SUN/238.9

Starlink Project
Starlink User Note 238.9

D.S. Berry
Malcolm J. Currie

2022 January 18

KAPLIBS – Internal subroutines used within the KAPPA package.
Version 3.5
Programmer's Reference

Abstract

KAPLIBS is a package of Fortran subroutine libraries which were originally written as part of the KAPPA package (a package of general-purpose image-processing and visualization tools). KAPLIBS provides software developers with access to many of the internal KAPPA routines, so that KAPPA-like applications can be written and built independently of KAPPA.

Contents

List of Figures

1 Introduction

KAPPA is package of general-purpose astronomical image-processing and visualization commands. It is documented in SUN/95. Over the long history of the KAPPA package, many internal Fortran subroutines have been written to provide facilities within KAPPA which have subsequently proved to be of more-general use outside KAPPA. In order to gain the benefit of these internal KAPPA facilities, software developers have in the past taken copies of the relevant routines and included them in their own projects. The disadvantages of this are obvious—it is easy to end up with many, potentially different, copies of the same routines within a large software suite such as the Starlink Software Collection, and bug fixes need to be implemented in many different places, rather than a in single master copy.

Another way of using the internal KAPPA routines is to link your applications directly against the libraries in the KAPPA package, but this requires the KAPPA package to be installed anywhere where your software is to be built, which is not always convenient.

To get round these problems, the KAPLIBS package was created to contain the internal routines from KAPPA which are deemed to be "generally useful". Now, you only need to have KAPLIBS installed to build your software, not the much larger KAPPA.

There are a large number of routines in KAPLIBS, making it potentially difficult to find the routines you want. To ease this problem, a search tool is provided which allows the contents of this document to be searched (see SEC:SEARCH).

1.1 Stability of the KAPLIBS Interface

KAPLIBS was created as a pragmatic solution to the problem of proliferation of KAPPA source code in several other packages. One of the reasons for previously keeping these routines hidden away within KAPPA was so that changes could be made to the argument lists or functionality of these routines without breaking software within other packages. In practice, it has hardly ever been necessary to change the interface to routines after an initial development period, but the possibility still exists that this may be necessary. For this reason, users of KAPLIBS should be aware that *it may occasionally be necessary to change the interface to KAPLIBS routines*. Such changes will be listed within this document at each release. To aid developers decisions over which routines to use, routines which are deemed to have a significant chance of being changed within the foreseeable future are highlighted later in this document. This class of routine will normally just include routines which have only recently been written.

1.2 The Scope of this Document

The purpose of this document is to give reference information about the argument lists and functionality of the internal KAPPA routines which are contained within KAPLIBS. It does not give a detailed explanation of how these routines should be used within a real application. Developers should usually study examples of existing code within KAPPA for this purpose. The source code for KAPPA is available from the source code repository at https://github.com/Starlink/starlink.

Applications within KAPPA are split up into several groups (called "monoliths'), with names such as `kapview`, `ndfpack` and (rather confusingly but at one time the only KAPPA monolith) `kappa`. The top-level routines for the applications in monolith `xyz` are in the tar file

`$KAPPA_DIR/xyz_sub.tar`. Each application may use some application-specific subroutines (*i.e.* subroutines which are so closely related to the purpose of a single application that they are not deemed as "generally useful"). These will be denoted by subroutines names with prefix "`KPS1_`" and will be contained within the tar file `$KAPPA_DIR/kapsub_sub.tar`.

2 Naming Conventions

Each subroutine within KAPLIBS has a name of the form `<prefix>_<name>` where `<prefix>` is a prefix indicating which KAPPA library the routine belongs to, and `<name>` is a unique name for the routine. The prefix associated with each KAPPA library indicates something of the purpose of that library. The following prefixes are currently included within KAPLIBS:

AIF_ These are old routines which were used to access parameter values or temporary work space. They are gradually being replaced within KAPPA by more modern routines provided by the PAR and PSX libraries.

FTS1_ These are routines used to access FITS files and headers - KAPPA's own `fitsio`.

IRA_ These routines were initially part of the IRAS90 Astrometry library (hence the IRA acronym). They are used to gain access to WCS information stored within NDFs in the form of IRAS90 Astrometry extension. This form of WCS is now deprecated in favour of the NDF WCS component, accessed through the AST library.

KPG1_ These are other general-purpose routines which do not fall into such obvious groups. Even within this library, though, there are loose associations of routines, usually indicated by some common element within the routine name. The following are some of the more significant associations:

- Routines associated with use of the PGPLOT graphics package usually have names which start "`KPG1_PG`".
- Routines associated with use of the AGI graphics database usually have names which start "`KPG1_GD`".
- Routines associated with accessing or using WCS information usually have names which start "`KPG1_AS`".

Within all these groups, some routines have different versions for processing data with different numerical types. The names of such routines are identical except for the trailing one or two characters which indicate the data type processed. These one- or two-character codes are:

d - Double-precision floating point

r - Single-precision floating point

c - Character

i - Single-precision integer

w - Word (usually two-byte integers)

uw - Unsigned word (usually two-byte unsigned integers)

bw - Byte (usually one-byte integers)

ub - Unsigned byte (usually one-byte unsigned integers)

l - Logical

Sometimes, routine names are documented as ending with a lower-case "x". This indicates that routines exist for various of the above numerical types. The *actual* routines names will not include the trailing "x", but will have one of the above codes in place of the "x".

3 Compiling and Linking

To compile and link an application with the KAPLIBS package, the following commands should be used (see SUN/144):

```
% alink adamprog.f ‘kaplibs_link_adam‘
```

Note the use of *opening* apostrophes (‘) rather than the more common closing apostrophe (’)[1].

This produces an executable image called **prog**. The `kaplibs_dev` command creates soft links within the current directory to the various include files provided by KAPLIBS. These are removed by the `kaplibs_dev remove` command.

3.1 Linking with Native PGPLOT

The commands described above will link the application with the Starlink GKS version of the PGPLOT graphics library. If you wish to link with the *native* version of PGPLOT, then include the switch "-nogks" as follows:

```
% alink adamprog.f ‘kaplibs_link_adam -nogks‘
```

This will include native PGPLOT in the link list, and cause all GKS and IDI related items to be removed.

[1]Currently, the parameter handling routines within KAPLIBS have not been separated out, and so it is not currently possible to link stand-alone (*i.e.* "non-Adam") applications against KAPLIBS.

A Changes in Version 3.5

- New generic KPG_STOCx calculates clipped ordered statistics, enabling the API of KPG_STOSx to remain unchanged.
- New routine kpgPutOutline creates and stores an STC polygon describing the spatial extent of an NDF. It has an option to define a convex hull enclosing specified pixels in an array.
- New generic KPG1_FMEDx derives the median of an array using Wirth's method. It is much faster than KPG1_MEDUx.
- KPG1_WRCAT can store arbitrary headers in the output catalogue.
- KPG1_CORRx has a new argument which returns the number of used points.
- The random number seed can now be set via environment variable STAR_SEED.
- kpgGtobj can now create an AST Region describing the coverage of a supplied NDF.
- Fixed a bug in CCG_FLX1 affecting variance calculations where the width was not squared in summations. This would affect the integ estimator in tasks such as KAPPA::COLLAPSE.
- Prevent values falling outside the histogram in KPG1_OPGR1.
- Fixed a memory leak in kpg1Kygp1.

B Changes in Version 3.4

- Handles 64-bit integers.
- New KPG1_KY2HD and kpg1Ky2hd have been moved to the ATL library.
- Added KPG1_QUOTE to put single quotes round a string and escape any embedded single quotes.

C Changes in Version 3.3

- New generic KPG1_CORRx routine to calculate the Pearson correlation coefficient of two arrays.
- New methods in CCG_COMNB1/2 to combine lines to give the number (or fraction) of good (or bad) pixels.

D Changes in Version 3.2

- New KPG_TYPSZ routine adapted from CONVERT's COF_TYPSZ that returns the number of bytes for a given numeric HDS data type.
- Routine KPG1_BADBX now has an extra argument that indicates how the calculated bounding box should be used.
- Routine KPG1_ASSET now allows the new pseudo-attribute `TEXTMARGIN` to be used to specify a margin to clear around a text string. The value of the attribute gives the width of the margin in units of the text height.
- C wrappers have been added for KPG1_BADBX, KPG1_GTOBJ, and KPG1_PXSCL.
- A bug handling default-configuration settings in kpg1Config, resulting in an "invalid object pointer" error, has been fixed.

E Changes in Version 3.1

- New C function kpg1Config gets a KeyMap containing configuration parameters from the user, with name checking and default values supplied.
- The new routine KPG1_DSSFM displays information about the storage form of an NDF array.
- There is a new generic routine KPG_FISEx that fills a section of a multi-dimensional array.
- The new KPG1_GTGPT routine is a variant of KPG1_GTGRP to obtain groups of strings. KPG1_GTGPT supports group members that are not 'sticky', in other words the strings only last for the current invocation of an application.
- The new more-general KPG_ISEQN replaces the old IRCAM-specific NXTNAM routine. This increments trailing sequence numbers in strings, most often used to generate a series of filenames.
- Add new routine KPG_LR2Ax that is the same as KPG_LDA2x (formerly LD2AR) except it processes single-precision sparse data.
- There is a new routine KPG_NORVx to replace the old NORMAL. It has the benefits of being generic and can operate on an array.
- KPG_OSTAx added. It uses recurrence formulae to calculate simple statistics robustly. It also extends KPG1_STATx by deriving the skewness and kurtosis.
- DIMLST is renamed KPG_DIMLS.
- ICMMM has been renamed KPG_IMMMx and made generic. The calculations are now in performed double precision. Note that the API has changed: the first two arguments have been transposed to the standard order, and the returned floating-point statistics are now double precision.

- LD2AR has been renamed KPG_LD2Ax and made generic.
- Remove FTS1_TREAD since the MAG library has now been removed from the main Starlink source tree.
- A number of long-deprecated old routines without a package prefix have been removed. These are listed below. The corresponding replacement routines appear in the right column.

BAD2Dx	KPG_FISEx setting the fill value to VAL__BADx
CHVALx	KPG1_CHVAx
COPAx	VEC_xTOx or KPG1_COPY
COPY1D	" "
COPY2D	" "
COPY3D	" "
CPSECR	KPG1_CPNDx
CREOUT	NDF_CREP and NDF_CPUT
ELNMBx	KPG1_ELNMx
EXPARR	KPG1_EXPOx
INSET	CHR_INSET
LOGARR	KPG1_LOGAx
NORMAL	KPG_NORVx
NXTNAM	KPG_ISEQN
POWARR	KPG1_POWx
ZERO1D	KPG1_FILLx setting the fill value to 0.0.
ZERO2D	KPG1_FILLx setting the fill value to 0.0.

- KPG1_ASSET now parses the supplied parameter name for a non-alphanumeric prefix and suffix to determine whether or not temporary attributes are supported via the new KPG1_GTGPT routine. A suffix indicates that KPG1_ASSET will be called again for the same parameter, affecting the value to write to the parameter file.
- The KPG1_PLTLN routine now accepts a value 6 for the MODE argument, which causes a staircase plot to be produced in which bad values are not flanked by vertical lines to the lower edge of the plot (as are produced by Mode 1), leaving a gap.
- KPG1_STDSx and KPG1_STFLx can respectively report or log the skewness and kurtosis via two additional arguments. Users of these routines will need to modify calling routines accordingly; without skewness or kurtosis values to document supply VAL__BADD for each undefined statistic.
- Support for FITS tapes was removed from FTS1_PHEAD, FTS1_RDATA, FTS1_RGRDA, FTS1_RSTAB, FTS1_SKIP. Argument MEDIUM in these routines now only accepts the

value `DISK`.

- Config files can now include `<def>` to revert to the default value.
- Config files can now include `<undef>` to store an undefined value using kpg1Kymap.

F Changes in Version 3.0

- CTG, IRQ, and LPG libraries are no longer in KAPLIBS. They are autonomous libraries each with its own documentation and link scripts.
- KPG1_MVBDx added. It modifies an array index (such as produced by PDA_QSIAx) to exclude indices that refer to bad data values.
- KPG_STOSx added. It calculates accurate order statistics by sorting an array by means of an index, returning the median and optionally percentiles.

G Changes in Version 2.8

- KPG1_BADBX added. It returns an NDF identifier for the smallest section of a supplied NDF that encloses all the good data values in the NDF.
- KPG1_BBOX<T> added. It obtains the pixel bounding box of a given value within an *N*-dimensional data array.
- KPG1_LIKE added. It obtains a section of an NDF that matches either the pixel or WCS shape of a given template NDF.
- KPG1_KYMAP can read vectors as well as scalars from a GRP group into an AST keyMap.
- `kaplibs_link` script now works without attempting to load ADAM symbols.
- Can now be built as a shared library.
- Added KPG1_ASTCMN to provide access to KPG_AST private common block
- The HLP helper routines are no longer part of kaplibs. Use the SHL library instead.
- KPG1_SCRSZ is no longer present. Please use ONE_SCRSZ instead.
- Now released under the GPL.
- kpgGtfts and kpgPtfts can now be used to read and write AST FitsChan objects from/to an NDF.
- KAPLIBS no longer links against Tk (the one Tk routine has been removed from KAPLIBS).
- KAPLIBS now has a public C interface (but not for all routines).

- Added KPG1_CGET, which returns the value of an NDF character component removing any AST or PGPLOT escape sequences.
- Added KPG1_MXMNX, which provides a single interface to the existing generic KPG1_MXMNx routines.
- Added KPG1_SCALX, which provides a single interface to the new doubly generic KPG1_SCLxx routines, for scaling and shifting data values.

H Changes in Version 2.7

- A new routine LPG_REPLA has been added. It controls a new option which allows a single NDF to be used as both input and output for an ADAM task. The default is for this option to be disabled.

I Changes in Version 2.6

- The GRF module (grf_kaplibs.c) has been upgraded to include the extra functions needed by AST V3.2.

J Changes in Version 2.5

- The KPG1_FFT... routines have been made significantly faster (at the expense of using slightly more memory). For instance, a speed gain of a factor 10 is typical for an array of 150000 points.

K Changes in Version 2.4

- The KPG1_WRLST routine now normalises the supplied positions (using AST_NORM) before writing them to the output catalogue.

L Changes in Version 2.3

- The routine GETHLP has been moved from KAPPA to the KAPLIBS:KAPGEN library.

M Changes in Version 2.2

- Script `kaplibs_link` has been added to enable linking of standalone applications.
- The routine KPG1_DSFRM has been modified to include details of AST SpecFrames. *The argument list has also been changed.*
- The routine KPG1_ASMRG has been modified to attempt alignment in Domain SPECTRAL.

N Changes in Version 2.1

- The KPG1_CPSTY routine has been added to copy AST Plotting styles from one graphical element to another.

O Changes in Version 2.0

- The "-nogks" switch has been added to the `kaplibs_link_adam` command, allowing applications to be linked with native PGPLOT instead of the Starlink GKS-based PGPLOT.
- The routine KPG1_WRLST now allows the user to choose the co-ordinate system in which the positions are stored in the catalogue columns. This is done using two ADAM parameters called COLFRAME and COLEPOCH (these names are hard-wired into the routine to encourage conformity). Consequently, applications which use this routine should add definitions for these two parameters to their IFL files.
- The routine KPG1_RDLST can now read positions from catalogues which do not contain a WCS FrameSet, in certain special cases. That is, if the catalogue contains columns called RA and DEC, or X and Y.
- The IRAS90 IRQ library has been included. IRQ is used to manage textual representations of NDF Quality bits.

P Routine Descriptions

AIF_ANTMP
Annuls a locator to a temporary object, thereby erasing the object

Description:

The routine annuls a locator to a temporary object created by AIF_TEMP, thereby causing the associated object to be erased and the file space associated with it to be released. If data are mapped to the object via HDS, then they are first unmapped.

