Object-Oriented Programming in AIPS Fortran

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

1 June 1992

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.

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 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.

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 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.

5 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.

5.1 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.

5.1.1 CREATE

Creates an image 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
```

5.1.2 DESTRY

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
```

5.1.3 ZAP

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
```

5.1.4 OCOPY

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
```

5.1.5 OCLONE

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
```

5.1.6 OOPEN

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
```

5.1.7 OCLOSE

Closes object updating disk resident information.

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

5.1.8 OGET

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)
                  Note: this is passed as a 1-D character array
                  even for multidimensional string arrays.
   IERR     I     Error return code, 0=OK
```

5.1.9 OPUT

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)
                  Note: this is passed as a 1-D character array
                  even for multidimensional string arrays.
Outputs:
   IERR     I     Error return code, 0=OK
```

5.2 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. A number of image operations have been defined and implemented. Images may consist of real or complex (scratch objects only) pixels and may be blanked.

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)
      CHARACTER IN*(*), OUT*36
      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, 'ROW',
     *   'ROW', IERR)
      IF (IERR.NE.0) GO TO 999
      CALL OPUT (OUT, 'ARRAY.ARRAY_PNT.ACCESS', OOACAR, DIM, 'ROW',
     *   'ROW', IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Determine size
      CALL OGET (IN, 'ARRAY.ARRAY_DESC.NAXIS', TYPE, DIM, NAXIS,
     *   NAXIS, 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
```

5.3 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.

5.4 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, OOARE = real, OOADP = double precision and OOACAR = character strings.

5.5 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
      CHARACTER IN*(*), OUT*36
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, NROW, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get number of columns
      CALL OGET (IN, 'NCOL', TYPE, DIM, NCOL, NCOL, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get range of rows from input
C                                       object.
      CALL OGET (IN, 'BCOUNT', TYPE, DIM, BC, BC, IERR)
      IF (IERR.NE.0) GO TO 999
      BC = MIN (MAX (BC, 1), NROW)
      CALL OGET (IN, 'ECOUNT', TYPE, DIM, EC, EC, 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
```

6 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 current contents of the $QOOP directory are QARRAY.FOR and QIMAGE.FOR which contain those routines in the ARRAY and IMAGE class which use "Q" routines. The routines in these files do not need to and cannot access the private data of their class. Task using the AIPS object oriented routines 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.

7 Examples

Examples of the use of these classes are shown in two AIPS tasks. The first, IMTST, does several unary and binary operations on images including a convolution. 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. The texts of these tasks are included as appendices. TBTSK is included in the standard distribution. Other examples of use are given in the section discussing the class interfaces.

8 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.

9 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.

9.1 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   MAXOBJ, MAXSIZ, MAXIO, BUFSIZ, MAXCLS, MAXVKW
C                                        MAXOBJ=max. number of objects
      PARAMETER (MAXOBJ = 20)
C                                        MAXSIZ=max. number words of
C                                         object memory.
```

```
      PARAMETER (MAXSIZ = 8192)
C                                       MAXIO=max. number of I/O
C                                       streams.
      PARAMETER (MAXIO = 5)
C                                       BUFSIZ=Size of I/O buffer
      PARAMETER (BUFSIZ = 8192)
C                                       MAXCLS = the number of
C                                       classes defined.
      PARAMETER (MAXCLS = 5)
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. The text follows:

```
C                                                      Include CLASSIO.
C                                      Class I/O include for
C                                      AIPS Object Oriented Fortran
C                                      system.
      INTEGER   OBJLUN(MAXOBJ), OBJFIN(MAXOBJ), BUFPNT(MAXOBJ)
      LOGICAL   OBNUSE(MAXIO)
      REAL      OBUFFR(BUFSIZ,MAXIO), SBUFF(512)
      COMMON /OBJBUF/ OBJLUN, OBJFIN, BUFPNT, OBNUSE, OBUFFR, SBUFF
C                                                      End CLASSIO.
```

