Object-Oriented Programming in AIPS Fortran

W. D. Cotton, N.R.A.O.

4 October 1993

ABSTRACT

This document describes a object oriented system running in the AIPS environment. The implementation was done using the AIPS Fortran preprocessor and results in many of the benefits of languages with explicit object oriented support. This document is intended both for the users and maintainers of this package.

Contents

1 Introduction

This document describes a partial implementation of an object oriented interface to AIPS data structures using the AIPS Fortran preprocessor. This implementation was intended to explore the possibilities of object oriented programming (OOP) in AIPS. These interfaces include image, uv data, and tables data structures. Aspects of object oriented programming are present in this implementation but the general style is best described as object based.

2 Object Oriented Programming (OOP)

The meaning of the term "object oriented programming" varies depending on the context but there are several aspects that need to be considered. These are 1) classes, 2) encapsulation, 3) inheritance, 4) polymorphism, and 5) instances of class (objects).

2.1 Classes

A class is an external definition of a data entity (public members) and the functions (member functions) which allow external access to, and manipulation of, this data entity. The run time realization of a class is an object. In this implementation, a class consists of a set of routines in a single file. Examples of classes are images, pixel arrays, and tables. The more complex classes such as images are composed of simpler classes such as pixel arrays. Any needed I/O and manipulation of AIPS catalogs are handled by the class member function transparently to the clients of these routines.

2.2 Encapsulation

A synonym for encapsulation is data hiding. Since data structures tend to change with time or application it is desirable to localize the knowledge of these structures to a well defined set of routines called a module or class. For this to work, these routines need to have data which they share with each other but is hidden from other access. Hiding of the details of the internal data structure allows changes in these structures to be transparent outside of the class. Hiding of class private data is difficult in non-OO languages; in Fortran, named commons are visible to all and the only viable approach is to put all the functions of a class into a single routine with many entry points. The AIPS Fortran preprocessor has a facility for a "LOCAL INCLUDE" file which can declare commons which are only available to routines in that file. This facility, which allows data to be shared by routines in a given file but hidden from other routines, has been used extensively in the implementation described in this document.

2.3 Inheritance

Inheritance is the ability to create new classes from existing classes and using the features of the existing classes if they are appropriate. Inheritance clearly is not a feature supported by Fortran 77 and the system described by this document cannot support true inheritance. Much of the functionality can be obtained by having a "derived" class "contain" an object of the base class rather than "be" an object of the base class as is the case with true inheritance. Derived classes in this system are somewhere between these two cases as the classes contained in the derived class do not have independent existence as objects but are parts of the derived object. This leads to behavior similar to true inheritance; for example, the image class is derived from the array class and the array member of the image object can be accessed as though the object were an array. The names for class members reflect their derivation. For example, the dimensionality member of the array descriptor member of the pixel array member of an image is named: "ARRAY.ARRAY_DESC.NAXIS". There are obvious limits to how far this naming convention can be carried.

2.4 Polymorphism

Polymorphism is the ability to deal with objects of different classes where the class of the object may be determined at run time. The ability to call the correct routine for a given object at run time is known as dynamic binding. There is obviously no support from Fortran 77 for this kind of operation so the use of "generic" functions is necessary. These generic functions determine the class of the object and call either a class specific function or, if possible, a truly generic function. Examples of these generic functions are creating, destroying and copying objects.

2.5 Instances of Class (Objects)

In most OOP languages objects are created and destroyed dynamically. In the implementation described in this document this is also the case. An object consists of a labeled collection of information (including the class name). Objects are given character string names and may be passed to subroutines or class member functions. Members of an object are addressed through a character string "keyword" which specifies the name and possibly the inheritance path of the element. It is possible to add relatively arbitrary information to an object. This allows attaching control information, such as windows in images, directly to the object. This control information can be accessed by any routine processing the object or ignored if it is not needed. This feature removes one of the primary difficulties of AIPS, that of passing this control information from the user to the routine where it is needed. The POPS adverbs passed are accessed as an "input" object. Members of input objects can be copied, and possibly renamed, to other arbitrary objects or the history file associated with an object. This greatly simplifies handling control information.

3 The Object Manager and Class I/O

The heart of this system is the object manager on top of which all classes are built. An object in this implementation consists primarily of a list of labeled information. Most of this information is kept in a linked list although some object specific information may kept elsewhere. For some classes some information is stored as part of an AIPS catalog header record (although this is nearly invisible at higher levels). Heavy use is made of the AIPS Fortran preprocessor LOCAL INCLUDE facility to provide data which is only available to the object manager. All access is through object manager functions. The Object manager is used only by Class modules and should not be visible at the applications level.

Relatively arbitrary collections of labeled information can be kept by the object manager. Items can be arrays (2 dimensions are actually supported but more could easily be implemented). Supported data types are double and single precision, character strings, integers and logicals.

"Virtual" keywords are used for either object values stored in the catalog header or derived values. The object manager knows explicitly about these virtual keywords and traps them. The object manager must be supplied with a routine to initialize and specify the virtual keywords. If the class has no virtual keywords then the object manager need not be modified. This use of virtual keywords hides that fact that some of the information is stored in an AIPS catalog header.

Although I/O to disk resident files are hidden from the applications level the class routines must still transfer data to and from disk as necessary. The buffers and control information are kept in the include file CLASSIO.INC. This include should NEVER appear in applications level routines. The values of various parameters used in the class I/O buffers and other system arrays are set in the include OBJPARM.INC which should also never be used outside of the object manager and class libraries.

4 Interface to AIPS

A user can send parameters to a task in AIPS by one of two non interactive methods; 1) the input parameters sent by GO and displayed by INPUT and 2) parameters sent after task activation by TELL and displayed by SHOW. These adverb values (and other necessary startup procedures) are obtained by members of the INPUT class and returned as an INPUT object. These routines take arrays defining the names, data types and dimensions of the expected arrays and MUST correspond EXACTLY to the adverbs POPS passes as defined in the *.HLP file.

AV2INP does the normal task startup functions and returns an INPUT object containing the passed parameters. IN2OBJ can then be used to copy lists of adverbs from the input object to another object with the possibility of renaming them. INTELL performs a similar function for adverbs passed using the TELL function in AIPS. INPUT objects can be be accessed like any other object.

5 History

History information is dealt with as text strings that are written to an object. This operation is only really defined for permanent, disk resident objects. Lists of labeled information can be copied from an arbitrary object, usually the inputs object, to the history of another object. The description of the history utility routines is described later in this document.

6 Class Interfaces

Each class has its own layer of interface routines. These interface routines can be used directly on objects of a derived type. In many cases these interface routines merely allow access to members of an object but in other cases support unary and binary operations on objects. An example of this is image arithmetic. There is also a generic set of the more common functions which will call the class specific routines. In particular access to data associated with an object can always be done using the generic routines OGET and OPUT. These routines are described in more detail in a following section.

For classes whose objects may contain much data there is a pair of efficient routines for read and write access to this data in addition to the more general access. Some of these are demonstrated in the following sections.

6.0.1 Keyword value pairs

Most of the descriptive information about an object is available as keyword/value pairs (where the value may be a 1 or 2 dimensional array). Keyword/value pairs are used to specify the underlying AIPS data structures, if any, as well as to carry arbitrary information that is required by class functions. There are a great variety of routines to access these keyword value pairs; generally a read and write routine for each class plus the generic OGET and OPUT. There is much overlap in the functionality of these routines. The main distinction is that the access functions for virtual classes (no actual instantiations, e.g. FILE_STATUS) will only access predefined class members whereas the generic and derived class access functions can process arbitrary keywords, including those of virtual base classes. The advantage of using specific virtual class access functions is that it reduces certain kinds of bugs.

In a similar mannar the use of explicit derivation heirarchy in member names ('CALEDIT.STOKES' vs. 'STOKES') is somewhat arbitrary. The advantages of the explicit derivation form are that 1) it is clear that the member belongs to a base class and 2) class membership and spelling are checked.

Some of these keyword/value pairs are mapped onto the underlying AIPS catalog header in a more or less invisible mannar. The main complication is that these keywords are only valid when the object is open. Before an object with an underlying AIPS structure is opened the appropriate keywords must be set. For images and UV data objects these are 'NAME', 'CLASS', 'IMSEQ' and 'DISK'. Table objects also require 'TBLTYPE' and 'VER'.

6.0.2 Multiple access

In general, an object only allows a single access at a time to underlying disk resident structures. That is, a single object cannot being both read and written at the same time. This problem is avoided by making a temporary, shallow copy of the object (same underlying structures) and reading from one and writing to the other. The temporary object should be deleted after use to free up the resources it uses. Generic function OCOPY makes a shallow copy of an image and DESTRY will delete an object but not its underlying files.

6.1 Generic Interface Routines

The following sections describe the generic routines in detail. Theses routines can be used for most operations involving AIPS objects although class specific routines may be desirable in some cases for efficiency reasons.

Class libraries should never reference the generic routines as this may result in a recursion. Routines in class libraies should always use the class specific access routines or the basic object access routines.

6.1.1 CREATE

CREATE (name, class, ierr)
Creates an object with specified name and class.

```
Inputs:
   NAME  C*?   The name of the object.
   CLASS C*8   Class of the object
Output:
   IERR  I     Error return code, 0=OK
```

6.1.2 DESTRY

DESTRY (name, ierr)
Destroys the object with name "name"; quasi-permanent forms are unaffected.

```
Inputs:
   NAME  C*?  The name of the object.
Output:
   IERR  I     Error return code, 0=OK
```

6.1.3 ZAP

ZAP (name, ierr)
Destroys the object with name "name"; quasi-permanent forms are deleted.

```
Inputs:
   NAME  C*?  The name of the object.
Output:
   IERR  I     Error return code, 0=OK
```

6.1.4 OCOPY

OCOPY (namein, namout, ierr)
Makes a shallow copy of one object to another. The same quasi permanent forms are used for both. This can be used to read and write to the same object.

```
Inputs:
   NAMEIN  C*?  The name of the input object.
   NAMOUT  C*?  The name of the output object.
Output:
   IERR    I     Error return code, 0=OK
```

6.1.5 OCLONE

OCLONE (namein, namout, ierr)
Clones (makes a deep copy of) an object. Each of the components parts is cloned.

```
Inputs:
   NAMEIN  C*?   The name of the input object.
   NAMOUT  C*?   The name of the output object.
Output:
   IERR    I     Error return code, 0=OK
```

6.1.6 OOPEN

OOPEN (name, status, ierr)
Opens an object for access. Obtains header info etc. Any disk resident files will be created if necessary.

```
Inputs:
   NAME   C*?  The name of the object.
   STATUS C*4   'READ', 'WRIT', 'DEST' (write but destroy on
                failure).
Output:
   IERR  I     Error return code, 0=OK, 5=data invalid
```

6.1.7 OCLOSE

OCLOSE (name, ierr)
Closes object updating disk resident information.

```
Inputs:
   NAME  C*?   The name of the object.
Output:
   IERR  I     Error return code, 0=OK
```

6.1.8 OGET

OGET (name, keywrd, type, dim, value, valuec, ierr)
Returns the dimensionality and value(s) associated with a given object member. Parameters in PAOOF.INC can be used for data type codes. The value returned will be VALUE or VALUEC depending on the value of TYPE, the other is undefined.

```
Inputs:
   NAME     C*?   The name of the object.
   KEYWRD   C*?   The name of the keyword in form 'MEM1.MEM2...'
Outputs:
   TYPE     I     Data type: 1=D, 2=R, 3=C, 4=I, 5=L
   DIM      I(*)  Dimensionality of value, an axis dimension of zero
                  means that that dimension and higher are
                  undefined.
   VALUE    ?(*)  The value associated with keyword.
   VALUEC   C*?   Associated value (character)
   IERR     I     Error return code, 0=OK
```

6.1.9 OPUT

OPUT (name, keywrd, type, dim, value, valuec, ierr)
Stores the dimensionality and value(s) associated with a given object member. Parameters in PAOOF.INC can be used for data type codes. The value stored will be VALUE or VALUEC depending on the value of TYPE, the other is ignored.

```
Inputs:
   NAME     C*?   The name of the object.
   KEYWRD   C*?   The name of the keyword in form 'MEM1.MEM2...'
   TYPE     I     Data type: 1=D, 2=R, 3=C, 4=I, 5=L
   DIM      I(*)  Dimensionality of value, an axis dimension of zero
                  means that that dimension and higher are
                  undefined.
   VALUE    ?(*)  The value associated with keyword.
   VALUEC   C*?   Associated value (character)
Outputs:
   IERR     I     Error return code, 0=OK
```

6.2 Objects and AIPS files

Many of the classes in this package are merely interfaces to AIPS data files which try to hide the details of the structures and fundamental access methods of these files. For this to work the objects must be connected to these underlying structures.

6.2.1 Names

The general method of associating AIPS file with objects is to attach keywords NAME, CLASS, IMSEQ and DISK with values corresponding to the AIPS file name components to the object and then "open" it. The process of opening the object will force the synchronization of memory resident structures with disk resident structures. The inverse operation, "closing" the object forces an update of the disk resident structures from the memory resident ones.

A related operation is when the "output" object is to be derived from another object via the clone operation. In this case the AIPS names of the new object are attached to the initial object as OUTNAME, OUTCLASS, OUTSEQ and OUTDISK.

6.2.2 Temporary Objects

Many AIPS utility routines refer to scratch files by the order number in the DFIL.INC common. When temporary or scratch objects (image or uvdata) are created they are given a keyword 'SCRCNO' whose associated value is this order number. Presence of this keyword is evidence that the underlying file exists and that it is a scratch file.

6.2.3 Catalog Status Flags

Image and UV data objects have a FILE_STATUS base class which has as members an access status and a data valid flag. The access status more or less corresponds to the AIPS catalog status flags and the access routines try to synchronize them. If the underlying files of an object are created then the validity flag is set false and read access is prohibited until cleared; usually on close after opening for write.

AIPS catalog status flags cause considerable logical difficulty. The current AIPS design has a single status flag for all components of the catalog entry. Catalog flags provide considerable protection for the user and should be invisible in applications software using this package. However, since catalog flags have been a rich source of bugs in the low level software some understanding of how they are handled is useful.

Image and uv objects attempt to set the corresponding AIPS catalog flags when they open and clear them when they close. The exceptions to this are 1) if the status is 'DEST' (destroy on failure) then the catalog status is not cleared until the status is changed or the program is terminated. 2) if the catalog entry already has a flag set by the same task then no attempt is made to set a new one (as this frequently doesn't work anyway). Table objects make no attempt to modify the AIPS catalog flags.

6.2.4 Logical unit numbers (LUNs)

This package has internally solved the problem of allocation of logical unit numbers by allocating them and deallocating them as needed. However, many utilities in the package are merely fronts to standard AIPS routines which may have hard coded LUNs. This may cause a conflict. The most serious case of this is in reading UV data which uses the standard AIPS calibration routines. These routines have hard coded LUNs and may use a considerable number of them especially if sorting of tables is required. If an input and output uv data are being accessed it is best to open the input object first. In general, when routines which are fronts to AIPS routines are called then all objects should be closed.

6.3 Persistance

All objects are ephemeral, existing only for the executation of a program. However, underlying disk resident structures may well be persistent (survive the executation of the program).

Keyword value pairs are persistent or ephemeral depending on whether or not they are mapped onto disk resident structures. The most common persistent keywords are the image, uv data, array and table descriptors which map onto AIPS catalog headers and table headers. Most other keyword value pairs are ephemeral.

Since the persistent keywords map onto disk resident structures they are not avalilable prior to the opening of the object. The exception to this is the case (common in tables) where some of these values must be set before the corresponding disk resident structures can be created. Opening the object forces any necessary file creation and synchronizes the disk and memory resident structures. Closing the object will update the disk resident structures with the current memory resident values.

Most of the persistent keywords are predefined so their usage is not different from other keywords. An exception to this case are the catalog header keywords which do not have a standarized usage. In this case routine OBVHKW (see the description of the object manager) must be used to assert that a given keyword is a catalog header keyword for the relevant class. This keyword then becomes a persistent keyword. Arrays and strings longer than 8 characters are not allowed as catalog header keywords.

6.4 Image Class

An image object consists of a pixel array object and a number of descriptive objects. These are dealt with as a single object by the object manager. Access to the pixel array can be either by pixel, row, plane or an entire image. Due to limitations of the AIPS I/O system subimages of a plane can be read but not written; thus to update a window in an image the entire plane must be read. Pixel values may be either real or complex although complex pixels are only allowed in temporary, scratch objects. Image class operators support real and complex images with possible blanking whenever possible.