Invocation:

```
CALL AIF_ANTMP( LOC, STATUS )
```

Arguments:

LOC = CHARACTER * (*) (Given and Returned)

HDS locator to the temporary object to be annulled. The character variable supplied is reset to a blank string by this routine.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- This routine attempts to execute even if STATUS is set on entry. However, no additional error report is made if it subsequently fails under these circumstances.

AIF_ASFIO
Opens a sequential file via a parameter

Description:

This routine opens a sequential file via FIO_ASSOC. Up to four attempts may be made to open the file. If a null response is supplied the file is not opened, and the flag returned indicates this fact.

Invocation:

```
CALL AIF_ASFIO( PNFILE, ACMODE, FORM, RECSZ, FD, OPEN, STATUS )
```

Arguments:

PNFILE = CHARACTER∗(∗)
Parameter name by which file is to be opened

ACMODE = CHARACTER∗(∗)
Expression giving the required access mode. Valid modes are: ' READ', ' WRITE', ' UPDATE' and ' APPEND'. For details, see FIO_OPEN.

FORM = CHARACTER∗(∗)(READ)
Expression giving the required formatting of the file. Valid formats are: ' FORTRAN', ' LIST', ' NONE' and ' UNFORMATTED'. For details, see FIO_OPEN.

RECSZ = INTEGER(READ)
Expression giving the maximum record size in bytes. Set it to zero if the Fortran default is required.

FD = INTEGER(WRITE)
Variable to contain the file descriptor.

OPEN = LOGICAL(WRITE)
If true the file has been opened.

STATUS = INTEGER(READ, WRITE)
Global status value

Method :

Check for error on entry - return if not o.k. Initialise looping flag Do while no error obtaining the name and opening the output file and maximum number of attempts not exceeded Get file name and open file If null returned then Set flag so that a log file will not be created Annul the error Exit from the loop Else if error occurred then If abort requested, do so Increment loop counter If maximum number of attempts not exceeded then Report error Else Set looping flag to exit Endif Cancel parameter used to get filename Else Set flag to indicate that the file has been opened Set looping flag to false Endif Enddo If error then Report and abort Endif Return

Bugs:

None known.

AIF_FLNAM
Returns the name of a file as a character string given its parameter name

Description:

Normally, the handles to HDS data files are locators and files are obtained via the parameter system. The name of a data file itself is not available to applications via the user-level parameter-system library, and so applications cannot place the file name in log files or use the it to generate compound names. This routine provides that functionality by calling internal parameter-system routines to obtain the name. This routine also works for Fortran files associated via an ADAM parameter.

Invocation:

```
CALL AIF_FLNAM( PARNAM, FILNAM, STATUS )
```

Arguments:

PARNAM = CHARACTER * (*) (Given)
Parameter name associated with the data object whose name is required.

FILNAM = CHARACTER * (*) (Returned)
The name of the file. If %FILNAM is not long enough a bad status is returned.

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

Notes:

- This cannot be used to obtain the names of objects within an HDS file.

Prior Requirements :

- Must have obtained the file via the input parameter.

AIF_GETVM
Obtains a pointer and a locator to mapped HDS workspace

Description:

This routine obtains and file maps a temporary array of a given data type and dimensions via HDS. A pointer to the mapped work array is returned, as is a locator so that the temporary array may be annulled when no longer required.

Invocation:

```
CALL AIF_GETVM( TYPE, NDIM, DIMS, PNTR, WKLOC, STATUS )
```

Arguments:

TYPE = CHARACTER * (*) (Given)
The HDS data type of the temporary array.

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

DIMS(NDIM) = INTEGER (Given)
The dimensions of the temporary array.

PNTR = INTEGER (Returned)
The pointer to the mapped temporary array.

WKLOC = CHARACTER * (DAT__SZLOC)(Returned)
The HDS locator to the temporary array.

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

Notes:

The current HDS tuning parameter MAP is stored, so that the temporary array may be accessed via file mapping, and upon exit restored to its former value.

AIF_OPFIO
Opens a Fortran sequential file by name

Description:

This routine opens a sequential file via FIO_OPEN. Up to four attempts may be made to open the file. If a null response is supplied the file is not opened, and the flag returned indicates this fact.

Invocation:

```
CALL AIF_OPFIO( FILNAM, ACMODE, FORM, RECSZ, FD, OPEN, STATUS )
```

Arguments:

FILNAM = CHARACTER∗(∗)
The name of the file to be opened.

ACMODE = CHARACTER∗(∗)
Expression giving the required access mode. Valid modes are: ' READ', ' WRITE', ' UPDATE' and ' APPEND'. For details, see FIO_OPEN.

FORM = CHARACTER∗(∗)(READ)
Expression giving the required formatting of the file. Valid formats are: ' FORTRAN', ' LIST', ' NONE' and ' UNFORMATTED'. For details, see FIO_OPEN.

RECSZ = INTEGER(READ)
Expression giving the maximum record size in bytes. Set it to zero if the Fortran default is required.

FD = INTEGER(WRITE)
Variable to contain the file descriptor.

OPEN = LOGICAL(WRITE)
If true the file has been opened.

STATUS = INTEGER(READ, WRITE)
Global status value

Method :

Check for error on entry - return if not o.k. Initialise looping flag Do while no error obtaining the name and opening the output file and maximum number of attempts not exceeded Get file name and open file If error occurred then If abort requested, do so Increment loop counter If maximum number of attempts not exceeded then Report error Else Set looping flag to exit Endif Else Set flag to indicate that the file has been opened Set looping flag to false Endif Enddo If error then Report and abort Endif Return

Bugs:

None known.

AIF_PTFNM
Writes the name of an HDS file to a parameter

Description:

Normally, the handles to HDS data files are locators and files are obtained via the parameter system. However, some applications can generate sensible names, especially when dealing a long series of files that are to be created without manual intervention. There is no direct mechanism in the user-level parameter-system library to put a name into the associated parameter. This routine provides that functionality.

Invocation:

```
CALL AIF_PTFNM( PARNAM, FILNAM, STATUS )
```

Arguments:

PARNAM = CHARACTER * (*) (Given)
Parameter name associated with the data object whose name is to be written.

FILNAM = CHARACTER * (*) (Given)
The name of the file.

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

Notes:

- This cannot be used to obtain the names of objects within an HDS file.

Prior Requirements :

- Must have obtained the file via the input parameter.

AIF_TEMP
Create a temporary HDS object

Description:

The routine creates a temporary HDS object with the specified type and shape. On the first invocation a temporary structure is created to contain such objects. Subsequently, temporary objects are created within this enclosing structure.

Invocation:

```
CALL AIF_TEMP( TYPE, NDIM, DIM, LOC, STATUS )
```

Arguments:

TYPE = CHARACTER * (*) (Given)
HDS type of object to be created.

NDIM = INTEGER (Given)
Number of object dimensions.

DIM(NDIM) = INTEGER (Given)
Object dimensions.

LOC = CHARACTER * (*) (Returned)
Locator to temporary object.

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

Notes:

- A blank string will be returned for the LOC argument if this routine is called with STATUS set, although no further processing will occur. The same value will also be returned if the routine should fail for any reason.
- This routine is a work-around to avoid the problems associated with calling DAT_TEMP if the objects created must subsequently be erased.

CCG_AD1x
Combines data lines using absolute deviations from an unweighted mean

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. The data values in the lines are then combined to form an unweighted mean, and then the absolute mean deviation from the mean. The output means are returned in the array RESULT. The output variances are standard-error values and are returned in RESVAR.

Invocation:

```
CALL CCG_AD1x( NPIX, NLINES, STACK, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

CCG_AD3x
Combines data lines using absolute deviations from an unweighted mean

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. The data values in the lines are then combined to form an unweighted mean, and then the absolute mean deviation from the mean. The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_AD3x( NPIX, NLINES, STACK, MINPIX, RESULT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

CCG_BM1x
Combines data lines using a broadened median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an broadened medium line. The broadened median is similar to a trimmed mean, except that the trimming fraction changes with the number of values (and is equivalent to a median for fewer than five values). The output means are returned in the array RESULT. The variances are propagation through the combination process and returned in the RESVAR array.

Invocation:

```
CALL CCG_BM1x( NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR, WRK1,
POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The variances of the data.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_BM3x
Combines data lines using a broadened median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an broadened medium line. The broadened median is similar to a trimmed mean, except that the trimming fraction changes with the number of values (and is equivalent to a median for fewer than five values). The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_BM3x( NPIX, NLINES, STACK, MINPIX, RESULT, WRK1, POINT, USED, NCON, NBAD, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace "x" in the routine name by D or R as appropriate. The STACK, RESULT, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_CLIPx
Sets any data outside a given value range BAD

Description:

This generic routine loops over all entries in RA. If any values are outside the range RMIN to VMAX then they are set BAD.

Invocation:

```
CALL CCG_CLIPR( VMIN, VMAX, EL, ARRAY, NGOOD, STATUS )
```

Arguments:

VMIN = ? (Given)
The minimum allowed value. All values below this are set BAD.

VMAX = ? (Given)
The maximum allowed value. All values above this are set BAD.

EL = INTEGER (Given)
The number of entries in ARRAY.

ARRAY(EL) = ? (Given and Returned)
The list of values to be clipped within the given range. On output this contains the clipped list.

NGOOD = INTEGER (Returned)
The number of values left after rejection.

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

Notes:

- There is a routine for real and double-precision: replace `" x"` in the routine name by D or R as appropriate. The ARRAY, VMIN, and VMAX arguments supplied to the routine must have the data type specified.

CCG_CNT1x
Combines data lines by counting pixels

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the count of good (or bad) values in each line. The output values are returned in the array RESULT. Each output variance value is set to zero and are returned in the array RESVAR.

Invocation:

```
CALL CCG_CNT1x( IMETH, NPIX, NLINES, STACK, RESULT, RESVAR, NCON, NBAD, STATUS )
```

Arguments:

IMETH = INTEGER (Given)
35 - Return the number of good values in each line. 36 - Return the number of bad values in each line.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

RESULT(NPIX) = INTEGER (Returned)
The output line of data. Note, the data type is always INTEGER.

RESVAR(NPIX) = INTEGER (Returned)
The output variances. Note, the data type is always INTEGER.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line. Each value in this array is set to NPIX.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. This is always returned as zero by this routine.

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

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK argument supplied to the routine must have the data type specified.

CCG_CNT3x
Combines data lines by counting pixels

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the count of good (or bad) values in each line. The output values are returned in the array RESULT.

Invocation:

```
CALL CCG_CNT3x( IMETH, NPIX, NLINES, STACK, RESULT, NCON, NBAD, STATUS )
```

Arguments:

IMETH = INTEGER (Given)
35 - Return the number of good values in each line. 36 - Return the number of bad values in each line.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

RESULT(NPIX) = INTEGER (Returned)
The output line of data. Note, the data type is always INTEGER.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line. Each value in this array is set to NPIX.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. This is always returned as zero by this routine.

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

Notes:

- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK argument supplied to the routine must have the data type specified.

CCG_COMB1x
Combines a stack of array lines into one line, using a variety of methods

Description:

The routine works along each line of the input stack of lines, combining the data. This variant uses a complete variance array propagates them.

The array NCON holds the actual numbers of pixels that were used in deriving the output value plus any values already present in the array; thus a cumulative sum of contributing pixel numbers may be kept.

Invocation:

```
CALL CCG_COMB1x( NPIX, NLINES, STACK, VARS, COORDS, IMETH, MINPIX, NITER, NSIGMA, ALPHA,
RMIN, RMAX, NMAT, RESULT, RESVAR, WIDTHS, COIND, WRK1, WRK2, PP, COVEC, POINT, USED,
NCON, NFLAG, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

COORDS(NPIX, NLINES) = ? (Given)
The co-ordinates along the collapse axis for each pixel. It is accessed only for IMETH = 22, 23, 33, 34.

IMETH = INTEGER (Given)
The method to use in combining the lines. It has a code of 1 to 300 which represent the following statistics. 1 = Mean 2 = Weighted mean 3 = Median 4 = Trimmed mean 5 = Mode 6 = Sigma-clipped mean 7 = Threshold exclusion mean 8 = Minmax mean 9 = Broadened median 10 = Sigma-clipped median 11 = Fast median 12 = Sum 13 = Standard deviation about the mean 14 = Sigma-clipped standard deviation about the mean 21 = Integrated value (sum of pixel co-ordinate width times value) 22 = Intensity-weighted co-ordinate 23 = Intensity-weighted dispersion of the co-ordinate 24 = Root mean square 25 = Absolute mean deviation 31 = Maximum 32 = Minimum 33 = Co-ordinate of maximum 34 = Co-ordinate of minimum 35 = Number of good pixel values 36 = Number of bad pixel values 37 = Fraction of good pixel values 38 = Fraction of bad pixel values 300 = Median, but estimating variance from mean variance

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NITER = INTEGER (Given)
The maximum number of iterations (IMETH = 5).

NSIGMA = REAL (Given)
The number of sigmas to clip the data at (IMETH = 5 and 6).

ALPHA = REAL (Given)
The fraction of data values to remove from data (IMETH = 4).

RMIN = REAL (Given)
The minimum allowed data value (IMETH = 7).

RMAX = REAL (Given)
The maximum allowed data value (IMETH = 7).

NMAT = INTEGER (Given)
Number of columns in covariance matrix. It should be at least NLINES $*$ (NLINES $+$ 1)/ 2.

RESULT(NPIX) = ? (Returned)
The output line of data. This will always be of type _INTEGER for methods 35 (NGOOD) and 36 (NBAD). Since this routine does not actually access the array values, the mismatch in data types does not matter.

RESVAR(NPIX) = ? (Returned)
The output variances. This will always be of type _INTEGER for methods 35 (NGOOD) and 36 (NBAD).

WIDTHS(NPIX, NLINES) = ? (Returned)
The widths along the collapse axis for each pixel. It is calculated only for IMETH = 21.

COIND(NPIX) = INTEGER (Returned)
Workspace to hold co-ordinate indices.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

PP(NLINES) = DOUBLE PRECISION (Returned)
Workspace for order statistics calculations.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Returned)
Workspace for storing ordered statistics variance-covariance matrix. Used for IMETHs 3 to 11, and 14.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NFLAG = INTEGER (Returned)
Number of output pixels set to bad because insufficient pixels were present to form the statistic for the collapsed axis, provided the minimum number of contributing data values is one.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, COORDS, RESULT, RESVAR, WIDTHS, WRK1, and WRK2 arguments supplied to the routine must have the data type specified (except for methods 35 and 36 for which RESULT and RESVAR are always _INTEGER).

- Various of the options are simply variations on a theme. The Broadened median is just a trimmed mean with a variable trimming fraction. The Mode is an iteratively carried out version of the sigma clipping (or more precisely the reverse). The minmax and threshold mean are also just trimmed means, but require their own mechanisms.
- The ' propagation' of the input variances assumes that the input data are fairly represented by a normal distribution. This fact is used together with the ' order statistics' of a normal population to form a new variance estimate. The order statistics are not independent so have non-zero covariances (off diagonal components of the variance-covariance matrix). All ' trimmed means' of any description use the order of the values to estimate which values are corrupt. This applies to all the methods supported here except the mean which rejects no data. The variance used to represent the input normal population is the reciprocal of the sum of the reciprocal variances. We have no other estimate of this value except from the population itself.
- Calculations are performed in double precision.

CCG_COMB3x
Combines a stack of array lines into one line, using a variety of methods

Description:

The routine works along each line of the input stack of lines, combining the data. This variant uses a single variance for each line and and does NOT propagate it.

The array NCON holds the actual numbers of pixels which were used in deriving the output value plus any values already present in the array; thus a cumulative sum of contributing pixel numbers may be kept.

Invocation:

```
CALL CCG_COMB3x( NPIX, NLINES, STACK, VARS, COORDS, IMETH, MINPIX, NITER, NSIGMA, ALPHA,
RMIN, RMAX, RESULT, WIDTHS, COIND, WRK1, WRK2, POINT, USED, NCON, NFLAG, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data.

