in the common in the includes DHDR.INC and CHDR.INC to the AIPS catalogue header in the form 1000nbytes + 100offset + position of pointer in common. Here nbytes given the number of bytes used in the AIPS catalogue header and the offset is the character offset past the position indicated by the header pointer. The text of these includes is in the following sections.

13.6.1 DFUV.INC

```
C                                                               Include DFUV
      INTEGER*2 ATYPE(150), APOINT(150), CTYPE(11), KTYPE(139),
     *   CPOINT(11), POINT(139), NKT, NCT
      REAL*4    AWORD(2,150), CWORD(2,11), KWORD(2,139), K1(2,73),
     *   K2(2,66)
C                                                               End DFUV
```

13.6.2 DFIT.INC

```
C                                                               Include DFUV
      INTEGER*2 ATYPE(82), APOINT(82), CTYPE(10), KTYPE(72), CPOINT(10),
     *   POINT(72), NKT, NCT
      REAL*4    AWORD(2,82), CWORD(2,10), KWORD(2,72)
C                                                               End DFUV
```

13.6.3 EFUV.INC

```
C                                                               Include EFUV
      EQUIVALENCE (AWORD(1,1), CWORD(1,1)),  (AWORD(1,85), K2(1,1)),
     *   (AWORD(1,12), KWORD(1,1), K1(1,1))
      EQUIVALENCE (APOINT(1), CPOINT(1)),    (APOINT(12), POINT(1))
      EQUIVALENCE (ATYPE(1), CTYPE(1)),      (ATYPE(12), KTYPE(1))
C                                                               End EFUV
```

13.6.4 EFIT.INC

```
C                                                               Include EFUV
      EQUIVALENCE (AWORD(1,1), CWORD(1,1)), (AWORD(1,11), KWORD(1,1))
      EQUIVALENCE (APOINT(1), CPOINT(1)),    (APOINT(11), POINT(1))
      EQUIVALENCE (ATYPE(1), CTYPE(1)),      (ATYPE(11), KTYPE(1))
C                                                               End EFUV
```