A number of image operations have been defined and implemented. Image arithmetic is implemented in the image and array classes. CLEANing of images is also supported by routines in $APLOOP/CLEAN.FOR and $QOOP/QCLEAN.FOR. Images may be interpolated using routines in $QOOP/QINTER.FOR.

The image operators can return output image objects. However, the object must have been created (CREATE or IMGCRE) and had naming information (NAME, CLASS, IMSEQ and DISK) attached. The image operators will fill in the geometry and descriptive information.

The following is a sample code fragment which adds 1.0 to the elements the pixel array of a 2-D image object:

```
      INTEGER   TYPE, IERR, DIM(7), I, J, LROW, NROW, NAXIS(7), DUMMY
      CHARACTER IN*(*), OUT*36, CDUMMY*1
      REAL      ROW(4096)
      INCLUDE 'INCS:PAOOF.INC'

     ...

C                                       Shallow copy image for write
      OUT = 'Temporary'
      CALL OCOPY (IN, OUT, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Open images
      CALL OOPEN (IN, 'READ', IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OOPEN (OUT, 'WRIT', IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Row access (the default)
      DIM(1) = 8
      DIM(2) = 1
      CALL OPUT (IN, 'ARRAY.ARRAY_PNT.ACCESS', OOACAR, DIM, DUMMY,
     *   'ROW', IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OPUT (OUT, 'ARRAY.ARRAY_PNT.ACCESS', OOACAR, DIM, DUMMY,
     *   'ROW', IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Determine size
      CALL OGET (IN, 'ARRAY.ARRAY_DESC.NAXIS', TYPE, DIM, NAXIS,
```

```
     *   CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
      LROW = NAXIS(1)
      NROW = NAXIS(2)
C                                       Loop over image
      DO 100 J = 1,NROW
         CALL ARREAD (IN, DIM, ROW, IERR)
         IF (IERR.NE.0) GO TO 999
         DO 50 I = 1,LROW
            ROW(I) = ROW(I) + 1.0
 50         CONTINUE
         CALL ARRWRI (OUT, DIM, ROW, IERR)
         IF (IERR.NE.0) GO TO 999
 100     CONTINUE
C                                       Close arrays
      CALL ARRCLO (IN, IERR)
      IF (IERR.NE.0) GO TO 999
      CALL ARRCLO (OUT, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Close images
      CALL OCLOSE (IN, IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OCLOSE (OUT, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Destroy temporary object
C                                       (doesn't affect output file).
      CALL DESTRY (OUT, IERR)
      IF (IERR.NE.0) GO TO 999
```

6.5 Complex Image Class

A complex image class has been implemented as part of a Complex CLEAN task. A complex image consists consists of a pair of simple images (e.g. Q and U images). In this case, two image objects are members of a complex image object. Complex Images are the only kind of complex images allowed to permanently reside in AIPS and exists as a pair of AIPS cataloged images. For many purposes it is more useful to have images with complex pixels which are allowed only as temporary or scratch objects. There are utilities in the CXUTIL module to allow conversion between Complex Images and Images with complex pixels.

6.6 UVdata Class

The uv data class provides for both access to individual visibility records as well as a number of operations on uv data objects as a whole. When uv data are read, optional selection, calibration and editing can be applied. Most of the parameters controlling selection, calibration and editing can be passed directly from the inputs object to the uv data object and most routine setup is done automatically.

On write, the file is extended automatically when it is full. The basic assess routines for visibility data are UVREAD and UVWRIT. These routines will open the object if necessary but an explicit close is required. Compressed data files can be read but only uncompressed can be written at present. UVCOPY can be used to copy one uv data to another applying any calibration, editing and selection.

Since UVREAD uses the AIPS calibration package only one uvdata can be opened 'READ' at a time. Also as the AIPS calibration package uses many hardcoded logical unit numbers a uvdata object to be read should be opened before one to be written.

Visibility records are passed in the usual AIPS form except that the random parameters and the visibility array are separated. Routine UVDPNT will return the pointers etc. needed to interprete these data structures. See "Going AIPS" for more details.

The uv data class read access uses the standard AIPS calibration, selection and editing package, The associated features are controled by the virtual CALEDIT class from which the UVDATA calss is derived.

The default values for each uv data object are generally to accept all data with no calibration or translation of Stokes type. Calibration, selection and editing can be enabled by setting appropriate CALEDIT members in an object before it is opened. When a uvdata is open its UV_DESC members reflect the specified selection; e.g. UV_DESC.NAXIS gives the number of channels, IFs and Stokes' parameters that will be returned by UVREAD. If this UVDATA is then copied (OUVCOP, OUVSCR, etc.) then the derived object will have the geometry appropriate for the selected data.

A number of high level utilities are available. Image model subtraction and division are available as the "Q" routines ($QOOP/QUVUTIL.FOR) OUMSUB and OUMDIV. Uv data can be uniform weighted and imaged using the "Q" routines OUVDFT, OUNFWT and OUVIMG. CLEANing using uv data is also supported by routines in $QAPLOOP/CLEAN.FOR and $QOOP/QCLEAN.FOR. Uv data can be sorted and rotated using "Q" routine OUVSRT and time averaged using UVBAVG or UVTAVG. Automatic editing of data is facilitated using UVCLIP.

Self calibration of a uv data object that has been divided by a model may be done using routine SLFCAL in $APLOOP/UVUTIL. If coherence of polarization data is desirable then this should be followed by SLFREF. The resulting SN table can be copied to the undivided uv data using UVTCOP. This calibration can be applied by setting CALEDIT.CLUSE in the uv data object to be calibrated to the appropriave version of the SN table.

The following routine, SCNAVG, from $APLOOP/UVUTIL.FOR demonstrates the use of the UV data class. In this routine a selected set of visibility data is read and averaged in time and frequency.

```
      SUBROUTINE SCNAVG (NAME, SOURCE, QUAL, TIMER, MAXANT, VIS, IERR)
C-----------------------------------------------------------------------
C   Vector average a specified set of data.
C   Inputs:
C      NAME    C*?    UV data object name
C      SOURCE  C*16   Source name
C      QUAL    I      Qualifier.
C      TIMER   R(2)   Start and stop time in days.
C      MAXANT  I      dimension of VIS.
C   Outputs:
C      VIS     R(2,maxant,maxant)  First entry (1,?,?):
C                     Real part in upper half (1,i,j) i<j.
C                     Imaginary part in lower half.
C                     Second entry (2,?,?):
C                     Weight in upper part, count in lower
C      IERR    I      Error code, 0=> OK, 1= no data found. else error.
C-----------------------------------------------------------------------
      INTEGER   QUAL, MAXANT, IERR
      CHARACTER NAME*(*), SOURCE*16
      REAL      TIMER(2), VIS(2,MAXANT,MAXANT)
C
      INTEGER   LOOP, I1, I2, TYPE, DIM(7), LIMS(2,7), D(7), NAXIS(7),
     *   NDIM, ANT1, ANT2, INDXB, DUMMY
      CHARACTER CSOU(30)*16, CDUMMY*1
      REAL      TR(8), RP(20), VS(5000), VT(3)
      INCLUDE 'INCS:PAOOF.INC'
      INCLUDE 'INCS:DMSG.INC'
C-----------------------------------------------------------------------
C                                   Add selection criteria to NAME
C                                   Source name.
      CSOU(1) = SOURCE
      DO 30 LOOP = 2,30
         CSOU(LOOP) = '   '
 30      CONTINUE
      DIM(1) = 16
      DIM(2) = 30
      DIM(3) = 0
      CALL SECPUT (NAME, 'SOURCS', OOACAR, DIM, DUMMY, CSOU, IERR)
```

```
      IF (IERR.NE.0) GO TO 995
C                                       qualifier
      DIM(1) = 1
      DIM(2) = 1
      CALL SECPUT (NAME, 'SELQUA', OOAINT, DIM, QUAL, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 995
C                                       Timerange
      DIM(1) = 8
      CALL RFILL (8, 0, TR)
      TR(1) = TIMER(1)
      TR(5) = TIMER(2)
      CALL SECPUT (NAME, 'TIMRNG', OOARE, DIM, TR, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 995
C                                       Init output
      DO 20 I2 = 1,MAXANT
         DO 10 I1 = 1,MAXANT
            VIS(1,I1,I2) = 0.0
            VIS(2,I1,I2) = 0.0
 10         CONTINUE
 20      CONTINUE
C                                       Open object
      CALL OUVOPN (NAME, 'READ', IERR)
      IF (IERR.GT.0) GO TO 995
C                                       Get vis info
      CALL UVDGET (NAME, 'NDIM', TYPE, DIM, NDIM, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 995
      CALL UVDGET (NAME, 'NAXIS', TYPE, DIM, NAXIS, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 995
      CALL UVDFND (NAME, 1, 'BASELINE', INDXB, IERR)
      IF (IERR.NE.0) THEN
         MSGTXT = 'TROUBLE FINDING RANDOM PARAMETER BASELINE'
         GO TO 990
         END IF
      CALL FILL (5, 1, D)
      CALL FILL (10, 0, LIMS)
      DO 50 LOOP = 1,NDIM
         D(LOOP) = NAXIS(LOOP)
         LIMS(1,LOOP) = 1
         LIMS(2,LOOP) = NAXIS(LOOP)
 50      CONTINUE
C                                       Loop reading, averaging data
 100     CALL UVREAD (NAME, RP, VS, IERR)
         IF (IERR.GT.0) GO TO 995
C                                       Done?
         IF (IERR.LT.0) GO TO 500
C                                       Crack baseline
         ANT1 = (RP(INDXB) / 256.0) + 0.001
         ANT2 = (RP(INDXB) - ANT1 * 256) + 0.001
C                                       Ignore antennas with too large
C                                       values.
         IF ((ANT1.LE.0) .OR. (ANT1.GT.MAXANT)) GO TO 100
         IF ((ANT2.LE.0) .OR. (ANT2.GT.MAXANT)) GO TO 100
C                                       Average vis
         CALL AVGVIS (NDIM, D(2), D(3), D(4), D(5), LIMS, VS, VT)
C                                       Update accumulation
         IF (VT(3).GT.0.0) THEN
C                                       Could have auto correlations
            VIS(1,ANT2,ANT1) = VIS(1,ANT2,ANT1) + VT(2)
            VIS(1,ANT1,ANT2) = VIS(1,ANT1,ANT2) + VT(1)
```

```
           VIS(2,ANT1,ANT2) = VIS(2,ANT1,ANT2) + VT(3)
           VIS(2,ANT2,ANT1) = VIS(2,ANT2,ANT1) + 1.0
           END IF
C                                    Loop until done
        GO TO 100
C                                    Close
 500  CALL OUVCLO (NAME, IERR)
      IF (IERR.GT.0) GO TO 995
C                                    Normalize by element count
      DO 520 I1 = 1,MAXANT-1
         DO 510 I2 = I1,MAXANT
            IF (VIS(2,I2,I1) .GT. 0.5) THEN
               VIS(1,I1,I2) = VIS(1,I1,I2) / VIS(2,I2,I1)
               IF (I1.NE.I2) VIS(1,I2,I1) = VIS(1,I2,I1) / VIS(2,I2,I1)
               END IF
 510        CONTINUE
 520     CONTINUE
      GO TO 999
C                                    Error
 990  CALL MSGWRT (8)
 995  MSGTXT = 'ERROR AVERAGING SCAN FOR ' // NAME
      CALL MSGWRT (8)
C
 999  RETURN
      END
```

6.7 Inputs Class.

The interface to POPS has been cleaned up with the introduction of the inputs class. An inputs object contains the labeled POPS adverbs passed to the task. Several arrays describing the passed adverbs can be filled with DATA statements and passed to AV2INP which does the routine AIPS startup procedures and returns an inputs object. Class member function IN2OBJ will copy a selected list of members of an inputs object, with possible renaming, to an arbitrary object. Usage of the inputs object is demonstrated in each of the example tasks given as appendices.

As part of defining the POPS adverbs a numeric data type code is used. The parameter include PAOOF.INC contains symbolic names for the defined data types: OOAINT = integer, OOALOG = logical, OOARE = real, OOADP = double precision and OOACAR = character strings.

6.8 Table Class

A basic table class has been implemented with most of the features of AIPS tables available. The following fragment demonstrates the use of the tables class. In this fragment all character entries in a specified range of rows are blanked.

```
      INTEGER   TYPE, IERR, DIM(7), I, ROW, NCOL, NROW, BC, EC, DUMMY
      CHARACTER IN*(*), OUT*36, CDUMMY*1
C                                    MAXSIZ = max table entry size as
C                                    reals or characters.
      PARAMETER (MAXSIZ = 50)
      REAL      NVALS(MAXSIZ)
      CHARACTER CVALS*(MAXSIZ)
      INCLUDE 'INCS:PAOOF.INC'

      ...

C                                    Shallow copy table for write
      OUT = 'Shallow copy'
      CALL OCOPY (IN, OUT, IERR)
```

```
      IF (IERR.NE.0) GO TO 999
C                                       Open tables
      CALL OOPEN (IN, 'READ', IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OOPEN (OUT, 'WRIT', IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get number of entries
      CALL OGET (IN, 'NROW', TYPE, DIM, NROW, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get number of columns
      CALL OGET (IN, 'NCOL', TYPE, DIM, NCOL, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get range of rows from input
C                                       object.
      CALL OGET (IN, 'BCOUNT', TYPE, DIM, BC, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
      BC = MIN (MAX (BC, 1), NROW)
      CALL OGET (IN, 'ECOUNT', TYPE, DIM, EC, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
      IF (EC.LE.0) EC = NROW
C                                       Process table
      DO 100 ROW = BC,EC
         DO 50 I = 1,NCOL
            CALL TABDGT (IN, ROW, I, TYPE, DIM, NVALS, CVALS, IERR)
            IF (IERR.NE.0) GO TO 999
C                                       Blank any characters
            IF (TYPE.EQ.OOACAR) CVALS = ' '
C                                       Rewrite
            CALL TABDPT (OUT, ROW, I, TYPE, DIM, NVALS, CVALS, IERR)
            IF (IERR.NE.0) GO TO 999
 50         CONTINUE
 100     CONTINUE
C                                       Close tables
      CALL OCLOSE (IN, IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OCLOSE (OUT, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Destroy temporary object
C                                       (doesn't affect output file).
      CALL DESTRY (OUT, IERR)
      IF (IERR.NE.0) GO TO 999
```

6.9 CLEAN Process Class

A CLEAN object contains one or more image object and allows access to CLEAN deconvolution products. Only one CLEAN object is allowed to be opened at a time. CLEANs may be restarted from a previous result.

Two types of CLEAN are supported in this package: 1) the "Clark" CLEAN in which all components are subtracted using images and their FFTs and 2) the Cotton-Schwab CLEAN for which the components are subtracted from the ungridded visibility data. The method is indicated using CLEANTYP and the usage of these two methods differs somewhat both in the values attached to the clean object and the cleaning routines (CLNIM for "IMAGE" and CLNUV for "UV") used to perform the CLEAN.

For the Clark or "IMAGE" clean a dirty image must be provided. Since this method can work for only a single image only a single dirty image can be provided. Since the clean image is a simple derivative of the dirty image it is cloned from the dirty image by the CLEAN process if it does not previously exist.

The Cotton-Schwab or 'UV' method can CLEAN several fields from the same set of UV data. For this method both the UV data object and the CLEAN objects must be specified (size etc.) although the CLEAN object need not have it's underlying files created prior to opening the CLEAN object. Routine U2IDES (UVUTIL module) is useful for defining the image produced from a uvdata. Any calibration, editing, selection and conversion to Stokes' "I" should be done prior to calling CLNUV as well as any uniform weighting. These steps can be done with UV2SCR and OUNFWT. The UV data object is modified during the CLEAN process; the residual data is returned. If no CLEANing is specified (NITER =0) then only the image and beam are made.

The residuals can optionally be scaled to the same physical units as the restored components before the restoration is done. This is requested via SCALRES and uses the parameters BMSCLSZ to define the region of the beam in which to determine its area.

All CLEANs in this class support both rectangular as well as round CLEAN windows. A round window is indicated by a value of -1 for the first element of the window. The second is the radius of the window in pixels and the third and fourth are the "x" and "y" pixel of the center of the window.

6.10 Printer Class

Access to printers is through the Printer class. If the user specified value, DOCRT, is greater than zero, the output is redirected to the user's terminal and the user can request that the task quit at the end of each page.

