SUN/183.6

Starlink Project
Starlink User Note 183.6

D.S. Berry

2nd October 2007

ARD — A Textual Language for Describing Regions within a Data Array Version 2.2 Programmer's Manual

Abstract

The ARD (ASCII Region Definition) system provides a textual language for describing regions within a data array, together with software for converting a textual description into a pixel mask, or plotting it on a graphics device. The textual language is based on a set of keywords identifying simple shapes (boxes, circles, lines, etc.). These keywords can be combined together using Boolean-style operators (AND, OR, NOT, etc.) to create more complex shapes. Data arrays can be multi-dimensional.

Contents

1 Introduction

Astronomical applications often require the user to identify regions of interest within a data array. For instance a statistics application may need to be told the region within the input image in which it is to evaluate the pixel statistics. Another example is a data calibration application which needs to be told the regions in which the detector was unreliable so that it can flag the corresponding output pixels as bad.

One way of identifying such regions is through the use of a pixel mask. In a pixel mask, different pixel values are used to differentiate between those pixels which are to be included by the application and those which are to be excluded. Such pixel masks are usually the same size and shape as the data array being processed by the application (so that a one-for-one correspondence exists between mask pixels and data pixels). This results in such masks occupying large amounts of disk space. More importantly, it means that different pixel masks are required for data arrays with different sizes or shapes.

ARD circumvents these problems by using textual expressions (known as "ARD descriptions") to describe the pixels to be included by the application. Multi-dimensional data arrays can be handled. A simple 2-dimensional ARD description such as:

```
CIRCLE( 20, 20, 5 ) .OR. RECT( 20, 20, 30, 30 )
```

tells the application to process all pixels which are *either* within a circle centred on pixel coordinates (20,20) with a radius of 5 pixels, *or* are within the rectangle with opposite corners at pixels coordinates (20,20) and (30,30). The ARD_WORK subroutine will convert an ARD description such as this into a pixel mask, with a shape and size specified by the application. The application will usually know the shape and size of the data array and so can ask ARD_WORK to create a pixel mask of the correct shape and size. Once the pixel mask is no longer needed, the storage space used to hold the mask can be released; there is no need to keep permanent copies of the pixel mask on disk.

As well as creating pixel masks, the ARD library also provides facilities to plot an ARD description on a graphics device. The ARD_PLOT routine draws the borders of the regions described in the ARD description, using a supplied *AST Plot* (see SUN/210) to perform the graphics.

In the above example, positions and distances in the ARD description were given in pixel coordinates. The ARD language includes two systems which allow positions and distances to be given in other coordinate systems:

(1) As of ARD version 2.0, the calling application can define an arbitrary collection of coordinate systems (which need not be linearly related to pixel coordinates) by supplying an *AST FrameSet* (see SUN/210). The FrameSet contains information which allows positions to be mapped from any of these coordinate systems into pixel coordinates. Positions within the ARD description can then be given in any of these coordinates systems (a statement in the ARD description indicating which system is being used). Thus, for instance, if the pixel array has an RA/DEC calibration, the application could supply a FrameSet indicating how to convert from RA/DEC to pixel coordinates. This would allow the ARD description to include positions in RA/DEC.

In addition, the ARD description itself can include a FrameSet defining a collection of inter-related coordinate systems (positions in the ARD description should be given in the "current" Frame of this FrameSet). In this case, an attempt is made to find a coordinate system which is contained both within the FrameSet supplied by the calling application, and within the FrameSet supplied in the ARD description. For instance, extending the previous example, if an ARD description contains position given in pixel coordinates in some other specific image and also contains a FrameSet which relates pixel positions within that image to RA/DEC, then positions will be mapped from pixel coordinates within the original image, into RA/DEC, and then into pixel coordinates within the required mask image. This effectively allows pixel positions to be given within one image and then transformed so that they can be used within another image.

(2) The calling application can also define an "application coordinate system" which is linearly related to pixel coordinates. Positions within the ARD description can then either be given directly in application coordinates, or in any coordinate system linearly related to application coordinates (in which case the ARD description must include statements describing the linear transformation). This system was present in version 1 of ARD and is still present in the current version, but is now deprecated in favour of the above more general system.

1.1 Some Example ARD Descriptions

ARD descriptions are made up by using logical operators (.AND., .OR., .NOT., etc.) to combine together keywords which represent the basic shapes known to ARD. statements can also be included which modify the way the ARD description is interpreted (for instance, by setting up an alternative coordinate system).

The ARD library includes two subroutines (ARD_GROUP and ARD_GRPEX) which simplify the task of obtaining ARD descriptions from the user or environment. Using these routines, an ARD description can be supplied to an application either directly, or by storing it in a text file and supplying the name of the text file to the application. The following examples represent lines stored in a text file. Such lines are effectively concatenated together into a single string before being processed by ARD:

```
ROTBOX( 0, 0, 20, 10, 30 )
```

This example is simply a single keyword representing one of the basic shapes known to ARD. It selects all pixels which have centres on or within a 2-dimensional rotated box. The box is centred on coordinates (0,0) and has sides of length 20 and 10. The first side of the box (i.e. the one with length 20) is at an angle of 30° to the array X axis (measured anti-clockwise).

```
CIR( 0, 0, 10 ) .AND. .NOT. ( COLUMN( 10 ) .OR. ROW( 5 ) )
```

This example uses logical operators and parentheses to combine several basic shapes together into a more complex shape. It also shows the use of abbreviated keywords. All pixels within the circle of radius 10 centred on (0,0) are selected, except for those which are on column 10 or row 5.

```
COFRAME( SKY, System=FK5, Equinox=2003.5 )
BOX( 12:23:41, -89:14, 1h40m, 20m )
```

This example shows the use of statements to specify the coordinate system in which positions are supplied. In this case, the COFRAME statement indicates that positions are supplied in FK5 equatorial (RA/DEC) coordinates, referred to the equinox of 2003.5. The BOX keyword then selects a box centred at RA $12^h23^m41^s$ and Dec. $-89^{\rm deg}14'$. The box covers an RA range of 1^h40^m and a Dec range of $20'$. The edges of a BOX region are always lines of constant axis value. Since this region is very close to the south equatorial pole, the pixel region containing this "box" will have quite strongly curved sides.

```
PIXEL(  1,  1 )
PIXEL( 20, 13 )
PIXEL(-55,122 )
PIXEL(112, 87 )
PIXEL( 33, 12 )
```

This example demonstrates the ability of ARD to recognise implicit .OR. operators. The list of PIXEL keywords specifies a set of individual pixels. Since these keywords have no intervening operators, ARD assumes that a .OR. operator is to be inserted between each pair, i.e. the union of all the individual pixels is assumed.

1.2 An Example ARD Application

This section presents Fortran code for an ADAM application which obtains a data array (in the form of an NDF structure, see SUN/33) and an ARD description from the environment, converts the ARD description into a pixel mask, and then finds and displays the data sum within the region specified by the ARD description. The application will deal with NDFs of any dimensionality up to the limit imposed by the NDF_ system.

```
      SUBROUTINE ARD_TEST( STATUS )                              [1]
      IMPLICIT NONE

*  Include definitions of global constants.
      INCLUDE 'SAE_PAR'                                          [2]
      INCLUDE 'NDF_PAR'
      INCLUDE 'PRM_PAR'
      INCLUDE 'GRP_PAR'
      INCLUDE 'AST_PAR'

*  Declare local variables.
      INTEGER STATUS, IGRP, INDF, NDIM, IPDATA, IPMASK, EL,
     :        LBND( NDF__MXDIM ), UBND( NDF__MXDIM ),             [3]
     :        LBNDI( NDF__MXDIM ), UBNDI( NDF__MXDIM ),
     :        LBNDE( NDF__MXDIM ), UBNDE( NDF__MXDIM ),
     :        REGVAL
      REAL SUM, TRCOEF( 1 )
      INTEGER IWCS

*  Check inherited global status.
      IF ( STATUS .NE. SAI__OK ) RETURN                          [4]

*  Obtain an identifier for the input NDF.
      CALL NDF_ASSOC( 'NDF', 'READ', INDF, STATUS )              [5]
```

```
*  Obtain the bounds of the NDF.
      CALL NDF_BOUND( INDF, NDF__MXDIM, LBND, UBND, NDIM,       [6]
     :                STATUS )

*  Map the DATA component of the NDF.
      CALL NDF_MAP( INDF, 'DATA', '_REAL', 'READ', IPDATA, EL,  [7]
     :              STATUS )

*  Obtain an ARD description specifying the region in which
*  the pixel values are to be summed.
      CALL ARD_GROUP( 'REGION', GRP__NOID, IGRP, STATUS )       [8]

*  Obtain workspace to hold the pixel mask corresponding to
*  the supplied ARD description.
      CALL PSX_CALLOC( EL, '_INTEGER', IPMASK, STATUS )         [9]

*  Get an AST FrameSet describing the WCS coordinate Frames stored
*  in the NDF.
      CALL NDF_GTWCS( INDF, IWCS, STATUS )                     [10]

*  Indicate that positions within the ARD description can be given in
*  any coordinate Frame included in the above WCS FrameSet.
      CALL ARD_WCS( IWCS, ' ', STATUS )                             [11]

*  Indicate that the value 2 should be used to represent
*  pixels specified by the first keyword in the ARD
*  description.
      REGVAL = 2

*  Call ARD_WORK to store positive values at all mask pixels
*  specified by the ARD description, and zero at all other
*  pixels.
      CALL ARD_WORK( IGRP, NDIM, LBND, UBND, TRCOEF, .FALSE.,   [12]
     :               REGVAL, %VAL( IPMASK ), LBNDI, UBNDI,
     :               LBNDE,  UBNDE, STATUS )

*  Call a subroutine to sum the data in the specified regions.
      CALL SUMIT( EL, %VAL( IPDATA ), %VAL( IPMASK ), SUM,
     :            STATUS )                                      [13]

*  Display the data sum.
      CALL MSG_SETR( 'SUM', SUM )
      CALL MSG_OUT( 'ARD_TEST_MSG1', '  Data sum: ^SUM',        [14]
     :              STATUS )

*  Release the work space used to hold the pixel mask.
      CALL PSX_FREE( IPMASK, STATUS )                           [15]

*  Delete the group used to hold the ARD description.
      CALL GRP_DELET( IGRP, STATUS )                            [16]

*  Annul the AST FrameSet identifier.
      CALL AST_ANNUL( IWCS, STATUS )                            [17]
```

```
*  Annul the NDF identifier.
      CALL NDF_ANNUL( INDF, STATUS )                              [18]

      END
```

