SUN/114.4

Starlink Project
Starlink User Note 114.4

Malcolm J. Currie
Alan J. Chipperfield

2019 November 30

PAR
Interface to the ADAM Parameter System
Version 2.4
Programmer's Manual

Abstract

PAR is a library of Fortran subroutines that provides convenient mechanisms for applications to exchange information with the outside world, through input-output channels called *parameters*. Parameters enable a user to control an application's behaviour. PAR supports numeric, character, and logical *parameters*, and is currently implemented only on top of the ADAM parameter system.

The PAR library permits parameter values to be obtained, without or with a variety of constraints. Results may be put into parameters to be passed onto other applications. Other facilities include setting a prompt string, and suggested defaults.

This document also introduces a preliminary C interface for the PAR library – this may be subject to change in the light of experience.

Contents

1 Introduction

Consider an application that subtracts four from every element of an array. This has a limited use – just one invocation. However, it would be more general and far more useful if we could subtract an arbitrary value from an arbitrary array. These arbitrary values are called *parameters* and through them we may qualify the behaviour of an application. In a sense they are like input-output channels, though instead of using logical-unit numbers they have memorable names.

PAR is a library of subroutines that makes it easier to import and export numeric, character, and logical parameters between applications and the outside world. In a contouring application, for example, these three types of parameters might be used to find the contour heights, the title for the graph, and whether or not the contours are to be annotated. Currently, PAR is only available for ADAM applications. However, this does mean that an application using PAR will be able to take advantage of features in the ADAM parameter system, such as a variety of defaulting mechanisms, a special null value, online help, and automatic type conversion. These make the application easier to use, more powerful, and more flexible.

PAR enables parameter values to be obtained, with or without constraints. Supported constraints are:

- even, and odd integers;
- within or beyond a range of values; and
- selection from a menu of options, whose contents may be abbreviated.

There are facilities for setting the prompt string, suggested-default values, and minimum and maximum permitted values. There is a routine that lets an application obtain new values repeatedly through the same parameter. Results may be put into parameters to be used in procedures and/or passed on to other applications. Finally, PAR can inquire the *state* of a parameter.

This document explains what a parameter is and the states it may reside in; it describes the usage of the PAR library with examples, progressing from the basic routines to the more sophisticated ones; and gives the linking procedures. There is a reference section containing full details of each subroutine, and classified lists of the routines.

As a consequence of PAR only being implemented using the ADAM parameter system (SUBPAR), the programmer must also write an *interface file* for each application. This controls the behaviour of the parameter, for example, whether it is defaulted or prompted for, where a suggested default should be obtained from, what is the default value, and where help is found. SUN/115 – the *Interface Module Reference Manual* – describes the syntax and facilities of interface files, and is required reading for those using PAR. Note that the examples in this document assume the ADAM implementation.

For the same reason, many error messages you see will be prefixed by 'SUBPAR:', indicating that they emanate from the SUBPAR library.

2 What is a Parameter?

From the programmer's point of view, a PAR parameter is a communication channel between an application and the outside world. In a traditional application you would probably use Fortran input-output statements to prompt the user, or read from a text file for the additional data needed by your programme. The concept of a PAR parameter is similar to this, but instead of passing the information through a logical-unit number, PAR uses a named communication channel. This has advantages over normal methods. A name is more memorable both to the user and to the programmer, especially if it describes the parameter's function. PAR also allows more sophisticated ways of controlling the behaviour of an application than the traditional Fortran I/O, and performs data-type conversion automatically.

3 Parameter States

A parameter has a number of states. Initially, a parameter has no value, and is said to be in the *ground* state. When the parameter has been given a value, the parameter moves to the *active* state. A parameter acquires a value in the first instance by looking for one supplied on the command line. Failing that, the parameter system will attempt to get a value, for example by prompting or adopting a fixed or dynamic default. If an application obtains a value from a parameter already in the active state, the existing value for that parameter is returned. To obtain a further value the application must first *cancel* the parameter and move it to the *cancelled* state. When the application next gets a value for that parameter (within the same invocation), the parameter returns to the active state. When an application completes the parameter returns to the ground state. PAR also permits users of your application to enter a null value for a parameter (a `!` in the ADAM parameter system). This puts the parameter into the *null* state, and can be used to select a special value or control the application in a special way.

4 Error Handling

Like other Starlink subroutine libraries, PAR adheres throughout to the error-handling strategy described in SUN/104. Subroutines have an integer final argument called STATUS that is an inherited status, and will return without action unless the status is set to the value SAI__OK when they are invoked. (SAI__OK is a symbolic constant defined in the include file SAE_PAR.) This strategy greatly simplifies the coding since you can make a series of subroutine calls without having to check STATUS after each call; if an error occurs the subsequent routines do nothing. [1] Where necessary, PAR makes error reports through the ERR_ routines in the manner described in SUN/104, and therefore sets STATUS to a value other than SAI__OK.

[1] PAR_CANCL and PAR_UNSET are exceptions to this behaviour. Since they are freeing resources they attempt to work regardless of the inherited status.

5 Basic Routines

5.1 Getting a Parameter's Value

The way an application obtains a value from a parameter is to *get* the value. In simple terms, a get operation is the equivalent of a Fortran read, though it can do much more. In this example,

```
            CALL PAR_GET0L( 'FLAG', VALUE, STATUS )
```

gets a logical value from the parameter FLAG and stores it in the logical variable VALUE. The data type is logical because of the L suffix in the routine's name. If VALUE were a character variable, you would use a similar routine, but this time with a C suffix as shown below.

```
            CALL PAR_GET0C( 'STRING', VALUE, STATUS )
```

There are similar routines for the other primitive data types, each with an appropriate suffix: D for double precision, I for integer and R for real. By convention where a routine has variants for different data types it has an "x" suffix, so in this case the routines are known collectively as PAR_GET0x. Note that there are no routines for obtaining byte and word values. These must be obtained using the integer version of the routine. The variable STATUS is the inherited status value adhering to the rules in SUN/104, and summarised in Section 4. The 0 in the routine name indicates that the routine passes a scalar (zero dimensions). Likewise a routine with a 1 in its name passes a vector, and with an N it passes an *n*-dimensional array.

There is a maximum length for the parameter name (set by ADAM) of fifteen characters, but this is quite adequate except perhaps for lexicographers.

The actual value of the parameter supplied to the application at run time need not necessarily have the same data type as you have requested in your application. The parameter system will handle conversions between the type of the value and the type required by the application. So you only need to concern yourself with what your application requires.

What does the user see? This will depend on the implementation, but suppose the user is prompted for a value. If you had in your code

```
            CALL PAR_GET0R( 'THRESHOLD', THRESH, STATUS )
```

The user might see

```
      THRESHOLD - Give the intensity threshold >
```

with the parameter name first, followed by a prompt string briefly describing the function of the parameter. (In the ADAM implementation this string is usually defined within the interface file, but there is a PAR routine for setting it within the application itself.) Suppose the user enters 32768 in response to the above prompt. The next time the user ran the application there could be a suggested default of the last-used value like this,

```
THRESHOLD - Give the intensity threshold /32768/ >
```

which could be accepted merely by pressing the return key.

In the ADAM implementation the PAR_GET calls will (re-)prompt whenever the user gives too many values or an invalid value. Invalid values include supplying a character string where a number is required, and being outside the interface file *range* or *in* constraints.

5.2 Getting an Array of Values

In addition to obtaining a scalar value, there are routines for getting arrays of values. At this point it is appropriate to say a few words about the shape of a parameter. A parameter's shape may be scalar, vector, or *n*-dimensional. When you get values from a parameter, the parameter acquires the shape of the values supplied by the user. Therefore, you could use the same parameter to obtain a scalar and a cube. How much is the user restricted by the parameter shape? When getting an array the application specifies the maximum number of dimensions and the maximum sizes along each dimension; the values supplied must not exceed these maxima, but there may be fewer. Now we return to the descriptions of the basic PAR_ routines.

Here is an example to get a vector of real values.

```
      REAL VALUES( 4 )

          :       :       :

      CALL PAR_GET1R( 'HEIGHTS', 4, VALUES, ACTVAL, STATUS )
```

It enables all or part of a four-element array to be obtained and stored in the real variable VALUES. ACTVAL is the actual number of values obtained from parameter HEIGHTS. Again there are versions of this routine for the different data types listed earlier.

In the above example the user might enter

```
[1,2.5D0,3.456]
```

where the brackets indicate an array and the commas separate the values.[2] For this input, ACTVAL becomes 3.

A parameter's values may be stored as an *n*-dimensional array. The following example shows a three-dimensional mask of integers being obtained from parameter MASK and being stored in an array called VALUES. MAXD gives the maximum dimensions of the array. The array itself must be large enough to hold the values. ACTD receives the actual dimensions of the array as specified by the user.

```
      INTEGER VALUES( 3, 2, 2 )

          :       :       :

      MAXD( 1 ) = 3
      MAXD( 2 ) = 2
      MAXD( 3 ) = 2
      CALL PAR_GETNI( 'MASK', 3, MAXD, VALUES, ACTD, STATUS )
```

[2]In the ADAM implementation the user can omit the brackets in response to a prompt.

This routine is unlikely to be in common usage because of the impracticality of the user of your application entering more than the smallest *n*-dimensional arrays of parameters, and having to know the parameter's shape. To illustrate the former point, in the above example the user might enter

```
[[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]]
```

where the nested brackets delimit the dimensions, and the commas separate the values within a dimension. Thus there are three values along the first dimension; each of these triples is bracketed in pairs along the second dimension; and this is repeated for each element along the third dimension, within the outermost brackets. In other words the first dimension increases fastest, followed by the second, and so on – the Fortran order. Thus, in the example, VALUES(3,1,2) is 9.

Usually, it will suffice to get a vector of values disregarding the dimensionality of the parameter.

```
      REAL VALUES( 12 )

          :       :       :

      CALL PAR_GETVR( 'MASK', 12, VALUES, ACTVAL, STATUS )
```

This has the same arguments as PAR_GET1x we saw above, though the actual number of array dimensions of the parameter need not be one.

There may be occasions when you want an exact number of values. Below, we obtain two character variables from parameter 'CONSTELLATION'. When too few values are supplied to the parameter, PAR_EXACx prompts for the remainder of the values.

```
      CHARACTER * ( 3 ) CONSTE( 2 )

          :       :       :

      CALL PAR_EXACC( 'CONSTELLATION', 2, CONSTE, STATUS )
```

An example of what the user sees is given in Section 7.2.

5.3 Putting a Parameter's Value

The opposite to *get* is *put*. Using traditional Fortran input-output, the equivalent of putting a value is to write the value to a file. In addition to getting parameter values, PAR can also put values into parameters. Suppose we have an application that computes statistical parameters for a data array. We should like to store these statistics in parameters, so that they may be passed to a later application. Thus we could have a number of puts – one per statistic. Here we just put the standard deviation value in the parameter SIGMA.

```
      CALL PAR_PUTOD( 'SIGMA', STDDEV, STATUS )
```

The value is written to a *parameter file* associated with the application. This file contains not only any values you have put there, but also the last-used values of all the parameters of that application.[3]

If we had the values corresponding to the quartiles and median of a data array we could use a routine that handles a one-dimensional array.

```
DOUBLE PRECISION QRTILE( 3 )

    :       :       :

CALL PAR_PUT1D( 'QUARTILES', 3, QRTILE, STATUS )
```

When you put an array, the subroutine arguments specify both the dimensionality and size of that array, and the size of an 'object' into which the array is to be put.[4]

Analogous to the PAR_GETNx and PAR_GETVx routines for getting arrays of values, there are equivalent generic routines, PAR_PUTNx and PAR_PUTVx, for putting values in parameters. Using PAR_PUTNx in one application and accessing the array via PAR_GETNx without prompting is the most likely way to handle *n*-dimensional arrays of parameter values.

See SUN/115 Appendix C for details of the specifications needed within the interface file to deal with scalar and array output parameters.