Multiple printer objects may exist but only one may be open at a given time. If a uv data, table or image object ('PRINTOBJ') is specified, a standard header is written to output when the printer object is opened. After the printer object is open, two lines of running titles ('TITLE1', 'TITLE2') may be provided to give column headings, etc. Single lines are processed using PRTWRI. The print object must be closed to spool the output to the printer.

7 Relationship with AIPS System Structure

One of the drawbacks of the object oriented style is that a task that uses a class must contain in its executable all routines that might possibly be applied to that class whether they are used or not. The AIPS object oriented routines are kept in special directories; class libraries and utility modules which do not use "Q" routines are kept in the $APLOOP directory; those that do are kept in the $QOOP directory. The "Q" routines in these files do not need to and cannot access the private data of their class. Tasks using the AIPS object oriented routines are kept in the $APGOOP directory if they do not use "Q" routines and $QPGNOT if they do. Class libraries and utility are in files whose name is close, if not identical to, the class or function name. Class libraries should not use the "Q" routines whereas utility packages might. This keeps from having to put all tasks in $QPGOOP. At very least, none of the generic routines should use "Q" routines.

8 Examples

Examples of the use of these classes are shown in two AIPS tasks which are given as appendices. The first, IMCOP, copies one object to another allowing subimaging. The second task, TBTSK, is a paraform task for simple table operations. The example given will copy a range of rows from one table to a similar table. TBTSK is included in the standard distribution. Other examples of use are given in the section discussing the class interfaces.

9 Discussion

The system presented here has a number of advantages over the more traditional AIPS. Encapsulation of data is very effective and much of the programming overhead of AIPS I/O, catalog manipulation etc. is eliminated. Attaching the control information directly to objects virtually eliminates the need for task specific commons as is used in AIPS. This makes it much simpler to write reuseable software. Many of the routines in the example tasks given in the appendices could be reused in other applications without change since they have no task specific connections (i.e. control information passed through a task specific common). In AIPS, control parameters are frequently passed between tasks and packages of related routines through commons. A change in one of the widely used commons frequently creates other problems. Passing control information directly attached to objects should signifigantly reduce the maintance costs. The ability to pass control information attached to objects is probably more important than the more common aspects of object-oriented programming except possibly the encapsulation of knowledge about data structures.

There are still some weaknesses in this system which need to be considered. In Fortran 77 there is no concept of classes so any dynamic binding must be done explicitly. This means that when new classes are added the relevant generic interface routines must also be modified. In Fortran 77 overloading of operators is not supported so all object operations must be explicitly written as routine calls which is less readable than the more algebraic notation allowed in some OO languages.

The error handling in this system is rather primitive. In some sections of the code half of the executable statments are error checks making the logic more difficult to follow.

The implementation of objects in this system allows only a relatively small number of large objects. However, the advantages of a large number of small objects (i.e. the individual pixels in an image array) are not readily apparent and would probably impose a severe performance penalty in any implementation.

10 Description of System Modules

The following sections describe the major modules of the system especially the object manager, the history utilities and the various classes. These descriptions include the public interface as well as the internal structures and functions. More complete details of the usage of the relevant routines are given in the precursor comments in the source code.

10.1 System includes OBJPARM.INC and CLASSIO.INC

There are two important system wide includes. The first is OBJPARM.INC which defines via PARAMETER statments various system wide parameters such as the maximum number of objects extant simultaneously. The text of this include follows:

```
C                                                Include OBJPARM.
C                                        System Parameter Include for
C                                        AIPS Object Oriented Fortran
C                                        system.
C                                         Parameter include for Objects
C                                         and Class I/O
      INTEGER   MAXIO, MAXCLS, MAXVKW
C                                         MAXIO=max. number of I/O
C                                         streams.
      PARAMETER (MAXIO = 5)
C                                         MAXCLS = the number of
C                                         classes defined.
      PARAMETER (MAXCLS = 10)
C                                         MAXVKW = Max number of
C                                         virtual keywords per class.
      PARAMETER (MAXVKW = 50)
C                                                End OBJPARM.
```

The other system include is CLASSIO.INC which contains class I/O buffers and other info. Note: this include includes OBJPARM.INC. The text follows:

```
C                                                Include CLASSIO.
C                                        Class I/O include for
C                                        AIPS Object Oriented Fortran
C                                        system.
      INCLUDE 'INCS:OBJPARM.INC'
```

```
      INCLUDE 'INCS:ZPBUFSZ.INC'
      INTEGER   BUFSIZ, HBFSZ
C                                         BUFSIZ=Size of I/O buffer
      PARAMETER (BUFSIZ = UVBFSS)
C                                        HBFSZ = size of HBUFF
      PARAMETER (HBFSZ = 1024)
      INTEGER   OBJLUN(MAXIO), OBJFIN(MAXIO), BUFPNT(MAXIO)
      LOGICAL   OBNUSE(MAXIO), LUNUSE(100)
      REAL      OBUFFR(BUFSIZ,MAXIO), SBUFF(512)
      HOLLERITH HBUFF(HBFSZ)
      CHARACTER BUFNAM(MAXIO)*32
      COMMON /OBJBUF/ OBJLUN, OBJFIN, BUFPNT, OBNUSE, LUNUSE, OBUFFR,
     *   SBUFF, HBUFF
      COMMON /OBCBUF/ BUFNAM
C                                                        End CLASSIO.
```

10.2 Object manager (Fortran)

This module is kept in $APLOOP/OBJMNGR.FOR.

This version of the Object manager uses c language routines to allocate and manage the linked list and CATBLK storage for each object.

```
     MAXCLS   PI           Number of classes defined.
     MAXVKW   PI           Maximum number of virtual keywords.
     VKWCLS   H*8(MAXCLS)  Class name for virtual keyword.
     VKWNUM   I(MAXCLS)    Number of defined virtual keywords.
     VKWTAB   I(7,MAXVKW,MAXCLS) Virtual keyword information array.
                           The second dimension is per keyword and the
                           third is per class.  The structure of the
                           first dimension is:
          Word  Type        Contents
          ----  ----  -------------------------------------------
           1     H     Keyword
           3     I     Category
           4     I     Pointer
           5     I     Data type
           6     I     Dim(1)
           7     I     Dim(2)
    Where
    keyword:   name of the virtual keyword as HOLLERITH (2 words = 8
               char)
    category:  1 = in fixed portion of catalog header, pointer is
               pointer into type dependent array.  D values must be
               copied from R array
               2 = in keyword/value portion of catalog header, some
               restrictions apply (not more than 2 words of data).
               3 = Special derived keywords read access only.  Pointer
               specifies a class specific function.
     pointer:  pointer to catalog header entry or function.
     data type: 1,2,3,4,5 for D, R, C, I, L data types of associated
               data.
     dim       Dimensionality of value, an axis dimension of zero
               means that that dimension and higher are undefined.  For
               character strings the length of the string is the first
               dimension.

     VKTKEY   PI                VKWTAB col. number for keyword
     VKTCAT   PI                VKWTAB col. number for category
```

```
  VKTPNT   PI                VKWTAB col. number for pointer
  VKTYPE   PI                VKWTAB col. number for data type
  VKTDM1   PI                VKWTAB col. number for dimension 1
  VKTDM2   PI                VKWTAB col. number for dimension 2

Shared data with Class I/O (CLASSIO.INC):
   MAXIO   PI              Maximum number of I/O simultaneous
                           streams
   BUFSIZ  PI              Buffer size in words
   OBUFFR  R(BUFSIZ,MAXIO) I/O buffers
   BUFPNT  I(MAXIO)        Buffer pointer
   OBJLUN  I(MAXOBJ)       LUNs for I/O
   OBJFIN  I(MAXOBJ)       FTAB pointer for I/O
   LUNUSE  L(100)          True if LUN=index allocated.

Public functions:
   OBINIT (ierr)
      Initialize Object manager.
   OBVHKW (class, keyword, type, ierr)
      Add a Catalog header keyword to the virtual keyword list.

Private functions:
  INVINI (ierr)
     Initialize virtual keywords for inputs class.
  IMVINI (ierr)
     Initialize virtual keywords for image class.
  OBKEYV (objnum, keywrd, keypnt, ierr)
     See if keyword is an object dependent, virtual keyword.
  OBRGET (objnum, keywrd, type, dim, value, valuec, ierr)
     Fetch the value (array) for a specified real (non-virtual)
     keyword.

Shared functions with class modules:
  OBCREA (name, class, ierr)
     Associate an object slot with an object name.
  OBFREE (name, ierr)
     Free the object slot associated with an object.
  OBNAME (name, objnum, ierr)
     Look up the object slot number of object with name "name".
  OBCLAS (obnum, clasno, name, ierr)
     Look up the class number of object number objnum.
  OBPUT (objnum, keywrd, type, dim, value, valuec, ierr)
     Save an entry in an object creating it if necessary.
  OBGET (objnum, keywrd, type, dim, value, valuec, ierr)
     Fetch the value (array) for a specified keyword.
  OBLUN (lun, ierr)
     Find a free LUN.
  OBLUFR (lun)
     Releases an LUN
  OBINFO (name, bufno, ierr)
     Look up I/O stream associated with an object.
  OBDSKC (name, disk, cno, ierr)
     Return Disk and slot information for object.
  OBHGET (name, cat, ierr)
     Return catalog header record for an object.
  OBHPUT (name, cat, ierr)
     Store catalog header record for an object.
  OBCOPY (namein, namout, ierr)
     Copies keywords from one object to another.
```

```
OBOPEN (name, ierr)
   Assigns a buffer.
OBCLOS (name, ierr)
   Closes a buffer associated with an object.
OBC2H (string, dim1, dim2, holl)
   Convert a string (possibly an array) to hollerith
OBH2C (holl, dim1, dim2, string)
   Convert a hollerith (possibly an array) to a string.
```

10.3 Object manager (c)

This module is kept in $APLGEN/ZOINTD.C.

The basic memory managment routines for AIPS objects is done using a number of "c" language "Z" routines which can dynamically allocate and deallocate memory. The data structures consist of an object directory and zero or more objects each of which contains a name, class, CATBLK and a linked list of arbitrary information. These routines should ONLY be called from Object manager routines (in $APLOOP/OBJMNGR.FOR). The following describes the structures and interfaces of these routines.

```
/*   Define the maximum number of objects                          */
#define MAXOBJ  1000

/* Linked list element Structure                                   */
/* The memory allocated for this structure will include a data area   */
/* after the end of the structure.                                 */
struct ll_element {
    char          keyword[8];   /* Keyword name 8 char  */
    struct ll_element *next;    /* Pointer to next element, void=> last */
    int           dataoff;      /* offset from struct to beginning of
                                   data array in bytes*/
    int           lendata;      /* Number of bytes in data array */
    int           type;         /* Data type code */
    int           ndim;         /* number of dimensions */
    int           dim[7];       /* dimensionality array */
                                /* data array follows dim[ndim-1]*/
};

/* AIPS object Structure                                           */
struct AIPS_object {
    char  name[32];             /* Object name */
    char  class[8];             /* Object class */
    int   catblk[256];          /* AIPS catalog header */
    struct ll_element *first;          /* First element of linked list */
    struct ll_element *last;           /* Last element of linked list */
    struct ll_element *select;         /* Selected element of linked list */
};


/* Object directory:                                               */
struct object_dir {
 struct AIPS_object *objpnt[MAXOBJ]; /* list of pointers to AIPS-Objects */
 int       numobj;               /* max number in objpnt assigned */
 char last_obj[32][3];           /* names of last several objects searched */
 int  last_num[3];               /* numbers of last few objects */
} task_dir;                      /* define task_dir for this task */

   Function Prototypes  and descriptions

Fortran callable:
```

```
void zointd_ ();
   Initialize AIPS object directory
void zocrob_ (char *name, char *class, int *objnum, int *ierr);
   Create/initialize object structure
void zodeob_ (int *objnum);
   Destroy object structure
void zofnob_ (char *name, int *objnum);
   Look up object
void zocpob_ (char *namein, char *nameout, int *ierr);
   Copy object
void zofnle_ (int *objnum, char *keyword, int *ierr);
   Look up keyword in object linked list; sets selected linked list
   element pointer.
void zoinle_ (int *objnum, char *keyword, int *type, int *ndim,
              int *dim, int *ierr);
   Lookup information about a linked list element
void zostdt_ (int *objnum, char *keyword, int *type, int *ndim,
              int *dim, char *data, int *ierr);
   Store data in AIPS Object linked list;  checks compatibility.
void zofedt_ (int *objnum, char *keyword, int *type, int *ndim,
              int *dim, char *data, int *ierr);
   Fetch data from AIPS Object linked list.
void zostct_ (int *objnum, int *catblk, int *ierr);
   Store CATBLK to AIPS Object.
void zofect_ (int *objnum, int *catblk, int *ierr);
   Fetch CATBLK from AIPS Object.
void zofenm_ (int *objnum, char *name, char *class, int *ierr);
   Fetch Object name and class from AIPS Object.

   c callable only:
struct ll_element* create_llelem_ (char *keyword, int *type,
                                   int *ndim, int *dim, int *ierr);
   Create/initialize AIPS Object linked list element
```

10.4 History

This module is kept in $APLOOP/HISTORY.FOR.

This module contains routines which can he used to manipulate histories associated with objects that are disk resident. Available functions are:

```
   OHCOPY (in, out, iret)
      Copies the history from object in the object out.  iret=0
      indicates success.
   OHWRIT (entry, out, iret)
      Writes up to 72 characters from character string entry to the
      history associated with object out.  iret=0 indicates success.
   OHLIST (in, list, nlist, out, iret)
      Writes the names and values of the members of object in specified
      by the first nlist elements in the character string array list to
      the history associated with object out.  iret=0 indicates
      success.
   OHTIME (out, iret)
      Adds task name and time and date stamp to history associated with
      object out.  iret=0 indicates success.  Assures that a history
      file exists.
```

10.5 Array Class

This module is kept in $APLOOP/ARRAY.FOR.

Array Class: name "ARRAY"

The ARRAY class is a virtual base class and as such cannot be instantiated. Arrays are created as part of an IMAGE object.

An array is a regular array of values with several descriptive base classes. General access to members is through ARRGET and ARRPUT although efficient access to the array may be had through ARREAD and ARRWRI. Access may be by element, row, plane or image. Arrays may contain either real or complex data type elements. Blanking is also allowed.

```
Class Data:
   array   R(*,*...)  Array of Pixel values.  May be memory or disk
                      resident.
   ARRAY_PNT  class   Array access pointer
   ARRAY_DESC class   Array description
   ARRAY_STAT class   Array statistical information

Private class data:
   ARRFDV     I(*,MAXIO) Internal array for I/O routine
   ARRTYP     I(MAXIO)   Access type: 1=pixel, 2=row, 3=plane,
                         4=array.
   ARRPT      I(MAXIO)   Row element pointer
   ARRDIM     I(7,MAXIO) Dimension of window in array

Public functions:
   ARRGET (name, keywrd, type, dim, value, valuec, ierr)
      Return array subarray.  Access may be by pixel, row, plane or
      array as determined from ARRAY_DESC.
   ARRPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store array subarray.  Access may be by pixel, row, plane or
      array as determined from ARRAY_DESC.
   ARREAD (name, dim, data, ierr)
      Read a section of an array
   ARRWRI (name, dim, data, ierr)
      Write a section of an array
   ARRCLO (name, ierr)
      Close I/O to an array
   ARROPN (name, status, ierr)
      Setup for I/O to an array
   ARSSET (name, ierr)
      Determines array statistics and updates ARRAY_STAT
   ARHIST (name, nhis, hmax, hmin, hist, ierr)
      Return histogram of array.
   CHKBLK (n, data, valid)
      Checks an array for blanking and returns a validity array
   SETBLK (n, valid, data)
      Blanks an array on the basis of a validity array.
   ARRNEG (in, out, ierr)
      Negate the values of an input array and write an output
      array.
   ARRFFT (dir, in, scr, out, ierr)
      FFT an array.
   ARRADD (in1, in2, out, ierr)
      Adds two arrays.
   ARRSUB (in1, in2, out, ierr)
      Subtracts array in2 from array in1.
   ARRMUL (in1, in2, out, ierr)
      Multiplies two arrays.
```

```
   ARRDIV (in1, in2, out, ierr)
      Divides array in1 by array in2.
   ARRPAD (in, out, ierr)
      Copy one array to another with zero padding.
   ARRSCL (in, factor, out, ierr)
      Scale an array with a real factor.
   ARRCOP (in, out, ierr)
      Copy the element of an array
   ARRFIL (scalar, out, ierr)
      Fill an array with a real scalar
   ARRSAD (in, scalar, out, ierr)
      Add a scalar to the elements of an array.
   ARRSMU (in, scalar, out, ierr)
      MUltiply a scalar times the elements of an array.
   ARRCWI (name, ierr)
      Clears the window (BLC, TRC) for first two dimensions of an
      array object.
Private Function:
   ARRCHK (in1, in2, ierr)
      Check that two arrays are compatible.
   ARRWIN (name, blc, trc, dim, ierr)
      Returns window information about an array
   ARRIO (opcode, name, fdvec, ibuff, ipnt, ierr)
      Handles I/O to disk resident array
   ARRMEM (????)  Not yet implemented
      Handles access to memory resident array.
```

10.6 Array descriptor Class

This module is kept in $APLOOP/ARRAYDESC.FOR.