Programming notes:

(1) The example is actually an ADAM A-task, and so consists of a subroutine with a single argument giving the inherited status value. See SUN/101 for further details about writing ADAM A-tasks. A "stand-alone" equivalent to the ARD_GROUP routine is available which can be used with non-ADAM applications.

(2) The INCLUDE statements are used to define the various "symbolic constants", which are used in this routine. Starlink software makes widespread use of such constants, which should always be defined in this way rather than by using actual numerical values. They are recognisable by the double underscore "__" (e.g. "SAI__OK") which distinguishes them from subroutine names. SAE_PAR defines constants starting with "SAI__", NDF_PAR defines constants starting with "NDF__" (see SUN/33), PRM_PAR defines constants starting with "VAL__" (see SUN/39), GRP_PAR defines constants starting with "GRP__" (see SUN/150), and AST_PAR defines constants starting with "AST__" (see SUN/210).

(3) This application is designed to be able to handle data arrays of any dimensionality, up to the limit set by the NDF library. This limit is given by the symbolic constant NDF__MXDIM, which is used in the declaration of various arrays used to hold information describing each axis.

(4) The value of the STATUS argument is checked. This is because the application uses the Starlink error handling strategy (see SUN/104), which requires that a subroutine should do nothing unless its STATUS argument is set to the value SAI__OK on entry. Here, we simply return without action if STATUS has the wrong value.

(5) The input NDF is now obtained using the ADAM parameter 'NDF'. This may involve prompting the user, or the NDF may be identified using some other means (for instance, the NDF may have been specified on the command line which invoked the application). An integer value is returned to the application in variable INDF. This is an NDF *identifier* and is used to refer to the NDF throughout the rest of the application.

(6) The shape and size of the NDF is now obtained. This returns the number of dimensions, and the upper and lower bounds on each axis. These bounds are needed to be able to correctly locate positions supplied within the ARD description.

(7) The DATA array in the NDF is then accessed by calling NDF_MAP. Rather than returning actual data values, this routine returns a *pointer* to the data values in IPDATA. The total number of pixels in the array is returned in EL.

(8) Next, the ARD description is obtained using ADAM parameter 'REGION'. The returned ARD description is stored in a "GRP group" (rather like a Fortran character array). The GRP package is described in SUN/150 and programmers using ARD should be aware of its contents. An integer value is returned in IGRP which is used to identify the group containing the ARD description throughout the rest of the application. The second argument

is a "null group identifier" (a symbolic constant defined within the include file GRP_PAR). It is used to indicate that there is no existing ARD description on which to base the new ARD description; a completely new ARD description must be supplied.

(9) We now obtain a pointer to a temporary array in which we can store the pixel mask corresponding to the ARD description. The mask has the same number of pixels as the data array, and each pixel stores an integer value. The PSX library is a Fortran interface to the POSIX library and is described in SUN/121.

(10) An NDF structure can have a range of "World Coordinate Systems" associated with it. Information describing these coordinate systems, and how to transform positions between them, is stored in the *WCS component* of the NDF. The NDF_GTWCS routine returns an identifier for an AST FrameSet which is a representation of the WCS component. The facilities of the AST library can then be used to manipulate the WCS information in many different ways.

(11) ARD_WCS stores the supplied FrameSet pointer for later use by the ARD_WORK routine. Making this call to ARD_WCS is optional. If it were not made, then positions within the ARD description would have to be supplied in pixel coordinates. Since we are in fact calling ARD_WCS, positions within the ARD description can be supplied in any coordinate system which can be related to any of the coordinate systems in the NDF's WCS FrameSet.

(12) The subroutine ARD_WORK is now called to create the pixel mask identifying the pixels specified by the ARD description. Positive values are stored in the mask for such pixels, and zero is stored for all other pixels. Note, no value need be assigned to the TRCOEF argument since it will be ignored anyway. This is because of the earlier call to ARD_WCS which specifies the WCS information, and makes the TRCOEF argument redundant.

(13) A subroutine is now called to add up the pixel values in the regions specified by the ARD description. The pointer values returned by NDF_MAP and PSX_CALLOC are turned into actual Fortran arrays at this point, which SUMIT can access. This is done using the %VAL function in the call to SUMIT. SUMIT is not part of the ARD package, but would be written by the application programmer. It may look like this:

```
      SUBROUTINE SUMIT( EL, DATA, MASK, SUM, STATUS )
      IMPLICIT NONE

*  Include definitions of global constants.
      INCLUDE 'SAE_PAR'
      INCLUDE 'PRM_PAR'

*  Arguments Given.
      INTEGER EL
      REAL DATA( EL )
      INTEGER MASK( EL )

*  Arguments Returned.
      REAL SUM

*  Arguments Given and Returned.
      INTEGER STATUS
```

```
*  Declare local variables.
      INTEGER I

*  Check inherited global status.
      IF ( STATUS .NE. SAI__OK ) RETURN

*  Initialise the sum of the valid data values.
      SUM = 0.0

*  Loop round every element in the data array.
      DO I = 1, EL

*  Check to see if this pixel was included in the ARD
*  description. It will have a positive mask value if it was.
*  Skip over the pixel if it was not included.
         IF( MASK( I ) .GT. 0 ) THEN

*  The regions selected by the ARD description may contain
*  pixels which are flagged as unusable in the input NDF.
*  Such pixels have the value given by the symbolic constant
*  VAL__BADR and should not be included in the returned data
*  sum.
            IF( DATA( I ) .NE. VAL__BADR ) THEN
               SUM = SUM + DATA( I )
            END IF

         END IF

      END DO

      END
```

The two arrays can be treated as one dimensional vectors because they are the same size and shape. This makes it easy to process arrays of any dimensionality.

(14) The data sum is displayed by assigning its value to an "MSG token" and then incorporating this token into a message to be displayed on the standard output device. See SUN/104 for a description of the MSG package.

(15) The storage space used to hold the pixel mask is released so that it can be re-used.

(16) The storage space used by the GRP group to hold the ARD description is released.

(17) We now tell the AST library to release the resources used to store the FrameSet read from the NDF.

(18) Finally, the NDF is closed.

1.3 Supplying ARD Descriptions to an Application

This section outlines some of the ways in which a user could supply an ARD description in response to a prompt for the 'REGION' parameter in the example application described in the previous section.

The first thing to be said is that the dimensionality of the ARD description (i.e. the number of values used to represent a single position in the ARD description) need not match that of the supplied NDF. The ARD description may refer to some subset of the axes in the NDF, in which case the mask will be applied independently to every value on the unspecified axes. By default, ARD descriptions are always assumed to be 2-dimensional, but this default can be over-ridden by including a DIMENSION statement in the ARD description. Thus for instance, if the positions in the ARD description are 3-dimensional, the user may want to give an ARD description such as:

```
DIMENSION( 3 )
CIRCLE( 40, 50, 60, 10) .OR. CIRCLE( 45, 55, 65, 10 )
```

The DIMENSION statement tells ARD to expect three values per position. The rest of the ARD description specifies the union of two spheres (i.e. "3-dimensional circles"), centred on (40,50,60) and (45,55,65), each of radius 10. This ARD description uses the default coordinate system established by the application's call to ARD_WORK. In the case of the application above, this default coordinate system is just the pixel coordinate system of the NDF.

ARD uses the versatile facilities of the GRP package to obtain the ARD description. The following examples outline some of the ways in which the above ARD description could be specified in response to a prompt for parameter 'REGION'. For more details on the facilities of GRP, see SUN/150. In each case the "$>$" represents the final character of the prompt string issued by the ADAM parameter system and is not actually typed in by the user:

(1) The entire ARD description could be given as a single literal string:

```
> DIMENSION(3) CIRCLE( 40, 50, 60, 10) .OR. CIRCLE( 45, 55, 65, 10 )
```

All blanks and tabs are ignored.