5.4 Cancelling a Parameter

Cancelling a parameter means that values associated with it are lost. It is most-commonly used for processing in a loop.

If we have a loop in which we wish to get new values for a parameter and do some calculations in each iteration, we must cancel any parameters obtained in the loop, otherwise the first value will be used repeatedly, since the parameter is in the active state.

Here is an illustration. ITER calculations are performed, using the value of parameter REJECT.

```
DO I = 1, ITER
   CALL PAR_GETOD( 'REJECT', ABC, STATUS )

      < perform calculation involving variable ABC >

   CALL PAR_CANCL( 'REJECT', STATUS )
END DO
```

[3] A parameter file is implementation specific, however it is expected to be part of any implementation. In the ADAM parameter system the parameter file is an HDS file named after the application and located in the ADAM_USER directory. So for an application called STATISTICS, the above parameter value could be 'read' by a parameter in a different application by the user giving the HDS object name with an `@` prefix, namely `@ADAM_USER:STATISTICS.SIGMA` for VMS and `@$ADAM_USER/statistics.SIGMA` for UNIX.

[4] In the ADAM implementation this is usually a component of the parameter file – in this case an object of the required size will be created. However, if indirection to a named HDS file is used the shape specified in the PAR routine must match the object's shape.

5.5 Setting the Dynamic Prompt

In the ADAM implementation, the prompt a user of an application sees usually originates from the interface file, and hence is fixed. However, it is possible to create a prompt string dynamically from within the application. For example,

```
      CALL PAR_PROMT( 'OK', 'The file '// NAME( :NC ) /
     :                / ' is to be erased. OK ?', STATUS )
```

will change the prompt string for parameter OK, so that the named file given by the variable NAME, may be substituted at run time.

5.6 Setting the Dynamic Default

When the parameter system prompts for a parameter, there is usually a suggested default given between / / delimiters, so that the user can just press <CR> to use the suggestion, or edit the suggestion. The suggested value can originate from a number of places. This includes from within an application, by setting a *dynamic* default.

In the following example the dynamic default for the scalar parameter SHADE depends on the value of variable COLOUR. If COLOUR is one, the dynamic default for parameter SHADE will be −1.0, and 0.0 otherwise.

```
*  Set the default shading (parameter SHADE) depending on the colour.
      IF ( COLOUR .EQ. 1 ) THEN
         CALL PAR_DEFOR( 'SHADE', -1.0, STATUS )
      ELSE
         CALL PAR_DEFOR( 'SHADE', 0.0, STATUS )
      END IF

*  Obtain the shading.
      CALL PAR_GETOR( 'SHADE', SHADE, STATUS )
```

Note that in ADAM applications the *ppath* in the interface file must begin `DYNAMIC` to ensure that the user of your application will be prompted with the dynamic default (see SUN/115).

The ADAM parameter system also allows the dynamic default to be used as the parameter value without prompting, through the interface file *vpath* (see SUN/115).

There are also routines that set the dynamic default for a vector parameter (PAR_DEF1x) and an array parameter (PAR_DEFNx). Again there are versions of this routine for the different data types listed earlier.

5.7 Setting Lower and Upper Limits

It is a common requirement to restrict a parameter's value to a specific range, or to be above some lower limit or less than some upper limit. For example, the order of a polynomial would have to be positive, but less than some maximum value that the polynomial-fitting software can handle. Whilst this can be done using the *range* field in the interface file, PAR offers dynamic control through two routines – PAR_MAXx to set the maximum value, and PAR_MINx to set the minimum value for the parameter. Note that there are no logical-type versions of these routines.

Here is an example of setting the minimum value.

```
*  The number of histogram bins must be positive.
      CALL PAR_MINI( 'NBINS', 1, STATUS )
      CALL PAR_GET0I( 'NBINS', NBINS, STATUS )
```

We can set an upper limit too.

```
*  The number of histogram bins must be in the range 10 to 1000
*  inclusive.
      CALL PAR_MINI( 'NBINS', 10, STATUS )
      CALL PAR_MAXI( 'NBINS', 1000, STATUS )
      CALL PAR_GET0I( 'NBINS', NBINS, STATUS )
```

If the user gives a value outside the legitimate range, an error report appears, and the user is prompted for an acceptable value. In the case of character parameters, the parameter's value follows the minimum value and precedes the maximum value in the ASCII collating list. [5]

If the minimum value is greater than the maximum at the time you get a parameter value, this instructs PAR not to permit values between the minimum and maximum. So the maximum is always the value immediately below the range to be excluded, and the minimum is the value immediately above. In case that is not clear here is an illustration.

```
*  Choose an integer excluding -1, 0, 1, and 2.
      CALL PAR_MINI( 'IVALUE', 3, STATUS )
      CALL PAR_MAXI( 'IVALUE', -2, STATUS )
      CALL PAR_GET0I( 'IVALUE', NUMBER, STATUS )
```

5.8 Cancelling Dynamic Values

5.8.1 Range

Having set a minimum or a maximum value, you can alter its value by a further call to PAR_MINx or PAR_MAXx. Like a parameter itself, these control values remain in effect until the end of an application, or until they are cancelled. Therefore, should you no longer want either or both of the maximum and minimum limits, you must cancel them with PAR_UNSET. This routine's second argument is a comma-separated list which selects the values to reset.

In the following example parameter NX is obtained twice, first using a range, and then with no maximum limit and a different minimum.

```
*  Get the integer value between 1 and 360.
      CALL PAR_MINI( 'NX', 1, STATUS )
      CALL PAR_MAXI( 'NX', 360, STATUS )
      CALL PAR_GET0I( 'NX', NX, STATUS )

*  Cancel the parameter, as a minimum or maximum value cannot be set
*  when the parameter is active, and because we want to obtain
```

[5] In the ADAM implementation the dynamic minimum and maximum must lie within the limits set by the range field in the interface file. If not, the get routine returns with an error status. The allowed values reside in the intersection set of the two ranges. Users can also specify MIN or MAX for a parameter value to assign the minimum or maximum value to that parameter.

```
      *  another value.
            CALL PAR_CANCL( 'NX', STATUS )

      *  Set a new minimum value.
            CALL PAR_MINI( 'NX', NX, STATUS )

      *  Cancel the maximum value.
            CALL PAR_UNSET( 'NX', 'MAX', STATUS )

      *  Obtain a new value, with a new minimum and no maximum.
            CALL PAR_GET0I( 'NX', NX2, STATUS )
```

See Appendix G for details of the options for the second argument.

5.8.2 Dynamic Defaults

PAR_UNSET need not be called to change the value of a default; a call to one of the PAR_DEF routines will suffice. However, if you wish to no longer have a dynamic default for parameter NX say, you will need to make the following call.

```
      CALL PAR_UNSET( 'NX', 'DEFAULT', STATUS )
```

5.9 Inquiring the State of a Parameter

PAR provides a routine for inquiring the state of a parameter. One practical use for this is to determine whether a parameter is specified on the command line or not, and hence to affect the behaviour of the application. For example, you have an application that loops for efficient and easy interactive use, but in a procedure or batch mode you want the application to process just one set of data.

```
            INCLUDE 'PAR_PAR'        ! PAR constants
            INTEGER STATE

                :       :       :

            CALL PAR_STATE( 'INIT', STATE, STATUS )

            LOOP = .TRUE.
            DO WHILE ( LOOP .AND STATUS .EQ. SAI__OK )
               CALL PAR_GETOR( 'INIT', START, STATUS )

                  < perform calculation >

               LOOP = STATE .NE. PAR__ACTIVE
               IF ( LOOP ) CALL PAR_CANCL( 'INIT', STATUS )
            END DO
```

INIT is a parameter required for each calculation. It is obtained within a code loop. If INIT is specified on the command-line, INIT is in the active state before the call to PAR_GET0R. So a logical expression involving the state decides whether there is but one cycle around the loop or many. The include file PAR_PAR contains the definitions of each of the states returned by PAR_STATE. The next section describes how we might end the loop in the interactive case.

6 Abort and Null

There are two special status values that can be returned by the parameter system when getting parameter values. They are PAR__ABORT and PAR__NULL, and are defined in the include file PAR_ERR.

6.1 Abort

PAR__ABORT is returned if a user enters `!!` in response to a prompt. In this event you should ensure that your application exits immediately. In practice this means that a) the user is not be prompted for any further parameters, though you do not have to check for an abort status after every PAR_GET call – the inherited status will look after that – and b) the application exits cleanly, freeing any resources used. There must be a status check before any of the parameter values are used, in case they have not been obtained successfully.

For example

```
      INCLUDE 'PAR_ERR'       ! PAR error constants

          :       :       :

      CALL PAR_GET0I( 'STEP', NSTEP, STATUS )
      CALL PAR_GET1R( 'HEIGHTS', 50, HEITS, NUMHTS, STATUS )
      CALL PAR_GET0L( 'COLOUR', COLOUR, STATUS )
      IF ( STATUS .EQ. PAR__ABORT ) THEN
         < perform any tidying operations >
         GOTO 999
      END IF

      < perform calculations >

  999 CONTINUE
      END
```

performs three parameter get calls, before testing for an abort; if an abort status is found, the routine exits. The status check against PAR__ABORT is unusual, and it is included to illustrate the symbolic constant. In normal circumstances you would test whether STATUS and SAI__OK were not equal, since it checks whether any error has occurred.

6.2 Null

PAR__NULL indicates that no value is available, and may be interpreted according to circumstances. This normally occurs when a user assigns a value of `!` to a parameter.

Null can be used in a variety of circumstances. These include to end a loop, to indicate that an optional file is not required, and to force a default value to be used.

Here is an example of handling null as a valid value for a parameter.

```
          CALL ERR_MARK
          CALL PAR_GETOR( 'WEIGHT', WEIGHT, STATUS )
          IF ( STATUS .EQ. PAR__NULL ) THEN
             WEIGHT = 1.0
             CALL ERR_ANNUL( STATUS )
          END IF
          CALL ERR_RLSE
```

When the PAR_GET0R returns with the null status, the variable WEIGHT is set to 1.0. This particular instance has limited value, but the constant could be dynamic, depending on the values of other parameters or data read from files. The call to ERR_ANNUL restores STATUS to SAI__OK, and also removes an error message that would otherwise be reported later (usually when the application completes). Since we do not want to lose any earlier error reports, ERR_MARK and ERR_RLSE bracket this fragment of code to set up and release an error context. See SUN/104 for further details.

Here null ends a loop, say to plot an histogram with different numbers of bins.

```
      100 CONTINUE
          CALL ERR_MARK
          CALL PAR_GETOI( 'NBINS', NBIN, STATUS )
          IF ( STATUS .EQ. PAR__NULL ) THEN
             CALL ERR_ANNUL( STATUS )
             CALL ERR_RLSE
             GOTO 200
          END IF
          CALL ERR_RLSE

              < Perform calculations >

          GOTO 100

    *   Come here at the end of the calculations.
      200 CONTINUE
```

7 Extended Routines

In addition to the basic routines there are some packaged facilities for common sequences of calls. Most deal with dynamic control of acceptable values when getting the parameter. Some are needed because the *in* choices and *range* limits imposed by the ADAM interface module are static, and therefore cannot be adjusted depending on the values of other parameters or data. Note that in the extended routines that follow, all call a PAR_GET*n*x routine which first checks that each supplied value satisfies these interface-file constraints, and prompts for a new value if it does not. Only after these tests are made will each value be compared with the constraints in the extended routines. Therefore, you are recommended not to use the in and range fields for parameters obtained using the extended routines.

Here is an example of handling your own constraints to illustrate why it is more convenient to use the packaged routines, where applicable, and also to demonstrate how you might write your own extended routine should you need one. It is a little contrived, since the PAR library already provides the most-common combinations.