COORDS(NPIX, NLINES) = ? (Given)
The co-ordinates along the collapse axis for each pixel. It is accessed only for IMETH = 22, 23, 33, 34.

IMETH = INTEGER (Given)
The method to use in combining the lines. It has a code of 1 to 300 which represent the following statistics. 1 = Mean 2 = Weighted mean 3 = Median 4 = Trimmed mean 5 = Mode 6 = Sigma-clipped mean 7 = Threshold exclusion mean 8 = Minmax mean 9 = Broadened median 10 = Sigma-clipped median 11 = Fast median 12 = Sum 13 = Standard deviation about the mean 14 = Sigma-clipped standard deviation about the mean 21 = Integrated value (sum of pixel co-ordinate width times value) 22 = Intensity-weighted co-ordinate 23 = Intensity-weighted dispersion of the co-ordinate 24 = Root mean square 25 = Absolute mean deviation 31 = Maximum 32 = Minimum 33 = Co-ordinate of maximum 34 = Co-ordinate of minimum 35 = Number of good pixel values 36 = Number of bad pixel values 37 = Fraction of good pixel values 38 = Fraction of bad pixel values 300 = Median, but estimating variance from mean variance

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NITER = INTEGER (Given)
The maximum number of iterations (IMETH = 5).

NSIGMA = REAL (Given)
The number of sigmas to clip the data at (IMETH = 5 and 6).

ALPHA = REAL (Given)
The fraction of data values to remove from data (IMETH = 4).

RMIN = REAL (Given)
The minimum allowed data value (IMETH = 7).

RMAX = REAL (Given)
The maximum allowed data value (IMETH = 7).

RESULT(NPIX) = ? (Returned)
The output line of data. This will always be of type _INTEGER for methods 35 (NGOOD) and 36 (NBAD). Since this routine does not actually access the array values, the mismatch in data types does not matter.

WIDTHS(NPIX, NLINES) = ? (Returned)
The widths along the collapse axis for each pixel. It is calculated only for IMETH = 21.

COIND(NPIX) = INTEGER (Returned)
Workspace to hold co-ordinate indices.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NFLAG = INTEGER (Returned)
Number of output pixels set to bad because insufficient pixels were present to form the statistic for the collapsed axis, provided the minimum number of contributing data values is one.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, COORDS, RESULT, WIDTHS, WRK1, and WRK2 arguments supplied to the routine must have the data type specified (except for methods 35 and 36 for which RESULT is always _INTEGER).
- Various of the options are simply variations on a theme. The Broadened median is just a trimmed mean with a variable trimming fraction. The Mode is an iteratively carried out version of the sigma clipping (or more precisely the reverse). The minmax and threshold mean are also just trimmed means, but require their own mechanisms.
- No propagation of variances is performed using this routine.
- Calculations are performed in double precision.

CCG_CS1x
Combines data lines using a sigma-clipped standard deviation

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The weighted mean and standard deviation of each input column in STACK is then used to estimate the range of values which represent the required sigma clipping. Values outside of this range are then rejected and the resulting output standard deviations are returned in the array RESULT. The variances are propagated through the combination process and returned in the RESVAR array.

Invocation:

```
CALL CCG_CS1x( NSIGMA, NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR,
WRK1, WRK2, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data, i.e. the clipped standard deviations.

RESVAR(NPIX) = ? (Returned)
The output variances of the clipped standard deviations.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace "`x`" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_CS3x
Combines data lines using a sigma-clipped standard deviation

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The weighted mean and standard deviation of each input column in STACK is then used to estimate the range of values that represent the required sigma clipping. The standard deviation is derived from the population of values at each position along the lines (cf. each image pixel). Values outside of this range are then rejected and the resulting output standard-deviation values are returned in the array RESULT.

Note that clipping will not be used when only two or three values are available (unless in the case of three values NSIGMA is less than 1.0).

Invocation:

```
CALL CCG_CS3x( NSIGMA, NPIX, NLINES, STACK, VARS, MINPIX, RESULT, WRK1, WRK2, POINT,
USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data. These are used as inverse weights when forming the mean and do not need to be real variances, as they are not propagated.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of clipped standard deviations.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_FLX1x
Combines data lines using the integrated value

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. It forms the sum of the data values multiplied by its pixel width lines are then summed, i.e a flux. The output integrated values are returned in the array RESULT. The output variances are estimated from the VARS values and are returned in RESVAR; this assumes no errors in the widths.

Invocation:

```
CALL CCG_FLX1x( NPIX, NLINES, STACK, VARS, WIDTHS, MINPIX, RESULT, RESVAR, NCON, NBAD,
STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

WIDTHS(NPIX, NLINES) = ? (Given)
The pixel widths in world co-ordinates.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = <TYPE> (Returned)
The output line of data.

RESVAR(NPIX) = <TYPE> (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace "x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.

CCG_FLX3x
Combines data lines using the integrated value

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. It forms the sum of the data values multiplied by its pixel width lines are then summed, i.e a flux. The output integrated values are returned in the array RESULT.

Invocation:

```
CALL CCG_FLX3x( NPIX, NLINES, STACK, WIDTHS, MINPIX, RESULT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

WIDTHS(NPIX, NLINES) = ? (Given)
The pixel widths in world co-ordinates.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D or R as appropriate. The STACK and RESULT arguments supplied to the routine must have the data type specified.

CCG_FM1x
Combines data lines using a fast median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form a (fast) median line. This method uses no weights and returns the estimated variances based on order stats for the population sizes used at each output pixel.

The method used is based on Wirth's algorithm for selecting the *K*th value, which is very fast compared with a full sort.

The output medians are returned in the array RESULT.

Invocation:

```
CALL CCG_FM1x( NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR, WRK1,
WRK2, POINT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR. This is only used when median is derived from the central-2 values.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variance.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_FM3x
Combines data lines using a fast median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form a (fast) median line. This method uses no weights and returns no variances.

The method used is based on Wirth's s algorithm for selecting the *K*th value, which is very fast compared with a full sort.

The output medians are returned in the array RESULT.

Invocation:

```
CALL CCG_FM3x( NPIX, NLINES, STACK, MINPIX, RESULT, WRK1, NCON, POINT, USED, NCON, NBAD,
STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace `" x"` in the routine name by D or R as appropriate. The STACK, RESULT, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_FRC1x
Combines data lines by counting pixels

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the fraction of good (or bad) values in each line. The output values are returned in the array RESULT. Each output variance value is set to zero and are returned in the array RESVAR.

Invocation:

```
CALL CCG_FRC1x( IMETH, NPIX, NLINES, STACK, RESULT, RESVAR, NCON, NBAD, STATUS )
```

Arguments:

IMETH = INTEGER (Given)
37 - Return the fraction of good values in each line. 38 - Return the fraction of bad values in each line.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line. Each value in this array is set to NPIX.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. This is always returned as zero by this routine.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, RESULT and RESVAR arguments supplied to the routine must have the data type specified.

CCG_FRC3x
Combines data lines by counting pixels

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the fraction of good (or bad) values in each line. The output values are returned in the array RESULT.

Invocation:

```
CALL CCG_FRC3x( IMETH, NPIX, NLINES, STACK, RESULT, NCON, NBAD, STATUS )
```

Arguments:

IMETH = INTEGER (Given)
37 - Return the fraction of good values in each line. 38 - Return the fraction of bad values in each line.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line. Each value in this array is set to NPIX.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. This is always returned as zero by this routine.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK and RESULT arguments supplied to the routine must have the data type specified.

CCG_I2WCx
Assigns world co-ordinates to an output array from an input list of indices

Description:

This routine assigns world co-ordinates to an output vector from an input two-dimensional array, corresponding to a a vector for each output element. The world co-ordinates are selected using a list of indices in each world-co-ordinate vector, there being one index per output value. A bad value or a value outside the bounds of the array in the list of indices causes a bad value to be assigned to the output array.

Invocation:

```
CALL CCG_I2WCx( NPIX, NLINES, INDICE, COORDS, RESULT, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data.

INDICE(NPIX) = INTEGER (Given)
The indices in the input array that point to the values to be assigned to the output vector.

COORDS(NPIX, NLINES) = ? (Given)
The world co-ordinates.

RESULT(NPIX) = ? (Returned)
The vector containing values copied from the input COORDS array.

NBAD = INTEGER (Returned)
The number of bad values in the output array.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for real and double-precision: replace "`x`" in the routine name by D or R as appropriate. The COORDS and RESULT supplied to the routine must have the data type specified.

CCG_IS2x
Sorts a list of data into increasing order, and applies the corresponding shuffle to an ancillary list

Description:

The routine uses an insert-sort method. This has proven itself the quickest for sorting small sets of data lots of times, as in image stacking using ordered statistics. The method looks at the second value, compares this with the first if swaps if necessary, then it looks at the third, looks at the previous values swaps with the lowest (or not at all) and so on until all values have been passed. It is fast (for the case above) simply because of the very few lines of operation. The sort is extended to the ancillary data ANCDAT, this maintains its correspondence with the ORDDAT dataset on exit.

Invocation:

```
CALL CCG_IS2x( EL, ORDDAT, ANCDAT, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of entries in ORDDAT.

ORDDAT(EL) = ? (Given and Returned)
The data to order. On output it contains the data in increasing order.

ANCDAT(EL) = INTEGER (Given and Returned)
A list of data associated with ORDDAT which needs to retain its correspondence with the items in ORDDAT (probably pointers).

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The ORDDAT argument supplied to the routine must have the data type specified.

CCG_IS3x
Sorts a list of data into increasing order, and applies the corresponding shuffle to two ancillary lists

Description:

The routine uses an insert sort method. This has proven itself the quickest for sorting small sets of data lots of times, as in image stacking using ordered statistics. The method looks at the second value, compares this with the first if swaps if necessary, then it looks at the third, looks at the previous values swaps with the lowest (or not at all) and so on until all values have been passed. It is fast (for the case above) simply because of the very few lines of operation. The sort is extended to the ancillary data RDAT and PDAT that maintain their correspondence with the ORDDAT dataset on exit.

Invocation:

```
CALL CCG_IS3x( EL, ORDDAT, RDAT, PDAT, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of entries in ORDDAT.

ORDDAT(EL) = ? (Given and Returned)
The data to order. On output it contains the data in increasing order.

RDAT(EL) = ? (Given and Returned)
A list of data associated with ORDDAT which needs to retain its correspondence with the items in ORDDAT.

PDAT(EL) = INTEGER (Given and Returned)
A list of data associated with ORDDAT which needs to retain its correspondence with the items in ORDDAT (probably pointers).

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The ORDDAT and RDAT arguments supplied to the routine must have the data type specified.

CCG_IWC1x
Combines data lines using the intensity-weighted co-ordinate

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data and their co-ordinates. For each pixel the subroutine forms a weighted mean co-ordinate and variance along a line of data. The weights are the product of the data values and their inverse variance. The output weighted co-ordinates are returned in the array RESULT. The output weighted variances are returned in RESVAR; this assumes no errors in the co-ordinates.

Invocation:

```
CALL CCG_IWC1x( NPIX, NLINES, STACK, VARS, COORDS, MINPIX, RESULT, RESVAR, NCON, NBAD,
STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

COORDS(NPIX, NLINES) = ? (Given)
The world co-ordinates.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D or R as appropriate. The STACK, VARS, COORDS, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.

CCG_IWC3x
Combines data lines using the intensity-weighted co-ordinate

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data and their co-ordinates. For each pixel the subroutine forms a weighted mean co-ordinate along a line of data. The weights are the data values. The output weighted co-ordinates are returned in the array RESULT.

Invocation:

```
CALL CCG_IWC3x( NPIX, NLINES, STACK, COORDS, MINPIX, RESULT, NCON, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

COORDS(NPIX, NLINES) = ? (Given)
The world co-ordinates.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D or R as appropriate. The STACK, COORDS, and RESULT arguments supplied to the routine must have the data type specified.

CCG_IWD1x
Combines data lines using the dispersion of the intensity-weighted co-ordinate

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data and their co-ordinates. For each pixel the subroutine forms the weighted dispersion of the co-ordinate and an estimated variance along a line of data. The weights are the data values and their inverse variances. The output co-ordinate dispersions are returned in the array RESULT. The output variances in RESVAR are the standard error of RESULT.

Invocation:

```
CALL CCG_IWD1x( NPIX, NLINES, STACK, VARS, COORDS, MINPIX, RESULT, RESVAR, NCON, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

COORDS(NPIX, NLINES) = ? (Given)
The world co-ordinates.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, COORDS, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.

- It uses the corrected two-pass algorithm of Chan, T.F., Golub, G.H., & LeVeque, R.J. (1983). " Algorithms for computing the sample variance: Analysis and recommendations" , American Statistician, 37(3), 242-247.

CCG_IWD3x
Combines data lines using the dispersion of the intensity-weighted co-ordinate

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data and their co-ordinates. For each pixel the subroutine forms the weighted dispersion of the co-ordinate along a line of data. The weights are the data values. The output co-ordinate dispersions are returned in the array RESULT.

Invocation:

```
CALL CCG_IWD3x( NPIX, NLINES, STACK, COORDS, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

COORDS(NPIX, NLINES) = ? (Given)
The world co-ordinates.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D or R as appropriate. The STACK, COORDS, and RESULT arguments supplied to the routine must have the data type specified.
- It uses the corrected two-pass algorithm of Chan, T.F., Golub, G.H., & LeVeque, R.J. (1983). " Algorithms for computing the sample variance: Analysis and recommendations" , American Statistician, 37(3), 242–247.

CCG_KTHx
Returns the Kth smallest value in an array

Description:

This routine returns the value of the Kth smallest element in the given array. It is an implementation of an algorithm of Niklaus Wirth from the book " Algorithms + data structures = programs" .

Invocation:

```
CALL CCG_KTHx( EL, K, ARRAY, ANCDAT, VALUE, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the input array.

K = INTEGER (Given and Returned)
On entry the ordered value to be returned, on exit index of ARRAY that contain the Kth smallest value.

ARRAY(EL) = ? (Given and Returned)
The array of values to be processed. Note this is modified on exit.

ANCDAT(EL) = INTEGER (Given and Returned)
Array of data whose association with ARRAY is to be preserved (such as a list of pointers to the original positions within ARRAY).

VALUE = ? (Returned)
The Kth smallest value in the input array.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The ARRAY and VALUE arguments supplied to the routine must have the data type specified.

CCG_MD1x
Combines data lines using a weighted median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form a weighted median line. The weights used are one-per-line and are the reciprocal of values given in the array VARS. The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_MD1x( CALCMV, NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR,
WRK1, WRK2, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

CALCMV = LOGICAL (Given)
If .FALSE. then the output variances are estimated by scaling the variance on the weighted mean (rather than the weighted median) by Pi/2. Otherwise, the output variances are calculated using the COVEC array.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR. Not used if CALCMV is .FALSE.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variance.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_MD3x
Combines data lines using a weighted median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form a weighted median line. The weights used are one-per-line and are the reciprocal of values given in the array VARS. The output medians are returned in the array RESULT.

Invocation:

```
CALL CCG_MD3x( NPIX, NLINES, STACK, VARS, MINPIX, RESULT, WRK1, WRK2, POINT, USED, NCON,
NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace "`x`" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_ME1x
Combines data lines using a weighted mean

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. The data values in the lines are then combined to form a weighted mean. The weights used are are the reciprocal of values given in the array VARS. The output means are returned in the array RESULT. The output variances are estimated from the VARS values and are returned in RESVAR.

Invocation:

```
CALL CCG_ME1x( NPIX, NLINES, STACK, VARS, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace `" x"` in the routine name by D or R as appropriate. The STACK, VARS, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.
- This routine performs its work in double precision.

