# Chapter 3

# Getting Started — Tasks

## 3.1 Overview

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

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

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

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

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

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

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

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

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

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

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

## 3.2 The Cost of Machine Independence

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

### 3.2.1 Character Strings

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

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

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

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

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

## 3.3 Task Name Conventions

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

## 3.4 Getting the Parameters

### 3.4.1 In AIPS (Help file)

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

```
; PRTIM
;---------------------------------------------------------------
;! Task displays a map on line-printer or terminal
;# TASK PRINTER
;-----------------------------------------------------------------------
;;  Copyright (C) 1995
;;  Associated Universities, Inc. Washington DC, USA.
;;
;;  This program is free software; you can redistribute it and/or
;;  modify it under the terms of the GNU General Public License as
;;  published by the Free Software Foundation; either version 2 of
;;  the License, or (at your option) any later version.
;;
;;  This program is distributed in the hope that it will be useful,
;;  but WITHOUT ANY WARRANTY; without even the implied warranty of
;;  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;;  GNU General Public License for more details.
;;
;;  You should have received a copy of the GNU General Public
;;  License along with this program; if not, write to the Free
;;  Software Foundation, Inc., 675 Massachusetts Ave, Cambridge,
;;  MA 02139, USA.
;;
;;  Correspondence concerning AIPS should be addressed as follows:
;;         Internet email: aipsmail@nrao.edu.
;;         Postal address: AIPS Project Office
;;                         National Radio Astronomy Observatory
;;                         520 Edgemont Road
;;                         Charlottesville, VA 22903-2475 USA
;-----------------------------------------------------------------------
;---------------------------------------------------------------
PRTIM     LLLLLLLLLLLLUUUUUUUUUUUU CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
PRTIM:  Task to print the image intensities in digital form
USERID      -32000.0      32000.0  User ID. 0 => current user,
                                     32000 => any user.
INNAME                             Image name(name).
INCLASS                            Image name(class).
INSEQ            0.0       9999.0  Image name(seq. #). 0 => high
INDISK           0.0          9.0  Disk drive #.      0 => any
BLC              0.0       4096.0  Bottom left corner of image
                                     0 => entire image
TRC              0.0       4096.0  Top right corner of image
                                     0 => entire image
NDIG             0.0          7.0  Digits in display.     0 => 1
FACTOR      -99999.0     999999.0  Multiplication factor. 0 => 1
XINC             0.0        100.0  Display every XINC col. 0=> 1
YINC             0.0        100.0  Display every YINC rows. 0=>1
DOCRT           -1.0        132.0  >0 -> use CRT, else printer
                                   >72 => CRT width in chars
---------------------------------------------------------------
PRTIM
Type: Task
Use:  PRTIM displays an image on the line printer or the user's
      terminal.  The input parameters specify (1) any
```

```
      rectangular solid in the image, (2) the number of digits
      used for the display, (3) the skipping of pixels and (4)
      the multiplication of the display after normalization.
      Normalization is done using the larger of Datamax and
      -10.0*Datamin rounded up to the next higher power of 10.
      For NDIG=1, Datamin is used instead of 10 * Datamin.  This
      default scaling (FACTOR = 1) causes the numbers to be
      printed in "natural" units (e.g. Jy/beam) scaled by some
      power of 10.  The output shows the scaling (e.g. 1000 =
      1.0 Jy/beam on NDIG = 3).
Adverbs:
  USERID......User ID of owner of image.  0 => current user,
                32000 => any user.
  INNAME......Image name(name).         Standard defaults.
  INCLASS.....Image name(class).        Standard defaults.
  INSEQ.......Image name(seq. #).       0 => highest.
  INDISK......Disk drive # of image.    0 => any.
  BLC.........The Bottom Left-hand pixel of the subarray of
              the map to be displayed. The value (0,0) means
              start at the bottom left of the entire image.
  TRC.........The Top Right-hand pixel of the subarray of
              the map to be displayed.  The value (0,0) means
              go to the top right of the entire image.
  NDIG........The number of digits in the display.  If NDIG <=
              0, an NDIG of 1 is used.
  FACTOR......The multiplication factor for the display.  If
              FACTOR <= 0.0, a FACTOR of 1.0 is used.
  XINC........Display every XINC column(s) in the display.
              If XINC <= 0, an XINC of 1 is used.
  YINC........Display every YINC row(s) in the display.
              If YINC <= 0, a YINC of 1 is used.
  DOCRT.......True (> 0.0) means to use the terminal, otherwise
              use the line printer.  If 72 < DOCRT <= 132, the
              task will assume that the terminal is DOCRT
              characters wide.
----------------------------------------------------------------
```