Here we obtain from parameter NAME an integer that is not exactly divisible by 3, set the dynamic default to 1, and interpret null to mean set the value to 3. See SUN/104 for details of the MSG_ calls.

```
      LOGICAL NOTOK
      INTEGER VALUE

          :       :       :

*  Set the dynamic default.
      CALL PAR_DEF0I( 'NAME', 1, STATUS )

*  Start a new error context.
      CALL ERR_MARK

*  Loop to obtain the value of the parameter.
*  ==========================================

*  Initialise NOTOK to start off the loop and indicate that a
*  satisfactory value has yet to be obtained.
      NOTOK = .TRUE.

  100 CONTINUE

*  The loop will keep going as long as a suitable value has not be read
*  and there is no error.
         IF ( .NOT. NOTOK .OR. ( STATUS .NE. SAI__OK ) ) GOTO 120

*  Get a value from the parameter.
         CALL PAR_GET0I( 'NAME', VALUE, STATUS )

*  Check for an error before using the value.
         IF ( STATUS .EQ. SAI__OK ) THEN

*  Test the value against the constraints.  Here it is just to see
*  if the value is divisible by 3.  You can replace this expression
*  with more complicated constraints.
            NOTOK = MOD( VALUE, 3 ) .EQ. 0

*  The value is not within the constraints, so report as an error,
*  including full information using tokens.  You would substitute a
*  routine name for fac_xxxxx.
            IF ( NOTOK ) THEN
               STATUS = SAI__ERROR
               CALL MSG_SETC( 'PARAM, 'NAME' )
               CALL MSG_SETI( 'VALUE', VALUE )

               CALL ERR_REP( 'fac_xxxxx_OUTR',
     :           '^VALUE is not permitted for ^PARAM.  Please give '/
     :           /'an integer not exactly divisible by 3.', STATUS )
```

```
*  The error is flushed so the user can see it immediately.
               CALL ERR_FLUSH( STATUS )

*  Cancel the parameter to enable a retry to get a value satisfying
*  the constraint.
               CALL PAR_CANCL( 'NAME', STATUS )
            END IF

*  Use the default value following an error.
         ELSE

*  Annul a null error to prevent an error report about null appearing.
*  Create a message informing the user of what has happened.
            IF ( STATUS .EQ. PAR__NULL ) THEN
               CALL ERR_ANNUL( STATUS )

*  If MSG verbose output is requested, inform the user what has happened.
*  You would substitute a routine name for fac_xxxxx.
               CALL MSG_SETC( 'PARAM', 'NAME' )
               CALL MSG_OUTIF( MSG__VERB, 'fac_xxxxx_DEFA',
    :            'A value of 3 has been adopted '/
    :            /'for parameter ^PARAM.', STATUS )
            END IF

*  Set the returned value to the special case.
            VALUE = 3

*  Terminate the loop.
            NOTOK = .FALSE.
         END IF

*  Go to the head of the loop.
         GOTO 100

*  Come here when the loop has been exited.  This includes when
*  an error status was returned by the PAR_GET0I routine.
  120 CONTINUE

*  Release the new error context.
      CALL ERR_RLSE
```

To set another constraint you have to modify the logical expression for variable NOTOK, and revise the ERR_REP error report, possibly with more tokens. Suppose variables VMIN and VMAX define a range of permitted values, which is inclusive when VMAX is the larger of the two and exclusive when VMAX is less than VMIN. The constraint expression could modified to the following.

```
*  Check that the value is within the specified include or exclude
*  range, and not divisible by 3.
      IF ( VMIN .GT. VMAX ) THEN
         NOTOK = ( ( VALUE .LT. VMIN ) .AND. ( VALUE .GT. VMAX ) )
     :           .OR. ( MOD( VALUE, 3 ) .EQ. 0 )
      ELSE
```

```
            NOTOK = ( VALUE .LT. VMIN ) .OR. ( VALUE .GT. VMAX ) .OR.
     :                ( MOD( VALUE, 3 ) .NE. 0 )
         END IF
```

and the error report would look something like this

```
*  The value is not within the constraints, so report as an error,
*  including full information using tokens.
         IF ( NOTOK ) THEN
            STATUS = PAR__ERROR
            CALL MSG_SETC( 'PARAM', 'NAME' )
            CALL MSG_SETI( 'VALUE', VALUE )
            CALL MSG_SETI( 'MIN', VMIN )
            CALL MSG_SETI( 'MAX', VMAX )
            IF ( VMIN .GT. VMAX ) THEN
               CALL MSG_SETC( 'XCLD', 'outside' )
            ELSE
               CALL MSG_SETC( 'XCLD', 'in' )
            END IF

            CALL ERR_REP( 'fac_xxxxx_OUTR',
     :        '^VALUE is not permitted for ^PARAM.  Please give '/
     :        /'an integer ^XCLD the range ^MIN to ^MAX, and not '/
     :        /'exactly divisible by 3.', STATUS )
```

In practice you would probably define a logical variable outside the loop to indicate whether the range was inclusive or exclusive.[6]

Having looked 'behind the scenes', we can now look at what the PAR_ extended routines offer.

7.1 Scalar Values between Limits

A parameter value may be forced to lie within a range using a call to the generic routine PAR_GDR0x. The name is derived from *G*et with a *D*efault and *R*ange. For example,

```
         CALL PAR_GDR0R( 'SCALE', 1.0, 0.0, 2.0, .FALSE., SCAFAC, STATUS )
```

gets the value for the real parameter SCALE using a dynamic default of 1.0, stores the value in the variable SCAFAC, and ensures that the value lies in the range 0.0–2.0. If a supplied value is out of this range, PAR_GDR0R reports the acceptable limits and prompts the user for another value. (This applies to all the routines with constraints described in Section 7.) Thus the user would see something like the following.

```
SCALE - Scale factor /1.0/ > 3.5
!! SUBPAR: 3.5 is greater than the MAXIMUM value 2.
SCALE - Scale factor /1.0/ >
```

[6]In the ADAM implementation users of your application may wish to take advantage of the MAX/MIN facility. If this is so you will need to add calls to PAR_MAXI and PAR_MINI in the previous example.

The fifth argument (NULL) will normally be false. When it is true, it instructs the PAR routine to return the dynamic-default value whenever a null value is supplied, and then to annul the error status. If the MSG filtering level (see SUN/104) is set to 'verbose', an informational message will be output.

In the above case if NULL were made .TRUE., and the user entered the null symbol whilst in 'verbose' mode, the dialogue would be as follows:

```
SCALE - Scale factor /1.0/ > !
!! A value of 1 has been adopted for parameter SCALE.
```

NULL should only be set true when the dynamic default will *always* give reasonable behaviour in the application.

For all but the last example in Section 7, the null flag is set to .FALSE., and will not be mentioned again until then.

Swapping the limits lets you exclude values in the range. Therefore,

```
      CALL PAR_GDROR( 'SCALE', 1.0, 2.0, 0.0, .FALSE., SCAFAC, STATUS )
```

would permit a value of 3.5, unlike before, as well as 0.0 and 2.0. However, 1.5 would not now be acceptable.

```
SCALE - Scale factor > 1.5
!! SUBPAR: 1.5 is in the excluded MIN/MAX range between 0 and 2.
SCALE - Scale factor >
```

Notice that the suggested default has disappeared.[7] This occurred because 1.0 violates the range constraint, and it is a feature of the PAR_ extended routines. If you do not want a dynamic default, set the dynamic-default argument to a value that violates the constraints. The range-exclusion feature is present in all the PAR_ extended routines that have a range constraint.

Not surprisingly, there are only versions of the range-constraint routines for integer, real, or double-precision parameters. These have the usual I, R, and D suffices respectively. The range limits have the same data type as the value.

7.2 Arrays of Values between Limits

There are similar generic routines for obtaining an array of values between limits. The first permits up to a given number of values to be obtained. We modify the example from the previous section to illustrate this routine.

```
      REAL SCAFAC( 10 )

          :       :       :

      CALL PAR_GDRVR( 'SCALE', 10, 0.0, 2.0, .FALSE., SCAFAC,
     :                NVAL, STATUS )
```

[7]This assumes that the ppath in the ADAM interface file starts with 'DYNAMIC'.

This time up to ten values may be obtained from parameter SCALE, and stored in the array SCAFAC. All the values must lie in the range 0.0–2.0. Since the number of values is not fixed, there is no dynamic-default argument. PAR_DEF1x can be called prior to PAR_GDRVx, if desired.

If you want an exact number of values, there is a routine to do this for you. For example, suppose that you want red, green, and blue intensities to define a colour, then the following code would obtain exactly three normalised intensities – one for each colour – between zero and one.

```
*  Get an RGB colour.  The default is yellow.
      DEFAUL( 1 ) = 1.0
      DEFAUL( 2 ) = 1.0
      DEFAUL( 3 ) = 0.0
      CALL PAR_GDR1R( 'RGB', 3, DEFAUL, 0.0, 1.0, .FALSE., RGB, STATUS )
```

The default must be an array. Should you not want a dynamic default in this case, just set the first element of DEFAUL to be negative or greater than one.

From the user's perspective, instructions will be given if additional values are required.

```
RGB - Red, green, and blue intensities /[1.0,1.0,0.0]/ > 1.0,0,0,0.5
!! SUBPAR: No more than 3 elements are allowed for parameter RGB.
RGB - Red, green, and blue intensities /[1.0,1.0,0.0]/ > 1.0,0.0
!! 1 more value is still needed.
RGB - Red, green, and blue intensities /[1.0,1.0,0.0]/ > 1.0,0.0,0.0
!! SUBPAR: No more than 1 element is allowed for parameter RGB.
RGB - Red, green, and blue intensities /[1.0,1.0,0.0]/ > 0.0
```

First the user gives too many values, and is told how many to give. Next too few are supplied, so PAR_GDR1R asks for another. Again the user is not paying attention and gives the whole RGB value instead of just the outstanding blue intensity. Finally, the user gives the last value, yielding an RGB of 1.0,0.0,0.0 or the colour red.

Another variation of the theme is when you want to have an exact number of values, and each value is constrained to its own range. For instance, suppose that you wanted to obtain the two-dimensional co-ordinates of a point within a rectangle, whose bounds are known. The following code could obtain the required values, where LBND and UBND define the lower and upper co-ordinates of the rectangle.

```
      DOUBLE PRECISION DEFAUL( 2 ), POS( 2 )
      DOUBLE PRECISION LBND( 2 ), UBND( 2 )

          :       :       :

*  Set the limits of the area in which the point must lie.
      LBND( 1 ) = 0.0D0
      UBND( 1 ) = 50.0D0
      LBND( 2 ) = -20.0D0
      UBND( 2 ) = 10.0D0

*  Get the co-ordinates of the point within the rectangle.  There
*  is no dynamic default because the default violates the range
*  constraint.
      DEFAUL( 1 ) = -1.0D0
      CALL PAR_GRM1D( 'POSITION', 2, DEFAUL, LBND, UBND,
     :                .FALSE., POS, STATUS )
```

PAR_GRM1x will inform the user of any value(s) that violate a range, and prompts for new values. The name is derived from *G*et with *R*anges for *M*ultiple dimensions.[8]

If you want to constrain each value of an array to its own range, but do not require an exact number of values, there is even a PAR routine to do that. PAR_GRMVx returns up to some maximum number of values of values, each within a defined range. In the following example an application needs integer compression factors (COMPRS) along each of NDIM dimensions. These must be positive and no greater than the size of the array along each dimension, and are set by the CMPMIN and CMPMAX arrays. ACTVAL returns the actual number values obtained from parameter COMPRESS, and in this example, it is used to set no compression for the higher dimensions.

```
*  Set the acceptable range of values from no compression to compress
*  to a single element in a dimension.
      DO I = 1, NDIM
         CMPMIN( I ) = 1
         CMPMAX( I ) = DIMS( I )
      END DO

*  Get the compression factors.
      CALL PAR_GRMVI( 'COMPRESS', NDIM, CMPMIN, CMPMAX, COMPRS, ACTVAL,
     :                STATUS )

*  Should less values be supplied than the number of dimensions, do
*  not compress the higher dimensions.
      IF ( ACTVAL .LT. NDIM ) THEN
         DO I = ACTVAL + 1, NDIM
            COMPRS( I ) = 1
         END DO
      END IF
```

7.3 Parity Constraint

There are occasions when you need an odd or even integer. For example, the size of a smoothing kernel must be odd. PAR provides two routines with these functions. Thus,

```
      CALL PAR_GODD( 'BOX', 5, 3, 11, .FALSE., BOXSIZ, STATUS )
```

obtains an odd integer between 3 and 11 from parameter BOX, and stores it in variable BOXSIZ. The dynamic default is 5. Similarly,

[8]In retrospect the name probably should have been PAR_GDM1x for *G*et with *D*efaults and *M*ultiple ranges, but the existing name is already in use in applications.

```
      CALL PAR_GEVEN( 'OFFSET', 2, -2, 10, .FALSE., OFFS, STATUS )
```

gets a even value in the range −2 to 10 from parameter OFFSET, with a dynamic default of 2. Zero counts as an even number, and so it would be acceptable here.