(2) The ARD description could be split up into several strings given in response to successive prompts. This is particularly useful for long ARD descriptions:

```
> DIMENSION(3) -
> CIRCLE( 40, 50, 60, 10) .OR. -
> CIRCLE( 45, 55, 65, 10 )
```

If an ARD description ends with a minus sign ("$-$"), the ARD_GROUP subroutine will issue another prompt and append any string supplied to the end of the previously supplied string. This continues until an ARD description is supplied which doesn't end with a minus sign.

(3) ARD descriptions can be split anywhere except in the middle of a numerical value. So, for instance, the following responses would be valid:

```
> DIMENSION(3) CIRCLE( 40, 50, -
> 60, 10) .OR. CIRCLE( 45, 55, -
> 65, 10 )
```

(4) If ARD finds adjacent keywords without any intervening operator, an implicit .OR. is inserted between them. So the following, in which the .OR. operator is not explicitly included, would give the same results as the previous examples:

```
> DIMENSION(3) -
> CIRCLE( 40, 50, 60, 10 ) -
> CIRCLE( 45, 55, 65, 10 )
```

(5) Instead of supplying the ARD description directly in response to the parameter prompt, it can be stored in a text file and the name of the text file given in response to the prompt. To do this, the file name must be preceded by an up-arrow symbol ("^"). If the file `desc.ard` contained the three lines:

```
DIMENSION(3)
CIRCLE( 40, 50, 60, 10 )
CIRCLE( 45, 55, 65, 10 )
```

then the user could give the following response:

```
> ^desc.ard
```

The ARD description can be split between the lines of the file in any way the user chooses.

(6) If only part of the ARD description is stored in a text file, then the two methods can be combined. For instance, if the DIMENSION statement is omitted from the file, so that `desc.ard` contains:

```
CIRCLE( 40, 50, 60, 10 )
CIRCLE( 45, 55, 65, 10 )
```

then the user could give the complete ARD description by giving the following response:

```
> DIMENSION(3);^desc.ard
```

This would cause the contents of the file `desc.ard` to be concatenated with the string preceding the ";" character. Note, the semi-colon is not included in the ARD description returned by ARD_GROUP.

(7) If the ARD description were to be split over several files, the contents of the files could be concatenated together in a similar way:

```
> ^desc1.ard;^desc2.ard;^desc3.ard
```

This would cause the contents of files `desc1.ard`, `desc2.ard` and `desc3.ard` to be combined to form the final ARD description.

(8) Indirection through text files can be nested. So if the string

```
^desc1.ard;^desc2.ard;^desc3.ard
```

were stored in a file `total.ard`, then the user could give the response:

```
> ^total.ard
```

2 ARD Description Syntax

An ARD description consists of a stream of *fields*, optionally separated by one or more spaces, or tabs. There are three types of fields:

(1) *Keyword* fields: These specify the basic shapes known to ARD (CIRCLE, BOX, etc.) from which more complex shapes are constructed.

(2) *Operator* fields: These are logical operators, as in Fortran (.AND., .OR., .XOR., .NOT. etc.).

(3) *Statement* fields: These are fields which effect the way that the keyword fields are interpreted, e.g. setting up the dimensionality of the system (DIMENSION), the recognized coordinate systems (COFRAME), etc.

Some keyword and statement fields require argument lists, and these are enclosed within parentheses following the keyword or statement. Arguments are delimited by commas.

keywords and statements can be abbreviated to three characters (operators cannot be abbreviated). ARD descriptions are case-insensitive, and white space (eg spaces, blank records in a file, blank group expressions) is ignored.

2.1 Restrictions on the Order of Fields

An ARD description is basically an algebraic expression in which the operators are *logical* operators (.AND., .OR., etc.) and the operands are represented by keywords (CIRCLE, BOX, etc.) which notionally take true or false values depending on whether or not the current pixel is inside or outside the specified region. Therefore, all the usual restrictions exist on the placing of operators and operands in algebraic expressions; binary operators (e.g. .AND.) must have an operand or bracketed expression on each side, unary operators (e.g. .NOT.) must be followed by an operand or bracketed expression, opening and closing parentheses must balance, etc.

There are, however, certain ways in which an ARD description can depart from this syntax:

- Statement fields can be embedded within an ARD description at any point. They are always removed before evaluating the ARD description as an algebraic expression.
- If .TRUE. is supplied for ARD_WORK argument CONCAT, then a single operator field may appear before the first keyword field. In this case the "missing" operand value to the left of the first operator is defined by the initial mask supplied to routine ARD_WORK. If CONCAT is supplied .TRUE. and no operator field is found before the first keyword field, then an implicit .OR. operator is assumed.
- If two keyword fields do not have an intervening operator field, then an implicit .OR. operator is assumed. This allows a simple list of regions to be specified such as:

```
#
#  ARD description file
#
POLYGON( 20, 20, 50, 50, 25, 75)      # A triangle
ELLIPSE( 10.0, 10.0, 8.0, 5.0, 45.0 )
CIRCLE( 22, 22, 40 )
```

This is effectively equivalent to:

```
POLYGON( 20, 20, 50, 50, 25, 75) .OR.
ELLIPSE( 10.0, 10.0, 8.0, 5.0, 45.0 ) .OR.
CIRCLE( 22, 22, 40 )
```

The returned integer mask will contain positive values within the union of the three regions, and zero outside.

- If an operand is directly followed by a .NOT. operator then an implicit .OR. will be inserted in front of the .NOT. This allows lists of regions such as the following to be given:

```
POLYGON( 20, 20, 50, 50, 25, 75)
.NOT.ELLIPSE( 10.0, 10.0, 8.0, 5.0, 45.0 )
.NOT.CIRCLE( 22, 22, 40 )
```

The returned mask will contain positive values for all pixels which are *either* within the polygon, or are *not* within the ellipse, or are *not* within the circle. It may need to be emphasised here that .OR. operators are inserted and not .AND. operators.

2.2 Group Expression Control Characters

The following two sub-sections supply details of the interaction of ARD with the GRP package, and may safely be skipped over on an initial reading through this document.

The stream of ARD fields is supplied in the form of a GRP group expression (and stored in a normal GRP group) with the following control characters (see SUN/150):

- "^" is used to mark the start of an indirection file, from which further ARD fields should be read.
- "#" is used to initiate a comment.
- ";" is used to delimit GRP elements. A GRP element may contain any number of ARD fields (within the restrictions on string length set by GRP). Argument lists for keyword and statement fields can be split between elements. An element may consist of a single reference to an indirection file (using the "^" character). In this case the element must not contain any explicit ARD fields. Note, GRP elements cannot span records. Thus, carriage returns (either in a file or entered at a prompt) also act as GRP element delimiters, in addition to the ";" character.
- "*" is used as the token which represents basis names within a modification element.
- "|" is used to separate old and new substitution strings when editing names.
- The "OPEN_NEST" and "CLOSE_NEST" control characters are not used (they are set "null").
- "-" is used as the flag character. If a group expression is supplied which ends with "-", then the user is re-prompted for a further group expression, which will be appended to the end of the earlier ones. The interpretation of "-" as the GRP flag character takes precedence over it's interpretation as a synonym for the .NOT. operator.

- "{" is used to open a new kernel.
- "}" is used to close a kernel. A typical use of the kernel characters is to allow the contents of an indirection file to be edited before being used. An element such as "`{^file1.ard}|100|200|`" would cause the contents of `file1.ard` to be read, and then all occurrences of 100 would be replaced by 200.

A complete ARD description can be broken across several lines of an indirection file, or across several directly supplied group expressions (each one terminated with the flag character "-"). However, breaks must occur either *between* fields or between values in a keyword or statement argument list.

2.3 Use of GRP Modification Elements

The GRP system provides a facility which allows a user to modify the contents of a group previously created by the calling application. To do this, a "modification element" is included in the supplied ARD description (or "group expression" to use GRP parlance). A GRP identifier for an existing group can be supplied to the routines ARD_GROUP and ARD_GRPEX (argument IGRP1). If the ARD description supplied by the user contains a modification element, then the contents of the group identified by IGRP1 will be modified according to the instructions in the modification element, and the results incorporated into the returned group. Note, the specified editing is applied separately to each element in the existing group, to produce the corresponding element for the new group. Elements are delimited in ARD descriptions by the ";" character, *and by carriage returns*.

For instance, suppose that the application has already created a group containing the following ARD description:

```
CIRCLE( 0, 10, 10 ) .OR. CIRCLE( 0, -10, 10 )
```

If the GRP identifier for this group is supplied as argument IGRP1 to routine ARD_GROUP, the user may supply an ARD description including a modification element such as:

```
( * ) .AND. BOX( 0, 0, 5, 5 )
```

The asterisk is replaced in turn by each element of the contents of the group identified by IGRP1 (the "basis" group). Thus, the above ARD description is equivalent to the following:

```
( CIRCLE( 0, 10, 10 ) .OR. CIRCLE( 0, -10, 10 ) ) .AND. BOX( 0, 0, 5, 5 )
```

Note, the original ARD description was supplied on a single line, and was thus stored as a single element in the group. If it had spanned lines, then each line would have been edited separately. For instance, if the original ARD description had been supplied as follows, on two lines:

```
CIRCLE( 0, 10, 10 ) .OR.
CIRCLE( 0, -10, 10 )
```

then the modified ARD description would be :

```
( CIRCLE( 0, 10, 10 ) .OR. ) .AND. BOX( 0, 0, 5, 5 )
( CIRCLE( 0, -10, 10 ) ) .AND. BOX( 0, 0, 5, 5 )
```

This group expression would not be accepted by ARD_WORK because of the missing operand after the .OR. operator.