9.2 Object manager

```
   Private data:
     MAXOBJ  PI                  Maximum number of simultaneous
                                 objects (parameter).
     MAXSIZ   PI                 Number of words of memory for an
                                 object. (parameter)
     NAMTAB  C*32(MAXOBJ)        Object name table, position gives
                                 object slot number.
     OBJMEM   ?(MAXSIZ, MAXOBJ)  Object memory, an association list,
                                 referenced under the aliases OBJMEI,
                                 OBJMER, OBJMEH, OBJMEL for I,R,C and
                                 L. D values must be copied.
                                 OBJMEM has the following structure:
          Word  Type       Contents
          ----  ----  --------------------------------------------
           0     I    Address of next cell, .LE. 0 => last entry.
           1     H*8  Keyword as HOLLERITH
           3     I    Type: 1=D, 2=R, 3=C, 4=I, 5=L
           4     I    First dimension (length of string for char)
           5     I    Second dimension
           6     ?    Data, characters as Hollerith
          ...         As many words as necessary for the data.

     OBJFRE   I(MAXOBJ)    Next free word in OBJMEM
     OBJCAT   I(256,MAXOBJ)Array of catalog header records
     OBJBUF   I(MAXOBJ)    Buffer number for associated I/O stream.
                           <0   => not active.
     LUNUSE   L(100)       True if LUN=index allocated.
     OBJCLS   C(MAXOBJ)*8  Object class
     MEMNXT   PI           OBJMEM offset for next cell pointer.
     MEMKEY   PI           OBJMEM offset for keyword
     MEMTYP   PI           OBJMEM offset for data type
```

```
 MEMDM1   PI           OBJMEM offset for 1st dimension
 MEMDM2   PI           OBJMEM offset for 2nd dimension
 MEMDAT   PI           OBJMEM offset for start of data
 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

Public functions:
   OBINIT (ierr)
      Initialize Object manager.

Private functions:
  INVINI (ierr)
```

```
      Initialize virtual keywords for inputs class.
   IMVINI (ierr)
      Initialize virtual keywords for image class.
   OBKEYW (objnum, keywrd, keypnt, ierr)
      Lookup keyword inOBJMEM; called by OBGET and OBPUT.
   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 one image to another.
   OBOPEN (name, ierr)
      Assigns a buffer.
   OBCLOS (name, ierr)
      Closes a buffer associated with an object.
```

9.3 History

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.
```

9.4 Array Class

Array Class: name "ARRAY"

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, ierr)
      Setup for I/O to an 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, out, ierr)
      FFT an array.
   ARRADD (in1, in2, out, ierr)
      Adds two arrays.
   ARRMUL (in1, in2, out, ierr)
      Multiplies two arrays.
   ARRPAD (in, out, ierr)
      Copy one array to another with zero padding.
```

```
   ARRSCL (in, factor, out, ierr)
      Scale an array with a factor.
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.
```

9.5 Array descriptor Class

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.
```

9.6 Array Pointer class

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.
```

9.7 Array Statistics Class

Array Statistics class: name="ARRAY_STAT"

The array statistics class provides various statistical data about an array.

```
Class data:
   DATAMAX  R    Maximum value
   PIXMAX   I(*) Position of maximum value
   DATAMIN  R    Mininum value
```

```
   PIXMIN   I(*) Position of minimum value
   DATARMS  R    RMS value
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.
```

9.8 Convolving Beam Class

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
   BMIN      R   Clean beam minor axis
   BPA       R   Clean beam position angle
Public functions:
   BEMGET (name, keywrd, type, dim, value, valuec, ierr)
      Return beam member.
   BEMPUT (name, keywrd, type, dim, value, valuec, ierr)
      Store beam member.
```

9.9 Complex Image Class

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.
```

9.10 File Name Class

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.
```

9.11 File Status Class

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.
```

9.12 Image Class

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.

```
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, SEQ and DISK associated with ``name''.
     Creates the file if necessary.
  IMCSET (name, status, ierr)
     Sets any file status)
  IMCCLR (name, ierr)
     Clears any file status)
  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
  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
```

9.13 Image Descriptor Class

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
   DATA-OBS C*8  Observing date as dd/mm/yy
   DATA-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
   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.
```

9.14 Inputs Class

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.
  INGET (name, keywrd, type, dim, value, valuec, ierr)
     Return adverb value.
  INPUTS (name, keywrd, type, dim, value, valuec, ierr)
     Store adverb value.
```

9.15 Position Class

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.
```

9.16 Table Class

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
  SEQ       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.
  TABCOP (namein, namout, ierr)
     Copys one object to another.  The same quasi permanent forms
     are used for both.
```

```
  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.

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.
```

9.17 Vector Class

Vector class: Name “VECTOR”