7.4 Menu Constraint

A common requirement is to select an option from a menu. PAR_CHOIC offers this functionality. You provide a list of options separated by commas. When the user selects a choice not in this menu, an error report appears that lists the available options, and the user is prompted.[9] PAR_CHOIC permits the user of your application to use an unambiguous abbreviation. However, the unshortened value is returned. The value is also in uppercase, though the list of options need not be so.

```
      CHARACTER * 10 FUNCT, OPTDEF
      CHARACTER * 72 OPLIST

          :       :       :

      OPLIST = 'Exit,Device,Histogram,List,Peep,'/
     :         /'Region,Save,Slice,Statistics,Value'
      OPTDEF = 'Region'
      CALL PAR_CHOIC( 'OPTION', OPTDEF, OPLIST, .FALSE., FUNCT, STATUS )
```

So in the above example there are ten options available for parameter OPTION. The dynamic default is 'Region'. If you assign the second argument with a value not in the main list, such as a blank string, it instructs PAR_CHOIC not to set a dynamic default. Note that PAR_CHOIC only returns character values.

This is what the user might see for the above example.

```
OPTION - Inspection option /'Region'/ > View
!! The choice View is not in the menu.  The options are
!     Exit,Device,Histogram,List,Peep,Region,Save,Slice,Statistics,Value.
!! Invalid selection for parameter OPTION.
OPTION - Inspection option /'Region'/ > S
!! The choice S is ambiguous.  The options are
!     Exit,Device,Histogram,List,Peep,Region,Save,Slice,Statistics,Value.
!! Invalid selection for parameter OPTION.
OPTION - Inspection option /'Region'/ > lust
Selected the nearest match "LIST" for parameter OPTION.
```

The first value is unacceptable as it is not in the menu. The second is ambiguous because there are several options beginning with an ess. Had the user entered `sl` say, the abbreviation would have selected a value of 'SLICE'. The final value appears not to be in the list of choices, but PAR_CHOIC allows the user one typing mistake, and so the user actually selects option 'LIST'. In this case, a warning message is output unless the MSG filtering level (see SUN/104) is set to 'quiet'.

There is a similar routine – PAR_CHOIV – to get a vector of character values from a menu.

[9]Remember that in the ADAM implementation the PAR_GET0C call made by PAR_CHOIC will first test the obtained value against the range or in fields in the interface file. If the supplied value is unacceptable, the user will be prompted by PAR_GET0C. Only once these constraints are passed will the value be tested against the menu.

7.5 Combined Constraints

Should you wish to combine the constraints of a numeric value within a range, and a list of options, there are PAR routines to do it for you. Here is an illustration to show how this might be profitably used. The application from which the following extract is taken wants to replace certain array values with a constant; this can either be a numeric value or set to the bad-pixel value.

```
      CHARACTER * ( VAL__SZR ) CNEWLO, THLDEF

          :       :       :

*  Get the replacement value.
      THLDEF = '0.0'
      CALL PAR_MIXOR( 'NEWLO', THLDEF, VAL__MINR, VAL__MAXR,
     :                'Bad', .FALSE., CNEWLO, STATUS )

      IF ( STATUS .EQ. SAI__OK ) THEN

*  It may be the bad-pixel value.
         IF ( CNEWLO .EQ. 'BAD' ) THEN
            NEWLO = VAL__BADR
         ELSE

*  Convert the output numeric string to its numeric value.
            CALL CHR_CTOR( CNEWLO, NEWLO, STATUS )
         END IF
      END IF
```

This obtains from parameter NEWLO a value which is either 'BAD' or a real value. The value is returned in variable CNEWLO. VAL__MINR and VAL__MAXR are the minimum and maximum values, and VAL__BADR is the bad value, for the real data type, and VAL__SZR is the maximum number of characters needed to store a real value; all symbolic constants are defined in the PRM_PAR include file (SUN/39). The dynamic default is '0.0'. Remember that the dynamic default and returned value are strings. Therefore, you must first test the returned value for being any of the items on the menu, before converting it to a number. Since the returned value may be undefined following an error, it is prudent to check the status before using the value.

Should the user give an unacceptable value, PAR_MIX0x informs the user of the error and lists the available options, and then invites the user to supply another value.

Here is a more subtle example. This only has numbers in the menu, but it is advantageous when only certain numeric values are acceptable.

```
*  Get the plate number.
      CALL PAR_MIX0I( 'PLATE', ' ', 101, 1500, '5,11,23,47,49',
     :                .FALSE., CPLATE, STATUS )
      IF ( STATUS .EQ. SAI__OK )
     :  CALL CHR_CTOI( CPLATE, PLATNO, STATUS )
```

Here PAR_MIX0I obtains an 'integer' that is either 5, 11, 23, 47, 49, or in the range 101–1500, and returns it in the *character* variable CPLATE. CHR_CTOI converts the string into a true integer value, PLATNO. Since the second argument of PAR_MIX0I is a blank string, there is no dynamic default. To produce a suggested default, the second argument would have to be one of the menu options, or satisfy the range constraint (when converted to an integer). If we had swapped the range limits, PAR_MIX0I would allow all values not in the range 102–1499.

7.6 Logical Value

PAR_GTD0L obtains a scalar logical value, with a dynamic default defined, and has the capability of handling a null status. Thus the following obtains a value from the parameter SWITCH and stores it in the variable called POWER. The dynamic default is .TRUE..

```
      NULL = .FALSE.
      CALL PAR_GTD0L( 'SWITCH', .TRUE., NULL, POWER, STATUS )
```

The third argument is the same as we met in Section 7.1. If we reverse the polarity of NULL, PAR_GTD0L will assign the dynamic default to POWER whenever the parameter is in the null state, and returns with STATUS set to SAI__OK. To reiterate NULL=.TRUE. should only be used when the dynamic default will *always* give reasonable behaviour in the application. This is highly likely for a logical value.

8 Other Types of Parameters

Besides the numeric, character, and logical parameters we have seen so far, there is another class of parameter to which some of the PAR routines may be applied. In ADAM, these are parameters that specify the name of an HDS or Fortran data file, a magnetic-tape or graphics device.

The values are obtained from such parameters by calling special routines within the particular subroutine library that connect to the ADAM parameter system. The routines are usually called *fac*_ASSOC, where *fac* is the facility name of the package, *e.g.* NDF_ASSOC accesses an NDF whose name is found via a parameter, and SGS_ASSOC obtains the name of an SGS graphics device. There are also *fac*_CANCL routines that free other resources opened by the corresponding *fac*_ASSOC routine, and therefore you should not use PAR_CANCL to cancel parameters obtained by using an *fac*_ASSOC routine. However, it is possible and quite legitimate to call PAR_PROMT to set a prompt string for this class of parameter, or to call PAR_DEF0C to set the dynamic default, or to find the state of a parameter using PAR_STATE.

9 Compilation and Linking

9.1 UNIX

To compile and link a UNIX ADAM application called prog with the PAR library, you should use the following command.

```
% alink prog.f
```

9.2 VMS

PAR is presently only available to ADAM programmers. Only the normal ADAM startup commands need be issued to access routines in the PAR library. These are

```
$ ADAMSTART
$ ADAMDEV
$ PAR_DEV
```

and they will ensure that all necessary definitions are made for compilation and linking.

PAR is distributed as a shareable image, and it is included in the ADAM ALINK command. So to link an ADAM A-task called PROG you just enter

```
$ FORTRAN PROG
$ ALINK PROG
```

10 Acknowledgements

Dennis Kelly and Alan Chipperfield wrote the original simple PAR library and the ADAM parameter system. Some of the extended routines are partly based on AIF routines written by Dave Baines and Steven Beard at ROE. Thanks to Alan Chipperfield for explaining the workings of the ADAM parameter system to the author, and for a critical reading of this document. Rodney Warren-Smith also suggested many improvements.

A Alphabetical List of Routines

The argument lists of the following routines, together with on-line help information, are available within the Starlink language-sensitive editor STARLSE (SUN/105).

PAR_CANCL (PARAM, STATUS)
Cancel a parameter

PAR_CHOIC (PARAM, DEFAUL, OPTS, NULL, VALUE, STATUS)
Obtain from a parameter a character value selected from a menu of options

PAR_CHOIV (PARAM, MAXVAL, OPTS, VALUES, ACTVAL, STATUS)
Obtain from a parameter a list of character values selected from a menu of options

PAR_DEF0x (PARAM, VALUE, STATUS)
Set a scalar dynamic default parameter value

PAR_DEF1x (PARAM, NVAL VALUES, STATUS)
Set a vector of values as the dynamic default for a parameter

PAR_DEFNx (PARAM, NDIM, MAXD, VALUES, ACTD, STATUS)
Set an array of values as the dynamic default for a parameter

PAR_EXACx (PARAM, NVALS, VALUES, STATUS)
Obtain an exact number of values from a parameter

PAR_GDR0x (PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS)
Obtain a scalar value within a given range from a parameter

PAR_GDR1x (PARAM, NVALS, DEFAUL, VMIN, VMAX, NULL, VALUES, STATUS)
Obtain an exact number of values within a given range from a parameter

PAR_GDRVx (PARAM, MAXVAL, VMIN, VMAX, VALUES, ACTVAL, STATUS)
Obtain a vector of values within a given range from a parameter

PAR_GET0x (PARAM, VALUE, STATUS)
Obtain a scalar value from a parameter

PAR_GET1x (PARAM, MAXVAL, VALUES, ACTVAL, STATUS)
Obtain a vector of values from a parameter

PAR_GETNx (PARAM, NDIM, MAXD, VALUES, ACTD, STATUS)
Obtain an array parameter value

PAR_GETVx (PARAM, MAXVAL, VALUES, ACTVAL, STATUS)
Obtain a vector of values from a parameter regardless of its shape

PAR_GEVEN (PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS)
Obtain an even integer value from a parameter

PAR_GODD (PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS)
Obtain an odd integer value from a parameter

PAR_GRM1x (PARAM, NVALS, DEFAUL, VMIN, VMAX, NULL, VALUES, STATUS)
Obtain from a parameter an exact number of values each within a given range

PAR_GRMVx (PARAM, MAXVAL, VMIN, VMAX, VALUES, ACTVAL, STATUS)
Obtain from a parameter a vector of values each within a given range

PAR_GTD0L (PARAM, DEFAUL, NULL, VALUE, STATUS)
Obtain a logical value from a parameter with a dynamic default

PAR_MAXx (PARAM, VALUE, STATUS)
Set a maximum value for a parameter

PAR_MINx (PARAM, VALUE, STATUS)
Set a minimum value for a parameter

PAR_MIX0x (PARAM, DEFAUL, VMIN, VMAX, OPTS, NULL, VALUE, STATUS)
Obtain from a parameter a character value either from a menu of options or as a numeric value within a given range

PAR_MIXVx (PARAM, MAXVAL, VMIN, VMAX, OPTS, VALUES, ACTVAL, STATUS)
Obtain from a parameter, character values either selected from a menu of options or as numeric values within a given range

PAR_PROMT (PARAM, PROMPT, STATUS)
Set a new prompt string for a parameter

PAR_PUT0x (PARAM, VALUE, STATUS)
Put a scalar value into a parameter

PAR_PUT1x (PARAM, NVAL, VALUES, STATUS)
Put a vector of values into a parameter

PAR_PUTNx (PARAM, NDIM, MAXD, VALUES, ACTD, STATUS)
Put an array of values into a parameter

PAR_PUTVx (PARAM, NVAL, VALUES, STATUS)
Put an array of values into a parameter as if the parameter were a vector

PAR_STATE (PARAM, STATE, STATUS)
Inquire the state of a parameter

PAR_UNSET (PARAM, WHICH, STATUS)
Cancel various parameter control values.