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

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

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

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

### 3.4.2 At Task Startup (GTPARM)

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

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

### 3.4.3 While a Task is Running (GTTELL)

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

## 3.5 Restarting AIPS

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

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

## 3.6 INCLUDE files

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

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

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

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

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

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

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

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

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

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

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

## 3.7 Initializing Commons

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

### 3.7.1 Device Characteristics Common

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

```
SYSNAM  C*20     System name
VERNAM  C*4      Version ID
RLSNAM  C*8      Release name
DEVNAM  C(10)*48 Names of files using non-FTAB I/O currently open.
NONNAM  C(8)*48  Names of files using non-map I/O currently open.
MAPNAM  C(12)*48 Names of files using map I/O currently open.
SYSTYP  C*4      system type: 'VMS ' or 'UNIX' or ???
SYSVER  C*8      system version: '4.5', 'BSD 4.2', 'SYS 5', ...
XPRDMM  R        Printer points per millimeter
```

```
XTKDMM  R         Graphics points per millimeter
TIMEDA  R(15)     Min. TIMDEST time for each disk (days)
TIMESG  R         Min. TIMDEST time for SAVE/GET files (days)
TIMEMS  R         Min. automatic destruction time for messages
TIMESC  R         Min. automatic destruction time for scratch
TIMECA  R         Min. destruction time for empty catalogs.
TIMEBA  R(4)      Times during which AP Batch jobs cannot start.
                  1, 2 start, stop times (hrs) on weekends
                  3, 4 start, stop times (hrs) on weekdays
TIMEAP  R(3)      1 => time between rolls (min)
                  2,3 polynomial terms for determining how long
                  a job must wait before grabbing the AP.
FBLANK  R         REAL value used to indicate blanking
DBLANK  D         DOUBLE PRECISION value used to indicate blanking.
HBLANK  H         HOLLERITH blank string (4 char)
RFILIT  R(14)     Spare
NVOL    I         Number of disk drives available to AIPS
NBPS    I         Number of bytes per disk sector
NSPG    I         Number of disk sectors per allocation granule
NBTB1   I         Number bytes in FTAB / non-FTAB device
NTAB1   I         Max number of non-FTAB devices open at once
NBTB2   I         Number bytes in FTAB / slow I/O device
NTAB2   I         Max number of slow I/O devices open at once
NBTB3   I         Number bytes in FTAB / fast I/O device
NTAB3   I         Max number of fast I/O devices open at once
NTAPED  I         Number of tape drives available to AIPS
CRTMAX  I         Number lines / CRT terminal page
PRTMAX  I         Number lines / printer page
NBATQS  I         Number batch AIPSs in system
MAXXPR  I(2)      Number of plotter dots / page in X, Y
CSIZPR  I(2)      Number of plotter dots / character in X, Y
NINTRN  I         Maximum # simultaneous interactive AIPSs
KAPWRD  I         # 1024s of words of array processor memory
NWDPDP  I         # words / double-precision floating point
NBITWD  I         # bits / word
NWDLIN  I         # words in a POPS input line
NCHLIN  I         # characters in a POPS input line
NTVDEV  I         # television display devices available
NTKDEV  I         # graphics display devices available
BLANKV  I         Integer magic value => blanked pixel
NTVACC  I         Number POPS programs allowed access to TV devices
NTKACC  I         Number POPS programs allowed access to graphics
UTCSIZ  I         Private catalog size (0=>public)
BYTFLP  I         Byte flip, 0=none, 1=bytes, 2=words, 3=both
USELIM  I         Maximum user number
NBITCH  I         # bits per character
NCHPRT  I         Width of line printer in characters.
KAP2WD  I         # 1024s words of secondary AP memory.
MAXXTK  I(2)      Graphics screen size x,y
CSIZTK  I(2)      Graphics character size x,y
DASSGN  I(8,15)   Lists of allowed users, 8 per disk for up to 15
                  disks.
SPFRMT  I         Single precision floating-point format code
                  0 => OTHER
```

```
                  1 => IEEE
                  2 => VAX F
                  3 => VAX G
                  4 => IBM (not supported yet)
DPFRMT  I         Double precision floating-point format code
                  (see codes for SPFRMT)
NSHORT  I         Shortest vector length to vectorize
TTYCAR  I         1 => TTY i/o uses carriage control characters.
DEVTAB  I(50)     Device type code numbers
FTAB    I(*)      I/O driving tables
```

### 3.7.2 Catalog Pointer Common

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

### 3.7.3 History Common

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

### 3.7.4 TV Common

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

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

### 3.7.5 UV data pointer common

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

### 3.7.6 Files common, DFIL.INC

Many tasks open a number of cataloged files and create several scratch files. The status of the cataloged files are marked 'READ' or 'WRIT' in the catalog directory and need to be cleared by the end of the program.