Array descriptor class: name = "ARRAY_DESC"

The array descriptor contains information about the dimensionality and data type of an array.

```
Class data:
   NDIM     I       Number of dimensions in the array
   NAXIS    I(*)    Dimension of each axis
   TRC      I(*)    Top right corner of subimage, 0's=>all
   BLC      I(*)    Bottom left corner of subimage, 0's=>all
   DATATYPE C*8     Element type, 'REAL', 'COMPLEX'
   ANAME    C*8     Name of array if memory resident
   FNAME    C*48?   Physical name of array file if disk resident.
   FDISK    I       Disk number for FNAME.
   BLANK    R       If 0.0 or absent then the array has no blanking.
Public functions:
   ARDGET (name, keywrd, type, dim, value, valuec, ierr)
      Return array descriptor member.
   ARDPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store array descriptor member.
   ARDCOP (namein, namout, ierr)
      Copy array descriptor info from namein to namout.
   ARDSCP (namein, namout, ierr)
      Copy array descriptor info not related to object size.
```

10.7 Array Pointer class

This module is kept in $APLOOP/ARRAYPNT.FOR.

Array pointer Class: name "ARRAY_PNT"

The array pointer class keeps track of the access type and the location in an array of the last access to support sequential access of an array.

```
Class Data:
   POSN    I(7)    Element number on each axis of first element of
                   last read or next write.
   ACCESS  C*8     Array access type: 'PIXEL', 'ROW', 'PLANE',
                   'ARRAY'
Public functions:
   ARPGET (name, keywrd, type, dim, value, valuec, ierr)
      Return array pointer.
   ARPPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store array pointer.
```

10.8 Array Statistics Class

This module is kept in $APLOOP/ARRAYSTAT.FOR.

Array Statistics class: name="ARRAY_STAT"

The array statistics class provides various statistical data about an array. The member values are best set using ARRAY class function ARSSET.

```
Class data:
   DATAMAX  R    Maximum value
   PIXMAX   I(*) Position of maximum value (1 per actual axis)
   DATAMIN  R    Mininum value
   PIXMIN   I(*) Position of minimum value (1 per actual axis)
   DATARMS  R    RMS value
   DATAMEAN R    Mean value
   NUMPIXEL I    Number of pixels examined.
Public functions:
   ARSGET (name, keywrd, type, dim, value, valuec, ierr)
      Return array statistics member.
   ARSPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store array statistics member.
```

10.9 Clean Process Class

This module is kept in $APLOOP/CLEAN.FOR.

CLEAN Class: name = "CLEAN"

This class performs a CLEAN deconvolution of images.

Note: Most of the useful functions of the class are in module $QOOP/QCLEAN.FOR as they use the "Array Processor".

Usage notes:

1) Before use a CLEAN object must be "opened" using CLNOPN. Only one CLEAN object is allowed to be "open" at a time.

2) Complete CLEANs can be done using CLNIM for Image CLEANs and CLNUV for UV data CLEANs. Multiple channels require separate calls to CLNIM or CLNUV but don't require separate opens and closes.

3) A CLEAN object must be closed (CLNCLO) before another can be opened. Closing the CLEAN object causes any scratch files used in the CLEAN to be destroyed.

4) Control parameters needed for imaging the uvdata in the "UV" clean are attached to the first CLEAN and BEAM image object. Any uniform weighting should be applied to the UVDATA before CLEANing.

```
 Inputs attached to CLEANI(1): (defaulted where approproate)
   FTTYPE    C*4   Fourier transform type 'FFT' or 'DFT'. ('FFT')
   IMSIZE    I(2,*) Image size per field (no default)
   CELLSIZE  R(2)  Cellsize in arcseconds in X and Y (no default)
   CHTYPE    C*4   'LINE',  or 'SUM ' for imaging ('SUM')
   SHIFT     R(2)  Shift in arcsec (DFT imaging)
   RASHIFT   R(*)  X position shift in arcseconds per field (0) FFT
   DECSHIFT  R(*)  Y position shift in arcseconds per field (0) FFT
   CENTERX   I(*)  Center X pixel position per field (std default)
```

```
  CENTERY   I(*)  Center Y pixel position per field (std default)
  CTYPX     I     X convolving function type (std default)
  XPARM     R(10) X convolving function parameters( std default)
  CTYPY     I     Y convolving function type (std default)
  YPARM     R(10) Y convolving function parameters (std default)
  DOZERO    L     IF true do Zero spacing flux (do if value given)
  ZEROSP    R(5)  Zero spacing parameters (no zero spacing flux)
  TFLUXG    R     Total flux to be subtracted from ZEROSP (0.0)
  DOTAPER   L     If true taper (do if non zero taper given)
  UVTAPER   R(2)  X and Y taper values (no taper)
Inputs attached to DIRTBEAM:
  IMSIZE    I(2)  Size of beam (no default)
Inputs attached to UVDATA (uv plane CLEAN only):
  MODMETH   C*4   Model method 'GRID', 'DFT ', '    '=> chose
                  Defaults to '    '.
  STOKES    C*4   Desired Stokes parameter (I)
  UVRANGE   R(2)  UV range in kilo wavelengths (all)

Class public members:
The following must be specified before opening the object:
  CLEANTYP    C*8          'IMAGE' or 'UV', ' ' => Image
The following are for uv plane cleans only:
  NIMAGE      I            Number of associated images (called
                           fields in class)
  UVDATA      C*32         UV data, contents will be current
                           residual data.
  UVCHAN      I            First channel number in uv data to
                           process.
  NCHAV       I            Number of channels to average.
The following are for image plane cleans only:
  DIRTYI      C(*)*32      Names of associated dirty images =
                           objects of type 'IMAGE', Should have
                           OUTNAME, OUTCLASS, OUTSEQ, OUTDISK set
                           to specify CLEANI
The following are for all CLEANs:
  CHANNEL     I            Selected frequency channel. (default=1)
  VERSION     I            CC file version number = CHANNEL for
                           spectral data. (default = new)
  DIRTBEAM    C*32         Name of dirty beam 'IMAGE' object.
  CLEANI      C(*)*32      Names of associated clean images =
                           objects of type 'IMAGE', May be either
                           residual or restored.
  BEAM        Base class   Beam object (default = fit)
                           NOTE: Sizes are in degrees NOT asec.
  NITER       I            Total number of iterations.
  BCOMP       I(*)         Restart using BCOMP components per
                           field. (default = 0)
  GAIN        R            Loop gain (default=0.2)
  PHAT        R            Prussian helment spike size (default=0)
  MINPATCH    I            Minimum BEAM-Half Width
  MAXPATCH    I            Maximum BEAM-Half Width
  MINFLUX     R            Min. residual flux. (default = 0)
  FACTOR      R            CLEAN depth factor. (default = 0)
  NORESTOR    L            If .true. don't restore components to
                           CLEANI. (default = .false.)
  DOBEAM      L            If true make a new dirty beam when
                           starting a CLEAN (default = .true.)
  NBOXES      I(*)         Number of windows (max. 10) per field.
  WINDOW      I(4,10,*)    CLEAN windows, one per IMAGE (default =
```

```
                           1 window per field excluding outer 5
                           pixels).
                           WINDOW(1,*)=-1 indicates a round box of
                           width WINDOW(2,*) pixels centered on
                           pixel (WINDOW(3,*), WINDOW(4,*))
   SCALERES    L           If true scale residuals by ratio of
                           dirty to restoring beam areaa (false)
   BMSCLSZ     I(2)        Halfwidth in x and y of box around beam
                           center to be used to determine dirty
                           beam area. (5,5)
   MAXNRES     I           Maximum number of residuals considered
                           each CLEAN cycle (20,000)
The following are available only after CLEANing
   NCOMP       I(*)        Number of components per IMAGE
   TFLUX       R           Total flux
   FLUX        R(*)        Flux for each field
   GRID        C*32        Name of grid object for image plane
                           CLEAN.
   TRANFN      C*32        Name of transfer function (FT of beam)
                           for image plane CLEAN.
   WORK1       C*32        Name of scratch image object
   WORK2       C*32        Name of scratch image object

Class  private data:
   ACTIVE      L        If true a CLEAN object is active. (open)
   MFIELD      I        Number of fields
   CTYPE       C*8      CLEAN type 'IMAGE' or 'UV'
   UVDATA      C*32     Name of UV data object
   DNAME       C(*)*32  Names of Dirty images
   CNAME       C(*)*32  Names of residual/CLEAN images
   DBNAME      C*32     Name of dirty beam image.

Public functions:
  CLNCRE (name, ierr)
     Creates an uv data set object with name "name".
  CLNDES (name, ierr)
     Destroys the uv data set object with name "name";
     quasi-permanent  forms are unaffected.
  CLNZAP (name, ierr)
     Destroys the uv data set object with name "name";
     quasi-permanent forms (CLEAN images only) are deleted.
  CLNOPN (name, status, ierr)
     Opens an uv data set object.  Checks for valid data.
  CLNCLO (name, ierr)
     Closes an uv data set object. Destroy scratch files.
  CLNGET (name, keywrd, type, dim, value, valuec, ierr)
     Return keyword value.
  CLNPUT (name, keywrd, type, dim, value, valuec, ierr)
     Store keyword value.

Private functions:
   CLBGET (name, keywrd, type, dim, value, valuec, ierr)
      Fetches member of a base class of uv data set class
   CLBPUT (name, keywrd, type, dim, value, valuec, ierr)
      Stores member of a base class of uv data set class
```

10.10 Clean Process "Q" Routines

This module is kept in $QOOP/QCLEAN.FOR.

The "Q" routine utility module contains most of the functional parts of the CLEAN class. Since these use "Q" routines they cannot be in the main CLEAN class module. See the discussion of the CLEAN process class for more details.

```
Module private data:
   ACTIVE      L         CLEAN common active
   MFIELD      I         Number of fields (max. 1 for IMAGE CLEAN)
   CNAME       C(*)*32   Names of residual/CLEAN images
   DBNAME      C*32      Name of dirty beam image.
   WORK1       C*32      Scratch image
   WORK2       C*32      Scratch image
   NCLNG       I(*)      Number of components per field.
   NSUBG       I(*)      Number of components subtracted per field.
   CCDISK      I(*)      Disk numbers of the fields.
   CCCNO       I(*)      Catalog slot numbers of the fields.
   CCVER       I(*)      Version numbers of the fields.
   CELLSG      R(2)      Grid increment in RA, Dec (asec)
   IMSIZE      I(2,*)    Image size of each field in pixels.
   ICNTRX      I(*)      Image center X pixel per field
   ICNTRY      I(*)      Image center Y pixel per field
   XPOFF       R(*)      X ref. pixel offset from center shift per
                         field (deg)
   YPOFF       R(*)      Y ref. pixel offset from center shift per
                         field (deg)
   MAPROT      R         Coordinate rotation (deg)
   NEWBEM      L         If true make a new beam when starting a
                         CLEAN.
   NXBEM       I         Number of cells in "x" dimension of beam
   NYBEM       I         Number of cells in "y" dimension of beam
   NBMHIS      I         Number of levels in beam histogram
   BEMMAX      R         Maximum value in beam (center value).
   BMHIS       I(516)    An array whose elements have values between
                         1 and NBMHIS + 1 indicating the maximum abs.
                         exterior sidelobe for a beam patch whose
                         size corresponds to the array index.
                         e.g. for a beam patch of half size I the
                         maximum fractional absolute sidelobe level
                         exterior to the  beam patch is BMHIS(I) /
                         NBMHIS
   BMAJ        R         Major axis size (FWHP in sec).
   BMIN        R         Minor axis size (FWHP in sec).
   BPA         R         Position angle of major axis (degrees).
   GAUSAA      R         Coefficient of u**2
   GAUSBB      R         Coefficient of u*v
   GAUSCC      R         Coefficient of v**2
   NBOXES      I(*)      Number of boxes given for field
   WIN         I(4,1,*0)  Boxes for fields
   RESHIS      R(1024)   The histogram of the distribution of pixel
                         values.
   NRSBIN      I         Number of bins in RESHIS
   RESMAX      R         Max. abs. residual map value.
   MAXRES      I         Max. number of residuals loaded.
   MAXPCH      I         Maximum beam patch size allowed.
   MINPCH      I         Minimum beam patch size allowed.
   PATCH       I         Beam patch size (max. distance from the
                         center)
   MAPLIM      R         Minimum abs. map level to be considered.
                         Only values GREATER than MAPLIM should be
                         used.
```

```
   NRESKP      I        Number of residuals to skip  when there are
                        too many to fit.
   APRESD      I        first location of the residuals.
   APCFLD(*)   I        start addresses of residuals for each field
   APCLCN(*)   I        element count for each field.
   GAIN        R        Clean loop gain.
   PHAT        R        Prussian helmet spike size (default=0)
   MINFLX      R        Min. residual flux. (default = 0)
   NOREST      L        If .true. don't restore components to
   FACTOR      R        Clean speed up factor.
   TFLUXG      R        Total CLEAN flux
   FLUXG       R(*)     CLEAN flux for each field.
   RESNUM      I        Number of residuals loaded.
   CLNLIM      I        Maximum number of clean components desired.
   FINISH      L        .TRUE. If minimum clean component flux
                        or maximum iteration encountered, otherwise
                        .FALSE.
   FSTCLN      L        Set to false to indicate this is not the
                        first clean cycle on this channel.
   CHANN       I        Image Channel number to process.
   APBEAM      I        Start location for beam patch
   GRID        C*32     Name of grid object for image plane CLEAN.
   DOSCAL      L        If true then scale residuals
   BMSSZ       I(2)     Half widths of box to determine beam area.
Used for IMAGE CLEAN only
   DNAME       C(*)*32  Names of Dirty images
   WFIRST      L        If true GRID is uninitialised
   TRANFN      C*32     Name of transfer function (FT of beam) for
                        image plane CLEAN.
Used for UV CLEAN only:
   UVDATA      C*32     residual UV data object.
   UVCHAN      I        First channel number in uv data to
                        process.
   NCHAV       I        Number of channels to average.

Public functions:
  CLNINI (name, ctype, ierr)
     Initializes CLEAN.
  CLNIM (name, ierr)
     Full image plane Clark CLEAN.
  CLNUV (name, ierr)
     Full ungridded uv plane Cotton-Schwab CLEAN.
  CLNCYC (name, ierr)
     One major cycle of B. Clark CLEAN.

Private functions:
   CLBHIS (iret)
      Determines the largest exterior sidelobe as a function of the
      beam patch size.
   CLRHIS (ifield, iret)
      Determines the histogram of pixel values.
   CDECID
      Determines a beam patch and limiting map value (PATCH and
      MAPLIM) which will optimize the use of memory.
   CLBSHV (iret)
      Loads the beam patch into memory.
   CLMPAC (iret)
      Takes points from the residual map greater than MAPLIM and
      places them with their addresses into memory
```

```
CLREST (iret)
   Convolves components with restoring beam and adds to
   residuals.
CLRGRD (ifield, iret)
   FT components and multiply by FT of restoring beam.
CLCCRM (ifield, dosum, dosub, aplo, apbuf, first, number, nload,
   nx, ny, iret)
   Creates structures in memory for the FT of CLEAN components.
CLGFIT (iret)
   Fits a Gaussian to the inner portion of a dirty beam
CLACLN (iret)
   Does the minor cycles for a single pass of Clark CLEAN.
CLGAUS (ifield)
   Initializes structures in memory for convolving Gaussians.
CLCSUM
   Sums flux densities in CC files.
CLGRID (ierr)
   Accumulates FT of components times the negative of the
   transfer function.
CLRMAX (nfield, cname, absmax, ierr)
   Find the maximum absolute pixel value of a set of images.
CLIINF (ierr)
   Update and check image info in common.
```

10.11 Convolving Beam Class

This module is kept in $APLOOP/BEAM.FOR.