A vector is a one dimensional array of numeric values. Blanking is supported. Arguments INn, and OUT are Fortran arays, 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
CVSCL (IN1, FACTOR, OUT, N)
   Complex Scale vector with real
RVBSCL (IN1, VALIN1, FACTOR, OUT, VALOUT, N)
   Real Scale vector with real with blanking
CVBSCL (IN1, VALIN1, FACTOR, OUT, VALOUT, N)
   Complex Scale vector with real with blanking
```

9.18 Velocity Class

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.
```

Appendix A

IMTST

Task IMTST will do one of a number of simple image operations depending on the value of the POPS adverb 'OPCODE'. One option, 'TEST', negates an image and then adds it to the original allowing a easy test for correctness since all valid pixels in the output should be zero.

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 TAFIN 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 'IMTST.INC'
C                                       Local include for IMTST
      CHARACTER OPCODE*4, INPUT1*36, INPUT2*36, OUTPUT*36
      COMMON /CHPARM/ OPCODE, INPUT1, INPUT2, OUTPUT
LOCAL END
LOCAL INCLUDE 'INPUT.INC'
C                                       Declarations for inputs
      INTEGER NPARMS
      PARAMETER (NPARMS=16)
      INTEGER AVTYPE(NPARMS), AVDIM(2,NPARMS)
      CHARACTER AVNAME(NPARMS)*8
LOCAL END
LOCAL INCLUDE 'INPUTDATA.INC'
C                                       DATA statments defining input
C                                       parameters.
C                                       Uses PAOOF.INC
C                    1          2          3          4          5
      DATA AVNAME /'INNAME', 'INCLASS', 'INSEQ', 'INDISK', 'IN2NAME',
C            6           7          8           9           10
     *   'IN2CLASS', 'IN2SEQ', 'IN2DISK', 'OUTNAME', 'OUTCLASS',
C           11          12       13      14        15         16
     *   'OUTSEQ', 'OUTDISK', 'BLC', 'TRC', 'OPCODE', 'BADDISK'/
C                   1        2        3        4        5        6
      DATA AVTYPE /OOACAR, OOACAR, OOAINT, OOAINT, OOACAR, OOACAR,
C          7        8        9        10       11       12       13
     *   OOAINT, OOAINT, OOACAR, OOACAR, OOAINT, OOAINT, OOAINT,
C          14       15       16
     *   OOAINT, OOACAR, OOAINT/
C                   1    2    3    4     5    6    7    8
      DATA AVDIM /12,1, 6,1, 1,1, 1,1, 12,1, 6,1, 1,1, 1,1,
C          9   10   11   12   13   14   15   16
     *   12,1, 6,1, 1,1, 1,1, 7,1, 7,1, 4,1, 10,1/
LOCAL END
      PROGRAM IMTST
C-----------------------------------------------------------------------
C! Test bed for image class software
C# Map-util Utility
C   This software is the subject of a User agreement and is confidential
C   in nature. It shall not be sold or otherwise made available or
C   disclosed to third parties.
C-----------------------------------------------------------------------
C   IMTST is a testbed for prototype image class.
C   Inputs:
C      AIPS adverb  Prg. name.          Description.
C      INNAME         NAMEIN        Name of input image.
C      INCLASS        CLAIN         Class of input image.
```