CCG_ME3x
Combines data lines using a weighted mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form a weighted mean. The weights used are one-per-line and are the reciprocal of values given in the array VARS. The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_ME3x( NPIX, NLINES, STACK, VARS, MINPIX, RESULT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, and RESULT arguments supplied to the routine must have the data type specified.
- This routine performs its work in double precision.

CCG_MM1x
Combines data lines using a min-max exclusion trimmed mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an trimmed mean in which the minimum and maximum values are excluded. The output means are returned in the array RESULT. The output variances are propagated from the inverse weights given in array VARS and are returned in the array RESVAR.

Invocation:

```
CALL CCG_MM1x( NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR, WRK1,
POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_MM3x
Combines data lines using a min-max exclusion trimmed mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an trimmed mean in which the minimum and maximum values are excluded. The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_MM3x( NPIX, NLINES, STACK, MINPIX, RESULT, WRK1, POINT, USED, NCON, NBAD, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_MN1x
Combines data lines using the minimum value

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the minimum value in each line. The output minima are returned in the array RESULT. Each output variance value is the corresponding variance of the minimum data value, propagated from the input variances given in array VARNCE and are returned in the array RESVAR. The vectorised pixel indices corresponding to the minima are also returned in POSIND.

Invocation:

```
CALL CCG_MN1x( BAD, NPIX, NLINES, STACK, VARNCE, RESULT, RESVAR, POSIND, WRK1, NCON,
NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true, there may be bad pixels present in the array. If false, it is safe not to check for bad values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARNCE(NPIX, NLINES) = ? (Given)
The data variances.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

POSIND(NPIX) = INTEGER (Returned)
The pixel indices of the minima in each output pixel.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, VARNCE, RESULT, RESVAR, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_MN3x
Combines data lines using the minimum value

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the minimum value in each line. The output minima are returned in the array RESULT. The vectorised pixel indices corresponding to the minima are also returned in POSIND.

Invocation:

```
CALL CCG_MN3x( BAD, NPIX, NLINES, STACK, RESULT, POSIND, WRK1, NCON, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true, there may be bad pixels present in the array. If false, it is safe not to check for bad values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

RESULT(NPIX) = <TYPE> (Returned)
The output line of data.

POSIND(NPIX) = INTEGER (Returned)
The pixel indices of the minima in each output pixel.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, RESULT, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_MO1x
Combines data lines using a mode

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an output mode line. The output modal values are returned in the array RESULT. The variances are propagated through the combination process and returned in RESVAR.

Invocation:

```
CALL CCG_MO1x( NSIGMA, NITER, NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT,
RESVAR, WRK1, WRK2, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NITER = INTEGER (Given)
Maximum number of refining iterations.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_MO3x
Combines data lines using a mode

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an output mode line. The output modal values are returned in the array RESULT.

Invocation:

```
CALL CCG_MO3x( NSIGMA, NITER, NPIX, NLINES, STACK, VARS, MINPIX, RESULT, WRK1, WRK2,
POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NITER = INTEGER (Given)
Maximum number of refining iterations.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_MX1x
Combines data lines using the maximum value

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the maximum value in each line. The output maxima are returned in the array RESULT. Each output variance value is the corresponding variance of the maximum data value, propagated from the input variances given in array VARNCE and are returned in the array RESVAR. The vectorised pixel indices corresponding to the maxima are also returned in POSIND.

Invocation:

```
CALL CCG_MX1x( BAD, NPIX, NLINES, STACK, VARNCE, RESULT, RESVAR, POSIND, WRK1, NCON,
NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true, there may be bad pixels present in the array. If false, it is safe not to check for bad values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARNCE(NPIX, NLINES) = ? (Given)
The data variances.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

POSIND(NPIX) = INTEGER (Returned)
The pixel indices of the maxima in each output pixel.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, VARNCE, RESULT, RESVAR, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_MX3x
Combines data lines using the maximum value

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then collapsed to the maximum value in each line. The output maxima are returned in the array RESULT. The vectorised pixel indices corresponding to the maxima are also returned in POSIND.

Invocation:

```
CALL CCG_MX3x( BAD, NPIX, NLINES, STACK, RESULT, POSIND, WRK1, NBAD, NCON, NBAD, STATUS
)
```

Arguments:

BAD = LOGICAL (Given)
If true, there may be bad pixels present in the array. If false, it is safe not to check for bad values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

RESULT(NPIX) = ? (Returned)
The output line of data.

POSIND(NPIX) = INTEGER (Returned)
The pixel indices of the maxima in each output pixel.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, RESULT, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_ORVAR
Returns the variances and covariances of the order statistics from n to 1, assuming an initially normal distribution

Description:

The routine returns the variances and covariances of the order statistics, assuming an initial (pre-ordered) normal distribution of mean 0 and standard deviation 1. The routine returns all variance/covariances in an array with the terms vectorised - that is following on after each row. This uses the symmetric nature of the matrix to compress the data storage, but remember to double the covariance components if summing in quadrature. The variances

- covariances are returned for all statistics from n to 1. The special case of n = 1 returns the variance of 2/pi (median).

Invocation:

```
CALL CCG_ORVAR( EL, NBIG, PP, VEC, MATRIX, STATUS )
```

Arguments:

EL = INTEGER (Given)
Number of members in ordered set.

NBIG = INTEGER (Given)
Maximum number of entries in covariance array row. equal to EL∗(EL+1)/2).

PP(EL) = DOUBLE PRECISION (Given)
Workspace for storing expected values of order statistics.

VEC(NBIG, EL) = DOUBLE PRECISION (Returned)
The upper triangles of the nset by nset variance-covariance matrix packed by columns. Each triangle is packed into a single row. For each row element Vij is stored in VEC(i+j∗(j-1)/2), for 1 <= i <= j <= nset.

MATRIX(EL, EL) = DOUBLE PRECISION (Returned)
Work space.

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

Notes:

- Data are returned as above to save on repeated calls (which are too slow). To obtain the actual variance of the data of order n you need to sum all the variances and twice the covariances and use these to modify the actual variance of the (unordered) data.

CCG_RMS1x
Combines data lines using the root mean square

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in each lines are then combined to form the roor mean square of the values. The output rot mean squares are returned in the array RESULT. The squared standard error of the variance is returned in RESVAR.

Invocation:

```
CALL CCG_RMS1x( NPIX, NLINES, STACK, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- This routine performs its work in double precision.
- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.

CCG_RMS3x
Combines data lines using the root mean square

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in each lines are then combined to form the roor mean square of the values. The output rot mean squares are returned in the array RESULT.

Invocation:

```
CALL CCG_RMS3x( NPIX, NLINES, STACK, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- This routine performs its work in double precision.
- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK and RESULT arguments supplied to the routine must have the data type specified.

CCG_SCR1
Combines data lines using a sigma-clipped mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The weighted mean and standard deviation of each input column in STACK is then used to estimate the range of values which represent the required sigma clipping. Values outside of this range are then rejected and the resulting output mean values are returned in the array RESULT. The variances are propagated through the combination process and returned in the RESVAR array, and include a covariance correction.

Invocation:

```
CALL CCG_SCR1( NSIGMA, NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR,
WRK1, WRK2, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output population variances.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_SC3x
Combines data lines using a sigma-clipped mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The weighted mean and standard deviation of each input column in STACK is then used to estimate the range of values which represent the required sigma clipping. The standard deviation is derived from the population of values at each position along the lines (cf. each image pixel). Values outside of this range are then rejected and the resulting output mean values are returned in the array RESULT.

Note that clipping will not be used when only two or three values are available (unless in the case of three values NSIGMA is less than 1.0).

Invocation:

```
CALL CCG_SC3x( NSIGMA, NPIX, NLINES, STACK, VARS, MINPIX, RESULT, WRK1, WRK2, POINT,
USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data. These are used as inverse weights when forming the mean and do not need to be real variances, as they are not propagated.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_SD1x
Combines data lines using an unweighted standard deviation

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in each lines are then combined to form an unweighted standard deviation. The output standard deviations are returned in the array RESULT. The squared standard error of the variance is returned in RESVAR.

Invocation:

```
CALL CCG_SD1x( NPIX, NLINES, STACK, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances. to the output line.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.
- This routine performs its work in double precision.

CCG_SD3x
Combines data lines using an unweighted standard deviation

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in each lines are then combined to form an unweighted standard deviation. The output standard deviations are returned in the array RESULT.

Invocation:

```
CALL CCG_SD3x( NPIX, NLINES, STACK, MINPIX, RESULT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data. to the output line.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- This routine performs its work in double precision.
- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK and RESULT arguments supplied to the routine must have the data type specified.

CCG_SM1x
Combines data lines using a sigma-clipped median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The weighted mean and standard deviation of each input column in STACK is then used to estimate the range of values which represent the required sigma clipping. Values outside of this range are then rejected and then the median of the remaining values is returned in the array RESULT. The variances are propagated through the combination processand returned in the RESVAR array.

Invocation:

```
CALL CCG_SM1x( NSIGMA, NPIX, NLINES, STACK, VARS, MINPIX, COVEC, NMAT, RESULT, RESVAR,
WRK1, WRK2, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Given and Returned)
The actual number of contributing pixels.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_SM3x
Combines data lines using a sigma clipped median

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The weighted mean and standard deviation of each input column in STACK is then used to estimate the range of values which represent the required sigma clipping. The standard deviation is derived from the population of values at each position along the lines (cf. each image pixel). Values outside of this range are then rejected and the resulting output mean values are returned in the array RESULT.

Note that clipping will not be used when only two or three values are available (unless in the case of 3 values NSIGMA is less than 1.0).

Invocation:

```
CALL CCG_SM3x( NSIGMA, NPIX, NLINES, STACK, VARS, MINPIX, RESULT, WRK1, WRK2, POINT,
USED, NCON, NBAD, STATUS )
```

Arguments:

NSIGMA = REAL (Given)
The number of sigma at which to reject data values.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data. These are used as inverse weights when forming the mean and do not need to be real variances, as they are not propagated.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_SUM1x
Combines data lines using the sum of values

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. The data values in the lines are then summed. The output sums are returned in the array RESULT. The output variances are estimated from the VARS values and are returned in RESVAR.

Invocation:

```
CALL CCG_SUM1x( NPIX, NLINES, STACK, VARS, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output RESULT array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK, VARS, RESULT, and RESVAR arguments supplied to the routine must have the data type specified. It is not wise to choose an integer type as the chances of overflow are high.

CCG_SUM3x
Combines data lines using the sum of values

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. The data values in the lines are then summed. The output sums are returned in the array RESULT.

Invocation:

```
CALL CCG_SUM3x( NPIX, NLINES, STACK, MINPIX, RESULT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The STACK and RESULT arguments supplied to the routine must have the data type specified. It is not wise to choose an integer type as the chances of overflow are high.

CCG_TC1x
Combines data lines using a threshold-clipped mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. All values outside of the range VMIN to VMAX are rejected before a estimate of the (weighted) mean is made. The output mean values are returned in the array RESULT. The variances are propagation through the combination process and returned in the RESVAR array.

Invocation:

```
CALL CCG_TC1x( VMIN, VMAX, NPIX, NLINES, STACK, VARS, MINPIX, COVEC, NMAT, RESULT, RESVAR,
WRK1, WRK2, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

VMIN = ? (Given)
Minimum allowed value.

VMAX = ? (Given)
Maximum allowed value.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCD1_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variance.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The VMIN, VMAX, STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_TC3x
Combines data lines using a threshold-clipped mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. All values outside of the range VMIN to VMAX are rejected before a estimate of the (weighted) mean is made. The output mean values are returned in the array RESULT.

Invocation:

```
CALL CCG_TC3x( VMIN, VMAX, NPIX, NLINES, STACK, VARS, MINPIX, RESULT, WRK1, WRK2, POINT,
USED, NCON, NBAD, STATUS )
```

Arguments:

VMIN = ? (Given)
Minimum allowed value.

VMAX = ? (Given)
Maximum allowed value.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NLINES) = ? (Given)
The variance to to used for each line of data.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

WRK2(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The VMIN, VMAX, STACK, VARS, RESULT, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_TM1x
Combines data lines using a trimmed mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an alpha trimmed mean line. The output means are returned in the array RESULT. The variances are propagated through the combination process and returned in the RESVAR array.

Invocation:

```
CALL CCG_TM1x( ALPHA, NPIX, NLINES, STACK, VARS, MINPIX, NMAT, COVEC, RESULT, RESVAR,
WRK1, POINT, USED, NCON, NBAD, STATUS )
```

Arguments:

ALPHA = REAL (Given)
The fraction of data to trim from upper and lower orders. It must be greater than 0.0 and less than 0.5.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

NMAT = INTEGER (Given)
Size of the first dimension of COVEC.

COVEC(NMAT, NLINES) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of sizes up to NLINES, produced by CCG_ORVAR.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variance.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, RESVAR, WRK1, and WRK2 arguments supplied to the routine must have the data type specified.

CCG_TM3x
Combines data lines using a trimmed mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an alpha trimmed mean line. The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_TM3x( ALPHA, NPIX, NLINES, STACK, MINPIX, RESULT, WRK1, POINT, USED, NCON,
NBAD, STATUS )
```

Arguments:

ALPHA = REAL (Given)
The fraction of data to trim from upper and lower orders. It must be greater than 0.0 and less 0.5.

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data.

WRK1(NLINES) = ? (Returned)
Workspace for calculations.

POINT(NLINES) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion in to the WRK1 array.

USED(NLINES) = LOGICAL (Returned)
Workspace used to indicate which values have been used in estimating a resultant value.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels.

NBAD = INTEGER (Returned)
The number of bad values in the output array.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, RESULT, and WRK1 arguments supplied to the routine must have the data type specified.

CCG_TMN2x
Forms the n-trimmed mean of the given set of ordered data

Description:

The routine forms the trimmed mean of the given dataset. The IGNORE upper and lower ordered values are removed from consideration. Then the remaining values are added and averaged. The SVAR value is a (given) best estimate of the original un-ordered population variance. The variance of the output value is formed assuming that the original dataset was normal in distribution, and is now fairly represented by the ordered statistics variances-covariances supplied in packed form in COVAR. The values not rejected are indicated by setting the flags in array used.

Invocation:

```
CALL CCG_TMN2x( IGNORE, ORDDAT, ORDVAR, EL, USED, COVAR, TMEAN, TVAR, STATUS )
```

Arguments:

IGNORE = INTEGER (Given)
The number of values to ignore from the upper and lower orders.

EL = INTEGER (Given)
The number of entries in ORDDAT.

ORDDAT(EL) = ? (Given)
The set of ordered data for which a trimmed mean is required.

SVAR = DOUBLE PRECISION (Given)
The variance of the unordered sample now ordered in ORDDAT.

USED(EL) = LOGICAL (Returned)
Flags showing which values have not been rejected.

COVAR($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size EL.

TMEAN = DOUBLE PRECISION (Returned)
The trimmed mean.

TVAR = DOUBLE PRECISION (Returned)
A variance estimate of the trimmed mean.

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

Notes:

- There is a routine for real and double-precision: replace "x" in the routine name by D or R as appropriate. The ORDDAT argument supplied to the routine must have the data type specified.

Prior Requirements :

- The variance-covariance array must have been generated in a fashion similar to that of ORDVAR.

CCG_TMN3x
Forms the n-trimmed mean of the given set of ordered data. This variant does not process variances

Description:

The routine forms the trimmed mean of the given dataset. The IGNORE upper and lower ordered values are removed from consideration. Then the remaining values are added and averaged. The values not rejected are indicated by setting the flags in array used.

Invocation:

```
CALL CCG1_TMN3x( IGNORE, EL, ORDDAT, USED, TMEAN, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of entries in ORDDAT.