B Classified List of Routines

B.1 Getting Values from a Parameter without Constraints

PAR_GET0x (PARAM, VALUE, STATUS)
Obtain a scalar value from a parameter

PAR_GET1x (PARAM, MAXVAL, VALUES, ACTVAL, STATUS)
Obtain a vector of values from a parameter

PAR_GETNx (PARAM, NDIM, MAXD, VALUES, ACTD, STATUS)
Obtain an array parameter value

PAR_GETVx (PARAM, MAXVAL, VALUES, ACTVAL, STATUS)
Obtain a vector of values from a parameter regardless of its shape

B.2 Getting Parameter Values from a Menu

PAR_CHOIC (PARAM, DEFAUL, OPTS, NULL, VALUE, STATUS)
Obtain from a parameter a character value selected from a menu of options

PAR_CHOIV (PARAM, MAXVAL, OPTS, VALUES, ACTVAL, STATUS)
Obtain from a parameter a list of character values selected from a menu of options

PAR_MIX0x (PARAM, DEFAUL, VMIN, VMAX, OPTS, NULL, VALUE, STATUS)
Obtain from a parameter a character value either from a menu of options or as a numeric value within a given range

PAR_MIXVx (PARAM, MAXVAL, VMIN, VMAX, OPTS, VALUES, ACTVAL, STATUS)
Obtain from a parameter, character values either selected from a menu of options or as numeric values within a given range

B.3 Getting Values within Ranges

PAR_GDR0x (PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS)
Obtain a scalar value within a given range from a parameter

PAR_GDR1x (PARAM, NVALS, DEFAUL, VMIN, VMAX, NULL, VALUES, STATUS)
Obtain an exact number of values within a given range from a parameter

PAR_GDRVx (PARAM, MAXVAL, VMIN, VMAX, VALUES, ACTVAL, STATUS)
Obtain a vector of values within a given range from a parameter

PAR_GEVEN (PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS)
Obtain an even integer value from a parameter

PAR_GODD (PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS)
Obtain an odd integer value from a parameter

PAR_GRM1x (PARAM, NVALS, DEFAUL, VMIN, VMAX, NULL, VALUES, STATUS)
Obtain from a parameter an exact number of values each within a given range

PAR_GRMVx (PARAM, MAXVAL, VMIN, VMAX, VALUES, ACTVAL, STATUS)
Obtain from a parameter a vector of values each within a given range

PAR_MIX0x (PARAM, DEFAUL, VMIN, VMAX, OPTS, NULL, VALUE, STATUS)
Obtain from a parameter a character value either from a menu of options or as a numeric value within a given range

PAR_MIXVx (PARAM, MAXVAL, VMIN, VMAX, OPTS, VALUES, ACTVAL, STATUS)
Obtain from a parameter, character values either selected from a menu of options or as numeric values within a given range

B.4 Putting Values into Parameters

PAR_PUT0x (PARAM, VALUE, STATUS)
Put a scalar value into a parameter

PAR_PUT1x (PARAM, NVAL, VALUES, STATUS)
Put a vector of values into a parameter

PAR_PUTNx (PARAM, NDIM, MAXD, VALUES, ACTD, STATUS)
Put an array of values into a parameter

PAR_PUTVx (PARAM, NVAL, VALUES, STATUS)
Put an array of values into a parameter as if the parameter were a vector

B.5 Miscellaneous

PAR_CANCL (PARAM, STATUS)
Cancel a parameter

PAR_GTD0L (PARAM, DEFAUL, NULL, VALUE, STATUS)
Obtain a logical value from a parameter with a dynamic default

PAR_MAXx (PARAM, VALUE, STATUS)
Set a maximum value for a parameter

PAR_MINx (PARAM, VALUE, STATUS)
Set a minimum value for a parameter

PAR_PROMT (PARAM, PROMPT, STATUS)
Set a new prompt string for a parameter

PAR_UNSET (PARAM, WHICH, STATUS)
Cancel various parameter control values.

B.6 Inquiry Routines

PAR_STATE (PARAM, STATE, STATUS)
Inquire the state of a parameter

C Error Codes

If a PAR routine completes its task successfully, it will return with status SAI__OK. If it detects an error condition, PAR defines the following error codes.

Symbolic name	Meaning
PAR__ACINV	Invalid access mode for the parameter
PAR__AMBIG	Ambiguous option list
PAR__ERROR	Some undefined error
PAR__NULL	Null parameter value

D Correspondence between PAR and SUBPAR States

PAR and SUBPAR states do not have a one-to-one correspondence. Each SUBPAR state is equated to the appropriate generic PAR states. The former version of PAR_STATE routine returned SUBPAR states, rather than the PAR states as happens now. Existing applications which compare the returned state with any of SUBPAR__GROUND, SUBPAR__ACTIVE, SUBPAR__CANCEL, and SUBPAR__NULL (defined in SUBPAR_PAR) will continue to function, since the PAR equivalents have the same numerical values.

Table 1: The PAR states.

PAR Symbolic name	Meaning	SUBPAR states
PAR__GROUND	The parameter has never had a value.	SUBPAR__GROUND
		SUBPAR__EOL
		SUBPAR__RESET
		SUBPAR__ACCEPT
		SUBPAR__RESACC
		SUBPAR__FPROMPT
		SUBPAR__RESPROM
		SUBPAR__ACCPR
		SUBPAR__RESACCPR
PAR__ACTIVE	The parameter has a value.	SUBPAR__ACTIVE
		SUBPAR__MAX
		SUBPAR__MIN
PAR__CANCEL	The parameter value has been cancelled.	SUBPAR__CANCEL
PAR__NULLST	The parameter is null.	SUBPAR__NULL

E The C Interface

A preliminary C interface is provided for trial purposes. It may be subject to change in the light of experience.

The interface obeys the rules defined in (PRO)LUN/10.

Briefly, the function name is generated from the Fortran subroutine name by forcing the name to lower case apart from the first character following any underscores, which is forced to upper case. Underscores are then removed.

For example: The C interface function for 'PAR_GET0R' is 'parGet0r'.

Arguments are provided in the same order as for the Fortran routine with the exception that CHARACTER arrays and returned CHARACTER strings have an additional argument of type `int` (passed by value) immediately following them to specify a maximum length for the output string(s) including the terminating null for which space must be allowed.

There is a fixed relationship between the type of the Fortran argument and the type of the argument supplied to the C function – it is as follows:

Fortran type	C type
INTEGER	int
REAL	float
REAL*8	double
DOUBLE PRECISION	double
LOGICAL	int
CHARACTER	char
FUNCTION	*type* (**name*)()
SUBROUTINE	void (**name*)()

Apart from any argument named 'status', given-only scalar arguments (not including character strings) are passed by value. All others are passed by pointer.

Arrays must be passed with the elements stored in the order required by Fortran.

All necessary constants and function prototypes can be defined by:

```
#include "par.h"
```

F The C Interface Function Prototypes

Where *T* is one of d, i, l, r and *TYPE* is the corresponding C type, the function prototypes are:

void parCancl (const char *param, int *status);
Cancel a parameter

void parChoic (const char *param, const char *defaul, const char *opts, int null, char *value, int value_length, int *status);
Obtain from a parameter a character value selected from a menu of options

void parChoiv (const char *param, int maxval, const char *opts, char *const *values, int values_length, int *actval, int *status);
Obtain from a parameter a list of character values selected from a menu of options

void parDef0c (const char *param, const char *value, int *status);
Set a scalar character dynamic default parameter value

void parDef0*T* (const char *param, *TYPE* value, int *status);
Set a scalar dynamic default parameter value

void parDef1c (const char *param, int nval, char *const *values, int values_length, int *status);
Set a vector of character values as the dynamic default for a parameter

void parDef1*T* (const char *param, int nval, const *TYPE* *values, int *status);
Set a vector of values as the dynamic default for a parameter

void parDefnc (const char *param, int ndim, const int *maxd, char *const *values, int values_length, const int *actd, int *status);
Set an array of character values as the dynamic default for a parameter

void parDefn*T* (const char *param, int ndim, const int *maxd, const *TYPE* *values, const int *actd, int *status);
Set an array of values as the dynamic default for a parameter

void parExacc (const char *param, int nvals, char *const *values, int values_length, int *status);
Obtain an exact number of character values from a parameter

void parExac*T* (const char *param, int nvals, *TYPE* *values, int *status);
Obtain an exact number of values from a parameter

void parGdr0*T* (const char *param, *TYPE* defaul, *TYPE* vmin, *TYPE* vmax, int null, *TYPE* *value, int *status);
Obtain a scalar value within a given range from a parameter

void parGdr1*T* (const char *param, int nvals, *TYPE* *defaul, *TYPE* vmin, *TYPE* vmax, int null, *TYPE* *values, int *status);
Obtain an exact number of values within a given range from a parameter

void parGdrv*T* (const char *param, int maxval, *TYPE* vmin, *TYPE* vmax, *TYPE* *values, int *actval, int *status);
Obtain a vector of values within a given range from a parameter

void parGet0c (const char *param, char *value, int value_length, int *status);
Obtain a scalar character value from a parameter

void parGet0*T* (const char *param, *TYPE* *value, int *status);
Obtain a scalar value from a parameter

void parGet1c (const char *param, int maxval, char *const *values, int values_length, int *actval, int *status);
Obtain a vector of character values from a parameter

void parGet1*T* (const char *param, int maxval, *TYPE* *values, int *actval, int *status);
Obtain a vector of values from a parameter

void parGetnc (const char *param, int ndim, const int *maxd, char *const *values, int values_length, int *actd, int *status);
Obtain a character array parameter value

void parGetn*T* (const char *param, int ndim, const int *maxd, *TYPE* *values, int *actd, int *status);
Obtain an array parameter value

void parGetvc (const char *param, int maxval, char *const *values, int values_length, int *actval, int *status);
Obtain a vector of character values from a parameter regardless of its shape

void parGetv*T* (const char *param, int maxval, *TYPE* *values, int *actval, int *status);
Obtain a vector of values from a parameter regardless of its shape

void parGeven (const char *param, int defaul, int vmin, int vmax, int null, int *value, int *status);
Obtain an even integer value from a parameter

void parGodd (const char *param, int defaul, int vmin, int vmax, int null, int *value, int *status);
Obtain an odd integer value from a parameter

void parGrm1*T* (const char ∗param, int nvals, const *TYPE* ∗defaul, const *TYPE* ∗vmin, const *TYPE* ∗vmax, int null, *TYPE* ∗values, int ∗status);
Obtain from a parameter an exact number of values each within a given range

void parGrmv*T* (const char ∗param, int maxval, const *TYPE* ∗vmin, const *TYPE* ∗vmax, const *TYPE* ∗values, int ∗actval, int ∗status);
Obtain from a parameter a vector of values each within a given range

void parGtd0l (const char ∗param, int defaul, int null, int ∗value, int ∗status);
Obtain a logical value from a parameter with a dynamic default

void parMaxc (const char ∗param, const char ∗value, int ∗status);
Set a maximum character value for a parameter

void parMax*T* (const char ∗param, *TYPE* value, int ∗status);
Set a maximum value for a parameter

void parMinc (const char ∗param, const char ∗value, int ∗status);
Set a minimum character value for a parameter

void parMin*T* (const char ∗param, *TYPE* value, int ∗status);
Set a minimum value for a parameter