Beam Class: name = "BEAM"

This class contains information about CLEANing that has been done and the restoring beam size.

```
Class Data:
   PRODUCT   I   Clean product code.
   NITER     I   Number of Clean components or MEM iterations
   BMAJ      R   Clean beam major axis (deg)
   BMIN      R   Clean beam minor axis (deg)
   BPA       R   Clean beam position angle (deg)
Public functions:
   BEMGET (name, keywrd, type, dim, value, valuec, ierr)
      Return beam member.
   BEMPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store beam member.
```

10.12 Complex Image Class

This module is kept in $APLOOP/CXIMAGE.FOR.

Complex Image Class: name "CX_IMAGE"

A complex image consists of a pair of real images. The real and imaginary parts of the object are accessed by prepending 'REAL.' or 'IMAG.' to the element name e.g. "REAL.ARRAY.ARRAY_DESC.NAXIS" is the name of the dimension array for the real image. Access to rows of the complex image are to member 'ARRAY' although CIGETX and CIPUTX allow more efficient access.

```
Class public members:
  ARRAY     CX(*,*,*)      Array(s) of complex pixels
  REAL      image object   Real part of the complex image
  IMAG      image object   Imaginary part of the complex image
  REALPART  C*32           Name of REAL
  IMAGPART  C*32           Name of IMAG
```

```
Class private members:
  CXIOBJ  C(2,MAXOBJ)*32 Name of the real and imaginary member
                         objects as a function of the complex image
                         object number.
  CXIOBN   I(2,MAXOBJ)   Object numbers of the real and imaginary
                         member objects
  CXIDIM   I(7,MAXIO)    Array dimension per I/O stream
  IOACTV   L(MAXOBJ)     If true image I/O is active

Public functions:
  CIMCRE (name, real, imag, iret)
     Creates a complex image object with name "name" and whose real
     and imaginary components are "real" and "imag".
  CIMDES (name, ierr)
     Destroys the image object with name "name"; quasi-permanent
     forms are unaffected.
  CIMZAP (name, ierr)
     Destroys the image object with name "name"; quasi-permanent
     forms are deleted.
  CIMCOP (namein, namout, ierr)
     Copys one object to another.  The same quasi permanent forms
     are used for both.
  CIMCLN (namein, namout, ierr)
     CLONES an object.  The component parts are cloned.
  CIGET (name, keywrd, type, dim, value, valuec, ierr)
     Return keyword value.
  CIPUT (name, keywrd, type, dim, value, valuec, ierr)
     Store keyword value.
  CIMOPN (name, status, ierr)
     Opens a complex image for array access
  CIMCLO (name, ierr)
     Closes a complex image for array access
  CIGETX (name, dim, row, ierr)
     Return complex row.
  CIPUTX (name, dim, row, ierr)
     Stores complex row.

Private functions:
  GETBAS (name, keywrd, base, object, mem, ierr)
     Checks base class reference and returns relevant information.
  CHKBAS (type, bascls, name, ierr)
      Checks that data type is valid for the base class.
  CIMIO (name, dim, data, ierr)
     Fetches or stores a row of a complex image.
```

10.13 CX_IMAGE Utility module

UV data utility library: $APLOOP/CXUTIL.FOR

This package of utility routines is related to the complex image (CX_IMAGE) class. These mostly convert between CX_IMAGE objects (the only type of complex image allowed permanently in AIPS) and IMAGE objects of type complex (the useful ones for image arithmetic).

```
Public functions:
IMG2CX (image, cximag, ierr)
   Convert image of type complex to a CX_IMAGE.
CX2IMG (cximag, image, ierr)
   Convert CX_IMAGE to image of type complex.
IMGR2C (rimage, cimage, ierr)
   Convert real image to image of type complex with zero imaginary.
```

10.14 File Name Class

This module is kept in $APLOOP/FILENAME.FOR.

File name class: name = "FILE_NAME"

The file name class contains the information about a disk resident data structure.

```
Class data:
   FTYPE   C*8    Type of file (e.g. AIPS, FITS)
Following for  files:
   NAME       C*12   AIPS adverb file name
   CLASS      C*6    AIPS adverb file class
   NAMCLSTY   C*20   AIPS Name (12 char), class(6 char) and TYPE
                  (2 char)
   IMSEQ   I      AIPS sequence number
   DISK    I      AIPS disk number
   CNO     I      AIPS catalog slot number
Public functions:
   FNAGET (name, keywrd, type, dim, value, valuec, ierr)
      Return array file name member.
   FNAPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store array file name member.
```

10.15 File Status Class

This module is kept in $APLOOP/FILESTATUS.FOR.

File Status: name = "FILE_STATUS"

The file status class contains information about the status and validity of a disk resident data structure.

```
Class data:
   STATUS  C*4   'READ', 'WRIT', 'DEST' (write but destroy on
                 failure) or '    '
   VALID   L     .TRUE. if file contains valid data.


 Public functions:
    FSTGET (name, keywrd, type, dim, value, valuec, ierr)
       Return array file status member.
    FSTPUT (name, keywrd, type, dim, value, valuec, ierr)
       Store array file status member.
```

10.16 Image Class

This module is kept in $APLOOP/IMAGE.FOR. Some "Q" members are in $QOOP/QIMAGE.FOR

Image Class: name "IMAGE"

An image consists of a pixel array as well as a number of base classes for descriptive information. General access is through IMGET and IMPUT although efficient access to the array member can be had directly through ARREAD and ARRWRI. ARRCLO should be used to close the access if ARREAD and/or ARRWRI are used. Even if ARRAY access is being used IMGOPN/IMGCLO should be used to insure the Image validity status and catalog status. If an underlying AIPS file is to be read and written at the same operation then the open for write should preceed the open for read or the bogey-man (CATIO) will get you.

When object are created and the underlying disk structures are also created the status is set to 'DEST' which is like write but if the task fails the underlying disk structures will be deleted. This status can be reset using IMCDES.

```
Class data:
   MAXROW  PI       Maximum length of an image row
   ROW1    R(*)     Row Buffer 1
```

```
   ROW2    R(*)     Row Buffer 2
   ROW3    R(*)     Row Buffer 3

Class base classes:
     ARRAY         Array of pixel values
     FILE_NAME     File name information
     FILE_STATUS   File status information
     IMAGE_DESC    Descriptive information about the image
     VELOCITY      Information for the conversion of frequency to
                   velocity.
     POSITION      Celestial position information
     TABLE         Tables
     BEAM          Beam size / deconvolution information

Public functions:
  IMGCRE (name, ierr)
     Creates an image object with name "name".
  IMGDES (name, ierr)
     Destroys the image object with name "name"; quasi-permanent
     forms are unaffected.
  IMGZAP (name, ierr)
     Destroys the image object with name "name"; quasi-permanent
     forms are deleted.
  IMGCOP (namein, namout, ierr)
     Copys one object to another.  The same quasi permanent forms
     are used for both.
  IMGCLN (namein, namout, ierr)
     CLONES an object.  A new object is created and any associated
     quasi-permanent forms are created.  The name, class etc. for
     the output quasi-permanent catalog entries are given by
     keywords OUTNAME, OUTCLASS, OUTSEQ and OUTDISK associated with
     namein.  The output image will represent the specified subimage
     in the input image.
  IMGSCR (name, dim, ierr)
     Creates an image scratch object of the size and structure given
     by dim.
  IMGOPN (name, status, ierr)
     Opens an image object.  Checks for valid data.
  IMGCLO (name, ierr)
     Closes an image object.  Updates data validity.
  IMGET (name, keywrd, type, dim, value, valuec, ierr)
     Return keyword value.
  IMPUT (name, keywrd, type, dim, value, valuec, ierr)
     Store keyword value.
  IMGATT (name, docrea, ierr)
     Attach an AIPS catalog data file to an object.  The name, class
     etc. for the output quasi-permanent catalog entries are given
     by keywords NAME, CLASS, IMSEQ and DISK associated with ``name''.
     Creates the file if necessary.
  IMCSET (name, status, ierr)
     Sets any file status except destroy on fail.
  IMCCLR (name, ierr)
     Clears any file status except destroy on fail.
  IMCDES (name, status, ierr)
     Clears destroy on fail status and resets file status.  If
     status is blank no status is set.
  IMGADD (in1, in2, out, ierr)
     Adds two image objects.
  IMGCVL (in1, in2, factor, out, ierr)
```

```
      Convolves two images
   IMGSUB (in1, in2, out, ierr)
      Subtracts in2 from in1 image objects
   IMGMUL (in1, in2, out, ierr)
      Multiplies in2 by in1 image objects
   IMGDIV (in1, in2, out, ierr)
      Divides in1 by in2 image objects
   IMGNEG (in, out, ierr)
      Negate the values of an image object.
   IMGFFT (dir, in, out, ierr)
      FFT an image
   IMCOPY (in, out, ierr)
      Copy one image to another.
   IMGPAD (in, out, ierr)
      Copy one image to another with zero padding.
   IMGSCL (in, factor, out, ierr)
      Scale an image with a factor.
   FFTPAD (in, out, ierr)
      Creates a scratch image suitable for FFTing an image and copies
      the selected subset of the input image into the scratch image
      with zero padding around the edges.  The scratch image is made
      twice the size of the input image if possible.

 Shared with derived classes
   IMGCHK (in1, in2, ierr)
      Checks that two images have compatible size and position.
   IMGWIN (in1, blc, trc, naxis, ierr)
      Determine specified window in an image.

 Private functions:
   CFLSET (name, disk, cno, status, ierr)
      Set AIPS catalog status, DFIL.INC common
   CFLCLR (name, disk, cno, status, ierr)
      Clear AIPS catalog status, DFIL.INC common
   IMCREA (name, ierr)
      Creates file structures for image "name"
   IMBGET (name, keywrd, type, dim, value, valuec, ierr)
      Fetches member of a base class of image class
   IMBPUT (name, keywrd, type, dim, value, valuec, ierr)
      Stores member of a base class of image class
   IMBTNF (catin, catout, ierr)
     Copies table information from CATBLK catin to catout.
   IMCLNX (namein, namout, ierr)
     If NAMOUT not fully instantiated copies descriptors from NAMEIN
     to NAMOUT allowing to subimaging in NAMEIN.
```

10.17 Image Descriptor Class

This module is kept in $APLOOP/IMAGEDESC.FOR.

Image descriptor class: name = "IMAGE_DESC."

The image descriptor contains descriptive information about an image.

```
 Class public members;
    OBJECT   C*8    Source name
    TELESCOP C*8    Telescope name
    INSTRUME C*8    Receiver name
    OBSERVER C*8    Observer name
    DATE-OBS C*8    Observing date as dd/mm/yy
    DATE-MAP C*8    Creation date as dd/mm/yy
```

```
   DOCHECK L       True if array labeling should be compared before
                   binary operations with other arrays.
   BUNIT   C*8     Units of the array
   EPOCH   R       Mean epoch of celestial position (1950, 2000)
   USERNO  I       User ID number
   CTYPE   C*8(*)  Label for each axis
   CRVAL   D(*)    Coordinate value at reference pixel
   CDELT   R(*)    Coordinate increment
   CRPIX   R(*)    Reference pixel for axis
   CROTA   R(*)    Coordinate rotation for each axis.

Public functions:
   IMDGET (name, keywrd, type, dim, value, valuec, ierr)
      Return image descriptor member.
   IMDPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store image descriptor member.
   IMDCOP (namein, namout, ierr)
      Copy image descriptor info from namein to namout.
```

10.18 Image Class Utility Module

This module is kept in $APLOOP/IMAGEUTIL.FOR.

This module contains Image class utility routines.

```
Public functions:
   IMTCOP (in, out, tbtype, tbver, ierr)
      Copy table between images. tbver=0 => copy all.
   IM2TAB (image, table, tbtype, tbver, ierr)
      Makes table object associated with a image object.
```

10.19 Image Interpolation Utility module

UV data utility library: $QOOP/QINTER.FOR

This is a package of image interpolation routines. The top level routines are IMGIMT and IMGHGE which interpolate one image to another. Unlike the *GEOM tasks these routines will interpolate over blanked pixels and can fill in small blanked regions. At the edges of the image being interpolated the interpolation kernal is adjusted to reduce edge effects.

```
Public functions:
IMGINT (in, shift, rotate, hwidth, out, ierr)
   Interpolate in to out with shift and rotate.
IMGHGE (in, hwidth, out, ierr)
   Interpolate in to out with full geometry calculation and optional
   W-corrections and radial scaling.
IMGIAE (in, parang, el, hwidth, out, ierr)
   Interpolate an az-el image at the positions of the output image.

Private functions:
IMINST (in, plane, hwidth, ierr)
   Set up for interpolation, read input plane.
IMINTR (x, y, val)
   Returns interpolation value at (x,y)
IMINTP (xpix, ypix, val)
   Returns interpolation value at pixel (xpix,ypix)
IMINFN (pos, hwidth, int, cen)
   Return interpolation function
```

10.20 Inputs Class

This module is kept in $APLOOP/INPUT.FOR.

Inputs Class: Name "INPUTS"

An INPUTS object contains the names and values of the POPS adverbs passed from the user.

```
Class data:
 Passed Adverbs

Public functions:
  AV2INP (prgn, nparm, parm, type, dim, out, ierr)
     Does AIPS startup and copies AIPS adverbs to an Inputs object.
  IN2OBJ (in, nkey, inkey, outkey, out, ierr)
     Copies a list of keywords to object out with possible renaming.
  INTELL (nparm, parm, type, dim, out, optell, ierr)
     Obtain "TELL" instructions as an INPUT object.
  INGET (name, keywrd, type, dim, value, valuec, ierr)
     Return adverb value.
  INPUTT (name, keywrd, type, dim, value, valuec, ierr)
     Store adverb value.
```

10.21 Position Class

This module is kept in $APLOOP/POSITION.FOR.

Observing position class: name = 'POSITION'

The observing position class contains information about the original position and any phase center shifts of the data.

```
Class data:
   OBSRA     D   Pointing position: RA (degrees)
   OBSDEC    D   Pointing position: Dec (degrees)
   XSHIFT    R   Phase shift in RA (degrees)
   YSHIFT    R   Phase shift in Dec (degrees)
Public functions:
   PSNGET (name, keywrd, type, dim, value, valuec, ierr)
      Return position member.
   PSNPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store position member.
   PSNCVT (in, xyzi, out, xyzo, ierr)
      Convert pixel position in one image to pixel position in
      another.
   PSNCV3 (in, xyzi, out, tza, cpa, spa, pbfwhm, fbwsq, c123, xyzo,
      ierr)
      Convert pixel positions correcting for array plane
      misorientation and radial scaling.
```

10.22 Printer Class

This module is kept in $APLOOP/PRINTER.FOR.

PRINTER Class: Name "PRINTER"

A Printer object is an interface to the line printer, or, as a user option, printer line output on the terminal. Only one printer object may be active (open) at a time. Lines are printed using PRTWRI; objects are opened and closed using PRTOPN and PRTCLO. Opening may cause a header for object PRINTOBJ (uvdata or image only) to be written and closing the object spools the output.

```
 Class public members:
   PRINTOBJ  C*32  Name of associated uvdata or image object.
   LPFILE    C*48  Name of printer file.
```

```
      DOCRT     I     If >0 use terminal, else printer.
    Following only after opening:
      TITLE1    C*132 First page running title/label
      TITLE2    C*132 Second page running title/label

   Class  private data:
      MYNAME    C*32  Name of open printer
      POBJ      C*32  Name of associated object
      DOCRT     R     Interactive/printer and width of terminal
      NACROS    I     Number of columns in output
      NLINE     I     Line number on page
      IPAGE     I     Page number.
      PRTLUN    I     Printer LUN
      PRTIND    I     Printer FTAB pointer.

   Public functions:
     PRTCRE (name, ierr)
        Creates a printer object with name "name".
     PRTDES (name, ierr)
        Destroys the printer object with name "name";
     PRTZAP (name, ierr)
        Destroys the printer object with name "name";
        (Same as PRTDES)
     PRTOPN (name, status, ierr)
        Opens a printer object, writing header if appropriate
     PRTCLO (name, ierr)
        Closes a printer object, spooling output if appropriate
     PRTGET (name, keywrd, type, dim, value, valuec, ierr)
        Return keyword value.
     PRTPUT (name, keywrd, type, dim, value, valuec, ierr)
        Store keyword value.
     PRTWRI (name, line, quit, ierr)
        Send a line to the printer, user may request quit.
     PRTNUP (name, ierr)
        Force new page on next write.
     PRTCNT (name, quit, ierr)
        For interactive use ask user if he/she wishes to continue.

   Private functions:
      PRBGET (name, keywrd, type, dim, value, valuec, ierr)
         Fetches private member.
      PRBPUT (name, keywrd, type, dim, value, valuec, ierr)
         Stores private member.
      PRTUVH (name, ierr)
         Print uvdata header
      PRTIMH (name, ierr)
         Print image header
      PRTTBH (name, ierr)
         Print table header
```

10.23 Table Class

This module is kept in $APLOOP/TABLE.FOR.