Scratch files must be destroyed by the end of the program. Since an error might terminate the program at any stage, the program must be prepared to clear catalog files and destroy scratch files under any circumstances in which it controls its death.

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

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

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

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

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

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

## 3.8 Input and Output File Names

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

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

## 3.9 Copying Extension Files

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

### 3.9.1 History

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

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

- HIINIT initializes the history common.
- HISCOP creates and catalogs a new history file, opens it, opens an old history file and copies it to the new history file, and leaves the old history file closed and the new file open.
- HIADD adds a history card to an open history file.
- HIADDN adds a history card to number of open history files.
- HIAD80 adds an 80-character card image into an open history file.
- HICLOS closes a history file, flushing the buffer if requested.
- HIMERG creates several history files by merging several old files.
- HIREAD reads the next history card from an open history file.

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

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

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

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

### 3.9.2 Extension tables (ALLTAB, TABCOP)

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

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

## 3.10 Communication with the user

### 3.10.1 Writing messages

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

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

```
Priority              Use
  0           Write to log file only
  1           Write to monitor terminal only
  2           Low interest normal messages
 3-4          Normal message
  5           High interest normal message.
 6-8          Error message
 9-10         Severe error messages
```

An example of the use of MSGWRT follows:

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

### 3.10.2 Turning off system messages

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

### 3.10.3 Writing to the line printer

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

```
      INTEGER   LPLUN, LPIND, BUFFER(256), IPCNT
      LOGICAL   T,F
      REAL      VALUE1, VALUE2
      CHARACTER LPNAME*48
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
      PARAMETER (LPLUN = 1)
      PARAMETER (LPNAME = '    ')
      INCLUDE 'INCS:DDCH.INC'
          .
          .
C                                       Open the printer.
      CALL ZOPEN (LPLUN, LPFIND, 1, LPNAME, F, T, T, IERR)
        (handle error condition if detected)
C                                       Header page if batch
      CALL BATPRT (1, BUFFER)
      IPCNT = 0
          .
          .
C                                       Increment line count
      IPCNT = IPCNT + 1
C                                       Check if page full.
      IF (IPCNT .LT. PRTMAX) GO TO 100
C                                       Write new page header
          .
          .
         ICPNT = 0
C                                       Write to printer
 100  WRITE (LPLUN,1000) VALUE1, VALUE2
          .
          .
C                                       Trailer page if batch
      CALL BATPRT (2, BUFFER)
C                                       Eject a page
      CALL ZENDPG (IPCNT)
C                                       Close printer and send to
C                                       spooler.
      CALL ZCLOSE (LPLUN, LPIND, IERR)
          .
          .
 1000 FORMAT (' VALUE1 =',F10.5, ' VALUE2 =',1PE12.6)
```

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

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

### 3.10.4 Writing to the Terminal (ZTTYIO)

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

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

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

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

```
                    .
C                                       Close terminal
      CALL ZCLOSE (TTYLUN, TTYIND, IERR)
                    .
                    .
 1000 FORMAT (' Hi there')
```

## 3.11 Scratch Files

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

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

An example of the use of SCREAT is the following:

```
      INTEGER   IRET, NX, NY, NP(2), BUFF(512), SIZE
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DDCH.INC'
           .
           .
           .
C                                       NX, NY are the size of an
C                                       image.  Make a scratch file
C                                       big enough for a copy of the
C                                       image.
C
C                                       Compute the size.
      NP(1) = NX
      NP(2) = NY
C                                       Compute size needed
      CALL MAPSIZ (2, NP, SIZE)
C                                       Create scratch file.
      CALL SCREAT (SIZE, BUFF, IRET)
C                                       Test for errors...
```

In the above example, the scratch file created will be entered in the DFIL.INC common as number N-SCR (which was incremented). The disk and catalog slot numbers are thus SCRVOL(NSCR) and SCRC-NO(NSCR). This scratch file can be opened as follows:

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

```
     *    .TRUE., IRET)
```