If the user has supplied the ARD description on a single line, and then modified it using the following ARD description:

```
( *|10|5| ) .AND. BOX( 0, 0, 5, 5 )
```

then the basis group would be edited by replacing all occurrences of the string "10" by the string "5", before being included in the final ARD description. Thus the above would be equivalent to:

```
( CIRCLE( 0, 5, 5 ) .OR. CIRCLE( 0, -5, 5 ) ) .AND. BOX( 0, 0, 5, 5 )
```

The syntax of such modification elements is described fully in SUN/150.

3 Interpretation of ARD Descriptions

After removal of all statement fields, and the insertion of any implicit .OR. operators, the fields in an ARD description are treated as a Fortran-like logical expression. Each keyword field forms a logical operand, acted upon by the adjoining operator fields. operators have their usual Fortran precedence (see Section 6). The order of evaluation can be changed by enclosing sub-expressions within parentheses as usual. The interpretation of operands depends on the type of keyword:

- Operands for regions which (in general) have non-zero volume (e.g. POLYGON, CIRCLE, BOX, etc.) are .TRUE. if the *centre* of the current pixel lies on or within the boundary of the region, and are .FALSE. otherwise.
- Operands for regions which have zero volume (e.g. POINT, LINE, ROW, COLUMN, etc.) are .TRUE. if the boundary of the region passes *through* the current pixel. The pixel with index I along some axis is assumed to cover a range of pixel coordinates P given by $(I-1) < P \leq I$.

3.1 Values within the Pixel Mask

The ARD_WORK subroutine creates an array of integer values (the "mask") which holds the result of evaluating the entire logical expression at each pixel. Positive integers represent .TRUE. (i.e. included pixels) and zero represents .FALSE. (i.e. excluded pixels).

Each keyword included in an ARD description is represented by a different positive integer. This provides the possibility for applications to differentiate between the different regions within an ARD description on the basis of the pixel mask alone.

3.1.1 Background Pixels

The .NOT. operator adds a few complications in that it requests pixels to be included which are *not* within a given region. For instance:

```
.NOT.CIRCLE( 0, 0, 10)
```

includes all pixels which are not within the circle of radius 10 centred on the origin. If (say) the value 2 was used to represent the CIRCLE region, what value should be used to represent the pixels which are *not* within the circle? The solution adopted by ARD is to consider all pixels *not* within a region to be "background" pixels, and to reserve the value 1 to represent such pixels. All .NOT. operators in an ARD description use the value 1 to represent included background pixels. Note, ARD doesn't know if a pixel is truly part of the background or not, it just assumes that all pixels selected by a .NOT. operator will be background. This gives rise to a possible anomaly which can be illustrated by the ARD description:

```
.NOT. ( .NOT. CIRCLE( 0, 0, 10 ) )
```

One might expect this to be equivalent to the ARD description:

```
CIRCLE( 0, 0, 10 )
```

but there will be a difference. The two ARD descriptions will store positive values at the same pixels (i.e. those within the circle), but the values stored will be different. In the first ARD description, the included pixels are generated by a .NOT. operator and so will be considered to be "background" pixels and will be represented by the value 1. In the second ARD description, the included pixels are generated directly by the CIRCLE keyword and will be represented by the value assigned to the keyword (which will be larger than 1).

The .EQV. operator can also cause background pixels to be included in the returned mask, and such background pixels are again represented by the value 1. The ARD description:

```
CIRCLE( 0, 0, 10 ) .EQV. CIRCLE( 10, 0, 10)
```

selects pixels which are either within both circles or within neither circle. Pixels which are within neither circle form part of the background, and are represented by the value 1. Pixels which are within both circles are represented by the larger of the two values assigned to the two keywords.

3.1.2 Assignment of Keyword Values

The choice of which positive value to use to represent each keyword in the ARD description is controlled by the REGVAL argument supplied to the ARD_WORK routine. If a positive value is supplied for REGVAL then the first keyword in the ARD description (working from left to right) is assigned the supplied value. Successive values are assigned to the remaining keywords. An error is reported if REGVAL is supplied equal to 1. This is because 1 is reserved to represent background pixels and may not be used to represent keywords.

If a zero or negative value is supplied for REGVAL, then the array of pixel values supplied to ARD_WORK in argument MASK is examined and the maximum value found. This value

is incremented by one and used to represent the first keyword in the ARD description (if the incremented value is less than 2, then 2 is used instead). Successive values are assigned to the remaining keywords.

There is one exception to these rules; pixels selected using an INPUT keyword (see section 3.2) are assigned the values of the corresponding pixels in the input mask supplied to ARD_WORK, irrespective of the position of the INPUT keyword in the ARD description. The inclusion of INPUT keywords within an ARD description does not effect the integer values used to represent other keywords. Thus if REGVAL is supplied equal to 2 and the ARD description is:

```
CIRCLE( 0, 0, 10 ) .AND. ( INPUT .OR. CIRCLE( 0, 10, 10 ) )
```

then the CIRCLE(0,0,10) region is assigned the value 2 and the CIRCLE(0,10,10) region is assigned the value 3 (i.e. the INPUT keyword is not included in the count of keywords).

On return from ARD_WORK the REGVAL argument will hold a value one larger than the value assigned to the last keyword in the ARD description.

3.1.3 Pixels Included in Several Regions

If a given pixel falls within more than one region, then the largest of the values associated with the regions is stored at the pixel's location. Specifically, the rules for determining the result of each operator are:

.AND. - A zero is returned unless both operands are positive, in which case the larger of the two positive values is returned.

.OR. - The larger of the two operand values is returned.

.XOR. - If either operand is zero, then the value of the other operand is returned. If neither operand is zero, then zero is returned.

.EQV. - If neither of the operands is zero, then the larger of the two operands is returned. If one of the operands is zero, then zero is returned. If both of the operands are zero, then the value 1 is returned.

.NOT. - If the operand is positive, then zero is returned. If the operand is zero, then 1 is returned.

Let's look at an example:

```
CIR( 0, 0, 50 )
CIR( 0, 0, 40 )
CIR( 0, 0, 30 )
CIR( 0, 0, 20 )
CIR( 0, 0, 10 )
```

An implicit .OR. operator is assumed to exist between each of the CIRCLE keywords. Let's assume that ARD_WORK is called with argument REGVAL set to 2. The CIR(0,0,50) keyword is evaluated first and causes a circular region of radius 50 to be filled with the value 2. Next, the CIR(0,0,40) keyword is evaluated. The value used to represent included pixels is incremented to 3, and a circular region of radius 40 is filled with this value, over-writing some of the 2's

which were written to the mask because of the previous keyword. Since the circular regions are concentric (both being centred on the (0,0)), this will leave an annulus containing the value 2 around the edge of the circle containing value 3. The remaining keywords are processed in the same way, each successive keyword over-writing all but an annulus of the circle created by the previous keyword. The final mask consists of a set of concentric annuli, each of thickness 10. Working outwards from the centre, the annuli have the values 6, 5, 4, 3 and 2. The REGVAL argument will be returned holding 7.

One consequence of allowing larger values to "over-write" smaller values is that background pixels always get over-written by pixels which are selected by virtue of being within a keyword region. Consider the following (assuming that REGVAL is again supplied equal to 2):

```
CIRCLE( 0, 0, 10 ) .AND. .NOT. CIRCLE( 0, 0, 5 )
```

This ARD description will be evaluated by first creating two intermediate masks, one containing the CIRCLE(0,0,10) region and another containing the .NOT.CIRCLE(0,0,5) region. These two masks will then be combined together using the .AND. operator to create the final mask. The first intermediate mask contains the value 2 at all pixels which are within the CIRCLE(0,0,10) region, and zero at all other pixels.

The second intermediate mask is initially set to hold the region CIRCLE(0,0,5). This means that pixels within the circle are given the value 3 and all others are given the value zero. The mask is then inverted to take account of the .NOT. operator. Pixels inside the circle which previously held the value 3 are changed to zero (indicating that these pixels have *not* been selected). Pixels outside the circle which previously held the value zero are assigned the value 1 (indicating that these pixels are background pixels).

The last stage is to combine the two intermediate masks together following the rules described above for an .AND. operator. Pixels which are further than 10 units from the origin (i.e. outside the CIRCLE(0,0,10) region) will hold zero in the first mask and 1 in the second mask, and so will be assigned a value zero in the final mask. Pixels which are between 5 and 10 units from the origin (i.e. inside the first circle but not the second) will have the value 2 in the first mask and the value 1 in the second. Both these values are positive and so the output mask pixel will also be positive. The rules for the .AND. operator above say that the positive value used will be the larger of the two values in the input masks. Thus the output mask pixels are assigned the value 2. Pixels which are less than 5 units from the origin will have the value 2 in the first mask and the value zero in the second mask, and so are assigned the value zero in the output mask. The final mask thus holds an annulus extending from radius 5 to radius 10 in which the pixels hold the value 2, all other pixels holding the value zero.