13.6.5 VFUV.INC

```
C                                                               Include VFUV
      DATA NCT/11/,     NKT/139/
      DATA CWORD/'SIMP','LE  ','BITP','IX  ','NAXI','S   ','NAXI',
     *  'S1  ','NAXI','S2  ','NAXI','S3  ','NAXI','S4  ','NAXI','S5  ',
     *         'NAXI','S6  ','NAXI','S7  ','NAXI','S8  '/
      DATA K1 /'OBJE','CT  ','TELE','SCOP','INST','RUME','OBSE','RVER',
     *         'DATE','-OBS','DATE','-MAP','BSCA','LE  ','BZER','0   ',
     *         'BUNI','T   ','CTYP','E1  ','CTYP','E2  ','CTYP','E3  ',
     *         'CTYP','E4  ','CTYP','E5  ','CTYP','E6  ','CTYP','E7  ',
     *         'CTYP','E8  ','CRVA','L1  ','CRVA','L2  ','CRVA','L3  ',
     *         'CRVA','L4  ','CRVA','L5  ','CRVA','L6  ','CRVA','L7  ',
     *         'CRVA','L8  ','CDEL','T1  ','CDEL','T2  ','CDEL','T3  ',
     *         'CDEL','T4  ','CDEL','T5  ','CDEL','T6  ','CDEL','T7  ',
     *         'CDEL','T8  ','CRPI','X1  ','CRPI','X2  ','CRPI','X3  ',
     *         'CRPI','X4  ','CRPI','X5  ','CRPI','X6  ','CRPI','X7  ',
     *         'CRPI','X8  ','CROT','A1  ','CROT','A2  ','CROT','A3  ',
     *         'CROT','A4  ','CROT','A5  ','CROT','A6  ','CROT','A7  ',
     *         'CROT','A8  ','EPOC','H   ','DATA','MAX ','DATA','MIN ',
     *         'BLAN','K   ','INHI','BIT ','IMNA','ME  ','IMCL','ASS ',
     *         'IMSE','Q   ','USER','NO  ','PROD','UCT ','NITE','R   ',
     *         'BMAJ','    ','BMIN','    ','BPA ','    ','VELR','EF  ',
     *         'ALTR','VAL ','ALTR','PIX ','OBSR','A   ','OBSD','EC  ',
     *         'REST','FREQ','XSHI','FT  ','YSHI','FT  ','DATE','    ',
     *         'ORIG','IN  '/
      DATA K2 /'GROU','PS  ','GCOU','NT  ','PCOU','NT  ','PTYP','E1  ',
     *         'PTYP','E2  ','PTYP','E3  ','PTYP','E4  ','PTYP','E5  ',
     *         'PTYP','E6  ','PTYP','E7  ','PTYP','E8  ','PTYP','E9  ',
     *         'PTYP','E10 ','PTYP','E11 ','PTYP','E12 ','PTYP','E13 ',
     *         'PTYP','E14 ','PTYP','E15 ','PTYP','E16 ','PTYP','E17 ',
     *         'PTYP','E18 ','PTYP','E19 ','PTYP','E20 ','PSCA','L1  ',
     *         'PSCA','L2  ','PSCA','L3  ','PSCA','L4  ','PSCA','L5  ',
     *         'PSCA','L6  ','PSCA','L7  ','PSCA','L8  ','PSCA','L9  ',
     *         'PSCA','L10 ','PSCA','L11 ','PSCA','L12 ','PSCA','L13 ',
     *         'PSCA','L14 ','PSCA','L15 ','PSCA','L16 ','PSCA','L17 ',
     *         'PSCA','L18 ','PSCA','L19 ','PSCA','L20 ','PZER','O1  ',
     *         'PZER','O2  ','PZER','O3  ','PZER','O4  ','PZER','O5  ',
     *         'PZER','O6  ','PZER','O7  ','PZER','O8  ','PZER','O9  ',
     *         'PZER','O10 ','PZER','O11 ','PZER','O12 ','PZER','O13 ',
     *         'PZER','O14 ','PZER','O15 ','PZER','O16 ','PZER','O17 ',
     *         'PZER','O18 ','PZER','O19 ','PZER','O20 ','TABL','ES  ',
     *         'SORT','ORDR','WTSC','AL  '/
C                                       1=Logical variable
C                                       2=Number
C                                       3=String
      DATA CTYPE /1,2,2,2, 2,2,2,2, 2,2,2/
      DATA KTYPE /3,3,3,3, 3,3,2,2, 3,3,3,3, 3,3,3,3, 3,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,3,3, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,3, 3,1,2,2,
     *            20*3, 20*2, 20*2, 2,3,2/
C                                       1000*nbytes + 100*offset +
C                                       position of pointer in common
      DATA CPOINT /   0, 2043, 2041, 2042, 2142, 2242, 2342, 2442,
```

```
     *              2542, 2642, 2742/
      DATA POINT / 8001, 8002, 8003, 8004, 8005, 8006, 8029, 8030,
     *             8007, 8009, 8109, 8209, 8309, 8409, 8509, 8609,
     *             8709, 8031, 8131, 8231, 8331, 8431, 8531, 8631,
     *             8731, 4010, 4110, 4210, 4310, 4410, 4510, 4610,
     *             4710, 4011, 4111, 4211, 4311, 4411, 4511, 4611,
     *             4711, 4012, 4112, 4212, 4312, 4412, 4512, 4612,
     *             4712, 4013, 4014, 4015, 4016, 2044,12017, 6218,
     *             2045, 2046, 2048, 2047, 4020, 4021, 4022, 2049,
     *             8035, 4023, 8032, 8033, 8034, 4024, 4025,    0,
     *                0, 1001, 2039, 2040,
     *             20*8008, 20*4004, 20*4004, 4004, 2048, 4004/
C                                                          End VFUV.
```