Table Class: Name "TABLE"

Tables are rectangular data structures which may contain elements of various data type. Most table access is through TABGET and TABPUT but rapid access to row data is possible through TABDGT and TABDPT.

```
Class public members:
  NAME      C*12     Catalog file name
  CLASS     C*6      Catalog file class
  IMSEQ     I        Catalog file sequence number
  DISK      I        Disk number
  TBLTYPE   C*2      Table type
  VER       I        Version number
```

The following must be set before a new table is opened and are unavailable before an existing table is opened.

```
  LABEL     C*56     Table label
  NCOL      I        Number of columns
  COLABEL   C(*)*24  Column labels
  COLUNIT   C(*)*8   Column units
  COLTYPE   I(*)     Column data type: 1=double, 2=real,
                     3=character, 4=integer, 5=logical, 7=bit
                     arrays.
  COLDIM    I(*)     Column dimension.
```

The following are available only when the table is open.

```
  NROW      I        Number of rows
  SORT      I(2)     Sort order
  CURROW    I        Current row number, if negative it has not yet
                     been read.
  ENTRY.nn  ?(?)     table entry for column number nn
  KEY.xxxx  ?        table keyword/value pair for keyword xxxx

Class private members:
  TBNCOL    I(MAXIO)       Number of columns, per I/O stream.
  TBCROW    I(MAXIO)       Current row number, per I/O stream.
  TBTYPE    I(128,MAXIO)   Column type codes, one set per I/O stream
  TBDIM     I(128,MAXIO)   Column element count
  TBPTR     I(128,MAXIO)   Column pointer to first element in array
                           of type.
  RECORD    I(2048,MAXIO)  Record buffer per I/O stream.
                           Equivalenced to RECR, RECD, RECH and, RECL
                           for real, double, Hollerith and logical.

Public functions:
  TABCRE (name, iret)
     Creates a table object.
  TABDES (name, ierr)
     Destroys the table object with name "name"; quasi-permanent
     forms are unaffected.
  TABZAP (name, ierr)
     Destroys the table object with name "name"; quasi-permanent
     forms are deleted.
  TABRMV (name, ierr)
     Removes any underlying AIPS files leaves object intact.
  TBCOPY (namein, namout, ierr)
     Copys one object to another.  The same quasi permanent forms
     are used for both.
  TBLCOP (namein, namout, ierr)
     Copys contents of one table object to another.  (Standard AIPS
     TABCOP)
  TABCLN (namein, namout, ierr)
     CLONES an object.  A new table is created.
  TABOPN (name, status, ierr)
```

```
      Opens a table for access.
   TABCLO (name, ierr)
      Closes a table for access.
   TABCOL (name, ncol, colab, colnum, ierr)
      Returns column numbers for a list of column labels.
   TABGET (name, keywrd, type, dim, value, valuec, ierr)
      Fetches table member
   TABPUT (name, keywrd, type, dim, value, valuec, ierr)
      Stores table member
   TABDGT (name, row, col, type, dim, value, valuec, ierr)
      Fetches table row data
   TABDPT (name, row, col, type, dim, value, valuec, ierr)
      Stores table row data
   TABKGT (name, keys, nkeys, klocs, kvals, ktype, ierr)
      Fetches values of specified table keywords.  If keys(1) is
      blank then all keywords up to a maximum of nkeys is read.  On
      return nkeys is the number read.
   TABKPT (name, keys, nkeys, klocs, kvals, ktype, ierr)
      Stores values of specified table keywords.
   TBLSRT (name, key1, key2, ierr)
      Sorts a table using a 2 key sort.
   TBLMRG (name, mkol, toler, skol, nskol, srtkol, ierr)
      Merge a table.

 Private functions:
   TBLMEM (keywrd, mem, arg, local, ierr)
      Parses keyword into components
   TBLKUP (name, tdisk, tcno, ttype, tver, ierr)
      Looks up information about table object.
   TBLHIV (name, nver, ierr)
      Returns highest table version number.
```

10.24 Table Utilities

This module is kept in $APLOOP/TABLEUTIL.FOR.

A number of routines which operate on table objects but which are not class members are in this module. A short description is given here; see the text of this module for full details.

```
 Public functions:
   COPHED (intab, outtab, ierr)
      Copy header info from one table to another, force create.
   ANTNO (anttab, subarr, maxant, ierr)
      Find maximum antenna number in an AN table.
   ANTNFO (table, subarr, anum, label, type, dim, value,
         valuec, ierr)
      Find info about antenna with id=anum
   SOUNFO (table, suid, label, type, dim, value, valuec, ierr)
      Find info about a given source id.
```

OOA Fronts to AIPS table specific routines: These routines will open the object.

```
   OANINI (table, opcode, anrow, arrayc, gstia0, degpdy, safreq,
         rdate, polrxy, ut1utc, datutc, timsys, aname, numorb,
         nopcal, anfqid, ierr)
      Open/create/init AIPS AN table. (ANTINI)
    OTABAN (table, opcode, anrow, anname, staxyz, orbprm, nosta,
          mntsta, staxof, poltya, polaa, polca, poltyb, polab, polcb,
          ierr)
```

```
   Access AIPS AN table (TABAN)

 OBLINI (table, opcode, blrow, numant, numpol, numif, ierr)
   Open/create/init AIPS BL table (BLINI)
 OTABBL (table, opcode, blrow, numpol, time, sourid, suba, ant1,
      ant2, freqid, facmul, facadd, ierr)
   Access AIPS BL table. (TABBL)

 OBPINI (table, opcode, bprow, numant, numpol, numif, numfrq,
      bchan, ierr)
   Open/create/init AIPS BP table (BPINI)
 OTABBP (table, opcode, bprow, numif, numfrq, numpol, time, interv,
      sourid, suba, ant, bandw, iffreq, freqid, refant, real,
      imag, ierr)
   Access AIPS BP table. (TABBP)

OBSINI (table, opcode, bsrow, mbdsln, numif, ierr)
  Open/create/init AIPS BS table (BSINI)
OTABBS (table, opcode, bsrow, numif, time, interv, baseln, subarr,
       stokes, source, rphase, rsbd, rrate, raccel, rmbd, wt,
       ierr)
  Access AIPS BS table. (TABBS)

OCCINI (table, opcode, ccrow, numcol, ierr)
  Open/create/init AIPS CC table (CCMINI)
OTABCC (table, opcode, ccrow, numcol, x, y, flux, type, parms,
       ierr)
  Access AIPS CC table. (TABCCM)

OCLINI (table, opcode, clrow, numant, numpol, numif, nterm, gmmod,
      ierr)
   Open/create/init AIPS CL table. (CALINI)
OTABCL (table, opcode, clrow, numpol, numif, time, timei, sourid,
      antno, suba, freqid, ifr, geodly, dopoff, atmos, datmos,
      mbdely, clock, dclock, disp, ddisp, creal, cimag, delay,
      rate, weight, refa, ierr)
   Access AIPS CL table. (TABCAL)

 OFGINI (table, opcode, fgrow, ierr)
   Open/create/init AIPS FG table (FLGINI)
 OTABFG (table, opcode, fgrow, sourid, suba, freqid, ants, timer,
      ifs, chans, pflags, reason, ierr)
   Access AIPS FG table. (TABFLG)

 OFQINI (table, opcode, fqrow, numif, ierr)
   Open/create/init AIPS FQ table (FQINI)
 OTABFQ (table, opcode, fqrow, numif, fqid, iffreq, ifchw, iftbw,
      ifside, ierr)
   Access AIPS FQ table. (TABFQ)
 OCHNDA (table, opcode, fqrow, nif, foff, isband, finc, freqid,
      ierr)
   Access AIPS FQ or CH table. (CHNDAT)
 OCHNCO (tablei, tableo, bif, eif, freqid, ierr)
   Copies selected portion of FQ or CH table. (CHNCOP)

 OHFINI (table, opcode, hfrow, ierr)
   Open/create/init AIPS HF table. (HFINI)
 OTABHF (table, opcode, hfrow,
      cc1, ic2, ic3, ic4, ic5, ic6, ic7, ic8, ic9, ic10, ic11,
```

```
      ic12, ic13, ic14, ic15, ic16, ic17, ic18, ic19, ic20, ic21,
      ic22, ic23, ic24, cc25, cc26, cc27, cc28, cc29, cc30, cc31,
      cc32, cc33, cc34, cc35, cc36, cc37, cc38, dc39, dc40, dc41,
      dc42, dc43, dc44, dc45, dc46, dc47, dc48, dc49, dc50, rc51,
      rc52, rc53, rc54, rc55, rc56, rc57, rc58, rc59, rc60, rc61,
      rc62, rc63, rc64, rc65, rc66, rc67, rc68, rc69, rc70, rc71,
      rc72, rc73, rc74, rc75, rc76, rc77, ierr)
   Does I/O to HF table (TABHF)

  ONXINI (table, opcode, nxrow, ierr)
     Open/create/init AIPS NX table (NDXINI)
  OTABNX (table, opcode, nxrow, time, dtime, idsour, subarr, vstart,
        vend, freqid, ierr)
     Access AIPS NX table. (TABNDX)

  OSNINI (table, opcode, snrow, numant, numpol, numif, numnod,
        gmmod, ranod, decnod, isappl, ierr)
     Open/create/init AIPS SN table (SNINI)
  OTABSN (table, opcode, snrow, numpol, time, timei, sourid, antno,
        suba, freqid, ifr, nodeno, mbdely, creal, cimag, delay,
        rate, weight, refa, ierr)
     Access AIPS SN table. (TABSN)

  OSUINI (table, opcode, surow, numif, veltyp, veldef, freqid,
        isurno, sukols, sunumv, ierr)
     Open/create/init AIPS SU table (SOUINI)
  OTABSU (table, opcode, surow, idsou, sounam, qual, calcod, flux,
        freqo, bandw, raepo, decepo, epoch, raapp, decapp, lsrvel,
        lrestf, pmra, pmdec, ierr)
     Access AIPS SU table. (TABSOU)

  OTYINI (table, opcode, tyrow, numpol, numif, ierr)
     Open/create/init AIPS TY table (TYINI)
  OTABTY (table, opcode, tyrow, numpol, numif, time, timei, sourid,
        antno, suba, freqid, tsys, tant, ierr)
     Access AIPS TY table. (TABTY)

  OSUELV (table, ant, subarr, sourid, time, elev, ierr)
     Returns source elevation for a given time and antenna.
  OSUPAN (table, ant, subarr, sourid, time, pangl, ierr)
     Returns parallactic angle for a given time, source and antenna.
```

10.25 Vector Class

This module is kept in $APLOOP/VECTOR.FOR.

Vector class: Name "VECTOR"

A vector is a one dimensional array of numeric values. Blanking is supported. Arguments INn, and OUT are Fortran arrays, VALINn and VALOUT are logical arrays specifying whether corresponding elements of INn or OUT are valid. The element count is N and the stride is assumed to the the minimum appropriate for the given data type. This is more of a utility library than a true class library.

```
Public functions

   RVNEG (IN, OUT, N)
      Real Vector negate.
   CVNEG (IN, OUT, N)
      Complex Vector negate
   RVBNEG (IN, VALIN, OUT, VALOUT, N)
      Real Vector negate with blanking
```

```
CVBNEG (IN, VALIN, OUT, VALOUT, N)
   Complex Vector negate with blanking
RVADD (IN1, IN2, OUT, N)
   Real Vector add
CVADD (IN1, IN2, OUT, N)
   Complex Vector add
RVBADD (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Real Vector add with blanking
CVBADD (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Complex Vector add with blanking
RVSUB (IN1, IN2, OUT, N)
   Real Vector subtract
CVSUB (IN1, IN2, OUT, N)
   Complex Vector subtract
RVBSUB (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Real Vector subtract with blanking
CVBSUB (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Complex Vector subtract with blanking
RVMUL (IN1, IN2, OUT, N)
   Real Vector multiply
CVMUL (IN1, IN2, OUT, N)
   Complex Vector multiply
RVBMUL (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Real Vector multiply with blanking
CVBMUL (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Complex Vector multiply with blanking
RVDIV (IN1, IN2, OUT, N)
   Real Vector divide first by second
CVDIV (IN1, IN2, OUT, N)
   Complex Vector divide first by second
RVBDIV (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Real Vector divide first by second with blanking
CVBDIV (IN1, VALIN1, IN2, VALIN2, OUT, VALOUT, N)
   Complex Vector divide first by second with blanking
RVSCL (IN1, FACTOR, OUT, N)
   Real Scale vector with real scalar
CVSCL (IN1, FACTOR, OUT, N)
   Complex Scale vector with complex scalar
RVBSCL (IN1, VALIN1, FACTOR, OUT, VALOUT, N)
   Real Scale vector with real scalar with blanking
CVBSCL (IN1, VALIN1, FACTOR, OUT, VALOUT, N)
   Complex Scale vector with complex scalar with blanking
RVCOPY (IN, OUT, N)
   Copy real vector
CVCOPY (IN, OUT, N)
   Copy complex vector
RVFILL (SCALAR, OUT, N)
   Real vector fill
CVFILL (SCALAR, OUT, N)
   Complex vector fill
RSVADD (IN, SCALAR, OUT, N)
   Real scalar vector add
CSVADD (IN, SCALAR, OUT, N)
   Complex scalar vector add
RSVADB (IN, VALIN, SCALAR, OUT, VALOUT, N)
   Real scalar vector add with blanking
CSVADB (IN, VALIN, SCALAR, OUT, VALOUT, N)
   Complex scalar vector add with blanking
```

10.26 Velocity Class

This module is kept in $APLOOP/VELOCITY.FOR.

Velocity class: name = "VELOCITY"

The velocity class contains informaton necessary for the transformation between frequency and velocity.

```
Class data:
   VELREF    I   Velocity definition code: 0 => none,
                 1 - 3 => LSR, Sun, Obs  + 256 if radio
   ALTRVAL   D   Alternate reference value: (frequency
                 in Hz or velocity in m/sec)
   ALTRPIX   R   Alternate reference pixel
   RESTFREQ  D   Line rest frequency (Hz)
Public functions:
   VELGET (name, keywrd, type, dim, value, valuec, ierr)
      Return velocity member.
   VELPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store velocity member.
```

10.27 Utility Module

This module is kept in $APLOOP/UTILITY.FOR.

This module contains general utility routines.

```
Public functions:
   OBSCNF (name, scrno, ierr)
      Fills in file description info for a scratch file created
      outside of the OOP package.
   OBFEXS (name, exist, ierr)
      Determines if and object exists, for objects with an underlying
      disk file the existance of this file is also checked. ierr=0
      mean it exists else not.
   OBACF (disk, cno, index)
      Determines if a specified AIPS disk and catalog slot already has
      an AIPS catalog flags set and registered in the DFIL.INC common.
  OBCSYN (name, ierr)
    Synchronizes the memory resident CATBLK with the one on disk,
    i.e. reads from disk.
  FNDSKC (name, disk, cno, ierr)
    Looks up disk, cno number; object need not have been previously
    opened.
```

10.28 UVdata Class

This module is kept in $APLOOP/UVDATA.FOR.