void parMix0*T* (const char ∗param, const char ∗defaul, *TYPE* vmin, *TYPE* vmax, const char ∗opts, int null, char ∗value, int value_length, int ∗status);
Obtain from a parameter a character value either from a menu of options or as a numeric value within a given range

void parMixv*T* (const char ∗param, int maxval, *TYPE* vmin, *TYPE* vmax, const char ∗opts, char ∗const ∗values, int values_length, int ∗actval, int ∗status);
Obtain from a parameter, character values either selected from a menu of options or as numeric values within a given range

void parPromt (const char ∗param, const char ∗prompt, int ∗status);
Set a new prompt string for a parameter

void parPut0c (const char ∗param, const char ∗value, int ∗status);
Put a scalar character value into a parameter

void parPut0*T* (const char ∗param, *TYPE* value, int ∗status);
Put a scalar value into a parameter

void parPut1c (const char ∗param, int nval, char ∗const ∗values, int values_length, int ∗status);
Put a vector of character values into a parameter

void parPut1*T* (const char ∗param, int nval, const *TYPE* ∗values, int ∗status);
Put a vector of values into a parameter

void parPutnc (const char ∗param, int ndim, const int ∗maxd, char ∗const ∗values, int values_length, const int ∗actd, int ∗status);
Put an array of character values into a parameter

void parPutn*T* (const char ∗param, int ndim, const int ∗maxd, const *TYPE* ∗values, const int ∗actd, int ∗status);
Put an array of values into a parameter

void parPutvc (const char ∗param, int nval, char ∗const ∗values, int values_length, int ∗status);
Put an array of character values into a parameter as if the parameter were a vector

void parPutv*T* (const char ∗param, int nval, const *TYPE* ∗values, int ∗status);
Put an array of values into a parameter as if the parameter were a vector

void parState (const char ∗param, int ∗state, int ∗status);
Inquire the state of a parameter

void parUnset (const char *param, const char *which, int *status);
Cancel various parameter control values.

G Reference Manual

PAR_CANCL
Cancels a parameter

Description:

The named parameter is cancelled. A subsequent attempt to get a value for the parameter will result in a new value being obtained by the underlying parameter system.

Invocation:

```
CALL PAR_CANCL( PARAM, STATUS )
```

Arguments:

PARAM = CHARACTER*(*) (Given)
The name of the parameter to be cancelled.

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

Notes:

The routine attempts to execute regardless of the value of STATUS. If the import value is not SAI__OK, then it is left unchanged, even if the routine fails to complete. If the STATUS is SAI__OK on entry and the routine fails to complete, STATUS will be set to an appropriate error number, and there will one or more additional error reports.

PAR_CHOIC
Obtains from a parameter a character value selected from a menu of options

Description:

This routine obtains a scalar character value from a parameter. The value must be one of a supplied list of acceptable values, and can be an abbreviation provided it is unambiguous. A dynamic default may be suggested.

Invocation:

```
CALL PAR_CHOIC( PARAM, DEFAUL, OPTS, NULL, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
The name of the parameter.

DEFAUL = CHARACTER ∗ (∗) (Given)
The suggested default value for the parameter. No default will be suggested when DEFAUL is not one of the options defined by OPTS. A status of PAR__AMBIG is returned if the default is ambiguous, *i.e.* an abbreviation of more than one of the options.

OPTS = CHARACTER ∗ (∗) (Given)
The list of acceptable options for the value obtained from the parameter. Items should be separated by commas. The list is case-insensitive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUE = CHARACTER ∗ (∗) (Returned)
The selected option from the list. It is in uppercase and in full, even if an abbreviation has been given for the actual parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the value of DEFAUL.

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

Notes:

The search for a match of the obtained character value with an item in the menu adheres to the following rules.

- All comparisons are performed in uppercase. Leading blanks are ignored.
- A match is found when the value equals the full name of an option. This enables an option to be the prefix of another item without it being regarded as ambiguous. For example, "10,100,200" would be an acceptable list of options.

- If there is no exact match, an abbreviation is acceptable. A comparison is made of the value with each option for the number of characters in the value. The option that best fits the value is declared a match, subject to two provisos. Firstly, there must be no more than one character different between the value and the start of the option. (This allows for a mistyped character.) Secondly, there must be only one best-fitting option. Whenever these criteria are not satisfied, the user is told of the error, and is presented with the list of options, before being prompted for a new value If a nearest match is selected, the user is informed unless the MSG filtering level (see SUN/104) is 'quiet'.

PAR_CHOIV
Obtains from a parameter a list of character values selected from a menu of options

Description:

This routine obtains a vector of character values from a parameter. Each value must be one of a supplied list of acceptable values, and can be an abbreviation provided it is unambiguous.

Invocation:

```
CALL PAR_CHOIV( PARAM, MAXVAL, OPTS, VALUES, ACTVAL, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

MAXVAL = INTEGER (Given)
The maximum number of values required. A PAR__ERROR status is returned when the number of values requested is less than one.

OPTS = CHARACTER $*$ ($*$) (Given)
The list of acceptable options for the values obtained from the parameter. Items should be separated by commas. The list is case-insensitive.

VALUES(MAXVAL) = CHARACTER $*$ ($*$) (Returned)
The selected options from the list in the order supplied to the parameter. They are in uppercase and in full, even if an abbreviation has been given for the actual parameter.

ACTVAL = INTEGER (Returned)
The actual number of values obtained.

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

Notes:

The search for a match of each obtained character value with an item in the menu adheres to the following rules.

- All comparisons are performed in uppercase. Leading blanks are ignored.
- A match is found when the value equals the full name of an option. This enables an option to be the prefix of another item without it being regarded as ambiguous. For example, "10,100,200" would be an acceptable list of options.
- If there is no exact match, an abbreviation is acceptable. A comparison is made of the value with each option for the number of characters in the value. The option that best fits the value is declared a match, subject to two provisos. Firstly, there must be no more than one character different between the value and the start of the option. (This allows for a mistyped character.) Secondly, there must be only one best-fitting option. Whenever these criteria are not satisfied, the user is told of the error, and is presented with the list of options, before being prompted for a new value If a nearest match is selected, the user is informed unless the MSG filtering level (see SUN/104) is 'quiet'.

PAR_DEF0x
Sets a scalar dynamic default parameter value

Description:

This routine sets a scalar as the dynamic default value for a parameter. The dynamic default may be used as the parameter value by means of appropriate specifications in the interface file.

If the declared parameter type differs from the type of the value supplied, then conversion is performed.

Invocation:

```
CALL PAR_DEF0x( PARAM, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

VALUE = ? (Given)
The dynamic default value for the parameter.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUE argument must have the corresponding data type.

PAR_DEF1x
Sets a vector of values as the dynamic default for a parameter

Description:

This routine sets a one-dimensional array of values as the dynamic default for a parameter. The dynamic default may be used as the parameter value by means of appropriate specifications in the interface file.

If the declared parameter type differs from the type of the array supplied, then conversion is performed.

Invocation:

```
CALL PAR_DEF1x( PARAM, NVAL, VALUES, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of a parameter of primitive type.

NVAL = INTEGER (Given)
The number of default values.

VALUES(NVAL) = ? (Given)
The array to contain the default values.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.

PAR_DEFNx
Sets an array of values as the dynamic default for a parameter

Description:

This routine sets an array of values as the dynamic default for a parameter. The dynamic default may be used as the parameter value by means of appropriate specifications in the interface file.

If the declared parameter type differs from the type of the array supplied, then conversion is performed.

Invocation:

```
CALL PAR_DEFNx( PARAM, NDIM, MAXD, VALUES, ACTD, STATUS )
```

Arguments:

PARAM = CHARACTER∗(∗) (Given)
The name of the parameter.

NDIM = INTEGER (Given)
The number of dimensions of the values array.

MAXD(NDIM) = INTEGER (Given)
The dimensions of the values' array.

VALUES(∗) = ? (Given)
The default values, given in Fortran order.

ACTD(NDIM) = INTEGER (Given)
The dimensions of the dynamic default object to be created.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- The current implementation of the underlying parameter system, SUBPAR, creates an *n*-dimensional HDS object, containing the specified values. The ACTD argument gives the dimensions of the object to be created. If the dynamic default is used as the suggested value in a prompt, the name of this object, rather than its contents, is offered.

PAR_EXACx
Obtains an exact number of values from a parameter

Description:

This routine obtains an exact number of values from a parameter and stores them in a vector. If the number of values obtained is less than that requested, a further get or gets will occur for the remaining values until the exact number required is obtained (or an error occurs). The routine reports how many additional values are required prior to these further reads. Should too many values be entered an error results and a further read is attempted.

Invocation:

```
CALL PAR_EXACx( PARAM, NVALS, VALUES, STATUS )
```

Arguments:

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

NVALS = INTEGER (Given)
The number of values needed. Values will be requested until exactly this number of values (no more and no less) has been obtained.

VALUES(NVALS) = ? (Returned)
The values obtained from the parameter system. They will only be valid if STATUS is not set to an error value.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- If more than one attempt is made is to obtain the values, the current value of the parameter will be only that read at the last get, and not the full array of values.
- A PAR__ERROR status is returned when the number of values requested is not positive.

PAR_GDR0x
Obtains a scalar value within a given range from a parameter

Description:

This routine obtains from a parameter a scalar value that lies within a supplied range of acceptable values. A dynamic default may be defined.

Invocation:

```
CALL PAR_GDR0x( PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

DEFAUL = ? (Given)
The suggested-default value for the parameter. No default will be suggested when DEFAUL is not within the range of acceptable values defined by VMIN and VMAX.

VMIN = ? (Given)
The value immediately above a range wherein the obtained value cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX = ? (Given)
The value immediately below a range wherein the obtained value cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUE = ? (Returned)
The value associated with the parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the value of DEFAUL.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace "x" in the routine name by D, I, K, or R respectively as appropriate. The DEFAUL, VMIN, VMAX, and VALUE arguments all must have the corresponding data type.
- If the value violates the constraint, the user is informed of the constraint and prompted for another value.

PAR_GDR1x
Obtains an exact number of values within a given range from a parameter

Description:

This routine obtains an exact number of values from a parameter. all of which must be within a supplied range of acceptable values. Dynamic defaults may be defined.

Invocation:

```
CALL PAR_GDR1x( PARAM, NVALS, DEFAUL, VMIN, VMAX, NULL, VALUES, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

NVALS = INTEGER (Given)
The number of values needed. Values will be requested until exactly this number (no more and no less) has been obtained.

DEFAUL(NVALS) = ? (Given)
The suggested-default values for the parameter. No default will be suggested when any of the DEFAUL elements is not within the range of acceptable values defined by VMIN and VMAX.

VMIN = ? (Given)
The value immediately above a range wherein the obtained values cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the obtained values. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX = ? (Given)
The value immediately below a range wherein the obtained values cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the obtained values. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUES(NVALS) = ? (Returned)
The values associated with the parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the values of DEFAUL.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace `"x"` in the routine name by D, I, K, or R respectively as appropriate. The DEFAUL, VMIN, VMAX, and VALUES arguments all must have the corresponding data type.
- If any of the values violates the constraint, the user is informed of the constraint and prompted for another vector of values. This is not achieved through the MIN/MAX system.

PAR_GDRVx
Obtains a vector of values within a given range from a parameter

Description:

This routine obtains up to a given number of values from a parameter. All the values must be within a supplied range of acceptable values.