13.6.6 VFIT.INC

```
C                                                          Include VFIT
      DATA NCT/10/,     NKT/72/
      DATA CWORD/'SIMP','LE  ','BITP','IX  ','NAXI','S   ','NAXI',
     *  'S1  ','NAXI','S2  ','NAXI','S3  ','NAXI','S4  ','NAXI','S5  ',
     *         'NAXI','S6  ','NAXI','S7  '/
      DATA KWORD /'OBJE','CT  ','TELE','SCOP','INST','RUME','OBSE',
     *  'RVER','DATE','-OBS','DATE','-MAP','BSCA','LE  ','BZER','O   ',
     *         'BUNI','T   ','CTYP','E1  ','CTYP','E2  ','CTYP','E3  ',
     *         'CTYP','E4  ','CTYP','E5  ','CTYP','E6  ','CTYP','E7  ',
     *         'CRVA','L1  ','CRVA','L2  ','CRVA','L3  ','CRVA','L4  ',
     *         'CRVA','L5  ','CRVA','L6  ','CRVA','L7  ','CDEL','T1  ',
     *         'CDEL','T2  ','CDEL','T3  ','CDEL','T4  ','CDEL','T5  ',
     *         'CDEL','T6  ','CDEL','T7  ','CRPI','X1  ','CRPI','X2  ',
     *         'CRPI','X3  ','CRPI','X4  ','CRPI','X5  ','CRPI','X6  ',
     *         'CRPI','X7  ','CROT','A1  ','CROT','A2  ','CROT','A3  ',
     *         'CROT','A4  ','CROT','A5  ','CROT','A6  ','CROT','A7  ',
     *         'EPOC','H   ','DATA','MAX ','DATA','MIN ','BLAN','K   ',
     *         'INHI','BIT ','IMNA','ME  ','IMCL','ASS ','IMSE','Q   ',
     *         'USER','NO  ','PROD','UCT ','NITE','R   ','BMAJ','    ',
     *         'BMIN','    ','BPA ','    ','VELR','EF  ','ALTR','VAL ',
     *         'ALTR','PIX ','OBSR','A   ','OBSD','EC  ','REST','FREQ',
     *         'XSHI','FT  ','YSHI','FT  ','DATE','    ','ORIG','IN  ',
     *         'TABL','ES  ','OPHR','AE11','OPHD','CE11','MAPN','AM11'/
C                                     1=Logical variable
C                                     2=Number
C                                     3=String
      DATA CTYPE /1,2,2,2, 2,2,2,2, 2,2/
      DATA KTYPE /3,3,3,3, 3,3,2,2, 3,3,3,3, 3,3,3,3, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,3,3,2, 2,2,2,2, 2,2,2,2,
     *            2,2,2,2, 2,2,3,3, 2,2,2,3/
C                                     1000*nbytes + 100*offset +
C                                     position of pointer in common
      DATA CPOINT /   0, 2043, 2041, 2042, 2142, 2242, 2342, 2442,
     *             2542, 2642/
      DATA POINT / 8001, 8002, 8003, 8004, 8005, 8006, 8029, 8030,
```

```
     *          8007, 8009, 8109, 8209, 8309, 8409, 8509, 8609,
     *          8031, 8131, 8231, 8331, 8431, 8531, 8631, 4010,
     *          4110, 4210, 4310, 4410, 4510, 4610, 4011, 4111,
     *          4211, 4311, 4411, 4511, 4611, 4012, 4112, 4212,
     *          4312, 4412, 4512, 4612, 4013, 4014, 4015, 4016,
     *          2044,12017, 6218, 2045, 2046, 2048, 2047, 4020,
     *          4021, 4022, 2049, 8035, 4023, 8032, 8033, 8034,
     *          4024, 4025,    0,    0, 4001, 4101, 4201,12017/
C                                                    End VFIT.
```

13.7 AIPS FITS PARSING ROUTINES

There are several AIPS utility routines which are useful for parsing (reading) FITS header records. These routines are briefly described in the following; details of the call sequences etc. will be given later.

- FPARSE parses a FITS header card, unpacking the card image, interpreting it and putting the data value into the correct location in the AIPS catalogue header. This routine is for standard FITS headers but with the substitution of the INCLUDEs DFIT.INC, EFIT.INC and VFIT.INC for DFUV.INC, EFUV.INC and VFUV.INC the routine will work for FITS image tapes written on the VLA pipeline.

- GETCRD unpacks a given card image from a header block of FITS data and looks for keywords in a supplied table.

- GETSYM finds the next symbol in an unpacked buffer. A symbol is defined to begin with a letter and have up to 8 alpha-numeric characters.

- GETLOG obtains the value of a logical variable from an unpacked buffer.

- GETNUM converts an ASCII numeric field into a REAL*8 value.

- GETSTR obtains a character string from an unpacked buffer.

Following are the details of the call sequence and function of the AIPS FITS parsing utility routines.