IGNORE = INTEGER (Given)
The number of values to ignore from the upper and lower orders.

ORDDAT(EL) = ? (Given)
The set of ordered data for which a trimmed mean is required.

USED(EL) = LOGICAL (Returned)
Flags showing which values have not been rejected.

TMEAN = DOUBLE PRECISION (Returned)
The trimmed mean.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The ORDDAT argument supplied to the routine must have the data type specified.

CCG_TRM2x
Forms the trimmed mean of the given set of ordered data, returning flags showing which values are used

Description:

The routine forms the trimmed mean of the given dataset. The alpha (as a fraction) upper and lower ordered values are removed from consideration. Then the remaining values are added and averaged. The SVAR value is a (given) best estimate of the original un-ordered population variance. The variance of the output value is formed assuming that the original dataset was normal in distribution, and is now fairly represented by the ordered statistics variances-covariances supplied in packed form in COVAR. The elements of the input array which actually contribute to the final value are flagged in the used array.

Invocation:

```
CALL CCG_TRM2x( ALPHA, EL, ORDDAT, ORDVAR, USED, COVAR, TMEAN, TVAR, STATUS )
```

Arguments:

ALPHA = REAL (Given)
The fraction of data to trim from upper and lower orders. (MUST BE GREATER THAN 0.0 AND LESS 0.5)

EL = INTEGER (Given)
The number of entries in ORDDAT.

ORDDAT(EL) = ? (Given)
The set of ordered data for which a trimmed mean is required.

SVAR = DOUBLE PRECISION (Given)
The variance of the unordered sample now ordered in ORDDAT.

USED(EL) = LOGICAL (Returned)
If the corresponding value in ORDDAT contributes to the final value then this is set true.

COVAR($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size EL.

TMEAN = DOUBLE PRECISION (Returned)
The trimmed mean.

TVAR = DOUBLE PRECISION (Returned)
A variance estimate of the trimmed mean.

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

Prior Requirements :

- The variance-covariance array must have been generated in a fashion similar to that of ORDVAR.

CCG_TRM3x
Forms the trimmed mean of the given set of ordered data, returning flags showing which values are used. This variant does not process variances

Description:

The routine forms the trimmed mean of the given dataset. The alpha (as a fraction) upper and lower ordered values are removed from consideration. Then the remaining values are added and averaged. The elements of the input array which actually contribute to the final value are flagged in the used array.

Invocation:

```
CALL CCG_TRM3x( ALPHA, EL, ORDDAT, USED, TMEAN, STATUS )
```

Arguments:

ALPHA = REAL (Given)
The fraction of data to trim from upper and lower orders. It must be greater than 0.0 and less than 0.5.

EL = INTEGER (Given)
The number of entries in ORDDAT.

ORDDAT(EL) = ? (Given)
The set of ordered data for which a trimmed mean is required.

USED(EL) = LOGICAL (Returned)
If the corresponding value in ORDDAT contributes to the final value then this is set true.

TMEAN = DOUBLE PRECISION (Returned)
The trimmed mean.

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

CCG_UM1x
Combines data lines using an unweighted mean

Description:

This routine accepts an array consisting of a series of (vectorised) lines of data. The data values in the lines are then combined to form an unweighted mean. The output means are returned in the array RESULT. The output variances are estimated from the VARS values and are returned in RESVAR.

Invocation:

```
CALL CCG_UM1x( NPIX, NLINES, STACK, VARS, MINPIX, RESULT, RESVAR, NCON, NBAD, STATUS
)
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

VARS(NPIX, NLINES) = ? (Given)
The data variances.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel or variance.

RESULT(NPIX) = ? (Returned)
The output line of data.

RESVAR(NPIX) = ? (Returned)
The output variances.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, RESULT, and RESVAR arguments supplied to the routine must have the data type specified.
- This routine performs its work in double precision.

CCG_UM3x
Combines data lines using an unweighted mean

Description:

This routine accepts an array consisting a series of (vectorised) lines of data. The data values in the lines are then combined to form an unweighted mean. The output means are returned in the array RESULT.

Invocation:

```
CALL CCG_UM3x( NPIX, NLINES, STACK, MINPIX, RESULT, NCON, NBAD, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

STACK(NPIX, NLINES) = ? (Given)
The array of lines which are to be combined into a single line.

MINPIX = INTEGER (Given)
The minimum number of pixels required to contribute to an output pixel.

RESULT(NPIX) = ? (Returned)
The output line of data. to the output line.

NCON(NLINES) = DOUBLE PRECISION (Returned)
The actual number of contributing pixels from each input line to the output line.

NBAD = INTEGER (Returned)
The number of bad values in the output array created while forming the statistics. It excludes those bad values whose corresponding values along the collapse axis are all bad.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The STACK, VARS, and RESULT arguments supplied to the routine must have the data type specified.
- This routine performs its work in double precision.

CCG_WCWIx
Creates a channel-width array from the channel co-ordinates

Description:

This routine accepts an array consisting of a series of (vectorised) lines of co-ordinates. For each pixel the subroutine forms a channel width by halving the difference of neighbouring co-ordinates along a line of data, i.e. it assumes that therre are no gaps. The widths for first and last pixels are the difference with its interior neighbour.

Invocation:

```
CALL CCG_WCWIx( NPIX, NLINES, COORDS, WIDTHS, STATUS )
```

Arguments:

NPIX = INTEGER (Given)
The number of pixels in a line of data.

NLINES = INTEGER (Given)
The number of lines of data in the stack.

COORDS(NPIX, NLINES) = ? (Given)
The world co-ordinates.

WIDTHS(NPIX, NLINES) = ? (Given)
The chanel widths.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D or R as appropriate. The COORDS and WIDTHS arguments supplied to the routine must have the data type specified.

CCG_WMD2x
Estimates the mean of a number of normally distributed data values, some of which may be corrupt

Description:

Not available. Remember to look in EDRS.

Invocation:

```
CALL CCG_WMD2x( EL, X, W, PBAD, NITER, TOLL, NSIGMA, XMODE, SIGMA, USED, NUSED, STATUS
)
```

Arguments:

EL = INTEGER (Given)
The number of data values.

X(EL) = ? (Given)
An array of data values.

W(EL) = ? (Given)
An array of data weights for each data value. The weights are inversely proportional to the square of the relative errors on each data point.

PBAD = REAL (Given)
An estimate of the probability that any one data point will be corrupt. (This value is not critical.)

NITER = INTEGER (Given)
The maximum number of iterations required.

TOLL = REAL (Given)
The absolute accuracy required in the estimate of the mean. Iterations cease when two successive estimates differ by less than this amount.

NSIGMA = REAL (Given)
The sigma level to reject data values at.

XMODE = DOUBLE PRECISION (Returned)
The estimate of the uncorrupted mean.

SIGMA = DOUBLE PRECISION (Returned)
An estimate of the uncorrupted normalised standard deviation of the data points. An estimate of the standard deviation of any one point is: SIGMA / SQRT(W) where W is its weight. It is returned as the bad value should there be only one or no valid data values.

USED(EL) = LOGICAL (Returned)
If a value is not rejected then its corresponding used element will be set true.

NUSED = INTEGER (Returned)
Number of the input data values which are actually used.

STATUS = INTEGER (Returned)
The global status.

Notes:

- There is a routine for real and double-precision: replace "_x" in the routine name by D or R as appropriate. The X and W arguments supplied to the routine must have the data type specified.

CCG_WMD3x
Estimates the mean of a number of normally distributed data values, some of which may be corrupt

Description:

This routine is based on maximising the likelihood function for a statistical model in which any of the data points has a constant probability of being corrupt. A weighted mean is formed with weights chosen according to the deviation of each data point from the current estimate of the mean. The weights are derived from the relative probability of being invalid or corrupt. A sequence of these iterations converges to a stationary point in the likelihood function. The routine approximates to a k-sigma clipping algorithm for a large number of points and to a mode estimating algorithm for fewer data points. Different weighting for each data point are allowed to accomodate known different intrinsic errors in the input data.

The variance of the input population is determined from the whole population and a new variance is computed, after the rejection passes, using the order statistics of a trimmed sample with the derived weights and initial numbers. Thus the input data should be ordered (either increasing or decreasing) so that means which are outliers (due to unstabilities from overly small sigma clipping) can have their variance properly estimated.

Invocation:

```
CALL CCG_WMD3x( EL, ORDDAT, WEIGHT, PBAD, NITER, TOLL, NSIGMA, COVEC, XMODE, FVAR, USED,
NUSED, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of data values.

ORDDAT(EL) = REAL (Given)
An ordered (increasing or decreasing) array of data values.

WEIGHT(EL) = REAL (Given)
An array of data weights for each data value. The weights should be the inverse variance of each data point. They are used to directly estimate the input population variance.

PBAD = REAL (Given)
An estimate of the probability that any one data point will be corrupt. (This value is not critical.)

NITER = INTEGER (Given)
The maximum number of iterations required.

TOLL = REAL (Given)
The absolute accuracy required in the estimate of the mean. Iterations cease when two successive estimates differ by less than this amount.

NSIGMA = REAL (Given)
The sigma level to reject data values at.

COVEC($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size EL, produced by CCG_ORVAR.

XMODE = DOUBLE PRECISION (Returned)
The estimate of the uncorrupted mean.

FVAR = DOUBLE PRECISION (Returned)
An estimate of the uncorrupted variance of the data points.

USED(EL) = LOGICAL (Returned)
If a value is not rejected then its corresponding used element will be set true.

NUSED = INTEGER (Returned)
Number of the input data values which are actually used when forming the estimate of the mean. This value will be zero or less if all values are rejected.

STATUS = INTEGER (Returned)
The global status.

Notes:

- There is a routine for real and double-precision: replace "`x`" in the routine name by D or R as appropriate. The X and W arguments supplied to the routine must have the data type specified.
- The input data must be sorted. The output variances are only accurate if the input data values have a normal distribution.

CCG_WTM2x
Forms the weighted median of a list of ordered data values. Incrementing the contributing pixel buffers

Description:

This routine finds a value which can be associated with the half- weight value. It sums all weights then finds a value for the half-weight. The comparison with the half-weight value proceeds in halves of the weights for each data point (half of the first weight, then the second half of the first weight and the first half of the second weight etc.) until the half weight is exceeded. The data values around this half weight position are then found and a linear interpolation of these values is the wtdmdn. The values which contribute to the result are flagged and passed through the USED array.

Invocation:

```
CALL CCG_WTM2x( EL. ORDDAT, WEIGHT, USED, RESULT, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of entries in the data array.

ORDDAT(EL) = ? (Given)
The list of ordered data for which the weighted median is required

WEIGHT(EL) = ? (Given)
The weights of the values.

USED = LOGICAL (Returned)
If a value contributes to the median value it is flagged as true in this array, otherwise the array is set to false.

RESULT = DOUBLE PRECISION (Returned)
The weighted median

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The ORDDAT and WEIGHT arguments supplied to the routine must have the data type specified.

Prior Requirements :

- The input data must be ordered increasing. No BAD values may be present.

CCG_WTM3x
Forms the weighted median of a list of ordered data values. Incrementing the contributing pixel buffers and estimating the variance change

Description:

This routine finds a value which can be associated with the half- weight value. It sums all weights then finds a value for the half-weight. The comparison with the half-weight value proceeds in halves of the weights for each data point (half of the first weight, then the second half of the first weight and the first half of the second weight etc.) until the half weight is exceeded. The data values around this half weight position are then found and a linear interpolation of these values is the weighted median. The values which contribute to the result are flagged and passed through the USED array. This routine also uses the order statistic covariance array (for a population EL big) to estimate the change in the variance from a optimal measurement from the given population, returning the adjusted variance.

Invocation:

```
CALL CCG_WTM3x( CALCMV, EL, ORDDAT, WEIGHT, VAR, COVAR, USED, RESULT, RESVAR, STATUS
)
```

Arguments:

CALCMV = LOGICAL (Given)
If .FALSE. then the output variances are estimated by scaling the variance on the weighted mean (rather than the weighted median) by pi/2. Otherwise, the output variances are calculated using the COVEC array.

EL = INTEGER (Given)
The number of entries in the data array.

ARR(EL) = ? (Given)
The list of ordered data for which the weighted median is required

WEIGHT(EL) = ? (Given)
The weights of the values.

VAR = DOUBLE PRECISION (Given)
The variance of the unordered sample now ordered in ARR.

COVAR($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size EL. Not used if CALCMV is .FALSE.

USED(EL) = LOGICAL (Returned)
If a value contributes to the median value it is flagged as true in this array, otherwise the array is set to false.

RESULT = DOUBLE PRECISION (Returned)
The weighted median

RESVAR = DOUBLE PRECISION (Returned)
The variance of result.

NBAD = INTEGER (Returned)
The number of bad values in the output array.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The ORDDAT and WEIGHT arguments supplied to the routine must have the data type specified.

Prior Requirements :

- The input data must be ordered increasing. No BAD values may be present.

CCG_WTM4x
Forms the weighted median of a list of ordered data values. Incrementing the contributing pixel buffers and estimating the variance change

Description:

This routine finds a value which can be associated with the half- weight value. It sums all weights then finds a value for the half-weight. The comparison with the half-weight value proceeds in halves of the weights for each data point (half of the first weight, then the second half of the first weight and the first half of the second weight etc.) until the half weight is exceeded. The data values around this half weight position are then found and a linear interpolation of these values is the weighted median. The values which contribute to the result are flagged and passed through the USED array. This routine also uses the order statistic covariance array (for a population EL big) to estimate the change in the variance from a optimal measurement from the given population, returning the adjusted variance.

This version is specialised to accept bounds for the values that can be used in ARR, these are from ARR(LBND:UBND). It is designed for use when some outliers of the population have already been flagged for removal (say by some clipping algorithm and the variance still needs to be determined from the original population size).

Invocation:

```
CALL CCG_WTM4x( EL, ORDDAT, WEIGHT, VAR, LBND, UBND, COVAR, USED, RESULT, RESVAR, STATUS
)
```

Arguments:

EL = INTEGER (Given)
The number of entries in the data array.

ARR(EL) = ? (Given)
The list of ordered data for which the weighted median is required

WEIGHT(EL) = ? (Given)
The weights of the values.

VAR = DOUBLE PRECISION (Given)
The variance of the unordered sample now ordered in ARR.

LBND = INTEGER (Given)
Lower bound of data to be considered in ARR.

UBND = INTEGER (Given)
Upper bound of data to be considered in ARR.

COVAR($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size EL.

USED(EL) = LOGICAL (Returned)
If a value contributes to the median value it is flagged as true in this array, otherwise the array element is set to false.

RESULT = DOUBLE PRECISION (Returned)
The weighted median

RESVAR = DOUBLE PRECISION (Returned)
The variance of result.

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

Notes:

- There is a routine for real and double-precision: replace " x" in the routine name by D or R as appropriate. The ORDDAT and WEIGHT arguments supplied to the routine must have the data type specified.

Prior Requirements :

- The input data must be ordered increasing. No BAD values may be present, although you may restrict the range (LBND, UBND).

FTS1_ASTWN
Displays any AST warning messages stored in the supplied FitsChan

Description:

The AST library can store warning messages in a FItsChan in the form of header cards with the keyword " ASTWARN" . This routine searches for such cards and displays them nicely if any are found.

Invocation:

```
CALL FTS1_ASTWN( FC, INDF, STATUS )
```

Arguments:

FC = INTEGER (Given)
The AST pointer to the FitsChan.

INDF = INTEGER (Given)
The NDF identifier for the NDF being created.