3.2 Supplying an Initial Pixel Mask

It may sometimes be necessary for applications to combine together pixel masks created from different ARD descriptions. To do this, the ARD descriptions should be processed in turn by ARD_WORK and they should use the "INPUT" keyword. This keyword refers to the pixel mask supplied to routine ARD_WORK in argument MASK; if a pixel holds a positive value in the supplied mask then the INPUT keyword is notionally .TRUE., and is notionally .FALSE. if the pixel value is zero or negative. This keyword can be used at any point in an ARD description, and can be included any number of times. A simple example of its use could be:

```
.NOT. INPUT
```

which inverts the supplied mask. If the ARD description does not include any references to the INPUT keyword, then the application can force an INPUT keyword to be inserted at the start of the ARD description by supplying a .TRUE. value for ARD_WORK argument CONCAT. For instance, if CONCAT is supplied .TRUE., then the ARD description:

```
BOX( 0, 0, 10, 10 ) .OR. CIR( 0, 0, 10 )
```

becomes

```
INPUT BOX( 0, 0, 10, 10 ) .OR. CIR( 0, 0, 10 )
```

There is now no operator between the INPUT and BOX keywords, and so ARD inserts an implicit .OR. so that the final ARD description used is:

```
INPUT .OR. BOX( 0, 0, 10, 10 ) .OR. CIR( 0, 0, 10 )
```

If the supplied ARD description had started with an operator, for instance:

```
.AND.( BOX( 0, 0, 10, 10 ) .OR. CIR( 0, 0, 10 ) )
```

then there would be no need to insert an implicit .OR. after the INPUT keyword, and so the final used ARD description would be:

```
INPUT .AND.( BOX( 0, 0, 10, 10 ) .OR. CIR( 0, 0, 10 ) )
```

To re-iterate, an INPUT keyword is inserted at the start of the ARD description only if the ARD description as supplied contains no INPUT keywords, *and* the ARD_WORK argument CONCAT is supplied .TRUE.. If either of these conditions is broken then the ARD description is left as supplied.

3.3 Bounding Boxes

To combine two large masks using a binary operator at every pixel can be wasteful of processor time, especially if not many of the pixels have actually been selected. To reduce this waste, ARD keeps track of the regions within the mask where the selected pixels are located, and only processes pixels in those regions. Precisely, ARD maintains a pair of "bounding boxes" throughout its evaluation of an ARD description. These are referred to as the internal and external bounding boxes. Each bounding box is a rectangular region of the mask. The internal bounding box contains *all* included pixels in the mask, but may also contain some excluded pixels. Conversely, the external bounding box contains all excluded pixels and may also contain some included pixels.

The upper and lower bounds of these boxes are returned to the calling application when the mask is complete. If an application is only interested in included pixels, it may then restrict its attention to the region of the data array contained within the internal bounding box (all pixels outside this box are guaranteed to be excluded). It is still necessary for the application to check

mask pixels to see if they are included or excluded, but the checks can at least be restricted to the region of the internal bounding box. The external bounding box can be used in a similar way if the application is only interested in excluded pixels.

If *all* mask pixels are included (i.e. if there are no excluded pixels) then the external bounding box will be returned "null", and the internal bounding box will be returned covering the entire mask. Likewise, if *all* mask pixels are excluded (i.e. if there are no included pixels) then the internal bounding box will be returned "null", and the external bounding box will be returned covering the entire mask. Null bounding boxes are identified by the fact that the lower bound of each axis is greater than the corresponding upper bound. This condition should always be checked for before using a bounding box.

4 Coordinate Systems

ARD_WORK needs to be able to locate pixels within the given pixel mask. To do this, every position supplied in an ARD description must be converted into *pixel coordinates*. The position of the origin of pixel coordinates within the mask is fixed by the upper and lower bounds of the mask supplied to ARD_WORK. The conventions for pixel coordinates used by Starlink software are described in Starlink System Note (SSN) 22.

However, it is not always appropriate for an ARD description to describe a region in terms of pixel coordinates. For instance, in a mosaic image a given position on the sky may have different pixel coordinates in each image. To describe a given region in pixel coordinates would therefore require a separate description for each image. In this case, we would ideally like to use a single ARD description in which the region was defined in terms of RA and Dec. positions. This can be done, so long as ARD_WORK knows how to transform an RA/Dec position into pixel coordinates within each of the specified pixel masks.

There are two schemes supported by the ARD library to allow the ARD description to include positions in a coordinate system other than pixel coordinates. As of version 2 of the ARD library, the facilities of the AST library (see SUN/210) can be used to specify positions within generalised non-linear coordinate systems. Prior to version 2, positions needed to be supplied in a coordinate system which was linearly related to pixel coordinates in the mask supplied to ARD_WORK. The version 1 scheme is still available in version 2 of the library, but is deprecated.

4.1 World Coordinate Systems in ARD Version 2

ARD version 2 uses the facilities of the AST library to manage coordinate systems. An AST "FrameSet" describes a collection of related coordinate systems (also called "Frames"), together with the mappings which allows positions to be transformed from one Frame to another. The calling application will always know how to locate a point within the mask if the point is specified in pixel coordinates. If, in addition, it can also locate points specified in one or more other coordinate systems, then it should create a FrameSet describing all the coordinate systems which it knows about, one of which must be pixel coordinates within the mask[1]. It then passes this FrameSet to the ARD system by calling routine ARD_WCS (this must be done prior to

[1] ARD recognizes this Frame by the fact that its Domain attribute is set to "PIXEL".

calling ARD_WORK)[2]. This FrameSet is known as the "Application FrameSet", and encapsulates knowledge of all the coordinate systems known to the application.

If ARD_WCS is not used, then a default application FrameSet containing a single Frame describing pixel coordinates is generated automatically.

The ARD description must then include statements describing the coordinate system in which positions are given within the ARD description. This can be done in one of three ways:

(1) By including a COFRAME statement before any keywords. A COFRAME statement describes a single coordinate system, namely that in which positions are given within the ARD description.

ARD_WORK will determine if there is any way of converting positions given within this coordinate system into any of the coordinate systems included in the Application FrameSet, and thus into pixel coordinates. If there is, then the corresponding transformation is used to locate positions within the pixel mask. Otherwise, an error is reported.

(2) By including a WCS statement before any keywords. A WCS statement specifies a FrameSet describing all the coordinate system known to the user. Positions within the ARD description are then assumed to be given within the "current" Frame of this FrameSet. This FrameSet is known as the "User FrameSet" to distinguish it from the "Application FrameSet".

ARD_WORK will search both FrameSets looking for a coordinate system which is present in both. If such a Frame is found, the User FrameSet is used to convert positions included in the ARD description into the common Frame, and the Application FrameSet is then used to convert positions from the common Frame into pixel coordinates. If no common Frame is found, then an error is reported. It may be that there is more than one common Frame, in this case priority is given to Frames describing celestial coordinate systems, followed by pixel coordinate Frames, followed by Frames with Domain GRID, followed by Frames with Domain ARDAPP. The highest priority Frame found within this list is used as the common Frame.

(3) If there is neither a WCS nor a COFRAME statement in the ARD description, then it is assumed that positions are supplied in a default coordinate system. This default is determined as follows:

- If an application FrameSet has been specified using ARD_WCS, then the default coordinate system is the current Frame of the application FrameSet. In this case, the *TR* argument for routine ARD_WORK is ignored.
- If no application FrameSet has been specified using ARD_WCS, then the default coordinate system is obtained by transforming the pixel coordinate system according to the values supplied for argument *TR* when calling ARD_WORK. This is the system which was used prior to ARD version 2 (see the next section for more details).

In addition, the deprecated version 1 statements COEFFS, OFFSET, TWIST, STRETCH, SCALE can be used to modify the relationship between the coordinate system used within

[2]You may be wondering why a separate routine is used instead of simply passing the information to ARD_WORK as an argument. This was done so that existing application which use ARD version 1 would continue to work without modification.

the ARD description and the default coordinate system describe above. If any of these statements are found, then any earlier COFRAME or WCS statements are ignored, and the requested operation (stretch, twist, offset, *etc*) is applied to the default coordinate system described above.

An ARD description can contain zero, one or more of the above WCS-related statements. The positions, *etc*, defining a given region in the description are interpreted using the most recent WCS-related statement. Thus, WCS information given later in an ARD description over-rides any given earlier.

If any of the deprecated version 1 statements (COEFFS, OFFSET, TWIST, STRETCH, SCALE) are found, then any earlier COFRAME or WCS statements are ignored, and the requested operation (stretch, twist, offset, *etc*) is applied to the default coordinate system described above. For this reason, you should not normally mix the old and the new statements.