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

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

## 3.12 Terminating the Program

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

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

## 3.13 Batch Jobs

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

## 3.14 Installing a New Task

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

## 3.15 INCLUDEs

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

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

### 3.15.1 DDCH.INC

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

### 3.15.2 DFIL.INC

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

```
C                                                           End DFIL.
```

### 3.15.3 DMSG.INC

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

### 3.15.4 DUVH.INC

```
C                                                           Include DUVH.
C                                       If you change this include you
C                                       must also change common
C                                       /CATHDR/ in DBCON
C                                       Include for uv header info
      INTEGER   NVIS
      INTEGER   ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, ILOCFQ,
     *   JLOCC, JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC,
     *   NCOR, INCS, INCF, INCIF, ICOR0, TYPUVD
      CHARACTER   SOURCE*8, ISORT*2
      DOUBLE PRECISION FREQ, RA, DEC
      COMMON /UVHDR/ FREQ, RA, DEC, NVIS, ILOCU, ILOCV, ILOCW, ILOCT,
     *   ILOCB, ILOCSU, ILOCFQ, JLOCC, JLOCS, JLOCF, JLOCR, JLOCD,
     *   JLOCIF, INCS, INCF, INCIF, ICOR0, NRPARM, LREC, NCOR, TYPUVD
      COMMON /UVHCHR/ SOURCE, ISORT
C                                                           End DUVH.
```

### 3.15.5 PUVD.INC

```
C                                                           Include PUVD
C                                       Parameters for uv data
      INTEGER   MAXANT,  MXBASE,  MAXIF, MAXFLG, MAXFLD, MAXCHA
C                                       MAXANT = Max. no. antennas.
      PARAMETER (MAXANT=45)
C                                       MXBASE = max. no. baselines
      PARAMETER (MXBASE= ((MAXANT*(MAXANT+1))/2))
C                                       MAXIF=max. no. IFs.
      PARAMETER (MAXIF=15)
C                                       MAXFLG= max. no. flags active
      PARAMETER (MAXFLG=1000)
C                                       MAXFLD=max. no fields
      PARAMETER (MAXFLD=16)
C                                       MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                       Parameters for tables
```

```
      INTEGER MAXCLC, MAXSNC, MAXANC, MAXFGC, MAXNXC, MAXSUC,
     *   MAXBPC, MAXBLC, MAXFQC
C                                       MAXCLC=max no. cols in CL table
      PARAMETER (MAXCLC=41)
C                                       MAXSNC=max no. cols in SN table
      PARAMETER (MAXSNC=20)
C                                       MAXANC=max no. cols in AN table
      PARAMETER (MAXANC=12)
C                                       MAXFGC=max no. cols in FG table
      PARAMETER (MAXFGC=8)
C                                       MAXNXC=max no. cols in NX table
      PARAMETER (MAXNXC=7)
C                                       MAXSUC=max no. cols in SU table
      PARAMETER (MAXSUC=21)
C                                       MAXBPC=max no. cols in BP table
      PARAMETER (MAXBPC=14)
C                                       MAXBLC=max no. cols in BL table
      PARAMETER (MAXBLC=14)
C                                       MAXFQC=max no. cols in FQ table
      PARAMETER (MAXFQC=5)
C                                                          End PUVD.
```

## 3.16 Routines

### 3.16.1 ALLTAB

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

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

### 3.16.2 CHCOPY

CHCOPY moves characters from one HOLLERITH string to another

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

### 3.16.3 CHCOMP

CHCOMP compares two HOLLERITH strings

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

### 3.16.4 CHFILL

CHFILL fills a HOLLERITH string with a character

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

### 3.16.5 CHLTOU

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

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

### 3.16.6 CHMATC

searches one HOLLERITH string for the occurrence of another string.

```
CHMATC (NA, JA, CA, NB, JB, CB, NP)
Inputs:
   NA   I        Number of characters in CA (start at JA)
   JA   I        Start at char position JA in CA
   CA   H(*)     Packed substring to be found in CB
```