```
C      INSEQ          SEQIN         Seq. of input image.
C      INDISK         DISKIN        Disk number of input image.
C      OUTNAME        NAMOUT        Name of the output image
C                                   Default output is input image.
C      OUTCLASS       CLAOUT        Class of the output image.
C                                   Default is input class.
C      OUTSEQ         SEQOUT        Seq. number of output image.
C      OUTDISK        DISKO         Disk number of the output image.
C      BLC(7)         BLC           Bottom left corner of subimage
C                                   of input image.
C      TRC(7)         TRC           Top right corner of subimage.
C      OPCODE         OPCODE        User specified opcode.
C-----------------------------------------------------------------------
      CHARACTER PRGM*6
      INTEGER  IRET
      INTEGER   INAX(7), IBLC(7), ITRC(7), BUFF1(256)
      REAL      FACTOR
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'IMTST.INC'
      DATA PRGM /'IMTST '/
C-----------------------------------------------------------------------
C                                       Startup
      CALL IMTIN (PRGM, IRET)
      FACTOR = 1.0
C                                       Operate on images.
      IF (OPCODE.EQ.'ADD ') CALL IMGADD (INPUT1, INPUT2, OUTPUT, IRET)
      IF (OPCODE.EQ.'NEG ') CALL IMGNEG (INPUT1, OUTPUT, IRET)
      IF (OPCODE.EQ.'CONV') CALL IMGCVL (INPUT1, INPUT2, FACTOR, OUTPUT,
     *   IRET)
      IF (OPCODE.EQ.'TEST') THEN
C                                       Negate an image and add it to
C                                       the original.
         CALL IMGWIN (INPUT1, IBLC, ITRC, INAX, IRET)
         IF (IRET.NE.0) GO TO 990
         CALL IMGSCR ('SCRATCH', INAX, IRET)
         IF (IRET.NE.0) GO TO 990
         CALL IMGNEG (INPUT1, 'SCRATCH', IRET)
         IF (IRET.NE.0) GO TO 990
         CALL IMGADD (INPUT1, 'SCRATCH', OUTPUT, IRET)
         IF (IRET.NE.0) GO TO 990
         END IF
C                                       History
      IF (IRET.EQ.0) CALL IMTHIS (INPUT1, INPUT2, OUTPUT)
C                                       Close down files, etc.
 990  CALL DIE (IRET, BUFF1)
C
 999  STOP
      END
      SUBROUTINE IMTIN (PRGN, IRET)
C-----------------------------------------------------------------------
C   IMTIN gets input parameters for IMTST 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-----------------------------------------------------------------------
      INTEGER   IRET
      CHARACTER PRGN*6
C
      INTEGER   NOPCO, NKEY1, NKEY2
C                                       NOPCO=number of Opcodes
      PARAMETER (NOPCO=4)
C                                       NKEY1=no. adverbs to copy to
C                                       INPUT1
      PARAMETER (NKEY1=10)
C                                       NKEY2=no. adverbs to copy to
C                                       INPUT2
      PARAMETER (NKEY2=6)
      INTEGER   IERR, LOOP, ICODE, DIM(7), TYPE
      LOGICAL   ISUNAR(NOPCO)
      CHARACTER OPCO(NOPCO)*4, INK1(NKEY1)*8, OUTK1(NKEY1)*32,
     *   INK2(NKEY2)*8, OUTK2(NKEY2)*32
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'IMTST.INC'
      INCLUDE 'INPUT.INC'
      INCLUDE 'INCS:PAOOF.INC'
      INCLUDE 'INPUTDATA.INC'
C                                       ISUNAR = .true. if OPCODE is a
C                                       unary function.
C                    1        2        3
      DATA ISUNAR / .FALSE., .TRUE., .TRUE., .FALSE./
      DATA OPCO /'ADD ', 'NEG ', 'TEST', 'CONV'/
C                                       Adverbs to copy to image objects
C                    1         2         3          4      5      6
      DATA INK1 /'INNAME', 'INCLASS', 'INSEQ', 'INDISK', 'BLC', 'TRC',
C            7          8          9         10
     *   'OUTNAME', 'OUTCLASS',  'OUTSEQ', 'OUTDISK'/
C                        1                 2
      DATA OUTK1 /'FILE_NAME.NAME', 'FILE_NAME.CLASS',
C                 3                 4                    5
     *   'FILE_NAME.IMSEQ', 'FILE_NAME.DISK',  'ARRAY.ARRAY_DESC.BLC',
C                   6                7           8           9
     *   'ARRAY.ARRAY_DESC.TRC', 'OUTNAME', 'OUTCLASS', 'OUTSEQ',
C          10
     *   'OUTDISK'/
C                     1          2          3           4      5
      DATA INK2 /'IN2NAME', 'IN2CLASS', 'IN2SEQ', 'IN2DISK', 'BLC',
C          6
     *   'TRC'/
C                        1                 2
      DATA OUTK2 /'FILE_NAME.NAME', 'FILE_NAME.CLASS',
C                 3                 4                    5
     *   'FILE_NAME.IMSEQ', 'FILE_NAME.DISK',  'ARRAY.ARRAY_DESC.BLC',
C                   6
     *   'ARRAY.ARRAY_DESC.TRC'/
C-----------------------------------------------------------------------
C                                       Startup
      CALL AV2INP (PRGN, NPARMS, AVNAME, AVTYPE, AVDIM, 'Input', IRET)
      IF (IRET.NE.0) GO TO 990
C                                       BADDISK
      CALL OGET ('Input', 'BADDISK', TYPE, DIM, IBAD, IBAD, IRET)
      IF (IRET.NE.0) GO TO 990
C                                       Lookup OPCODE
```