4.2 Coordinate Handling in ARD Prior to Version 2

ARD version 1 required the transformation between "user coordinates" (*i.e.* the coordinate system in which positions are supplied in the ARD description) to pixel coordinates to be linear. This linear transformation was split up into two components; the application specified a linear transformation from pixel coordinates to "application coordinates" when calling ARD_WORK, and the ARD description included statements which defined a linear transformation from application coordinates to user coordinates (by default, user coordinates were assumed to be identical with application coordinates). These two components were concatenated to get the total transformation from pixel to user coordinates.

4.2.1 Application Coordinates

Version 1 applications use the TRCOEF argument of the ARD_WORK routine to define the application coordinate system. TRCOEF would be supplied holding the coefficients of the linear mapping from application coordinates to pixel coordinates. For instance, if a 2-dimensional application coordinates system (x_a, y_a) is required to be equal to pixel coordinates (x_p, y_p), but with the origin shifted to pixel coordinates $(10, 20)$, then the linear mapping from (x_a, y_a) to (x_p, y_p) is:

$$\begin{aligned} x_p &= 10 + 1.x_a + 0.y_a \\ y_p &= 20 + 0.x_a + 1.y_a \end{aligned}$$

In this case TRCOEF would be supplied holding the 6 coefficient values (10,1,0,20,0,1). In general, if (Y_1, Y_2, Y_3, ..., Y_N) are a set of application coordinates in N dimensions, and (Z_1, Z_2, Z_3, ..., Z_N) are the corresponding pixel coordinates, then the application supplies a set of constants C_1 to $C_{N*(N+1)}$, where:

$$\begin{aligned} Z_1 &= C_1 + C_2.Y_1 + C_3.Y_2 + ... + C_{N+1}.Y_N \\ Z_2 &= C_{N+2} + C_{N+3}.Y_1 + C_{N+4}.Y_2 + ... + C_{2.(N+1)}.Y_N \\ &.... \\ Z_N &= C_{N.N} + C_{N.(N+1)}.Y_1 + C_{N.(N+2)}.Y_2 + ... + C_{N.(N+1)}.Y_N \end{aligned}$$

These constants can be supplied to ARD_WORK as a 1-dimensional vector C with N.(N+1) elements, or as a 2-dimensional array T with dimensions (0:N, N) where:

$$\begin{aligned} Z_1 &= T(0,1) + T(1,1).Y_1 + T(2,1).Y_2 + ... + T(N,1).Y_N \\ Z_2 &= T(0,2) + T(1,2).Y_1 + T(2,2).Y_2 + ... + T(N,2).Y_N \\ &.... \\ Z_N &= T(0,N) + T(1,N).Y_1 + T(2,N).Y_2 + ... + T(N,N).Y_N \end{aligned}$$

The order in which the coefficient values are stored is the same in both cases.

There will be many cases in which application coordinates are required to be just equal to pixel coordinates. In this case the diagonal elements of the T array (T(1,1), T(2,2), etc.) would be set to one and all other elements of T would be set to zero. This is likely to be a common requirement, and so ARD_WORK has a special facility for creating such a "unit" mapping. If the first coefficient (C(1) or T(0,1)) is set equal to the symbolic constant VAL__BADR (defined in include file PRM_PAR), then ARD_WORK will ignore the supplied values of TRCOEF and use a unit mapping instead.

4.2.2 User Coordinates

The positions and displacements within a Version 1 ARD description are interpreted as application coordinates by default. To supply positions and displacements in some other linearly-related coordinate system, the ARD description would contain statement fields defining the mapping from user coordinates to application coordinates. Such mappings were set up using the statements COEFFS, OFFSET, SCALE, TWIST and STRETCH.

For instance, if the application expects coordinates to be given in arc-seconds but the user wishes to give them in arc-minutes, then the statement SCALE(60) should be included in the ARD description before the first keyword. This mapping is then concatenated with the mapping supplied by the application to get the mapping from user coordinates to pixel coordinates.

The user coordinate system may be changed at any point within an ARD description using suitable statement (COEFFS, etc.), and the new mapping will be used to interpret all further positions until the user coordinate system is modified again.

Arguments which specify distances (such as the radius of a circle for instance) are subject to the current mapping. Thus, for instance, a circle may be transformed into an ellipse if the coordinate axes have different scales.

5 Keywords

Keywords are fields within an ARD description which specify one of the basic shapes known to ARD. Most are followed by an argument list giving values for the size, position, orientation, etc., of the shape. Argument lists are contained within parentheses, and arguments are separated by commas. Positions and distances should be supplied in a format appropriate to the current user coordinate system as determined by the WCS and/or COFRAME statements. Keywords can be abbreviated to three characters. The following keywords are currently supported (N represents the dimensionality of the ARD description, and all positions and distances are given in N-dimensional user coordinates):

BOX - A rectangular box with sides parallel to the user coordinate axes. The argument list should contain $2 * N$ values; the first N values give the coordinates of the box centre, and the remaining N values give the lengths of the box sides. It may sometimes be more convenient to use the RECT keyword which specifies a rectangular box in terms of two diagonally opposite corners.

CIRCLE - A circle (for $N = 2$) or sphere (for $N > 2$). The argument list should contain $N + 1$ values; the first N values give the coordinates of the centre of the circle or sphere, and the remaining value gives the radius (in user coordinates).

COLUMN - A set of lines parallel to the second axis of the user coordinate system. The argument list can contain any number of values (one for each of the lines). Each argument value X specifies that the corresponding line should pass through the position $(X, 0)$. This keyword can only be used in 2-dimensional ARD descriptions.

ELLIPSE - A 2-dimensional ellipse. The argument list should contain 5 values; the first pair give the user coordinates of the centre of the ellipse, the second pair give the half-lengths of the two axes of the ellipse (in user coordinates), and the fifth value gives the angle (in degrees) from the first axis of the user coordinate system to the first axis of the ellipse (positive rotation is from the first to the second user axis). This keyword can only be used in 2-dimensional ARD descriptions.

FRAME - The entire mask excluding a border of given width. The argument list should contain a single value giving the width of the border. This keyword can only be used in 2-dimensional ARD descriptions. A further restriction on its use is that the current mapping from user coordinates to pixel coordinates must be isomorphic (i.e. each mask pixel must correspond to a square area in user coordinates; the square may be rotated, shifted and/or scaled, but it must still be a square). If this is not the case an error will be reported by ARD_WORK.

INPUT - The pixel mask supplied as input to routine ARD_WORK.

LINE - A straight line between two given positions. The argument list should contain $2 * N$ values, the first set of N values giving the user coordinates of the first position, and the second set of N values giving the second position. Only the section of the line between (and including) the two positions is included in the mask.

PIXEL - A set of individual pixels. The argument list should contain an integer multiple of N values; each set of N values giving the user coordinates of a point to be included.

POINT - POINT is a synonym for PIXEL.

POLYGON - A 2-dimensional polygonal area. The argument list should contain an even number of values; each pair giving the user coordinates of a vertex of the polygon. These vertices are joined together in the order given to form the polygon. The last vertex is joined to the first to close the polygon. This keyword can only be used in 2-dimensional ARD descriptions. Note, the edges of a polygon are *geodesics* within the user coordinate Frame. So, for instance, if user coordinates are RA and DEC then the edges of a polygon will correspond to great circles on the sky.

RECT - A rectangular box with sides parallel to the user coordinate axes. The argument list should contain $2 * N$ values; each set of N values giving the coordinates of a pair

of diagonally opposite corners. It may sometimes be more convenient to use the BOX keyword which specifies a rectangular box in terms of its centre and dimensions.

ROTBOX - A rotated box. The argument list should contain 5 values; the first pair give the coordinates of the box centre, the second pair give the lengths of the two sides, and the fifth value gives the angle (in degrees) from the first axis of the user coordinate system to the first side of the box (positive rotation is from the first to the second user axis). This keyword can only be used in 2-dimensional ARD Note, the edges of a rotated box are *geodesics* within the user coordinate Frame. So, for instance, if user coordinates are RA and DEC then the edges of a rotated box will correspond to great circles on the sky.

ROW - A set of lines parallel to the first user axis. The argument list can contain any number of values (one for each of the lines). Each argument value Y specifies that the corresponding line should pass through the position $(0, Y)$. This keyword can only be used in 2-dimensional ARD descriptions.

WHOLE - This keyword selects all pixels in the mask. It has no argument list.

6 Operators

Operators are fields within an ARD description which specify an operation to perform on one or two keyword fields. They cannot be abbreviated, and the leading and trailing dots must be included. The following operators are supported (they are listed in order of decreasing precedence):

.NOT. - Invert the region specified by the following keyword or expression (i.e. included pixels become excluded, and excluded pixels become included). A minus sign is recognised as a synonym for .NOT..

.AND. - Take the intersection of the two regions. A pixel is included only if it is within *both* of the regions given on either side of the .AND. field.

.OR. - Take the union of the two regions. A pixel is included if it is within *either* of the regions given on either side of the .OR. field.

.XOR. - Take the exclusive OR of the two regions. A pixel is included if it is within one, but not both, of the regions given on either side of the .XOR. field.

.EQV. - Take the equivalence of the two regions. A pixel is included if it is within *both* of the regions given on either side of the .EQV. field, or if it is within *neither*.