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

### 3.16.7 CHR2H

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

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

### 3.16.8 CHWMAT

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

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

### 3.16.9 DIE

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

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

### 3.16.10 DIETSK

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

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

### 3.16.11 EXTCOP

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

```
EXTCOP (TYPE, INVER, OUTVER, LUNOLD, LUNNEW, VOLOLD,
  *   VOLNEW, CNOOLD, CNONEW, CATNEW, BUFF1, BUFF2, BUFF3, IRET)
Inputs:
   TYPE    C*2     Extension file type eg 'CC','AN'
   INVER   I       Version number to copy, 0=>copy all.
   OUTVER  I       Version number on output file, if more than one
                   copied (INVER=0) this will be the # of the first
                   file.  If OUTVER=0 the EXTINI defaults are used.
   LUNOLD  I       LUN for old file
   LUNNEW  I       LUN for new file
   VOLOLD  I       Disk number for old file.
   VOLNEW  I       Disk number for new file.
   CNOOLD  I       Catalog slot number for old file
   CNONEW  I       Catalog slot number for new file
   CATNEW  I(256)  Catalog header for new file.
In/out:
   BUFF1   I(>512) Work buffer: 256 words + n * 256 words (enough
                   to hold at least one logical record)
   BUFF2   I(>512) Work buffer: as BUFF1
   BUFF3   I(*)    Buffer large enough to hold one logical record
Output:
   IRET    I     Return error code  0 => ok
                                    1 => files the same, no copy
                                    2 => no input files exist
                                    3 => failed
                                    4 => no output files created
```

### 3.16.12 GTPARM

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

```
GTPARM (NAME, NPARMS, RQUICK, RPARM, SCRTCH, IERR)
Inputs:
   NAME    C*6        Task name
   NPARMS  I          Number of real variables wanted
Outputs:
   RQUICK  L          T => release POPs as soon as possible
```

```
                    F => wait until you have finished
   RPARM   R(NPARMS)  Parameters received
   SCRTCH  I(256)     Scratch buffer
   IERR    I          Error code: 0 -> ok
                                  1 -> initiator (AIPS) not found
                                  2 -> disk troubles
                                  3 -> initiator zeroed
```

### 3.16.13 GTTELL

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

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

### 3.16.14 HIADD

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

```
HIADD (HLUN, CARD, BUFFER, IERR)
Inputs:
   HLUN    I         LUN of HI file (must be open!!)
   CARD    C*72      new card
In/out:
   BUFFER  I(256)    HI work buffer
Output:
   IERR    I         Error return: 0 => ok, other set by HIIO
```

### 3.16.15 HIADDN

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

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

### 3.16.16 HIAD80

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

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

### 3.16.17 HICLOS

HICLOS closes a history file updating it if requested.

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

### 3.16.18 HIINIT

HIINIT initializes the history common area /HICOM /

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

### 3.16.19 HIMERG

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

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

### 3.16.20 HIREAD

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

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

### 3.16.21 HISCOP

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

```
HISCOP (LUNOLD, LUNNEW, VOLOLD, VOLNEW, CNOOLD,
  *   CNONEW, CATBLK, BUFER1, BUFER2, IERR)
Inputs:
   LUNOLD   I        LUN for old history file.
   LUNNEW   I        LUN for new history file.
   VOLOLD   I        Vol. number for old history file.
```

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

### 3.16.22 H2CHR

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

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

### 3.16.23 MAKOUT

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

```
OUTS  <=  -1         becomes  OUTS = INSEQ
OUTN  =  ' '         becomes  OUTN = INN
         'yy*zz '    becomes  OUTN = INN   with first n characters
                     replaced by yy and last m chars with zz - if
                     yy or zz contain ?'s don't replace those char
                     positions
OUTCL =  '   '       becomes  OUTCL= DEFCLS
         'yy*zz '    becomes  OUTCL= DEFCLS with same as OUTN
        If the 1st character of OUTCL is a '_' then the default
        is replaced with INCL and the remaining 5 characters of
        OUTCL are used as normal.

MAKOUT (INN, INCL, INS, DEFCLS, OUTN, OUTCL, OUTS)
Inputs:
   INN     C*12     Input file name
   INCL    C*6      Input file class
   INS     I        Input file sequence number
   DEFCLS  C*6      Default output file class 6 packed chars
```

```
                    if = ' ', use task name
In/Out:
   OUTN    C*12     User-supplied OUTNAME adverb
   OUTCL   C*6      User-supplied OUTCLASS adverb
   OUTS    I        User-supplied OUTSEQ adverb in integer
NOTE: the actual Input file name parameters must be supplied, not
the user adverbs (which can themselves contain wild cards, pure
blank fields, zeros, and the like.
```