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

FTS1_AXIS
Creates an axis structure within an NDF from FITS header information

Description:

The routine searches the FITSD header for the keywords that describe the axis structure. If at least one reference value, CRVALn, exists then an axis component is created and filled with the appropriate values. CDELTn defines the step between axis values. If it is not present in the header it is set to 1. CRPIXn defines the reference pixel to which the reference value corresponds. If is absent form the header pixel 1 is assumed to be the reference pixel. If CTYPEn is in the header it is used to assigned a value to the nth axis's label component.

The precision of the output axis-centre array depends on the absolute relative size of the offset to the scale. Single precision is used if this ratio is greater than one hundred times the floating-point precision.

Invocation:

```
CALL FTS1_AXIS( NCARD, HEADER, SCARD, NDF, STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the buffer.

HEADER(NCARD) = CHARACTER * 80 (Given)
The FITS header 'cards', each element corresponding to a 80-character card.

SCARD = INTEGER (Given)
The number of the card from where the search will begin. This is needed because the headers make contain a dummy header prior to an extension.

NDF = INTEGER (Given)
The identifier for the NDF to contain the axis structure.

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

FTS1_BLCAR
Determines whether or not the first card in a FITS record has a blank keyword

Description:

This routine is needed for UNIX portability, since the calling application only knows the pointer to the FITS record, and not the actual values. Hence a function is required.

Invocation:

```
RESULT = FTS1_BLCAR( RECORD )
```

Arguments:

RECORD(36) = CHARACTER * (80) (Given)

The FITS record to be tested.

Returned Value:

FTS1_BLCAR = LOGICAL

If true, the first card image in RECORD has a blank keyword, i.e. characters 1 to 8 in the card image are blank.

FTS1_BLVAL
Blanks out the value from a FITS-header card

Description:

This routine modifies a FITS-header card by replacing the value with blank characters, leaving the keyword, value indicator and any comment in situ. If the header has no value, the header is returned unchanged.

Invocation:

```
CALL FTS1_BLVAL( HEADER, STATUS )
```

Arguments:

HEADER = CHARACTER * (80) (Given & Returned)
The FITS-header card.

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

FTS1_BSWAP
Swaps adjacent bytes in an array of bytes

Description:

This swaps adjacent pairs of bytes in an array in situ. This is VAX specific.

Invocation:

```
CALL FTS1_BSWAP( NBYTE, BYTES, STATUS )
```

Arguments:

NBYTE = INTEGER (Returned)

Number of bytes. An SAI__ERROR will be returned if this is not an even number.

BYTES(NBYTE) = BYTE (Given and Returned)

The 16-bit words whose bytes are to be swapped.

STATUS = INTEGER (Given and Returned)

The global status.

FTS1_CHVAx
Replaces all occurrences of a value in an array with another value

Description:

This routine copies the input array to the output array, except where a specified value occurs, and this is replaced with a new value throughout the output array.

Invocation:

```
CALL FTS1_CHVAx( EL, INARR, OLDVAL, NEWVAL, OUTARR, NREP, STATUS )
```

Arguments:

EL = INTEGER (Given)
The dimension of the input and output arrays.

INARR(EL) = ? (Given)
The input array.

OLDVAL = ? (Given)
Value to be replaced.

NEWVAL = ? (Given)
New value to be substituted for the old value.

OUTARR(EL) = ? (Returned)
The output array containing the modified values.

NREP = INTEGER (Returned)
The number of replacements made.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays and values supplied to the routine must have the data type specified.

FTS1_COMNT
Gets the value of a FITS COMMENT card from a buffer of cards

Description:

This routine searches a buffer containing the header card images from a FITS file for the next card with keyword COMMENT. The search begins at a defined card image; and ends when the next end of a header block, marked by the END keyword, is encountered or the buffer is exhausted. The routine returns the comment string, and the number of the card image within the buffer array that contains the comment. If the keyword is is present %THERE is true, otherwise it is false.

Invocation:

```
CALL FTS1_COMNT( NCARD, BUFFER, STCARD, THERE, VALUE, CARD, :  STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the buffer.

BUFFER(NCARD) = CHARACTER * (*) (Given)
The buffer containing the header card images.

STCARD = INTEGER (Given)
The number of the card from which to search for the next comment card

THERE = LOGICAL (Returned)
If true the parameter %NAME is present.

VALUE = CHARACTER * (*) (Returned)
The comment string of the first COMMENT keyword found at or after %STCARD. The length should be at least 72 characters.

CARD = INTEGER(Returned)
The number of the card containing the first COMMENT card. If no COMMENT card could be found this is returned with a value of zero.

STATUS = INTEGER (Given)
Global status value.

FTS1_CRNDF
Creates an NDF for a FITS data array, generating the NDF's name in some circumstances

Description:

This is a server routine for FITSIN. It packages up the operations required to create an output NDF. In automatic mode the filename is generated and put into the parameter to prevent prompting. In manual mode the user is prompted for the file name of the NDF.

Invocation:

```
CALL FTS1_CRNDF( PNNDF, FILROO, GCOUNT, NG, AUTO, FORMAT, NDIM, :  DIMS, NDF, FILNAM,
ASSOC, STATUS )
```

Arguments:

PNNDF = CHARACTER * (*) (Given)
The name of the parameter by which the filename of the output NDF will be obtained.

FILROO = CHARACTER * (*) (Given)
The rootname of the output NDF. The suffix Gn, where n=%NG, is appended in group-format mode to generate the filename. Otherwise in automatic mode the NDF filename is the rootname.

GCOUNT = INTEGER (Given)
Number of data arrays in the FITS sub-file. It should be one if there are no groups.

NG = INTEGER (Given)
Number of the group. It should be one if there are no groups.

AUTO = LOGICAL (Given)
If true the processing should be done in automatic mode, where the user is not prompted for file names, since these are generated from the prefix, file and sub-file numbers.

FORMAT = CHARACTER * (*) (Given)
The destination HDS format of the data array in the output file.

NDIM = INTEGER (Given)
Dimensionality of the NDF.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the NDF.

NDF = INTEGER (Returned)
The identifier of the created NDF.

FILNAM = CHARACTER * (*) (Returned)
The filename of the output NDF.

ASSOC = LOGICAL (Returned)
If false the NDF name was generated automatically.

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

FTS1_DREAD
Obtains a FITS record from a disk file

Description:

This routine reads the byte stream from the FITS disk file and extracts a FITS record of 2880 bytes. The blocksize of the disk file is arbitrary save that it be no more than the maximum FITS blocksize of 28800.

Invocation:

```
CALL FTS1_DREAD ( LU, BLKSIZ, ACTSIZ, LINIT, BUFFER, OFFSET, :  RECORD, STATUS )
```

Arguments:

LU = INTEGER (Given)
The logical unit number for the disk-FITS file.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the file block.

ACTSIZ = INTEGER (Given and Returned)
The actual block size. This need not be a multiple of the FITS record length of 2880 bytes. It must be known on input.

LINIT = LOGICAL (Given)
If true, the current record counter (RECNUM) is reset to one. Should be true if a file is being read for the first time.

BUFFER(ACTSIZ) = BYTE (Given and Returned)
The buffer containing the block of data. This is only read when %OFFSET does not equal %ACTSIZ, i.e. there are some non-header data within it.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted.

RECORD(2880) = BYTE (Returned)
The current FITS record. Successive calls will read of the FITS records in sequence.

STATUS = INTEGER(Given and Returned)
Global status value.

Prior requirements :

- The disk file should already be open and the first block read.

FTS1_DTYPE
Obtains the input data type, scales and offsets for a FITS file

Description:

This is a server routine for FITSIN. It packages up the operations required to define the input data types, the scale factor and offset, the data-blank value, the number of bytes per data values. This includes the group count, number of parameters per group and their group scalings for a non-standard data array.

Invocation:

```
CALL FTS1_DTYPE( DARRAY, NONSDA, BITPIX, SCARD, NCARD, HEADER, :  MXPARM, NDIM, DIMS,
BSCALE, BZERO, BLANK, BADPIX, :  IEEE, GCOUNT, PCOUNT, PTYPE, PSCALE, PZERO, :  STATUS
)
```

Arguments:

DARRAY = LOGICAL (Given)
If true there is a data array present if the FITS file.

NONSDA = LOGICAL (Given)
If true the data array is not standard, i.e. in group format. It is ignored if %DARRAY is false.

BITPIX = INTEGER (Given)
The value of the BITPIX keyword in the FITS header, i.e. the number of bits per data value. If it is negative this indicates an IEEE-format file.

SCARD = INTEGER (Given)
The number of the card from where the searches of the header will begin. This is needed because the headers make contain a dummy header prior to an extension.

NCARD = INTEGER (Given)
The number of card images in the header.

HEADER(NCARD) = CHARACTER * 80 (Given)
The FITS headers in 80-character records.

MXPARM = INTEGER (Given)
The maximum number of group parameters, and the dimension size for the various group arrays.

NDIM = INTEGER (Given and Returned)
Dimensionality of the data array. It may be modified if a) there is no data array, because a valid NDF must have a data array with physical dimensions (set to 1); or b) it is a non-standard array (i.e. groups) where the first dimension is zero, and so the dimensionality is reduced by one.

DIMS(DAT__MXDIM) = INTEGER (Given and Returned)
The dimensions of the table. It may be modified if a) there is no data array, because a valid NDF must have a data array with physical dimensions (set to 2 in 1-d); or b) it is a non-standard array (i.e. groups) where the first dimension is zero, and so the dimension sizes are shifted down one dimension.

BSCALE = REAL (Returned)
The scale factor of the FITS integer data for their conversion to the true floating-point values.

BZERO = REAL (Returned)
The offset of the FITS integer data for their conversion to the true floating-point values.

BLANK = INTEGER (Returned)
The data-blank value equivalent to the bad-pixel flag. It should be ignored if %BADPIX is false.

BADPIX = LOGICAL (Returned)
If true the data-blank was defined in the FITS header.

IEEE = LOGICAL (Returned)
If true the FITS data are in IEEE floating-point format.

GCOUNT = INTEGER (Returned)
The number of groups in the FITS sub-file.

PCOUNT = INTEGER (Returned)
The number of group parameters in each group.

PTYPE(MXPARM) = CHARACTER * (*) (Returned)
The type (descriptive name) of each group parameter.

PSCALE(MXPARM) = DOUBLE PRECISION (Returned)
The scale factors of the group parameters so that the parameters may be converted to the true floating-point values.

PZERO(MXPARM) = DOUBLE PRECISION (Returned)
The offsets of the group parameters so that the parameters may be converted to the true floating-point values.

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

Notes:

The non-standard case of a floating-point data-blank value (BLANK) is handled assuming it is the true blank value as opposed to the blank value in the FITS data. This is achieved by subtracting the offset and dividing by the scale factor and taking the nearest integer.

FTS1_EDFEX
Edits non-reserved keyword cards in a FITS extension of an NDF

Description:

This subroutine edits a number of non-reserved keyword cards in the FITS extension of an NDF file. This subroutine inserts, updates, moves, deletes, reports, and tests the existence of a number of keyword cards in the FITS extension of an NDF. The occurrence of keywords may be defined, when there are more than one cards of the same name. The location of each insertion or move is immediately before some occurrence of a corresponding keyword.

This routine itself merely deals with accessing and enlarging the FITS extension, obtaining worksapce, and being able to pass the mapped character arrays in the desired order to routine FTS1_EDKEY via a dummy routine. FTS1_EDKEY actually performs the editing; look there for more details of the editing functions and rules.

Invocation:

```
CALL FTS1_EDFEX( NKEY, EDITS, NAMES, PSTNS, KOCCUR, POCCUR, VALUES, COMNTS, TYPES, FTSLOC,
THERE, STATUS )
```

Arguments:

NKEY = INTEGER (Given)

The number of keyword cards to be inserted into the FITS extension.

EDITS(NKEY) = CHARACTER * (*) (Given)

The editing commands. These need only be one character per element. Allowed values are 'Amend', 'Delete', 'Exist', 'Move', 'Null' ', 'Rename', 'Print', 'Update', and 'Write', which can be abbreviated to the initial letter.

'Amend' acts as 'Update' if the keyword exists, but as 'Write' if the keyword is absent.

'Delete' removes a named keyword.

'Exist' reports TRUE to standard output if the named keyword exists in the header, and FALSE if the keyword is not present.

'Move' relocates a named keyword to be immediately before a second keyword. When this positional keyword is not supplied, it defaults to the END card, and if the END card is absent, the new location is at the end of the headers.

'Null' nullifies the value of the named keyword. Spaces substitute the keyword's value.

'Print' causes the value of a named keyword to be displayed to standard output. This will be a blank for a comment card.

'Rename' renames a keyword, using the value as the new keyword.

'Update' revises the value and/or the comment. If a secondary keyword is defined explicitly, the card may be relocated at the same time. If the secondary keyword does not exist, the card being edited is not moved. Update requires that the keyword being edited exists.

'Write' creates a new card given a value and an optional comment. Its location uses the same rules as for the Move command.

NAMES(NKEY) = CHARACTER * (*) (Given)

The names of the keywords to be edited in the FITS card array. A name may be compound to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Each keyword must be no longer than 8 characters, and be a valid FITS keyword comprising alphanumeric characters, hyphen, and unsderscore. Any

lowercase letters are converted to uppercase and blanks are removed before inserted or comparison with the existing keywords.

The keywords ' ', 'COMMENT', and 'HISTORY' are comment cards and do not have a value.

The keyword must exist except for the Write and Exist commands.

PSTNS(NKEY) = CHARACTER * (*) (Given)
The position keyword names. A position name may be compound to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is twenty. Each keyword must be no longer than eight characters. When locating the position card, comparisons are made in uppercase and with the blanks removed.

The new keywords are inserted immediately before each corresponding position keyword. If any name in it does not exist in FITS array, the consequences will be as follows. In the Write, Amend (new keyword), and Move edits its corresponding keyword will be inserted just before the END card or appended to FITS array when the END card does not exist; however, the card is not relocated for an Update or Amend (with an existing keyword) edit. If two or more new cards have the same position name, they will all be put just before the position name in the same order as they are in NAMES.

A positional keyword is used by the Move, Write, Amend, and Update editing commands.

KOCCUR(NKEY) = INTEGER (Given)
The occurrences of the NAMES keywords to use. Values less than or equal to 1 will manipulate the first occurrence of the keyword to insert.

POCCUR(NKEY) = INTEGER (Given)
The occurrences of the PSTNS keywords to use. Values less than or equal to 1 will situate the inserted keyword immediately before the first occurrence of the positional keyword.

VALUES(NKEY) = CHARACTER * (*) (Given)
The new values of the NAMES keywords for the Update and Write editing commands. The special value '$V' means use the current value of the NAMES keyword. This makes it possible to modify a comment, leaving the value unaltered. In addition $V(keyword) requests that the value of the keyword given between the parentheses be assigned to the keyword being edited. This positional keyword must exist and have a value for a Write edit; whereas the FITS-header value is unchanged for Update if there are problems with this positional keyword.

For a Rename edit, VALUES has a different meaning; in this case it stores the replacement keyword name.

COMNTS(NKEY) = CHARACTER * (*) (Given)
The comments to be written to the NAMES keywords for the Update and Write editing commands. The special value '$C' means use the current comment. In addition $C(keyword) requests that the comment of the keyword given between the parentheses be assigned to the keyword being edited. If this positional keyword does not exist the comment is unchanged for Update, and is blank for a Write edit.

TYPES(NKEY) = CHARACTER * (*) (Given)
The data types of the values to Write or Update. This does allow some numeric or logical values to be written as strings. These will be one of the following: '_CHAR', '_DOUBLE', '_INTEGER', '_LOGICAL', '_REAL'. In addition there are two special values: 'COMMENT' to indicate that the card is a comment (so strictly it has no type), and ' ' to indicate that the data type is unknown, as occurs for a value defined by a reference keyword. The length should be at least 8 characters.