UV data class: name = "UVDATA"

UV data generally implies data measured with an interferometer. A UV data object consists of an set of arrays of correlation values with a set of "random parameters" as labels for each set of correlation values. Also included are a set of descriptive classes. General access to members is through member functions OUVGET and OUVPUT. Efficient access to visibility array data is through UVREAD and UVWRIT.

```
 Class public members:
   VIS         R(*)          Random parameters and correlation
                             array.
   FILE_NAME   Base class    File name information
   FILE_STATUS Base class    File status information
   UV_DESC     Base class    Descriptive information about the uv
                             data set
```

```
   VELOCITY    Base class    Information for the conversion of
                             frequency to velocity.
   POSITION    Base class    Celestial position information
   CALEDIT     Base class    Calibration and editing control
                             information.
   OLDFILE     Logical       If true then the underlying files
                             existed before the object was created.

Class  private data:
   UVNRPM   I(MAXIO)   Number of random parameters
   UVLREC    I(MAXIO)   Length of visibility record.
   UVNVIS    I(MAXIO)   Number of visibility records.
   UVBIND    I(MAXIO)   Buffer pointer.
   UVSIZE    I(MAXIO)   Maximum file size in vis records.
   UVVNO     I(MAXIO)   Last visibility number to be written.

Public functions:
  OUVCRE (name, ierr)
     Creates an uv data set object with name "name".  Initializes
     sel/cal/edit criteria.
  OUVDES (name, ierr)
     Destroys the uv data set object with name "name";
     quasi-permanent  forms are unaffected.
  OUVZAP (name, ierr)
     Destroys the uv data set object with name "name";
     quasi-permanent forms are deleted.
  OUVCOP (namein, namout, ierr)
     Copys one object to another.  The same quasi permanent forms
     are used for both.
  OUVCLN (namein, namout, ierr)
     CLONES an object.  A new object is created and any associated
     quasi-permanent forms are created.  The name, class etc. for
     the output quasi-permanent catalog entries are given by
     keywords OUTNAME, OUTCLASS, OUTSEQ and OUTDISK associated with
     namein.
  OUVSCR (name, name2, nvis, ierr)
     Creates an uv data set scratch object of the structure given by
     name2 and number of visibilities given by nvis.
  OUVOPN (name, status, ierr)
     Opens an uv data set object.  Checks for valid data.
     Initializes I/O, selection, calibration, and editing for read.
  OUVCLO (name, ierr)
     Closes an uv data set object.  Updates data validity.
  OUVINI (name, ierr)
     Reinitialize I/O.
  OUVGET (name, keywrd, type, dim, value, valuec, ierr)
     Return keyword value.
  OUVPUT (name, keywrd, type, dim, value, valuec, ierr)
     Store keyword value.
  UVREAD (name, rp, vis, ierr)
     Read next visibility record.  The UV_DESC descriptor for an
     open file will describe the data as returned rather than as on
     disk.  Will open file if necessary.
  UVWRIT (name, rp, vis, ierr)
     Write next visibility record.  Will open file if necessary.
Following are public but shouldn't be needed from applications.
  OUVATT (name, docrea, ierr)
     Attach an AIPS catalog data file to an object.  The name, class
     etc. for the output quasi-permanent catalog entries are given
```

```
    by keywords NAME, CLASS, IMSEQ and DISK associated with NAME.
    Creates the file if necessary.
  OUCSET (name, status, ierr)
    Sets any file status)
  OUCCLR (name, ierr)
    Clears any file status except 'DEST'.
  OUCDES (name, status, ierr)
    Clears destroy on fail status and resets file status.  If
    status is blank no status is set.
  OUVSLT (name, name2, ierr)
    Modify descriptors in name to correspond to selection criteria
    in name2.
 OUVCGT (name, cat, ierr)
    Return catalog header record for an uvdata indicating correct
    compression. This function should be used to get the CATBLK to
    be passed to nonOOP routines.

Shared with derived classes

Private functions:
  OUCFST (name, disk, cno, status, ierr)
    Set AIPS catalog status, DFIL.INC common
  OUCFCL (name, disk, cno, status, ierr)
    Clear AIPS catalog status, DFIL.INC common
  OUCREA (name, ierr)
    Creates file structures for uv data set "name"
  OUBGET (name, keywrd, type, dim, value, valuec, ierr)
    Fetches member of a base class of uv data set class
  OUBPUT (name, keywrd, type, dim, value, valuec, ierr)
    Stores member of a base class of uv data set class
  OUVXPN (name, ierr)
    Expands size of uv data file
  OUVCPR (name, ierr)
    Compresses size of uv data file
  OUVTNF (catin, catout, ierr)
    Copies table information from CATBLK catin to catout.
```

10.29 UVdata Utility module

UV data utility library: $APLOOP/UVUTIL.FOR

```
Public functions:

AVGVIS (naxis, d2, d3, d4, d5, lim, visin, visout)
   Vector average a visibility record
SCNAVG (name, source, qual, timer, maxant, vis, ierr)
   Vector average a specified set of data
SCNHAY (name, source, qual, timer, maxant, vis, ampfrc, ierr)
   Extract scan information needed for making Haystack HF tables.
NXTAVG (opcode, name, interv, avgif, avgpol, antwt, maxan, maxifs,
   maxpol, timec, timei, sid, fqid, vis, ierr)
   Return all data averaged for the next time interval.
UVCOPY (uvin, uvout, ierr)
   Copies a uv data object to another with application of any
   calibration editing, and selection criteria.
UV2SCR (uvdata, uvscr, ierr)
   Copies a uv data object to a scratch object with application of
   any calibration editing, and selection criteria.
UVCLIP (uvin, uvout, ierr)
```

```
   Copy uvdata flagging data with excessive amplitudes.
U2IDES (uvdata, image, dodft, ierr)
   Copies descriptive info from a Uvdata object to an image and
   initializes the image descriptors.
UVFRQS (uvdata, uvfreq, freqs, ierr)
   Return reference frequency and an array of frequencies for each
   channel/IF.
UVREFQ (uvdata, chtype, refreq, refpix, ierr)
   Determines reference frequency and pixel for a uvdata set subject
   to frequency selection and averaging.
UVDTCO (uvin, uvout, ierr)
   Copy descriptive tables (AN, FQ, SU) from uvin to uvout.
UVTCOP (uvin, uvout, type, ver, ierr)
   Copy a specified table from one uvdata to another, ver=0 => all.
UVFCOP (uvin, uvout, ierr)
   Copy portion of FQ table given by input uvdata selection criteria
   to another uvdata.
UV2TAB (uvdata, table, tbtype, tbver, ierr)
   Makes table object associated with a uvdata object.
UVBAVG (uvin, uvout, ierr)
   Baseline dependent time averaging of a uv data set.
UVTAVG (uvin, uvout, ierr)
   Time average time sorted data.
UVCALT (uvin, uvout, ierr)
   Copies all tables from one uvdata object to another.
UVAMPS (uvin, uvran, namps, amps, ierr)
  Determine range of baselines and a histogram of average amplitude
  vs. baseline length.
UVFRSC (uvname, snname, ierr)
   Faraday rotation self cal solutions.

OOA Fronts to AIPS uv data specific routines:
 SLFCAL (uvname, snname, ierr)
   Does self cal solutions
 SLFREF (snname, isuba, refant, ierr)
   Refererence all phases to a common reference antenna.  This
   should generally be used after SLFCAL.


OOA Front private routines
 SLFPA (vobs, maxan, maxifs, maxpol, numif, numpol, refant, avgif,
   avgpol, dol1, dogcon, gaerr, mode, minno, confac, snrmin, prtlv,
   creal, cimag, cwt, refan, gotant)
   Compute self calibration solutions from averaged data.
 SLFSOU (snname, numan, numif, numpol, timec, timei, suba, sid, fqid,
   creal, cimag, cwt, refan, gotant, summgm, cntmgm, ierr)
   Write solution to SN object.
 FRSOLN (vobs, maxan, maxifs, maxpol, numif, numpol, avgif, minno,
   snrmin, ratmin, ratmax, prtlv, creal, cimag, cwt, refan, gotant)
   Compute Faraday rotation solutions.
```

10.30 UVdata "Q" Utility module

UV data utility library: $QOOP/QUVUTIL.FOR

```
Public functions:

Generic functions:
OUVIMG (uvdata, mfield, image, beam, work1, work2, dobeam, chan,
```

```
      nchan, imchan, ierr)
      Make image or beam from a uv data set; called from UV2IMG.
   UVSUBM (uvin, uvout, mfield, cname, chanl, nchan, ierr)
      Generic routine to subtract the FT of a model from a uv data set.
   UVDIVM (uvin, uvout, mfield, cname, chanl, nchan, ierr)
      Generic routine to divide the FT of a model into a uv data set.
   OUVSRT (uvin, uvout, ierr)
      Sort and optionally rotate uv data.

   Specific functions:
   OUVDFT (uvdata, beam, image, ierr)
      DFT a uv data set to an image.  High level.
   ODFT (uv, chinc, chtype, image, beam, factor, ierr)
      Low level routine to DFT a uv data set.  Called by OUVDFT.

  OOA Fronts to AIPS uv data specific routines:
   OUNFWT (uv, image, ierr)
      Applies uniform weighting correction (UVTBUW)
   OUMSUB (uvin, uvout, mfield, cname, chanl, nchan, ierr)
      Subtract the FT of a model from a uv data set; called from
      UVSUBM.
   OUMDIV (uvin, uvout, mfield, cname, chanl, nchan, ierr)
      Divide the FT of a model into a uv data set; called from UVDIVM.
   OUCDFT (uvin, uvout, nfield, image, chanl, nchan, ierr)
      Add DFT of "W" corrected model to uv data.
   OUFDFT (uvin, uvout, nfield, image, chanl, nchan, ierr)
      Add DFT of bandwidth smearing corrected model to uv data.
   OUSETF (tflux, nfield, flux)
      Sets values of total flux densities in DGDS.INC commons
```

10.31 Frequency dependent antenna correction utility

Frequency dependent antenna beam utility module: $QOOP/PBUTIL.FOR

This module provides utilities for imaging related functions which include the effects of frequency dependant primary antenna patterns. The model of the antenna pattern used is that it is a uniformly illuminated disk. The parameters required are:

```
      DOPBFM     L    If true then do modeling with frequency dependent
                       primary beam corrections.
      PBFREQ     D(*) Frequencies (Hz) going into the average.
      PBFSIZ     R    Antenna diameter (m)
```

These parameters are passed attached to the input uv data in each case.

For uv model computations a CC table or corrected image is generated for each frequency channel in which the fluxes, assumed an average of all channels, are corrected to that channel. The model computation is done for each channel independently. For imaging operations, each channel is imaged separately, corrected and then averaged.

```
   Public functions:
   PBFSUB (uvin, uvout, mfield, cname, chanl, nchan, ierr)
      Subtract the FT of a model from a uv data set.
   PBFDIV (uvin, uvout, mfield, cname, chanl, nchan, ierr)
      Divide the FT of a model into a uv data set.

   Private functions:
   PBFCCT (nfield, image, iccver, ichan, nfreq, pbfreq, pbfsiz, occver,
      outcc, ierr)
      Make temporary, frequency scaled CC table.
```

```
PBFSCI (nfield, image, ichan, nfreq, pbfreq, pbfsiz, tmpimg, ierr)
   Make temporary, frequency scaled image.
PBFACT (nfreq, pbfreq, pbfsiz, freq, radius)
   Function which returns relative primary beam correction.
```

10.32 "Q" Utility module

UV data utility library: $QOOP/APUTIL.FOR The routines in this module mostly copy data to and from "AP" memory and will only work for the pseudo AP as they directly access "AP" memory.

```
Public functions:

AP2IMG (ap, image, plane, ierr)
   Copy an image plane from "AP" memory to an image object.
IMG2AP (image, plane, ap, ierr)
   Copy an image plane from an image object to "AP" memory.
AP2UV (ap, uv, nv, lv, nr, ierr)
   Copy a set of visibilities from the "AP" to uv object.
UV2AP (uv, ap, nv, lv, nr, ierr)
```

10.33 UVdata descriptor Class

This module is kept in $APLOOP/UVDESC.FOR.

UV data descriptor class: name = "UV_DESC"

The uv data descriptor contains descriptive information about an uv data set.

```
Class public members;
   OBJECT   C*8    Source name
   TELESCOP C*8    Telescope name
   INSTRUME C*8    Receiver name
   OBSERVER C*8    Observer name
   DATE-OBS C*8    Observing date as dd/mm/yy
   DATE-MAP C*8    Creation date as dd/mm/yy
   BUNIT    C*8    Units of the correlation array
   EPOCH    R      Mean epoch of celestial position (1950, 2000)
   USERNO   I      User ID number
   NDIM     I      Number of dimensions in the array
   NAXIS    I(*)   Dimension of each axis
   CTYPE    C*8(*) Label for each axis
   CRVAL    D(*)   Coordinate value at reference pixel
   CDELT    R(*)   Coordinate increment
   CRPIX    R(*)   Reference pixel for axis
   CROTA    R(*)   Coordinate rotation for each axis.
   GCOUNT   I      Number of visibilities
   SORTORD  C*2    Sort order.
   NRPARM   I      Number of random parameters
   PTYPE    C*8(*) Random parameter labels
   LREC     I      Visibility record length.
   NCORR    I      Number of correlators in visibility
   VISOFF   I      Offset in visibility number (valid on read only
                   for single source datasets)
   ISCOMP   L      If true, data on disk are in compressed format.
   ANAME    C*8    Name of array if memory resident
   FNAME    C*48?  Physical name of file if disk resident.
   FDISK    I      Disk number for FNAME.
   REFFREQ  D      Reference frequency (that of u,v,w) in Hz.

Public functions:
```

```
UVDGET (name, keywrd, type, dim, value, valuec, ierr)
   Return image descriptor member.
UVDPUT (name, keywrd, type, dim, value, valuec, ierr)
   Store image descriptor member.
UVDPNT (name, ilocu, ilocv, ilocw, iloct, ilocb, ilocsu, ilocfq,
*   jlocc, jlocs, jlocf, jlocr, jlocd, jlocif, incs, incf, incif,
*   ierr)
   Returns pointers etc for interpreting a visibility data
   record.
UVDCOP (namein, namout, ierr)
   Copy UV descriptor info from namein to namout.
UVDSCP (namein, namout, ierr)
   Copy UV descriptor info not related to object size.
UVDFND (name, itype, label, index, ierr)
   Look up the random parameter or axis "label" number
```

10.34 Selection/Calibration/Editing Class

This module is kept in $APLOOP/CALEDIT.FOR.

Selection/Calibration/Editing class: name = "CALEDIT"

The parameters controling data selection, calibration and editing are stored here.

```
Class public members.
   SOURCS   C(30)*16  Names of up to 30 sources, *=>all
                      First character of name '-' => all except
                      those specified.
   SELQUA   I         Qualifier wanted (-1 => all)
   SELCOD   C*4       Cal code ('    ')
   TIMRNG   R(8)      Start day, hour, min, sec, end day, hour,
                      min, sec. 0's => all
   UVRNG    R(2)      Minimum and maximum baseline lengths in
                      1000's wavelengths. 0's => all
   STOKES   C*4       Stokes types wanted.
                      'I','Q','U','V','R','L','IQU','IQUV'
                      '    '=> Leave data in same form as in input.
   BCHAN    I         First channel number selected, 1 rel. to first
                      channel in data base. 0 => all
   ECHAN    I         Last channel selected. 0=>all
   BIF      I         First IF number selected, 1 rel. to first
                      IF in data base. 0 => all
   EIF      I         Last IF selected. 0=>all
   DOCAL    L         If true apply calibration, else not.
   DOPOL    L         If true then correct for feed polarization
                      based on antenna file info.
   DOSMTH   L         True if smoothing requested.
   DOACOR   L         True if autocorrelations wanted (false)
   DOXCOR   L         True if cross-correlations wanted (true)
   DOWTCL   L         True if weight calibration wanted.
   DOFQSL   L         True if FREQSEL random parm present (false)
   FRQSEL   I         Default FQ table entry to select (-1)
   SELBAN   R         Bandwidth (Hz) to select (-1.0)
   SELFRQ   R         Frequency (Hz) to select (-1.0)
   DOBAND   I         >0 if bandpass calibration. (-1)
   DOSMTH   L         True if smoothing requested. (false)
   SMOOTH   R(3)      Smoothing parameters (0.0s)
   DXTIME   R         Integration time (days). Used when applying
                      delay corrections to correct for delay error.
   ANTENS   I(50)     List of antennas selected, 0=>all,
                      any negative => all except those specified
```

```
   SUBARR   I         Subarray desired, 0=>all
   FGVER    I         FLAG file version number, if < 0 then
                      NO flagging is applied. 0 => use highest
                      numbered table.
   CLUSE    I         Cal (CL or SN) file version number to apply.
   BLVER    I         BL Table to apply .le. 0 => none
   BPVER    I         BP table to apply .le. 0 => none

Public functions:
   SECGET (name, keywrd, type, dim, value, valuec, ierr)
      Return member.
   SECPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store  member.
   SECINI (name, ierr)
      Initializes default values (mostly no sel. edit, cal.)
   SECSLT (name, bif, eif, bchan, echan, stokes, ierr)
      Returns IF, channel and Stokes selection.
   SECSAV (name, bif, eif, bchan, echan, stokes, ierr)
      Resets IF, channel and Stokes selection.
Private to UV data class:
   SECSET (name, ierr)
      Copy current sel/edt/cal criteria to common
```

Appendix A

IMCOP

Example task IMCOP will copy one image to another with subimaging using an IMAGE class function IMCOPY. The main body of the tast does little more that create the necessary objects, call the library routine and the copy and update the history.