.XOR. and .EQV. have equal precedence. Opening and closing parentheses ("(" and ")") can be used to bracket sub-expressions within an ARD description.

7 Statements

Statements are fields within an ARD description which modify the way in which keywords are interpreted. They are followed by an argument list giving various numerical or textual values. Statements can be abbreviated to three characters. The following statements are currently supported (N represents the dimensionality of the ARD description):

DIMENSION(N) - This establishes the number of coordinates (N) required to describe a single point in the ARD description. If no DIMENSION statement is found before the first keyword field, a value of 2 is assumed for N. The dimensionality may not be changed after the first keyword field, but multiple DIMENSION statements may be included (so long as they all specify the same value for N) to improve readability, and to allow the concatenation of multiple ARD descriptions which contain compatible DIMENSION statements. Note, N gives the number of dimensions in the user co-ordinate system; this may not necessaily be the saem as the number of pixel axes in the mask array.

COEFFS(C1, C2, ...) - **DEPRECATED. Use the COFRAME or WCS statement instead.** This establishes a new linear mapping from user coordinates to application coordinates replacing any previous mapping (see Section 4). The statement must have $N * (N + 1)$ arguments. If (X_1, X_2, X_3, ..., X_N) are a set of user coordinates in N dimensions, and (Y_1, Y_2, Y_3, ..., Y_N) are the corresponding application coordinates, then the argument list should contain a set of constants C_1 to $C_{N*(N+1)}$, where:

$$\begin{aligned} Y_1 &= C_1 + C_2.X_1 + C_3.X_2 + ... + C_{N+1}.X_N \\ Y_2 &= C_{N+2} + C_{N+3}.X_1 + C_{N+4}.X_2 + ... + C_{2.(N+1)}.X_N \\ &.... \\ Y_N &= C_{N.N} + C_{N.(N+1)}.X_1 + C_{N.(N+2)}.X_2 + ... + C_{N.(N+1)}.X_N \end{aligned}$$

The mapping established replaces any previous mapping, and is used to transform all following coordinates, until another mapping is established. The default mapping results in user coordinates being identical with application coordinates. The OFFSET, SCALE, TWIST and STRETCH statements allow complex mappings to be created without the use of COEFF statement in certain cases.

COFRAME(DOMAIN,...) - Specifies the user coordinate system (*i.e.* the coordinate system in which positions are supplied within the remainder of the ARD description). The first argument is the Domain of the coordinate system. There are several special values that causes specialised AST objects to be created to descibe specific forms of coordinate systems:

- "SKY": If the ARD description is 2-dimensional, an AST SkyFrame will be created.
- "TIME": If the ARD description is 1-dimensional, an AST TimeFrame will be created.
- "SPECTRUM": If the ARD description is 1-dimensional, an AST SpecFrame will be created.
- "DSBSPECTRUM": If the ARD description is 1-dimensional, an AST DSBSpecFrame will be created.

Any other Domain value will cause a simple AST Frame to be created instead. Any subsequent arguments should be of the form `<keyword>=<value>` where `<keyword>` is the name of an AST attribute and `<value>` is the value to assign to the attribute.

OFFSET(X, Y, Z, ...) - DEPRECATED. Use the COFRAME or WCS statement instead. This modifies the current mapping from user coordinates to application coordinates so that the origin of the user coordinate system is moved by the given offsets in application coordinates. The initial position for the origin of user coordinates is (0,0,0,..). In a 2-dimensional ARD description, the first statement OFFSET(10,15) would put the origin at (10,15) in the application coordinates system. A second statement OFFSET(2,3) would move it to (12,18). The statement must have *N* arguments.

SCALE(F) - DEPRECATED. Use the COFRAME or WCS statement instead. This modifies the current mapping from user coordinates to application coordinates so that the user coordinate system is magnified by the given factor F along all axes. The magnification is centred on the origin of the application coordinate system.

STRETCH(F1, F2, F3, ...) - DEPRECATED. Use the COFRAME or WCS statement instead. This modifies the current mapping from user coordinates to application coordinates so that the user coordinate system is magnified by the given factor F_i along axis i. Unlike the SCALE statement, the magnifications are centred on the origin of the *user* coordinate system (i.e. the position of the origin of the user coordinate system within the application coordinate system is not changed by this statement). The statement must have *N* arguments.

TWIST(T) - DEPRECATED. Use the COFRAME or WCS statement instead. This statement modifies the current mapping from user coordinates to application coordinates so that the user coordinate system is rotated by an angle T (in degrees). The rotation is about the origin of the application coordinate system. If the ARD description has more than 2 dimensions, then the rotation takes place in the X-Y plane. Rotation from the X axis to the Y axis is positive. "TWIST" is used rather than the more obvious "ROTATE" in order to avoid a name clash with the keyword ROTBOX.

WCS(...) - Specifies an AST FrameSet in which the current Frame is the user coordinate system (*i.e.* the coordinate system in which positions are supplied within the remainder of the ARD description). The "argument list" should be a textual dump of an AST FrameSet as produced by the AST Channel class (*e.g.* using the AST_SHOW routine). Each line of the dump should be stored as a separate GRP element in the supplied ARD expression - the simplest way to do this is probably to supply the ARD description within a text file, and put each line of the dump on a separate line in the file:

```
WCS(<!!
    Begin FrameSet
    IsA Frame
    Nframe = 6
    ...
    ...
    End FrameSet
!!>)
```

The <!! and !!> strings tell GRP to treat the enclosed text as verbatim text. That is, any GRP control characters found within the body of the FrameSet dump are treated as literal characters.

8 Compiling and Linking

To compile and link a UNIX ADAM application with the ARD package, the following commands should be used (see SUN/144):

```
% alink adamprog.f -L/star/lib `ard_link_adam`
```

(note the use of *opening* apostrophes (`) rather than closing apostrophe (')). To compile and link a stand-alone UNIX application with the ARD package, the following commands should be used:

```
% f77 prog.f -o prog -L/star/lib `ard_link`
```

This produces an executable image called `prog`.

The ADAM and stand-alone versions of the ARD_ system differ, in that the stand-alone version does not contain the routine ARD_GROUP.

A Alphabetical List of Routines

ARD_GROUP (PARAM, IGRP1, IGRP2, STATUS)
Obtain an ARD description from the environment

ARD_GRPEX (DESC, IGRP1, IGRP2, FLAG, STATUS)
Store an explicitly supplied ARD description in a GRP group

ARD_GTWCS (IGRP, NDIM, IWCS, STATUS)
Return a FrameSet connecting pixel and user co-ordinates

ARD_PLOT (IGRP, IPLOT, GBOX, REGVAL, STATUS)
Draw the outline of an ARD description on a graphics device

ARD_PTWCS (IWCS, IGRP, STATUS)
Construct an ARD WCS statement and append it to a GRP group

ARD_WCS (IWCS, DOMAIN, STATUS)
Specify WCS information to be used in future calls to ARD_WORK

ARD_WORK (IGRP, NDIM, LBND, UBND, TRCOEF, CONCAT, REGVAL, MASK, LBNDI, UBNDI, LBNDE, UBNDE, STATUS)
Create a pixel mask from an ARD description

B Routine Descriptions

ARD_GROUP
Obtain an ARD description from the environment

Description:

An ARD description is obtained from the environment using the supplied parameter name and stored in a group identified by the returned value of IGRP2. If the last character in the ARD description is a minus sign ("- ") then the parameter value is then cancelled and further ARD descriptions are obtained and appended to the returned group. This process continues until an ARD description is supplied which does not end with a minus sign, or a null value is supplied.

If a GRP identifier for an existing group is supplied for IGRP1 then the group will be used as the basis for any modification elements contained within the ARD descriptions obtained from the environment. No checks are made for modification elements if the symbolic constant GRP__NOID is supplied for IGRP1.

Invocation:

```
CALL ARD_GROUP( PARAM, IGRP1, IGRP2, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The parameter name.

IGRP1 = INTEGER (Given)
GRP identifier for a group to be used as a basis for modification elements.

IGRP2 = INTEGER (Returned)
GRP identifier for the created group.

STATUS = INTEGER (Given and Returned)
The global status.

Notes:

- No checks are made on the syntax of the ARD description.
- The returned GRP identifier (IGRP2) should be deleted using GRP_DELET when it is no longer needed.
- If an error occurs either before or during this routine then IGRP2 will be returned holding the symbolic value GRP__NOID (defined in include file GRP_PAR).
- An error is returned if the first value obtained for the parameter is a null value.

ARD_GRPEX
Store an ARD description in a GRP group

Description:

The supplied ARD description is appended to the group identified by IGRP2. If the symbolic constant GRP__NOID is supplied for IGRP2 then a new group is first created and its identifier is returned in IGRP2.

If a GRP identifier for an existing group is supplied for IGRP1 then the group will be used as the basis for any modification elements contained within the ARD description. No checks are made for modification elements if the symbolic constant GRP__NOID is supplied for IGRP1.

Invocation:

```
CALL ARD_GRPEX( DESC, IGRP1, IGRP2, FLAG, STATUS )
```

Arguments:

DESC = CHARACTER * (*) (Given)
The ARD description.

IGRP1 = INTEGER (Given)
GRP identifier for a group to be used as a basis for modification elements.

IGRP2 = INTEGER (Given and Returned)
GRP identifier for the group holding the ARD description.

FLAG = LOGICAL (Returned)
Returned .TRUE. if the last non-blank character in the supplied ARD description is a minus sign ("- ").

STATUS = INTEGER (Given and Returned)
The global status.

Notes:

- No checks are made on the syntax of the ARD description.
- The returned GRP identifier (IGRP2) should be deleted using GRP_DELET when it is no longer needed.
- The symbolic constant GRP__NOID is defined in the include file GRP_PAR.

ARD_GTWCS
Return a FrameSet connecting pixel and user co-ordinates

Description:

This routine returns an AST pointer for a FrameSet describing the relationship between the user co-ordinate system used by the supplied ARD description, and the pixel coordinate Frame of the FrameSet stored by the most recent call to ARD_WCS.

Invocation:

```
CALL ARD_GTWCS( IGRP, NDIM, IWCS, STATUS )
```

Arguments:

IGRP = INTEGER (Given)
A GRP identifier for the group holding the ARD description.

NDIM = INTEGER (Given)
The number of pixl axes in the mask array.

IWCS = INTEGER (Returned)
An AST pointer to the FrameSet. The base Frame will be the pixel coordinate Frame in the FrameSet supplied via the most recent call to ARD_WCS. The current Frame will be the user co-ordinate system specified by the supplied ARD description.

Notes:

- An error is reported if the dimensionality of the ARD description is different to that of the mask array (as specified by argument NDIM).

ARD_PLOT
Plot the boundary of an ARD description

Description:

This routine draws a curve marking the boundary of the ARD description supplied within group IGRP. It can also draw a boundary round a given sub-region by supplying a positive value for REGVAL. The ARD description must be 2-dimensional.

Invocation:

```
CALL ARD_PLOT( IGRP, IPLOT, GBOX, REGVAL, STATUS )
```

Arguments:

IGRP = INTEGER (Given)

A GRP identifier for the group holding the 2-dimensional ARD description.

IPLOT = INTEGER (Given)

An AST pointer to a Plot which will be used to draw the boundary. The Plot and the ARD description will be aligned in a suitable common coordinate Frame, present in both the Plot and the WCS FrameSet implied by the ARD description. If no such common Frame is available, an error is reported.

GBOX(4) = REAL (Given)

An array giving the position and extent of the plotting area (on the plotting surface of the underlying graphics system) in which graphical output is to appear. This must be specified in the base (i.e. GRAPHICS) Frame of the supplied Plot. This can be smaller than the area covered by the supplied Plot, in which case the graphics will be truncated.

The first pair of values should give the coordinates of the bottom left corner of the plotting area and the second pair should give the coordinates of the top right corner. The coordinate on the horizontal axis should be given first in each pair.

REGVAL = INTEGER (Given and Returned)

The index of the region within the ARD description to be outlined. If the value zero is supplied, the entire ARD description is outlined. If a positive value is supplied, then only the region with the specified index is outlined. If a negative value is supplied, then regions with indices greater than or equal to the absolute value are outlined. If the supplied value is not zero, then REGVAL is modified on return to hold one more than the largest value used to represent any of the keywords in the ARD description. The supplied value is left unchanged if it zero.

STATUS = INTEGER (Given and Returned)

The global status.

ARD_PTWCS
Construct an ARD WCS statement and append it to a GRP group

Description:

This routine creates a WCS statement describing the supplied FrameSet, and appends the statement to the end of the supplied GRP group.

Invocation:

```
CALL ARD_PTWCS( IWCS, IGRP, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
An AST pointer to a FrameSet.

IGRP = INTEGER (Given and Returned)
A GRP group identifier. If GRP__NOID is supplied, a new group is created and its identifier returned.

STATUS = INTEGER (Given and Returned)
The global status.

ARD_WCS
Specify WCS information to be used in future calls to ARD_WORK

Description:

This routine can be used to specify the coordinate systems which can be used in subsequent calls to ARD_WORK. ARD descriptions passed to subsequent calls to ARD_WORK can include positions in any of the Frames included in the supplied FrameSet. The ARD description should include suitable COFRAME or WCS statements to indicate which coordinate system is being used. If no COFRAME or WCS statements are included in the ARD description, then it is assumed that positions within the ARD description are given in the current Frame of the supplied FrameSet, IWCS.

If this routine is not called prior to ARD_WORK (or if it is called with IWCS set AST__NULL), then the ARD description must provide (either directly or through a WCS statement) positions in pixel coordinates.

The FrameSet pointer supplied is simply stored by this routine. If any changes are subsequently made to the FrameSet by the calling routine, then these changes will be visible within ARD_WORK. In particular, if the calling routine annuls the FrameSet pointer, then ARD_WORK will fail.

The supplied FrameSet will be used by all subsequent calls to ARD_WORK until a new FrameSet is specified by calling ARD_WCS again.

Invocation:

```
CALL ARD_WCS( IWCS, DOMAIN, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
An AST pointer to a FrameSet, or AST__NULL.

DOMAIN = CHARACTER * (*) (Given)
The Domain name corresponding to pixel coordinates within the mask array passed to routine ARD_WORK. If a blank value is supplied, "PIXEL " will be used. The IWCS FrameSet (if supplied) must contain a Frame with this Domain. If the supplied string is longer than 40 characters, the trailing characters are ignored.

STATUS = INTEGER (Given and Returned)
The global status.

ARD_WORK
Convert an ARD description into a pixel mask

Description:

This routine returns an array which contains a positive value for all pixels within the areas specified by a given ARD description, and zero for all other pixels.

Invocation:

```
CALL ARD_WORK( IGRP, NDIM, LBND, UBND, TRCOEF, CONCAT, REGVAL, MASK, LBNDI, UBNDI, LBNDE,
UBNDE, STATUS )
```

Arguments:

IGRP = INTEGER (Given)
A GRP identifier for the group holding the ARD description.

NDIM = INTEGER (Given)
The number of pixel axes in the mask array.

LBND(NDIM) = INTEGER (Given)
The lower pixel index bounds of the mask array.

UBND(NDIM) = INTEGER (Given)
The upper pixel index bounds of the mask array.

TRCOEF(0:NDIM, NDIM) = REAL (Given)
The coefficients of the mapping from application coordinates (i.e. default user coordinates) to pixel coordinates. If the first element is equal to VAL__BADR, then a unit mapping is used. This argument is ignored if a call to ARD_WCS has already been made to establish WCS Information.

CONCAT = LOGICAL (Given)
If .TRUE., then an INPUT keyword is inserted at the start of the ARD description so long as the ARD description does not already contain any INPUT keywords. If .FALSE., the ARD description is left as supplied.

REGVAL = INTEGER (Given and Returned)
A positive integer to use to represent the first keyword in the ARD description (excluding INPUT keywords). An error is reported if the value 1 is supplied. If the supplied value is negative or zero, then the value used is one greater than the maximum pixel value supplied in MASK (except that 2 is used if the maximum mask value is 1 or less). On return, REGVAL holds one more than the largest value used to represent any of the keywords in the ARD description.

MASK($*$) = INTEGER (Given and Returned)
The mask array. Any negative values in the supplied array are treated as zero.

LBNDI(NDIM) = INTEGER (Returned)
The lower pixel bounds of a box which encompasses all internal pixels. If there are no internal pixels in the returned mask, each lower bound is returned greater than the corresponding upper bound.

UBNDI(NDIM) = INTEGER (Returned)
The upper pixel bounds of a box which encompasses all internal pixels.

LBNDE(NDIM) = INTEGER (Returned)
The lower pixel bounds of a box which encompasses all external pixels. If there are no external pixels in the returned mask, each lower bound is returned greater than the corresponding upper bound.

UBNDE(NDIM) = INTEGER (Returned)
The upper pixel bounds of a box which encompasses all external pixels.

STATUS = INTEGER (Given and Returned)
The global status.

C Acknowledgements

The current ARD system arose out of previous system produced by Peter Draper. I am grateful to the following people for useful suggestions: Peter Draper, Malcolm Currie, Richard Saxton, Rodney Warren-Smith and Grant Privet.

D Changes Introduced in Version 2.0

- Routines ARD_PLOT has been added to allow border of an ARD description to be drawn on a graphics device.
- Routines ARD_WCS and ARD_PTWCS have been added to support the use of WCS within ARD description.
- The WCS and COFRAME statements have been introduced.
- The COEFFS, STRETCH, OFFSET, SCALE and TWIST statements have been deprecated in favour of the new WCS and COFRAME statements.

E Changes Introduced in Version 2.1

- The COFRAME statement has been extended to include specific support for TimeFrame, SpecFrame and DSBSpecFrame co-ordinate systems.
- The user coordinate system used to describe positions and shapes within an ARD expression can now refer to a sub-space of the axes defined in the application WCS FrameSet. For instance, if the application is an (RA,Dec,freq) cube, an ARD expression can be used that refers only to the freq axis, in which case the same frequency region will be masked in all (ra,dec) planes.

F Changes Introduced in Version 2.2

- A new routine ARD_GTWCS has been added that returns a FrameSet connecting PIXEL coordinates and user coordinates within a specified ARD description.