### 3.16.24 PRTLIN

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

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

### 3.16.25 PSFORM

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

```
PSFORM (NC, PS, IPT)
Inputs:
   NC   I       Number characters in pattern possible
   PS   C*(*)   Pattern string
Output:
   IPT  I(NC)   Coded array:
                    value = -2 => position is *
                    value = -1 => position is ?
                    value = 0  => position is a blank
                    value > 0  => there are IPT(i) real chars
                                  including the present following
```

### 3.16.26 RELPOP

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

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

### 3.16.27 SCREAT

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

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

### 3.16.28 TABCOP

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

```
TABCOP (TYPE, INVER, OUTVER, LUNOLD, LUNNEW, VOLOLD,
  *   VOLNEW, CNOOLD, CNONEW, CATNEW, BUFF1, BUFF2, IRET)
Inputs:
   TYPE     C*2     Extension file type (e.g. 'CC','AN')
   INVER    I       Version number to copy, 0 => copy all.
   OUTVER   I       Version number on output file, if more than one
                    copied (INVER=0) this will be the number of the
                    first file.  If OUTVER = 0, it will be taken as
                    1 higher than the previous highest version.
   LUNOLD   I       LUN for old file
   LUNNEW   I       LUN for new file
   VOLOLD   I       Disk number for old file.
   VOLNEW   I       Disk number for new file.
   CNOOLD   I       Catalog slot number for old file
   CNONEW   I       Catalog slot number for new file
In/out:
```

```
   CATNEW   I(256)  Catalog header for new file.
Output:
   BUFF1    I(256)  Work buffer
   BUFF2    I(256)  Work buffer
   IRET     I       Return error code  0 => ok
                       1 => files the same, no copy.
                       2 => no input files exist
                       3 => failed
                       4 => no output files created.
                       5 => failed to update CATNEW
                       6 => output file exists
```

### 3.16.29 UVPGET

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

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

```
UVPGET (IERR)
Inputs: From common /MAPHDR/ (DCAT.INC)
   CATBLK   I(256)   Catalog block
   CATH     H(256)   same as CATBLK
   CATR     R(256)   same as CATBLK
   CATD     D(128)   same as CATBLK
Output: In common /UVHDR/ (DUVH.INC)
   SOURCE   C*8      Source name.
   ILOCU    I        Offset from beginning of vis record of U
                     or longitude for single dish format data.
   ILOCV    I        Offset from beginning of vis record of V
                     or longitude for single dish format data.
   ILOCW    I        Offset from beginning of vis record of W.
   ILOCT    I                     "                        Time
   ILOCB    I                     "                      Baseline
                                                         (or beam)
   ILOCSU   I                     "                    Source id.
   ILOCFQ   I                     "                    Freq id.
   JLOCC    I        0-rel. order in data of complex values
   JLOCS    I        Order in data of Stokes' parameters.
   JLOCF    I        Order in data of Frequency.
   JLOCR    I        Order in data of RA
   JLOCD    I        Order in data of dec.
   JLOCIF   I        Order in data of IF.
   INCS     I        Increment in data for stokes (see above)
   INCF     I        Increment in data for freq. (see above)
   INCIF    I        Increment in data for IF.
   ICOR0    I        Stokes value of first value.
   NRPARM   I        Number of random parameters
   LREC     I        Length in values of a vis record.
   NVIS     I        Number of visibilities
```

```
   FREQ     D        Frequency (Hz)
   RA       D        Right ascension (1950) deg.
   DEC      D        Declination (1950) deg.
   NCOR     I        Number of correlators (Stokes' parm.)
   ISORT    C*2      Sort order 1st 2 char meaningful.
   TYPUVD   I        UV data type, 0=interferometer,
                        1=single dish unprojected,
                        2=single dish projected RA and Dec.
   IERR     I        Return error code: 0=>OK,
                        1, 2, 5, 7 : not all normal rand parms
                        2, 3, 6, 7 : not all normal axes
                        4, 5, 6, 7 : wrong bytes/value
```

### 3.16.30 ZDCHIN

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

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

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

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

### 3.16.31 ZTTYIO

Perform I/O to a terminal.

```
ZTTYIO (OPER, LUN, FIND, NCHARS, BUFF, IERR)
Inputs:
   OPER     C*4   Operation code 'READ' or 'WRIT'
   LUN      I     Logical unit number
   FIND     I     Index in FTAB to file control block for LUN
   NCHARS   I     # characters to transfer (<= 132)
In/out:
   BUFF     C*(*) I/O buffer containing characters (1-256)
Output:
   IERR     I     Error return code: 0 => no error
                     1 => file not open
                     2 => input error
                     3 => I/O error
                     4 => end of file
```