Invocation:

```
CALL PAR_GDRVx( PARAM, MAXVAL, VMIN, VMAX, VALUES, ACTVAL, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

MAXVAL = INTEGER (Given)
The maximum number of values required. A PAR__ERROR status is returned when the number of values requested is less than one.

VMIN = ? (Given)
The value immediately above a range wherein the obtained values cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the obtained values. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX = ? (Given)
The value immediately below a range wherein the obtained values cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the obtained values. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VALUES(MAXVAL) = ? (Returned)
The values associated with the parameter. They will only be valid if STATUS is not set to an error value.

ACTVAL = INTEGER (Returned)
The actual number of values obtained.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace "x" in the routine name by D, I, or R respectively as appropriate. The VMIN, VMAX, and VALUES arguments all must have the corresponding data type.
- Should too many values be read, the parameter system will repeat the get in order to obtain a permitted number of values.
- If any of the values violates the constraint, the user is informed of the constraint and prompted for another vector of values. This is not achieved through the MIN/MAX system.

PAR_GET0x
Obtains a scalar value from a parameter

Description:

This routine obtains a scalar value from a parameter. If it is necessary, the value is converted to the required type.

Invocation:

```
CALL PAR_GET0x( PARAM, VALUE, STATUS )
```

Arguments:

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

VALUE = ? (Returned)
The parameter value.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUE argument must have the corresponding data type.
- Note that a scalar (zero-dimensional) parameter is different from a vector (one-dimensional) parameter containing a single value.

PAR_GET1x
Obtains a vector of values from a parameter

Description:

This routine obtains a vector of values from a parameter. If it is necessary, the values are converted to the required type.

Invocation:

```
CALL PAR_GET1x( PARAM, MAXVAL, VALUES, ACTVAL, STATUS )
```

Arguments:

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

MAXVAL = INTEGER (Given)
The maximum number of values that can be obtained.

VALUES(MAXVAL) = ? (Returned)
The array to receive the values associated with the parameter.

ACTVAL = INTEGER (Returned)
The actual number of values obtained.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- Note that this routine will accept a scalar value, returning a single-element vector.

PAR_GETNx
Obtains an array parameter value

Description:

This routine obtains an *n*-dimensional array of values from a parameter. If necessary, the values are converted to the required type.

Invocation:

```
CALL PAR_GETNx( PARAM, NDIM, MAXD, VALUES, ACTD, STATUS )
```

Arguments:

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

NDIM = INTEGER (Given)
The number of dimensions of the values array. This must match the number of dimensions of the parameter.

MAXD(NDIM) = INTEGER (Given)
Array specifying the maximum dimensions of the array to be read. These may not be smaller than the dimensions of the actual parameter nor greater than the dimensions of the VALUES array.

VALUES($*$) = ? (Returned)
The values obtained from the parameter. These are in Fortran order.

ACTD(NDIM) = INTEGER (Returned)
The actual dimensions of the array. Unused dimensions are set to 1.

STATUS = INTEGER
The global status.

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace `"x"` in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- Note that this routine will accept a scalar value, returning a single-element array with the specified number of dimensions.

PAR_GETVx
Obtains a vector of values from a parameter regardless of the its shape

Description:

This routine obtains an array of values from a parameter as if the parameter were vectorized (*i.e.* regardless of its dimensionality). If necessary, the values are converted to the required type.

Invocation:

```
CALL PAR_GETVx( PARAM, MAXVAL, VALUES, ACTVAL, STATUS )
```

Arguments:

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

MAXVAL = INTEGER (Given)
The maximum number of values that can be held in the values array.

VALUES(MAXVAL) = ? (Returned)
Array to receive the values associated with the object.

ACTVAL = INTEGER (Returned)
The actual number of values obtained.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- Note that this routine will accept a scalar value, returning a single-element vector.

PAR_GEVEN
Obtains an even integer value from a parameter

Description:

This routine obtains a scalar integer value from a parameter. This value must be even and within a supplied range of acceptable values. A dynamic default may be defined.

Invocation:

```
CALL PAR_GEVEN( PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

DEFAUL = INTEGER (Given)
The suggested-default value for the parameter. No default will be suggested when DEFAUL is not within the range of acceptable values defined by VMIN and VMAX, or DEFAUL is odd.

VMIN = INTEGER (Given)
The value immediately above a range wherein the obtained value cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX = INTEGER (Given)
The value immediately below a range wherein the obtained value cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUE = INTEGER (Returned)
The value associated with the parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the value of DEFAUL.

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

Notes:

- Zero is deemed to be even.
- If the value violates the constraint, the user is informed of the constraint and prompted for another value.

PAR_GODD
Obtains an odd integer value from a parameter

Description:

This routine obtains a scalar integer value from a parameter. This value must be odd and within a supplied range of acceptable values. A dynamic default may be defined.

Invocation:

```
CALL PAR_GODD( PARAM, DEFAUL, VMIN, VMAX, NULL, VALUE, STATUS )
```

Arguments:

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

DEFAUL = INTEGER (Given)
The suggested-default value for the parameter. No default will be suggested when DEFAUL is not within the range of acceptable values defined by VMIN and VMAX, or DEFAUL is odd.

VMIN = INTEGER (Given)
The value immediately above a range wherein the obtained value cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX = INTEGER (Given)
The value immediately below a range wherein the obtained value cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUE = INTEGER (Returned)
The value associated with the parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the value of DEFAUL.

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

Notes:

- Zero is deemed to be even.
- If the value violates the constraint, the user is informed of the constraint and prompted for another value.

PAR_GRM1x
Obtains from a parameter an exact number of values each within a given range

Description:

This routine obtains an exact number of values from a parameter. Each value must be within its own range of acceptable values supplied to the routine. Dynamic defaults may be defined.

This routine is particularly useful for obtaining co-ordinate information, where each co-ordinate has different bounds, and a series of calls to PAR_GDR0x would not permit an arbitrary number of dimensions.

Invocation:

```
CALL PAR_GRM1x( PARAM, NVALS, DEFAUL, VMIN, VMAX, NULL, VALUES, STATUS )
```

Arguments:

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

NVALS = INTEGER (Given)
The number of values needed. Values will be requested until exactly this number (no more and no less) has been obtained.

DEFAUL(NVALS) = ? (Given)
The suggested-default values for the parameter. No default will be suggested when any of the DEFAUL elements is not within the range of acceptable values defined by VMIN and VMAX for that value.

VMIN(NVALS) = ? (Given)
The values immediately above a range wherein each obtained value cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the corresponding obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX(NVALS) = ? (Given)
The values immediately below a range wherein each obtained value cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the corresponding obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUES(NVALS) = ? (Returned)
The values associated with the parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the values of DEFAUL.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace `"x"` in the routine name by D, I, or R respectively as appropriate. The DEFAUL, VMIN, VMAX, and VALUES arguments must have the corresponding data type.
- If any of the values violates the constraints, the user is informed of the constraints and prompted for another vector of values. This is not achieved through the MIN/MAX system.

PAR_GRMVx
Obtains from a parameter a vector of values each within a given range

Description:

This routine obtains from a parameter up to a given number of values. Each value must be within its own range of acceptable values supplied to the routine.

This routine is particularly useful for obtaining values that apply to *n*-dimensional array where each value is constrained by the size or bounds of the array, and where the number of values need not equal *n*. For example, the size of a smoothing kernel could be defined by one value that applies to all dimensions, or as individual sizes along each dimension.

Invocation:

```
CALL PAR_GRMVx( PARAM, MAXVAL, VMIN, VMAX, VALUES, ACTVAL, STATUS )
```

Arguments:

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

MAXVAL = INTEGER (Given)
The maximum number of values required. A PAR__ERROR status is returned when the number of values requested is less than one.

VMIN(MAXVAL) = ? (Given)
The values immediately above a range wherein each obtained value cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum allowed for the corresponding obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VMAX(MAXVAL) = ? (Given)
The values immediately below a range wherein each obtained value cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum allowed for the corresponding obtained value. However, should VMAX be less than VMIN, all values are acceptable except those between VMAX and VMIN exclusive.

VALUES(MAXVAL) = ? (Returned)
The values associated with the parameter. They will only be valid if STATUS is not set to an error value.

ACTVAL = INTEGER (Returned)
The actual number of values obtained.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace "x" in the routine name by D, I, or R respectively as appropriate. The VMIN, VMAX, and VALUES arguments must have the corresponding data type.
- Should too many values be obtained, the parameter system will repeat the get in order to obtain a permitted number of values.

- If any of the values violates the constraints, the user is informed of the constraints and prompted for another vector of values. This is not achieved through the MIN/MAX system.

PAR_GTD0L
Obtains a logical value from a parameter with a dynamic default

Description:

This routine obtains a scalar logical value from a parameter. A dynamic default is defined.

Invocation:

```
CALL PAR_GTD0L( PARAM, DEFAUL, NULL, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the parameter.

DEFAUL = LOGICAL (Given)

The suggested default value for the parameter.

NULL = LOGICAL (Given)

NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUE = LOGICAL (Returned)

The value associated with the parameter. It will only be valid if STATUS is not set to an error value.

STATUS = INTEGER (Given and Returned)

The global status.

PAR_MAXx
Sets a maximum value for a parameter

Description:

This routine sets a maximum value for the specified parameter. The value will be used as an upper limit for any value subsequently obtained for the parameter.

If the routine fails, any existing maximum value will be unset.

Invocation:

```
CALL PAR_MAXx( PARAM, VALUE, STATUS )
```

Arguments:

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

VALUE = ? (Given)
The value to be set as the maximum. It must not be outside any RANGE specified for the parameter in the interface file.

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

Notes:

- There is a routine for each of the data types character, integer, real, and double precision: replace "x" in the routine name by C, I, R, or D respectively as appropriate. The VALUE argument must have the corresponding data type. If the parameter has a different type, the maximum value will be converted to the type of the parameter which must be character, double precision, integer, or real.
- If a minimum value has been set (using PAR_MINx) that is greater than the maximum at the time the parameter value is obtained, only values between the limits will not be permitted.
- The maximum value set by this routine overrides any upper RANGE value which may have been specified in the interface file. The specified value must not be outside any RANGE values. The maximum value may also be selected as the parameter value – again in preference to any upper RANGE value – by specifying MAX as the parameter value on the command line or in response to a prompt.

PAR_MINx
Sets a minimum value for a parameter

Description:

This routine sets a minimum value for the specified parameter. The value will be used as a lower limit for any value subsequently obtained for the parameter.

If the routine fails, any existing minimum value will be unset.

Invocation:

```
CALL PAR_MINx( PARAM, VALUE, STATUS )
```

Arguments:

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

VALUE = ? (Given)
The value to be set as the minimum. It must not be outside any RANGE specified for the parameter in the interface file.

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

Notes:

- There is a routine for each of the data types character, integer, real, and double precision: replace `"x"` in the routine name by C, I, R, or D respectively as appropriate. The VALUE argument must have the corresponding data type. If the parameter has a different type, the minimum value will be converted to the type of the parameter which must be character, double precision, integer, or real.
- If a maximum value has been set (using PAR_MAXx) that is less than the minimum at the time the parameter value is obtained, only values between the limits will not be permitted.
- The minimum value set by this routine overrides any lower RANGE value which may have been specified in the interface file. The specified value must not be outside any RANGE values. The minimum value may also be selected as the parameter value – again in preference to any lower RANGE value – by specifying MIN as the parameter value on the command line or in response to a prompt.

PAR_MIX0x
Obtains from a parameter a character value from either a menu of options or within a numeric range

Description:

This routine obtains a scalar character value from a parameter. The value must be either:

- one of a supplied list of acceptable values, with unambiguous abbreviations accepted; or
- a numeric character string equivalent to a number, and the number must lie within a supplied range of acceptable values.

A dynamic default may be suggested.

Invocation:

```
CALL PAR_MIX0x( PARAM, DEFAUL, VMIN, VMAX, OPTS, NULL, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER $*$ ($*$) (Given)
The name of the parameter.

DEFAUL = CHARACTER $*$ ($*$) (Given)
The suggested-default value for the parameter. No default will be suggested when DEFAUL is not numeric within the range of acceptable values defined by VMIN and VMAX, and not one of the options defined by OPTS. A status of PAR__AMBIG is returned if the default is ambiguous, *i.e.* an abbreviation of more than one of the options.

VMIN = ? (Given)
The value immediately above a range wherein the obtained numeric value cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum numeric value allowed for the obtained value. However, should VMAX be less than VMIN, all numeric values are acceptable except those between VMAX and VMIN exclusive.

VMAX = ? (Given)
The value immediately below a range wherein the obtained numeric value cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum numeric value allowed for the obtained value. However, should VMAX be less than VMIN, all numeric values are acceptable except those between VMAX and VMIN exclusive.