13.7.1 FPARSE - (parse FITS card) will unpack and interpret a card image from a block of FITS data and put that data into the internal AIPS header. Works for standard uv or image FITS headers.

```
      FPARSE (ICARD, FITBLK, PSCAL, POFF, PTYPES, TABLES,
     *   END, IERR)
```

Inputs:		
ICARD	I*2	The card number (1-36) in block to interpret.
FITBLK	I*2(1440)	A block of FITS header data.
Outputs:		
PSCAL	R*8(20)	Random parameter scalings
POFF	R*8(20)	Random parameter offsets
PTYPES	R*4(20)	Random parameter types (packed chars every other one)
TABLES	I*2	# Tables extension
END	L*2	True if end card found, else false.
IERR	I*2	error code 0=ok. 1=error.
COMMON /MAPHDR/		

13.7.2 GETCRD - (get card) will unpack a given card image from a header block of FITS data, look for a recognizable key word from a supplied table and return information to the calling routine.

```
      GETCRD (ICARD, NOSYM, STRSYM, SYMTAB, FITBLK, NPNT,
     *   KL, SYMBOL, TABNO, ISHIST, END, IERR)
```

Inputs:		
ICARD	I*2	the card image (1-36) in FITS data block.
NOSYM	I*2	the number of entries in key word table.
STRSYM	I*2	Start search with symbol # STRSYM
SYMTAB	I*2(2,NOSYM)	unpacked keywords, two per I*2.
FITBLK	I*2(1440)	the block of FITS header cards.
In/Out:		
NPNT	I*2	The position to start scan in array KL. Returns the last position scanned plus one.
KL	I*2(80)	input the unpacked card image if NPNT > 1, else returns the unpacked card image.
Outputs:		
SYMBOL	I*2(2)	the unpacked symbol found on the card.
TABNO	I*2	SYMBOL matches SYMTAB(1&2,TABNO).
ISHIST	L*2	True if history card else false.
END	L*2	True if end card found, else false.
IERR	I*2	0=match found, 1=no match on otherwise valid keyword, 2=card ends or other trouble

13.7.3 GETLOG - obtains the value of a logical variable from loose buffer.

```
GETLOG (KB, LIMIT, KBP, IL)

Inputs:
   KB(80)    I*2   Loose buffer of card image
   LIMIT     I*2   Number of words in loose buffer
   KBP       I*2   Pointer position at start
Outputs:
   KBP       I*2   Pointer position of next field
   IL        I*2   Value of logical field
                   0--> .false.
                   1--> .true.
                   2--> invalid
```

13.7.4 GETNUM - converts ASCII numeric field into REAL*8 number.

```
GETNUM (KB, KBPLIM, KBP, X)

Inputs:  KB    I*2( )  loose character buffer
         KBPLIM I*2    # characters in buffer
         KBP   I*2     start of numeric field
Outputs: KBP   I*2     start of next field (incl blanks)
         X     R*8     numerical value
```

13.7.5 GETSTR - obtains a hollerith value from a loose buffer.

```
GETSTR (KB, KBPLIM, NMAX, KBP, ISTR, NCHAR)

Inputs:  KB      I*2(80)   loose buffer
         KBPLIM  I*2       size of loose buffer
         NMAX    I*2       max string length in characters
         KBP     I*2       start position in KB
Outputs: KBP     I*2       start position in KB next field
         ISTR    R*4(*)    packed string, blank filled
         NCHAR   I*2       # characters (0 => no string found)
```

13.7.6 GETSYM - scrutinizes a card image to look for the next symbol. A symbol begins with a letter and contains up to eight alpha-numeric characters (A-Z,0-9,_). This routine is used for interpreting a FITS tape and for interpreting the HI files.

```
GETSYM (LBUFF, NPNT, SYM, IERR)

Inputs:
   LBUFF(80)   I*2   Loose packed card image
```

NPNT	I*2	Pointer to first character
Output:		
NPNT	I*2	Pointer value after getting symbol
SYM(2)	R*4	Symbol, padded with blanks
IERR	I*2	Return code 0--> Found legal symbol followed by '=' 1--> Ran off the end of the card 2--> Symbol had >8 characters 3--> Found legal symbol with no '=' or SYM is HISTORY or COMMENT 4--> Found a '/' symbol 5--> Symbol contains an illegar char

13.8 REFERENCES

Wells, Greisen, and Harten 1981, Astronomy and Astrophysics Supplement series, vol. 44, pp 363 - 370.

Greisen and Harten, 1981, Astronomy and Astrophysics Supplement Series, vol. 44, pp 371 - 374.

Harten, Grosbol, Tritton, Greisen and Wells 1984, draft reproduced in the IAU Comission 9 Astronomical Image Processing Circular.

INDEX