FTSLOC = CHARACTER * (*) (Given)
The locator to the FITS extension of the NDF.

THERE = LOGICAL (Returned)
Result of final " Exist" operation.

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

Prior Requirements :

Unless all edits are read-only, the The FITS extension is mapped for update access. It therefore must have some values assigned before using the routine.

FTS1_EDKEY
Edits keywords in a FITS card array

Description:

This subroutine inserts, updates, moves, deletes, reports, and tests the existence of a number of keyword cards in a FITS-header array. The occurrence of keywords may be defined, when there are more than one cards of the same name. The location of each insertion or move is immediately before some occurrence of a corresponding keyword. The routine returns the modified FITS-header array.

Invocation:

```
CALL FTS1_EDKEY( NOCARD, NKEY, MXCARD, EDITS, NAMES, PSTNS, KOCCUR, POCCUR, VALUES,
COMNTS, TYPES, FTSCAR, ACTNUM, IARY1, IARY2, CARY, EXISTS, STATUS )
```

Arguments:

NOCARD = INTEGER (Given)

Number of cards in the original FITS array.

NKEY = INTEGER (Given)

The number of keyword cards to be inserted into the FITS-header array.

MXCARD = INTEGER (Given)

Maximum number of cards in the resultant FITS array. This must be not less than NOCARD + NKEY + 1.

EDITS(MAXMOD) = CHARACTER * (*) (Returned)

The editing commands. Allowed values are 'Amend', 'Delete', 'Exist', 'Move', 'Rename', 'Print', 'Update', and 'Write'. Each element can be abbreviated to the initial letter.

'Amend' acts as 'Update' if the keyword exists, but as 'Write' if the keyword is absent.

'Delete' removes a named keyword.

'Exist' reports TRUE to standard output if the named keyword exists in the header, and FALSE if the keyword is not present.

'Move' relocates a named keyword to be immediately before a second keyword. When this positional keyword is not supplied, it defaults to the END card, and if the END card is absent, the new location is at the end of the headers.

'Null' nullifies the value of the named keyword. Spaces substitute the keyword's value.

'Print' causes the value of a named keyword to be displayed to standard output. This will be a blank for a comment card.

'Rename' renames a keyword, using the value as the new keyword.

'Update' revises the value and/or the comment. If a secondary keyword is defined explicitly, the card may be relocated at the same time. If the secondary keyword does not exist, the card being edited is not moved. Update requires that the keyword being edited exists.

'Write' creates a new card given a value and an optional comment. Its location uses the same rules as for the Move command.

NAMES(NKEY) = CHARACTER * (*) (Given)

The names of the keywords to be edited in the FITS card array. A name may be compound to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is twenty. Each keyword must be no longer than eight characters, and be a valid FITS keyword comprising alphanumeric characters, hyphen, and

underscore. Any lowercase letters are converted to uppercase and blanks are removed before inserted or comparison with the existing keywords.

The keywords ' ', 'COMMENT', and 'HISTORY' are comment cards and do not have a value.

The keyword must exist except for the Amend, Write, and Exist commands.

PSTNS(NKEY) = CHARACTER * (*) (Given)
The position keyword names. A position name may be compound to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is twenty. Each keyword must be no longer than eight characters. When locating the position card, comparisons are made in uppercase and with the blanks removed.

The new keywords are inserted immediately before each corresponding position keyword. If any name in it does not exist in FITS array, the consequences will be as follows. In the Write, Amend (new keyword), and Move edits its corresponding keyword will be inserted just before the END card or appended to FITS array when the END card does not exist; however, the card is not relocated for an Update or Amend (with an existing keyword) edit. If two or more new cards have the same position name, they will all be put just before the position name in the same order as they are in NAMES.

A positional keyword is used by the Move, Write, Amend, and Update editing commands.

KOCCUR(NKEY) = INTEGER (Given)
The occurrences of the NAMES keywords to use. Values less than or equal to 1 will manipulate the first occurrence of the keyword to insert.

POCCUR(NKEY) = INTEGER (Given)
The occurrences of the PSTNS keywords to use. Values less than or equal to 1 will situate the inserted keyword immediately before the first occurrence of the positional keyword.

VALUES(NKEY) = CHARACTER * (*) (Given)
The new values of the NAMES keywords for the Update and Write editing commands. The special value '$V' means use the current value of the NAMES keyword. This makes it possible to modify a comment, leaving the value unaltered. In addition $V(keyword) requests that the value of the keyword given between the parentheses be assigned to the keyword being edited. This positional keyword must exist and have a value for a Write edit; whereas the FITS-header value is unchanged for Update if there are problems with this positional keyword. edited.

For a Rename edit, VALUES has a different meaning; in this case it stores the replacement keyword name.

COMNTS(NKEY) = CHARACTER * (*) (Given)
The comments to be written to the NAMES keywords for the Amend, Update, and Write editing commands. The special value '$C' means use the current comment. In addition $C(keyword) requests that the comment of the keyword given between the parentheses be assigned to the keyword being edited. If this positional keyword does not exist the comment is unchanged for Update, and is blank for a Write edit.

TYPES(NKEY) = CHARACTER * (*) (Given)
The data types of the values to Write or Update. This does allow some numeric or logical values to be written as strings. These will be one of the following: '_CHAR', '_DOUBLE', '_INTEGER', '_LOGICAL', '_REAL'. In addition there are two special values: 'COMMENT' to indicate that the card is a comment (so strictly it has no type), and ' ' to indicate that the data type is unknown, as occurs for a value defined by a reference keyword. The length should be at least eight characters.

FTSCAR(MXCARD) = CHARACTER * (*) (Given and Returned)
On entry its first NOCARD elements hold the original FITS cards. On exit, its first ACTNUM elements hold the FITS cards after the insersion.

ACTNUM = INTEGER (Returned)
The actual number of cards in the FITS array after inserting.

IARY1(MXCARD) = INTEGER (Returned)
The first temporary working space.

IARY2(MXCARD) = INTEGER (Returned)
The second temporary working space.

CARY(MXCARD) = CHARACTER * (*) (Returned)
A temporary working space.

EXISTS = LOGICAL (Returned)
The result of the last " Exist" operation.

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

Notes:

- When an error occurs during editing, warning messages are sent at the normal reporting level, and processing continues to the next editing command.
- The FITS fixed format is used for writing or updating headers, except for double-precision values requiring more space. The comment is delineated from the value by the string ' / '.
- The comments in comment cards begin one space following the keyword or from column 10 whichever is greater.
- At present the following reserved keywords are neither modifiable nor movable: SIMPLE, BITPIX, NAXIS, NAXISn, EXTEND, PCOUNT, GCOUNT, XTENSION, BLOCKED, and END. This is because order in the extension should be fixed and should not be changed by any routine. There is one exception: an END keyword may be appended if one does not exist.

FTS1_EVKEY
Extracts a keyword and occurrence, and validates the keyword

Description:

This routine serves FTS1_RFMOD. It takes a string containing a keyword and an optional occurrence in brackets, and extracts the keyword and any occurrence. It validates the keyword or the hierarchical keywords. It also returns the length of the keyword.

Invocation:

```
CALL FTS1_EVKEY( STRING, KEYWRD, LENGTH, OCCUR, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string containing the keyword or hierarchical keyword in the form keyword1.keyword2.keyword3 etc. There may be a trailing [number] string which defines the occcurence of the keyword.

KEYWRD = CHARACTER * (*) (Returned)

The extracted keyword in uppercase. The supplied length is recommended to be at least 48 to allow for six full-length keywords.

LENGTH = INTEGER (Returned)

The length in characters of the keyword, so the keyword is KEYWRD(:LENGTH).

OCCUR = INTEGER (Returned)

The occurrence of the keyword to use. If none is supplied in the STRING or the value is not a positive integer, OCCUR is assigned the value 1.

STATUS = INTEGER (Given and Returned)

The global status.

FTS1_FNDFS
Reads the first FrameSet from the supplied FitsChan

Description:

This routine reads Objects from the supplied FitsChan until a FrameSet is obtained, and returns the FrameSet.

Invocation:

```
CALL FTS1_FNDFS( FC, OBJ, STATUS )
```

Arguments:

FC = INTEGER (Given)
An AST pointer to the FitsChan.

OBJ = INTEGER (Given)
The AST pointer to the FrameSet.

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

Notes:

- The FitsChan is not rewound before reading. The first read starts at the current Card in the FitsChan.
- No value is set for the FitsChan Encoding attribute.
- OBJ is returned equal to AST__NULL if no FrameSet can be read from the supplied FitsChan, or if an error occurs.

FTS1_FRMT
Obtains the input and output data format for a FITS file

Description:

This is a server routine for FITSIN. It packages up the operations required to define the input and output data formats (HDS types).

Invocation:

```
CALL FTS1_FRMT( BITPIX, IEEE, FMTCNV, BPV, FMTIN, FMTOUT, :  STATUS )
```

Arguments:

BITPIX = INTEGER (Given)
The value of the BITPIX keyword in the FITS header, i.e. the number of bits per data value. If it is negative this indicates an IEEE-format file.

IEEE = LOGICAL (Given)
If true the FITS data are in IEEE floating-point format.

FMTCNV = LOGICAL (Given)
If true, format conversion from the integer FITS data to the real output data array is required.

BPV = INTEGER (Returned)
The number of bytes per data value.

FMTIN = CHARACTER * (*) (Returned)
The HDS format of the data in the FITS file.

FMTOUT = CHARACTER * (*) (Returned)
The destination HDS format of the data array in the output file.

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

FTS1_FTWCS
Uses co-ordinate system information in the supplied FITS headers to create WCS and AXIS components in an NDF

Description:

This constructs an AST FrameSet from the supplied FITS headers and adds it into the existing WCS information in the supplied NDF. It can also create AXIS structures (see below).

The information needed to create the FrameSet can be stored several times in a single FITS header, using different keywords each time. Each of these descriptions is known as an " encoding" and AST supports several different encoding schemes (e.g. FITS-WCS, FITS-IRAF, DSS, NATIVE). If the supplied FITS header contains more than one encoding then we need to choose which one to use. This decision is important because is is possible for encodings to be inconsistent (i.e. software may modify one encoding without making equivalent modifications to the other encodings). The simplest way to make this decision is to hand responsibility for it over to the user. In this case, the user supplies a list of preferred encodings, and the first of these encodings that exists in the FITS header gets used. If the user does not know which encoding to use, then we can make an intelligent guess by comparing the encodings to see which ones are consistent and which ones are not.

In addition to the WCS component, this routine also creates AXIS Centre, Label and Units components in the NDF, but only if they do not already exist, and if the FrameSet read from the FITS header contains an AXIS Frame. NDF2FITS does not write out the AXIS Frame if it is equivalent to pixel co-ordinates, and so no AXIS structures will be created by this routine in this case. Also, if the AXIS Frame represents linear axis co-ordinates, then there will already be AXIS structures in the NDF (created earlier within FITSIN), and so again no AXIS structures will be created by this routine. Thus, this routine will only create AXIS structures in the cases where the axis co-ordinates are non-linear.

Invocation:

```
CALL FTS1_FTWCS( NCARD, HEADER, SCARD, INDF, NENCOD, ENCODS, STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of cards in the array of headers, from the start of the first header section to the end of the current one.

HEADER(NCARD) = CHARACTER * 80 (Given)
The buffer containing the header card images.

SCARD = INTEGER (Given)
The number of the card from where searches will begin, and copying of the headers to the FITS extension. Therefore NCARD - SCARD + 1 headers will appear in the extension. This argument is needed because the headers make contain a dummy header prior to an extension.

INDF = INTEGER (Given)
The NDF identifier.

NENCOD = INTEGER (Given)
The number of encodings supplied in ENCODS.

ENCODS(NENCOD) = CHARACTER * (*) (Given)
The user's preferred AST encodings. If NENCOD is zero, then this is ignored, and an intelligent guess is made as to which encoding to use (see FTS1_WCSIM).

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

FTS1_GKEYC
Gets the value and comment of a named header of type CHARACTER from a buffer of FITS-header card images

Description:

This routine searches a buffer containing the header card images from a FITS file for a keyword NAME; and returns its value (as a character string) and comment, and the number of the card image within the buffer array that contains the named keyword. The search ends when the next end of a header block, marked by the END keyword, is encountered or the buffer is exhausted. If the keyword is present THERE is true, otherwise it is false. If the keyword is expected to be present more than once then the argument NOCCUR controls which occurrence will be retrieved. If a keyword is not found then no error results and the argument VALUE remains unmodified.

The name may be compound to permit reading of hierarchical keywords (with a blank regulation keyword). This routine will also work for HISTORY, COMMENT and the ' ' (blank) keyword comment cards. Cards without an equals sign present are also regarded as comment cards. Comment cards have no returned value, only a comment.

Invocation:

```
CALL FTS1_GKEYC( NCARD, BUFFER, SCARD, NAME, NOCCUR, THERE, VALUE, COMENT, CARD, STATUS
)
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the buffer.

BUFFER(NCARD) = CHARACTER * 80 (Given)
The buffer containing the header card images.

SCARD = INTEGER (Given)
The number of the card from where the search will begin. This is needed because the headers may contain a dummy header prior to an extension.

NAME = CHARACTER * (*) (Given)
The name of the keyword whose value is required. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Comparisons are performed in uppercase and blanks are removed. Each keyword must be no longer than 8 characters.

NOCCUR = INTEGER (Given)
The value of this argument specifies which occurrence of a keyword should be used, if multiple ones are expected. Any value less than or equal to 1 indicates the first occurrence.

THERE = LOGICAL (Returned)
If .TRUE., the keyword given by argument NAME is present, regardless of the exit status.

VALUE = CHARACTER * (*) (Returned)
The value of the keyword. The string is truncated to the length of VALUE if the FITS value contains more characters than that.

COMENT = CHARACTER * (*) (Returned)
The comment associated with the keyword.

CARD = INTEGER (Returned)
The number of the card containing the named keyword. If the card could not be found this is set to zero.

STATUS = INTEGER (Given)
Global status value.

FTS1_GKEYx
Gets the value and comment of a named header from a buffer of FITS-header card images

Description:

This routine searches a buffer containing the header card images from a FITS file for a keyword NAME; and returns its value and comment, and the number of the card image within the buffer array that contains the named keyword. The search ends when the next end of a header block, marked by the END keyword, is encountered or the buffer is exhausted. If the keyword is present, THERE is .TRUE., otherwise it is .FALSE. Since all cards images are in character format, type conversion is performed. An error status will be returned if the conversion has failed. If the keyword expected to be present more than once, then the argument NOCCUR controls which occurrence will be retrieved. If a keyword is not found, then no error results and the argument VALUE remains unmodified.

The name may be compound to permit reading of hierarchical keywords. This routine will probably only work for HISTORY, COMMENT and ' ' (blank) if there is just one value given on the line, i.e. only one " keyword = value" before any comment marker. An error will result otherwise.

Invocation:

```
CALL FTS1_GKEYx( NCARD, BUFFER, SCARD, NAME, NOCCUR, THERE, VALUE, COMENT, CARD, STATUS
)
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the buffer.

BUFFER(NCARD) = CHARACTER * 80 (Given)
The buffer containing the header card images.

SCARD = INTEGER (Given)
The number of the card from where the search will begin. This is needed because the headers may contain a dummy header prior to an extension.

NAME = CHARACTER * (*) (Given)
The name of the keyword whose value is required. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Comparisons are performed in uppercase and blanks are removed. Each keyword must be no longer than 8 characters.

NOCCUR = INTEGER (Given)
The value of this argument specifies which occurrence of a keyword should be used, if multiple ones are expected. Any value less than or equal to 1 indicates the first occurrence.