OPTS = CHARACTER $*$ ($*$) (Given)
The list of acceptable options for the value obtained from the parameter. Items should be separated by commas. The list is case-insensitive.

NULL = LOGICAL (Given)
NULL controls the behaviour of this routine when the parameter is in the null state. If NULL is .FALSE., this routine returns with STATUS=PAR__NULL. If NULL is .TRUE., the returned VALUE takes the value of DEFAUL and, if the MSG filtering level (see SUN/104) is 'verbose', a message informs the user of the value used for the parameter. The routine then returns with STATUS=SAI__OK. This feature is intended for convenient handling of null values. NULL should only be set to .TRUE. when the value of DEFAUL will always give a reasonable value for the parameter.

VALUE = CHARACTER $*$ ($*$) (Returned)
The selected option from the list or the character form of the numeric value. The former is in uppercase and in full, even if an abbreviation has been given for the actual parameter. If STATUS is returned not equal to SAI__OK, VALUE takes the value of DEFAUL.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace `"x"` in the routine name by D, I, or R respectively as appropriate. The VMIN and VMAX arguments must have the corresponding data type.
- There are two stages to identify or validate the character value obtained from the parameter.

  In the first the value is converted to the data type specified by the `"x"` in the routine name. If this is successful, the derived numeric value is compared with the range of acceptable values defined by VMIN and VMAX. A value satisfying these constraints is returned in VALUE and the routine exits.

  The second stage searches for a match of the character value with an item in the menu. This step adheres to the following rules.

  – The value is converted to the data type specified by the `"x"` in the routine name. If this is successful, the numeric value is compared with the range of acceptable values defined by VMIN and VMAX. A value satisfying these constraints is returned and the matching process terminates.
  – All comparisons are performed in uppercase. Leading blanks are ignored.
  – A match is found when the value equals the full name of an option. This enables an option to be the prefix of another item without it being regarded as ambiguous. For example, "10,100,200" would be an acceptable list of options.
  – If there is no exact match, an abbreviation is acceptable. A comparison is made of the value with each option for the number of characters in the value. The option that best fits the value is declared a match, subject to two provisos. Firstly, there must be no more than one character different between the value and the start of the option. (This allows for a mistyped character.) Secondly, there must be only one best-fitting option. Whenever these criteria are not satisfied, the user is told of the error, and is presented with the list of options, before being prompted for a new value If a nearest match is selected, the user is informed unless the MSG filtering level (see SUN/104) is 'quiet'.

PAR_MIXVx
Obtains from a parameter character values from either a menu of options or within a numeric range

Description:

This routine obtains a vector of character values from a parameter. Each value must be either:

- one of a supplied list of acceptable values, with unambiguous abbreviations accepted; or
- a numeric character string equivalent to a number, and the number must lie within a supplied range of acceptable values.

Invocation:

```
CALL PAR_MIXVx( PARAM, MAXVAL, VMIN, VMAX, OPTS, VALUES, ACTVAL, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
The name of the parameter.

MAXVAL = INTEGER (Given)
The maximum number of values required. A PAR__ERROR status is returned when the number of values requested is less than one.

VMIN = ? (Given)
The value immediately above a range wherein the obtained numeric values cannot lie. Thus if VMAX is greater than VMIN, VMIN is the minimum numeric value allowed for the obtained values. However, should VMAX be less than VMIN, all numeric values are acceptable except those between VMAX and VMIN exclusive.

VMAX = ? (Given)
The value immediately below a range wherein the obtained numeric values cannot lie. Thus if VMAX is greater than VMIN, VMAX is the maximum numeric value allowed for the obtained values. However, should VMAX be less than VMIN, all numeric values are acceptable except those between VMAX and VMIN exclusive.

OPTS = CHARACTER ∗ (∗) (Given)
The list of acceptable options for each value obtained from the parameter. Items should be separated by commas. The list is case-insensitive.

VALUES(MAXVAL) = CHARACTER ∗ (∗) (Returned)
The selected values that are either options from the list or the character form of numeric values that satisfy the range constraint. The former values are in uppercase and in full, even if an abbreviation has been given for the actual parameter. Note that all values must satisfy the constraints. The values will only be valid if STATUS is not set to an error value.

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

Notes:

- There is a routine for each of the data types double precision, integer, and real: replace "x" in the routine name by D, I, or R respectively as appropriate. The VMIN and VMAX arguments must have the corresponding data type.

- There are two stages to identify or validate each character value obtained from the parameter.

  In the first the value is converted to the data type specified by the `"x"` in the routine name. If this is successful, the derived numeric value is compared with the range of acceptable values defined by VMIN and VMAX. A value satisfying these constraints is returned in the VALUES.

  The second stage searches for a match of the character value with an item in the menu. This step adheres to the following rules.

  – The value is converted to the data type specified by the `"x"` in the routine name. If this is successful, the numeric value is compared with the range of acceptable values defined by VMIN and VMAX. A value satisfying these constraints is returned and the matching process terminates.

  – All comparisons are performed in uppercase. Leading blanks are ignored.

  – A match is found when the value equals the full name of an option. This enables an option to be the prefix of another item without it being regarded as ambiguous. For example, "10,100,200" would be an acceptable list of options.

  – If there is no exact match, an abbreviation is acceptable. A comparison is made of the value with each option for the number of characters in the value. The option that best fits the value is declared a match, subject to two provisos. Firstly, there must be no more than one character different between the value and the start of the option. (This allows for a mistyped character.) Secondly, there must be only one best-fitting option. Whenever these criteria are not satisfied, the user is told of the error, and is presented with the list of options, before being prompted for a new value If a nearest match is selected, the user is informed unless the MSG filtering level (see SUN/104) is 'quiet'.

  This routine exits when all the values satisfy the criteria.

PAR_PROMT
Sets a new prompt string for a parameter

Description:

Replace the prompt string for the indicated parameter by the given string.

Invocation:

```
CALL PAR_PROMT( PARAM, PROMPT, STATUS )
```

Arguments:

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

PROMPT = CHARACTER ∗ (∗) (Given)
The new prompt string.

STATUS = INTEGER
The global status.

PAR_PUT0x
Puts a scalar value into a parameter

Description:

This routine puts a scalar value into a parameter. If necessary, the specified value is converted to the type of the parameter.

Invocation:

```
CALL PAR_PUT0x( PARAM, VALUE, STATUS )
```

Arguments:

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

VALUE = ? (Given)
The value to be put into the parameter.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUE argument must have the corresponding data type.
- A scalar (zero-dimensional) parameter is different from a vector (one-dimensional) parameter containing a single value.
- In order to obtain a storage object for the parameter, the current implementation of the underlying ADAM parameter system will proceed in the same way as it does for input parameters. This can result in users being prompted for ‘a value’. This behaviour, and how to avoid it, is discussed further in the Interface Module Reference Manual (SUN/115).
- Limit checks for IN, RANGE, MIN/MAX are not applied.

PAR_PUT1x
Puts a vector of values into a parameter

Description:

This routine puts a one-dimensional array of values into a parameter. If necessary, the specified array is converted to the type of the parameter.

Invocation:

```
CALL PAR_PUT1x( PARAM, NVAL, VALUES, STATUS )
```

Arguments:

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

NVAL = INTEGER (Given)
The number of values that are to be put into the parameter.

VALUES(NVAL) = ? (Given)
The array of values to be put into the parameter.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- A scalar (zero-dimensional) parameter is different from a vector (one-dimensional) parameter containing a single value.
- In order to obtain a storage object for the parameter, the current implementation of the underlying ADAM parameter system will proceed in the same way as it does for input parameters. This can result in users being prompted for 'a value'. This behaviour, and how to avoid it, is discussed further in the Interface Module Reference Manual (SUN/115).
- Limit checks for IN, RANGE, MIN/MAX are not applied.

PAR_PUTNx
Puts an array of values into a parameter

Description:

This routine puts an *n*-dimensional array of values into a parameter. If necessary, the specified array is converted to the type of the parameter.

Invocation:

```
CALL PAR_PUTNx ( PARAM, NDIM, MAXD, VALUES, ACTD, STATUS )
```

Arguments:

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

NDIM = INTEGER (Given)
The number of dimensions of the values array. This must match the number of dimensions of the object.

MAXD(NDIM) = INTEGER (Given)
The array specifying the dimensions of the array to be put. These may not be greater than the actual dimensions of the parameter (ACTD) nor those of the VALUES array.

VALUES(∗) = ? (Given)
The array of values to be put into the parameter. These must be in Fortran order.

ACTD(NDIM) = INTEGER (Given)
The dimensions of the parameter storage to be created. These are unlikely to be different from MAXD.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "`x`" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- In order to obtain a storage object for the parameter, the current implementation of the underlying ADAM parameter system will proceed in the same way as it does for input parameters. This can result in users being prompted for ‘a value’. This behaviour, and how to avoid it, is discussed further in the Interface Module Reference Manual (SUN/115).
- Limit checks for IN, RANGE, MIN/MAX are not applied.

PAR_PUTVx
Puts an array of values into a parameter as if the parameter were a vector

Description:

This routine puts a one-dimensional array of primitive values into a parameter as it if the parameter were vectorized (*i.e.* regardless of its actual dimensionality). If necessary, the specified array is converted to the type of the parameter.

Invocation:

```
CALL PAR_PUTVx( PARAM, NVAL, VALUES, STATUS )
```

Arguments:

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

NVAL = INTEGER (Given)
The number of values that are to be put into the parameter. It must match the actual parameter's size.

VALUES(NVAL) = ? (Given)
The values to be put into the parameter.

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

Notes:

- There is a routine for each of the data types character, double precision, integer, logical, and real: replace "x" in the routine name by C, D, I, K, L, or R respectively as appropriate. The VALUES argument must have the corresponding data type.
- In order to obtain a storage object for the parameter, the current implementation of the underlying ADAM parameter system will proceed in the same way as it does for input parameters. This can result in users being prompted for 'a value'. This behaviour, and how to avoid it, is discussed further in the Interface Module Reference Manual (SUN/115).
- Limit checks for IN, RANGE, MIN/MAX are not applied.

PAR_STATE
Inquires the state of a parameter

Description:

This routine returns the current state of the indicated parameter. The possible states are GROUND, ACTIVE, CANCELLED and ANNULLED.

Invocation:

```
CALL PAR_STATE( PARAM, STATE, STATUS )
```

Arguments:

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

STATE = INTEGER (Returned)
The current state value of the parameter.

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

Notes:

The symbolic names for these state values are as follows. PAR__GROUND is the ground state, PAR__ACTIVE is the active state, PAR__CANCEL is the cancelled state, and PAR__NULLST is the null state. These are defined in the Fortran INCLUDE file 'PAR_PAR'.

PAR_UNSET
Cancels various parameter control values.

Description:

This routine cancels one or more control values of a parameter. These are currently the parameter's dynamic default value (set by PAR_DEFnx or DAT_DEF), its minimum value (set by PAR_MINx), its maximum value (set by PAR_MAXx), or its prompt string (set by PAR_PROMT).

The routine will operate regardless of the given STATUS value and will not report or set STATUS if the specified values have not been set or are already cancelled.

Invocation:

```
CALL PAR_UNSET( PARAM, WHICH, STATUS )
```

Arguments:

PARAM = CHARACTER∗(∗) (Given)

The name of the parameter.

WHICH = CHARACTER∗(∗) (Given)

A comma-separated list of the control values to be cancelled, selected from the following options:

- 'DEFAULT' to cancel the dynamic default,
- 'MAXIMUM' to cancel the maximum value, and
- 'MINIMUM' to cancel the minimum value.

Unambiguous abbreviations are permitted.

STATUS = INTEGER (Unused)

The global status. The routine is executed regardless of the import value of STATUS. If the import value is not SAI__OK, then it is left unchanged, even if the routine fails to complete. If the import value is SAI__OK on entry and the routine fails to complete, STATUS will be set to an appropriate value.