The names and descriptions of the POPS adverbs passed are declared and DATAed in the LOCAL INCLUDEs INPUT.INC and INPUTDATA.INC. These arrays are passed to AV2INP which returns an input object. The initialization routine IMCOIN then creates the relevant objects and copies adverb values from the input object to these objects using lists in DATA statments and calls to IN2OBJ.

```
LOCAL INCLUDE 'INPUT.INC'
C                                     Declarations for inputs
      INTEGER NPARMS
      PARAMETER (NPARMS=10)
      INTEGER AVTYPE(NPARMS), AVDIM(2,NPARMS)
      CHARACTER AVNAME(NPARMS)*8
LOCAL END
LOCAL INCLUDE 'INPUTDATA.INC'
C                                     DATA statments defining input
C                                     parameters.
      INCLUDE 'INCS:PAOOF.INC'
      DATA AVNAME /'INNAME', 'INCLASS', 'INSEQ', 'INDISK',
     *   'BLC', 'TRC',
     *   'OUTNAME', 'OUTCLASS', 'OUTSEQ', 'OUTDISK'/
      DATA AVTYPE /OOACAR, OOACAR, OOAINT, OOAINT,
     *   OOAINT, OOAINT,
     *   OOACAR, OOACAR, OOAINT, OOAINT/
      DATA AVDIM /12,1, 6,1, 1,1, 1,1,
     *   7,1, 7,1,
     *   12,1, 6,1, 1,1, 1,1/
LOCAL END

      PROGRAM IMCOP
C-----------------------------------------------------------------------
C! Copies image with subimaging using Image class library
C# Task MAP-UTIL AP OOP
C   This software is the subject of a User agreement and is confidential
C   in nature. It shall not be sold or otherwise made available or
C   disclosed to third parties.
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
      CHARACTER PRGM*6, IN*32, OUT*32
      INTEGER  IRET, BUFF1(256)
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DHDR.INC'
      DATA PRGM /'IMCOP'/
C-----------------------------------------------------------------------
C                                     Startup
      CALL IMCOIN (PRGM, IN, OUT, IRET)
      IF (IRET.NE.0) GO TO 990
C                                     Copy using IMAGE class library
C                                     routine IMCOPY
      CALL IMCOPY (IN, OUT, IRET)
      IF (IRET.NE.0) GO TO 990
C                                     History
```

```
      CALL IMCOHI (IN, OUT)
***??? copy CC tables
C                                       Close down files, etc.
 990  CALL DIE (IRET, BUFF1)
C
 999  STOP
      END

      SUBROUTINE IMCOIN (PRGN, IN, OUT, IRET)
C-----------------------------------------------------------------------
C   IMCOIN gets input parameters for IMCOP and creates the output.
C   Inputs:
C      PRGN    C*6  Program name
C   Output:
C      IN      C*?  Input object
C      OUT     C*?  Output object, only name filled in.
C      IRET    I    Error code: 0 => ok
C-----------------------------------------------------------------------
      INTEGER   IRET
      CHARACTER PRGN*6, IN*(*), OUT*(*)
C
      INCLUDE 'INCS:PUVD.INC'
      INTEGER   NKEY1, NKEY2
C                                       NKEY1=no. adverbs for IN
      PARAMETER (NKEY1=6)
C                                       NKEY2 = no. adverb for OUT
      PARAMETER (NKEY2=4)
      CHARACTER INK1(NKEY1)*8, OUTK1(NKEY1)*32, INK2(NKEY2)*8,
     *   OUTK2(NKEY2)*32
      INCLUDE 'INPUT.INC'
      INCLUDE 'INPUTDATA.INC'
C                                       Adverbs for IN
C                    1          2        3        4        5      6
      DATA INK1 /'INNAME', 'INCLASS', 'INSEQ', 'INDISK', 'BLC', 'TRC'/
C                                       Rename
C                    1       2       3        4       5      6
      DATA OUTK1 /'NAME', 'CLASS', 'IMSEQ', 'DISK', 'BLC', 'TRC'/
C                                       Adverbs for OUT
C                    1          2          3          4
      DATA INK2 /'OUTNAME', 'OUTCLASS', 'OUTSEQ', 'OUTDISK'/
C                                       Rename
C                   1       2         3       4
      DATA OUTK2 /'NAME', 'CLASS', 'IMSEQ', 'DISK'/
C-----------------------------------------------------------------------
C                                       Startup
      CALL AV2INP (PRGN, NPARMS, AVNAME, AVTYPE, AVDIM, 'Input', IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Create IN
      IN = 'Input image object'
      CALL CREATE (IN, 'IMAGE', IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Copy adverbs to object
      CALL IN2OBJ ('Input', NKEY1, INK1, OUTK1, IN, IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Open and close to check that
C                                       it's OK
      CALL OOPEN (IN, 'READ', IRET)
      IF (IRET.NE.0) GO TO 999
      CALL OCLOSE (IN, IRET)
```

```
      IF (IRET.NE.0) GO TO 999
C                                          Create output object
      OUT = 'Output image'
      CALL CREATE (OUT, 'IMAGE', IRET)
      IF (IRET.NE.0) GO TO 999
C                                          Copy adverbs to object
      CALL IN2OBJ ('Input', NKEY2, INK2, OUTK2, OUT, IRET)
      IF (IRET.NE.0) GO TO 999
C
 999  RETURN
      END

      SUBROUTINE IMCOHI (IN, OUT)
C-----------------------------------------------------------------------
C   Routine to write history file to output.
C   Inputs:
C      IN      C*?  Input object
C      OUT     C*?  Output object
C-----------------------------------------------------------------------
      CHARACTER IN*(*), OUT*(*)
C
      INTEGER   NADV
      PARAMETER (NADV=6)
      CHARACTER LIST(NADV)*8
      INTEGER   IERR
      INCLUDE 'INCS:DMSG.INC'
C                                          Adverbs to copy to history
      DATA LIST /'INNAME', 'INCLASS', 'INSEQ', 'INDISK', 'BLC', 'TRC'/
C-----------------------------------------------------------------------
C                                           Copy old history
      CALL OHCOPY (IN, OUT, IERR)
      IF (IERR.NE.0) GO TO 990
C                                           New additions - copy adverb
C                                           values.
      CALL OHLIST ('Input', LIST, NADV, OUT, IERR)
      IF (IERR.NE.0) GO TO 990
      GO TO 999
C                                          Error
 990  MSGTXT = 'ERROR WRITING HISTORY FOR ' // OUT
      CALL MSGWRT (6)
 999  RETURN
      END
```

Appendix B

TBTSK

TBTSK is a paraform task for manipulating tables. The example shown copies a range of rows from one table to another.

```
LOCAL INCLUDE 'INPUT.INC'
C                                       Declarations for inputs
      INTEGER NPARMS
C                                       NPARMS=no. adverbs passed.
      PARAMETER (NPARMS=10)
      INTEGER AVTYPE(NPARMS), AVDIM(2,NPARMS)
      CHARACTER AVNAME(NPARMS)*8
LOCAL END
LOCAL INCLUDE 'INPUTDATA.INC'
C                                       DATA statments defining input
C                                       parameters.
C                                       NOTE: Uses values in PAOOF.INC
C                                       Adverb names
C                     1        2         3          4        5
      DATA AVNAME /'USERID', 'INNAME', 'INCLASS', 'INSEQ', 'INDISK',
C           6          7        8          9         10
     *   'INEXT', 'INVERS', 'OUTVERS',  'BCOUNT' ,'ECOUNT'/
C                                       Adverb data types (PAOOF.INC)
C                    1        2      3       4        5       6
      DATA AVTYPE /OOAINT, OOACAR, OOACAR, OOAINT, OOAINT, OOACAR,
C          7       8       9       10
     *   OOAINT, OOAINT, OOAINT, OOAINT/
C                                       Adverb dimensions (as 2D)
C                  1    2    3    4    5    6    7    8    9
      DATA AVDIM /1,1, 12,1, 6,1, 1,1, 1,1, 2,1, 1,1, 1,1, 1,1, 1,1/
LOCAL END
      PROGRAM TBTSK
C-----------------------------------------------------------------------
C! Paraform AIPS OOP task processing a table.
C# Calibration VLBI
C   This software is the subject of a User agreement and is confidential
C   in nature. It shall not be sold or otherwise made available or
C   disclosed to third parties.
C-----------------------------------------------------------------------
C   Paraform AIPS OOP task processing a table.
C-----------------------------------------------------------------------
      CHARACTER PRGM*6, INTAB*36, OUTTAB*36
      INTEGER  IRET, BUFF1(256)
      DATA PRGM /'TBTSK '/
C-----------------------------------------------------------------------
C                                       Startup
      CALL TABTIN (PRGM, INTAB, OUTTAB, IRET)
C                                       Process table
      IF (IRET.EQ.0) CALL TABTAB (INTAB, OUTTAB, IRET)
C                                       History
      IF (IRET.EQ.0) CALL TABTHI (OUTTAB)
C                                       Close down files, etc.
 990  CALL DIE (IRET, BUFF1)
C
 999  STOP
      END

      SUBROUTINE TABTIN (PRGN, INTAB, OUTTAB, IRET)
```

```
C-----------------------------------------------------------------------
C   TABTIN gets input parameters for TBTSK and creates the input and
C   output objects
C   Inputs:
C      PRGN    C*6  Program name
C   Output:
C      IRET    I    Error code: 0 => ok
C                               4 => user routine detected error.
C                               5 => catalog troubles
C                               8 => can't start
C   Commons: /INPARM/ all input adverbs in order given by INPUTS
C                     file
C-----------------------------------------------------------------------
      INTEGER   IRET
      CHARACTER PRGN*6, INTAB*36, OUTTAB*36
C
      INTEGER   NKEY1, NKEY2
C                                       NKEY1=no. adverbs to copy to
C                                       INTAB
      PARAMETER (NKEY1=9)
C                                       NKEY2=no. adverbs to copy to
C                                       OUTTAB
      PARAMETER (NKEY2=7)
      CHARACTER INK1(NKEY1)*8, OUTK1(NKEY1)*32, INK2(NKEY2)*8,
     *   OUTK2(NKEY2)*32
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INPUT.INC'
      INCLUDE 'INCS:PAOOF.INC'
      INCLUDE 'INPUTDATA.INC'
C                                       Adverbs to copy to INTAB
C                    1         2          3        4         5
      DATA INK1 /'USERID', 'INNAME', 'INCLASS', 'INSEQ', 'INDISK',
C          6        7         8         9
     *   'INEXT', 'INVERS', 'BCOUNT', 'ECOUNT'/
C                                       May rename adverbs to INTAB
C                    1         2       3         4      5        6
      DATA OUTK1 /'USERID', 'NAME', 'CLASS', 'IMSEQ', 'DISK', 'TBLTYPE',
C         7       8         9
     *   'VER', 'BCOUNT', 'ECOUNT'/
C                                       Adverbs to copy to OUTTAB
C                    1         2          3        4         5
      DATA INK2 /'USERID', 'INNAME', 'INCLASS', 'INSEQ', 'INDISK',
C          6         7
     *   'INEXT', 'OUTVERS'/
C                                       May rename adverbs to OUTTAB
C                    1         2       3        4      5       6
      DATA OUTK2 /'USERID', 'NAME', 'CLASS', 'IMSEQ', 'DISK', 'TBLTYPE',
C         7
     *   'VER'/
C-----------------------------------------------------------------------
C                                       Startup,  returns "Input" object
C                                       containing POPS adverbs
      CALL AV2INP (PRGN, NPARMS, AVNAME, AVTYPE, AVDIM, 'Input', IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Create input object
      INTAB = 'Input table'
      CALL CREATE (INTAB, 'TABLE', IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Copy adverbs to object
```

```
      CALL IN2OBJ ('Input', NKEY1, INK1, OUTK1, INTAB, IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Create Output Object
      OUTTAB = 'Output table'
      CALL CREATE (OUTTAB, 'TABLE', IRET)
      IF (IRET.NE.0) GO TO 999
C                                       Copy adverbs to object
      CALL IN2OBJ ('Input', NKEY2, INK2, OUTK2, OUTTAB, IRET)
      IF (IRET.NE.0) GO TO 999
C
 999  RETURN
      END

      SUBROUTINE TABTAB (INTAB, OUTTAB, IERR)
C-----------------------------------------------------------------------
C   Convert table.
C   This example simply copys one table to another
C   Inputs:
C      INTAB   C*   Name of input table object.
C      OUTTAB  C*   Name of output table object.
C   Output:
C      IERR    I    Error code: 0 => ok
C-----------------------------------------------------------------------
      CHARACTER INTAB*(*), OUTTAB*(*)
      INTEGER   IERR
C
C
      INTEGER   MAXSIZ
C                                       MAXSIZ = max table entry size as
C                                       reals or characters
      PARAMETER (MAXSIZ = 5000)
      INTEGER   IROW, OROW, NROW, ICOL, NCOL, BC, EC, TYPE, DIM(3),
     *   DUMMY
      REAL      NVALS(MAXSIZ)
      CHARACTER CVALS*(MAXSIZ), CDUMMY*1
C-----------------------------------------------------------------------
C                                       Example: copy a subset of one
C                                       table to another.
C
C                                       Create output table
C                                       This copies header stuff
C                                       including any keywords.
      CALL COPHED (INTAB, OUTTAB, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Open input table
      CALL OOPEN (INTAB, 'READ', IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Open output table
      CALL OOPEN (OUTTAB, 'WRIT', IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get number of entries
      CALL OGET (INTAB, 'NROW', TYPE, DIM, NROW, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Number of columns
      CALL OGET (INTAB, 'NCOL', TYPE, DIM, NCOL, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get range of rows.
      CALL OGET (INTAB, 'BCOUNT', TYPE, DIM, BC, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
```

```
      BC = MIN (MAX (BC, 1), NROW)
      CALL OGET (INTAB, 'ECOUNT', TYPE, DIM, EC, CDUMMY, IERR)
      IF (IERR.NE.0) GO TO 999
      IF (EC.LE.0) EC = NROW
C                                       Copy selected rows.
      OROW = 0
      DO 100 IROW = BC,EC
         OROW = OROW + 1
         DO 50 ICOL = 1,NCOL
            CALL TABDGT (INTAB, IROW, ICOL, TYPE, DIM, NVALS, CVALS,
     *         IERR)
            IF (IERR.NE.0) GO TO 999
            CALL TABDPT (OUTTAB, OROW, ICOL, TYPE, DIM, NVALS, CVALS,
     *         IERR)
            IF (IERR.NE.0) GO TO 999
 50         CONTINUE
 100     CONTINUE
C                                       Close tables
      CALL OCLOSE (INTAB, IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OCLOSE (OUTTAB, IERR)
      IF (IERR.NE.0) GO TO 999
C
 999  RETURN
      END

      SUBROUTINE TABTHI (OUTTAB)
C-----------------------------------------------------------------------
C   Routine to write history file to output table object.  This assumes
C   that a previous history exists and merely adds the information from
C   the current task.
C   Inputs:
C      OUTTAB  C*?  Output table object
C-----------------------------------------------------------------------
      CHARACTER OUTTAB*(*)
C
      INTEGER   NADV
      PARAMETER (NADV=8)
      CHARACTER LIST(NADV)*8
      INTEGER   IERR
      INCLUDE 'INCS:DMSG.INC'
C                                       Adverbs to copy to history
      DATA LIST /'INNAME', 'INCLASS', 'INSEQ', 'INEXT', 'INVERS',
     *   'OUTVERS',  'BCOUNT' ,'ECOUNT'/
C-----------------------------------------------------------------------
C                                       Add task label to history
      CALL OHTIME (OUTTAB, IERR)
      IF (IERR.NE.0) GO TO 990
C                                       Copy adverb values.
      CALL OHLIST ('Input', LIST, NADV, OUTTAB, IERR)
      IF (IERR.NE.0) GO TO 990
      GO TO 999
C                                       Error
 990  MSGTXT = 'ERROR WRITING HISTORY FOR ' // OUTTAB
      CALL MSGWRT (4)
C
 999  RETURN
      END
```