```
      CALL OGET ('Input', 'OPCODE', TYPE, DIM, OPCODE, OPCODE, IRET)
      IF (IRET.NE.0) GO TO 990
      ICODE = -1
      DO 200 LOOP = 1,NOPCO
         IF (OPCODE.EQ.OPCO(LOOP)) ICODE = LOOP
 200     CONTINUE
C                                        Recognized OPCODE
      IF (ICODE.LE.0) THEN
         IRET = 1
         MSGTXT = 'UNKNOWN OPCODE: ' // OPCODE
         GO TO 990
         END IF
C                                        Create input objects
      INPUT1 = 'Image 1'
      CALL IMGCRE (INPUT1, IRET)
      IF (IRET.NE.0) GO TO 999
C                                        Copy adverbs to object
      CALL IN2OBJ ('Input', NKEY1, INK1, OUTK1, INPUT1, IERR)
C                                        Second input object if binary
C                                        function.
      IF (.NOT.ISUNAR(ICODE)) THEN
         INPUT2 = 'Image 2'
         CALL IMGCRE (INPUT2, IRET)
         IF (IRET.NE.0) GO TO 999
C                                        Copy adverbs to object
         CALL IN2OBJ ('Input', NKEY2, INK2, OUTK2, INPUT2, IERR)
         END IF
C                                        Create Output Object - attached
C                                        output file naming to INPUT1
      OUTPUT = 'Output'
      CALL OCLONE (INPUT1, OUTPUT, IRET)
      IF (IRET.NE.0) GO TO 999
      GO TO 999
C
 990  CALL MSGWRT (8)
C
 999  RETURN
      END

      SUBROUTINE IMTHIS (INPUT1, INPUT2, OUTPUT)
C-----------------------------------------------------------------------
C   Routine to write history file to output image object.
C   Inputs:
C      INPUT1  C*?  First input image
C      INPUT2  C*?  Second input image
C      OUTPUT  C*?  Output image
C-----------------------------------------------------------------------
      CHARACTER INPUT1*(*), INPUT2*(*), OUTPUT*(*)
C
      INTEGER   NADV
      PARAMETER (NADV=12)
      CHARACTER LIST(NADV)*8, HILINE*72
      INTEGER   IERR, TYPE, DIM(7)
      REAL      SUMQ, SUMU
      INCLUDE 'INCS:DMSG.INC'
C                                        Adverbs to copy to history
      DATA LIST /'INNAME', 'INCLASS', 'INSEQ', 'IN2NAME', 'IN2CLASS',
     *   'IN2SEQ', 'OUTNAME', 'OUTCLASS', 'OUTSEQ', 'BLC', 'TRC',
     *   'OPCODE'/
```

```
C-----------------------------------------------------------------------
C                                        Copy old history
      CALL OHCOPY (INPUT1, OUTPUT, IERR)
      IF (IERR.NE.0) GO TO 990
      CALL OHCOPY (INPUT2, OUTPUT, IERR)
      IF (IERR.NE.0) GO TO 990
C                                        New additions - copy adverb
C                                        values.
      CALL OHLIST ('Input', LIST, NADV, OUTPUT, IERR)
      IF (IERR.NE.0) GO TO 500
C                                       Error
 990  MSGTXT = 'ERROR WRITING HISTORY FOR ' // OUTPUT
      CALL MSGWRT (4)
 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', 'SEQ', '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', 'SEQ', '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)
      REAL      NVALS(MAXSIZ)
      CHARACTER CVALS*(MAXSIZ)
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, NROW, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Number of columns
      CALL OGET (INTAB, 'NCOL', TYPE, DIM, NCOL, NCOL, IERR)
      IF (IERR.NE.0) GO TO 999
C                                       Get range of rows.
      CALL OGET (INTAB, 'BCOUNT', TYPE, DIM, BC, BC, IERR)
      IF (IERR.NE.0) GO TO 999
      BC = MIN (MAX (BC, 1), NROW)
```

```
      CALL OGET (INTAB, 'ECOUNT', TYPE, DIM, EC, EC, 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)
 999  RETURN
      END
```