THERE = LOGICAL (Returned)
If .TRUE., the keyword given by argument NAME is present, regardless of the exit status.

VALUE = ? (Returned)
The value of the keyword.

COMENT = CHARACTER * (*) (Returned)
The comment associated with the keyword.

CARD = INTEGER (Returned)
The number of the card containing the named keyword. If the card could not be found this is set to zero.

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for each of the data types logical, integer, 64-bit integer, real, and double precision: replace "`x`" in the routine name by L, I, K, R or D as appropriate.
- The comments are written from column 32 or higher if the value demands more than the customary 20 characters for the value. A comment may be omitted if the value is so long to leave no room.

FTS1_GPARM
Adds the group parameters to the FITS header records

Description:

This is a server routine for FITSIN. It packages up the operations required to add the group parameters to the HDS structure that contains the FITS header cards. It also writes the group parameters to a log file is required.

Each group parameter is evaluated in double precision from its scale and offset, and with the parameter name a pseudo FITS card is generated. In the header structure the last card—has the END keyword—is overwritten. Once all the group parameters have been copied into the header structure, an END card is placed after them to preserve a valid FITS header section.

Invocation:

```
CALL FTS1_GPARM( NCARD, HEADER, PCOUNT, PARAMS, PTYPE, PSCALE, :  PZERO, BAD, BLANK,
LOGHDR, FD, STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of header 80-character cards in the original FITS header. It excludes the number of group parameters.

HEADER(NCARD + PCOUNT) = CHARACTER * 80 (Given)
The FITS headers in 80-character cards. The aditional PCOUNT elements are for the group parameters.

PCOUNT = INTEGER (Given)
The number of group parameters in each group.

PARAMS(PCOUNT) = INTEGER (Given)
The values of the group parameters in the header, i.e. before any scale and offset have been applied.

PTYPE(PCOUNT) = CHARACTER * (*) (Given)
The type (descriptive name) of each group parameter.

PSCALE(PCOUNT) = DOUBLE PRECISION (Given)
The scale factors of the group parameters so that the parameters may be converted to the true floating-point values.

PZERO(PCOUNT) = DOUBLE PRECISION (Given)
The offsets of the group parameters so that the parameters may be converted to the true floating-point values.

BAD = LOGICAL (Given)
If true, testing and replacement of undefined parameters is to occur. A blank value, as specified by %BLANK, is replaced by the standard magic value. If false, the value of %BLANK is ignored.

BLANK = INTEGER (Given)
Value of an undefined parameter.

LOGHDR = LOGICAL (Given)
If true the evaluated group parameters written in FITS-card format will be written to the log file.

FD = INTEGER (Given)
The file descriptor for the log file. It is ignored if %LOGHDR is false.

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

FTS1_GVALC
Extracts a string value from a FITS header

Description:

This routines determines the location of a character value within a FITS header and hence the value itself. The location of the delimiting quote marks and string value are returned.

Invocation:

```
CALL FTS1_GVALC( HEADER, CSTART, CEND, VALUE, STATUS )
```

Arguments:

HEADER = CHARACTER * (*) (Given)
The FITS header 'card' whose value is to be extractd.

CSTART = INTEGER (Given)
The column containing the leading quote of the string value. This is set to zero if no string value could be found.

CEND = INTEGER (Given)
The column containing the trailing quote of the string value. This is set to zero if no string value could be found.

VALUE = CHARACTER * (*) (Returned)
The character value. CARD should contain at least 68 characters. A blabnnk string is returned if the header failed to contain a character value.

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

Notes:

- It is assumed that the header conforms to the FITS Standard, with one exception. Single quotes within a string—these should be doubled—are detected provided that the next solidus after a single quote will be a comment delimiter rather than literal text in the string.

FTS1_HDLOG
Outputs the FITS header cards to an ASCII file

Description:

This is a server routine for FITSIN. It packages up the operations required to write the FITS header cards (80-character) into an ASCII file. A heading is also written giving the file and sub-file numbers. The maximum record length of the ASCII file supported by this routine is 132 characters. The minimum is 80 characters—an error is reported if it is smaller than 80.

The file name appears in the caption before the headers. It may be truncated if it is longer than the file recordsize less 30 characters. Truncation occurs at the start of the name and is designated via an ellipsis.

Invocation:

```
CALL FTS1_HDLOG( HEADER, FD, CFN, SUBFIL, NHEADS, HDNUM, STATUS )
```

Arguments:

HEADER(*) = CHARACTER * 80 (Given)
The FITS headers in 80-character records.

FD = INTEGER (Given)
The file descriptor for the log file.

CFN = CHARACTER * (*) (Given)
The input filename or tape file number of the FITS file being processed.

SUBFIL = INTEGER (Given)
The number of the sub-file/extension within the current FITS file being processed.

NHEADS = INTEGER (Given)
The number of header sections in the sub-file. This includes dummy FITS header sections.

HDNUM(*) = INTEGER (Given)
The number of header cards within each header in the sub-file.

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

Prior Requirements :

- The ASCII file must be opened.

FTS1_I2VXD
Converts a vector of 64-bit IEEE floating-point numbers to Vax-D format

Description:

This is a dummy routine used to build the KAPPA FITS readers on UNIX.

Invocation:

```
CALL FTS1_I2VXD( BSWAP, EL, BUF, STATUS )
```

Arguments:

BSWAP = LOGICAL (Given)

Whether or not adjacent bytes are to be swapped. If and only if BSWAP is 1 will the bytes be swapped. Swapping must occur either prior or within this routine to obtain the correct Vax-D values. An expression must not be given for this argument. Bytes in the order 1 2 3 4 5 6 7 8 become 2 1 4 3 6 5 8 7 after swapping.

EL = INTEGER (Given)

The number of IEEE numbers to be converted.

BUF(EL) = DOUBLE PRECISION (Given and Returned)

On input the IEEE numbers to be converted. On return these are converted to Vax-D format.

STATUS = INTEGER (Given and Returned)

The global status.

References :

- " IEEE Standard for Binary Floating-Point Arithmetic" , ANSI/IEEE 754, 1985.
- " Floating Point Agreement for FITS" , D.C. Wells & P. Grosbol, 1990.

FTS1_I2VXR
Converts a vector of 32-bit IEEE floating-point numbers to Vax-F format

Description:

This is a dummy routine used to build the KAPPA FITS readers on UNIX.

Invocation:

```
CALL FTS1_I2VXR( BSWAP, WSWAP, EL, BUF, STATUS )
```

Arguments:

BSWAP = INTEGER (Given)

Whether or not byte swapping is to take place. If and only if BSWAP is 1 will the bytes be swapped. Swapping must occur either prior or within this routine to obtain the correct Vax-F values. An expression must not be given for this argument.

WSWAP = INTEGER (Given)

Whether or not word swapping is to take place. If and only if WSWAP is 1 will the words be swapped. Swapping must occur either prior or within this routine to obtain the correct Vax-F values. An expression must not be given for this argument.

EL = INTEGER (Given)

The number of IEEE numbers to be converted.

BUF(EL) = DOUBLE PRECISION (Given and Returned)

On input the IEEE numbers to be converted. On return these are converted to Vax-F format.

STATUS = INTEGER (Given and Returned)

The global status.

References :

- " IEEE Standard for Binary Floating-Point Arithmetic" , ANSI/IEEE 754, 1985.
- " Floating Point Agreement for FITS" , D.C. Wells & P. Grosbol, 1990.

FTS1_INKEY
Inserts keywords to a FITS card array

Description:

This subroutine inserts a number of keyword cards into a FITS-header array just before some given keywords, and returns the position of these newly inserted cards in the FITS array after insertion. If a given position keyword is not in the FITS header or is blank, its corresponding keyword card will be inserted just before the end card or appended to the present FITS array when end card does not exist. If a keyword card to be inserted already exists in the FITS array, that card will be deleted and moved to the specified position. For those new keyword cards, an equals sign '=' will be put at the 9th column for simple keyword or immediately after keywords for compound ones; and a character value '{undefined}' will be given to them.

The following reserved keywords are not modified: SIMPLE, BITPIX, NAXIS, NAXISn, EXTEND, PCOUNT, GCOUNT, XTENSION, BLOCKED, and END.

Invocation:

```
CALL FTS1_INKEY( NOLDCA, NKEY, NAMES, PSTNS, FTSCAR, :  ACTNUM, IARY1, IARY2, CARY,
STATUS )
```

Arguments:

NOLDCA = INTEGER (Given)
Number of cards in the original FITS array.

NKEY = INTEGER (Given)
The number of keyword cards to be inserted into the FITS-header array.

NAMES(NKEY) = CHARACTER * (*) (Given)
The keywords to be inserted into FITS card array.

PSTNS(NKEY) = CHARACTER * (*) (Given)
The position keyword names, before them the new keywords is inserted. If any name in it does not exit in FITS array, its corresponding keyword will be inserted just before the end card or appended to FITS array when end card does not exist. If two or more new cards have the same position name, they will all be put just before the position name in the same order as they are in NAMES.

FTSCAR(NOLDCA + NKEY) = CHARACTER * (*) (Given and Returned)
On entry its first NOLDCA elements hold the original FITS cards. On exit, its first ACTNUM elements hold the FITS cards after the insersion.

ACTNUM = INTEGER (Returned)
The actual number of cards in the FITS array after inserting.

IARY1(NOLDCA + NKEY) = INTEGER (Returned)
The first temporary working space.

IARY2(NOLDCA + NKEY) = INTEGER (Returned)
The second temporary working space.

CARY(NOLDCA + NKEY) = CHARACTER * (*) (Returned)
A temporary working space.

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

FTS1_ISKEY
Inquires whether or not a string is a valid FITS keyword

Description:

This routine tests whether a given string would be a valid FITS header keyword or not. For a keyword to be valid it must be no more than eight characters, and must comprise only uppercase Latin letters, numbers, underscore, and hyphen.

Invocation:

```
CALL FTS1_ISKEY( KEYWRD, VALID, STATUS )
```

Arguments:

KEYWRD = CHARACTER * (*) (Given)
The string to be tested.

VALID = LOGICAL (Returned)
If true, the string is a valid FITS header keyword.

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

Notes:

This routine does not convert the string to uppercase or remove leading blanks before validation.

FTS1_LOKEY
Locates an occurrence of a keyword in a FITS header

Description:

This routines find the location of a certain occurrence of a named keyword in an array of FITS headers. Hierarchical keywords are allowed. A bad status is returned if the desired keyword is not present in the header array.

Invocation:

```
CALL FTS1_LOKEY( NCARD, HEADER, KEYWRD, OCCUR, CARD, STATUS )
```

Arguments:

NCARD = INTEGER (Given)

The number of cards in the FITS header array.

HEADER(NCARD) = CHARACTER * (80) (Given)

The array of FITS headers.

KEYWRD = CHARACTER * (*) (Given)

The keyword to search for in the array. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Comparisons are performed in uppercase and blanks are removed. Each keyword must be no longer than 8 characters.

OCCUR = INTEGER (Given)

The occurrence of the keyword to locate. Values less than 1 obtain the first occurrence.

CARD = INTEGER (Returned)

The number of card containing the desired keyword.

STATUS = INTEGER (Given and Returned)

The global status.

FTS1_MANDH
Obtains the values of the mandatory headers in a FITS file

Description:

This routine searches for the mandatory FITS header keywords stored in a buffer, and their values are returned, if they are present. Should an item be missing or have an unsupported value an error is reported, a bad status is set and the routine exits. This version supports mandatory descriptors that are not in the correct order.

Currently, only simple FITS and group-format FITS are supported.

The number of dimensions is reduced when the highest dimension is one.

Invocation:

```
CALL FTS1_MANDH( FIRST, NCARD, HEADER, SCARD, BITPIX, NDIM, AXIS, DARRAY, NONSDA, SIZE,
STATUS )
```

Arguments:

FIRST = LOGICAL (Given)
If true the buffer contains the first header of a FITS file. It is used to validate the header.

NCARD = INTEGER (Given)
The number of card images in the header.

HEADER(NCARD) = CHARACTER * 80 (Given)
The array of headers (80-character cards) to be searched for the mandatory keywords.

SCARD = INTEGER (Given)
The number of the card from where the searches of the header will begin. This is needed because the headers make contain a dummy header prior to an extension.

BITPIX = INTEGER (Returned)
The number of bits per pixel of the data array.

NDIM = INTEGER (Returned)
The number of active dimensions.

AXIS(DAT__MXDIM) = INTEGER (Returned)
The dimensions of the data array.

DARRAY = LOGICAL (Returned)
If true there is a data array.

NONSDA = LOGICAL (Returned)
If true the data array is non-standard.

SIZE = INTEGER (Returned)
The number of pixels in the (or each) data array.

STATUS = INTEGER (Given and Returned)
Global status value.

FTS1_NDFCM
Creates the title, units, axes, WCS, and FITS extension in an NDF from the FITS headers

Description:

This routine adds the character components, axis structure, WCS component and FITS extension to an NDF. It searches a buffer containing the FITS header card images for the OBJECT keyword whose value, if present, becomes the NDF title. Similarly BUNIT is mapped to the NDF units. The supplied header structure is copied to the FITS extension.

Invocation:

```
CALL FTS1_NDFCM( NCARD, HEADER, SCARD, NDF, NENCOD, ENCODS, STATUS )
```

Arguments:

NCARD = INTEGER (Given)

The number of cards in the array of headers, from the start of the first header section to the end of the current one.

HEADER(NCARD) = CHARACTER * 80 (Given)

The buffer containing the header card images.

SCARD = INTEGER (Given)

The number of the card from where searches will begin, and copying of the headers to the FITS extension. Therefore NCARD - SCARD + 1 headers will appear in the extension. This argument is needed because the headers make contain a dummy header prior to an extension.

NDF = INTEGER (Given)

Identifier of the NDF to which to write the additional components and the FITS extension.

NENCOD = INTEGER (Given)

The number of AST encodings supplied in ENCODS.

ENCODS(NENCOD) = CHARACTER * (*) (Given)

The user's preferred AST encodings. If NENCOD is zero, then this is ignored, and an intelligent guess is made as to which encoding to use. The encoding determines which FITS headers are used to create the NDF WCS component.

STATUS = INTEGER (Given)

Global status value.

FTS1_NDF
Makes an NDF from a simple or group-format FITS file

Description:

This is a server routine for FITSIN/FITSDIN, hence the large argument list. It packages up the operations required to create and name an NDF; copy the FITS data to the NDF's data array, performing a data conversion if requested and flagging blank data with the standard bad-pixel values; generate the other components: title, units, WCS, axis structure and the FITS extension. For group-format FITS data, a series of NDFs are created, one per group, each with a generated filename. A null NDF may be given and this routine will exit, but permit the calling routine to continue to the next FITS file.

Invocation:

```
CALL FTS1_NDF( HEADER, BFPNTR, RCPNTR, AUTO, PNNDF, MEDIUM, MD, VMS, LENDIA, SIZE, NDIM,
DIMS, BPV, FMTCNV, FMTIN, FMTOUT, IEEE, BADPIX, BLANK, BSCALE, BZERO, DARRAY, NONSDA,
GCOUNT, PCOUNT, MXPARM, PTYPE, PSCALE, PZERO, FILROO, LOGHDR, FD, CFN, SUBFIL, GEXTND,
NCARD, HEADER, SCARD, NENCOD, ENCODS, BLKSIZ, ACTSIZ, OFFSET, CURREC, NEXT, PARAMS,
STATUS )
```

Arguments:

HEADER(*) = CHARACTER * 80 (Given)
The FITS headers in 80-character records.

BFPNTR = INTEGER (Given)
Pointer to BUFFER(BLKSIZ) = CHARACTER * (1) (Given and Returned). The buffer containing the block of data. This is only read when %OFFSET does not equal %ACTSIZ, i.e. there are some non-header data within it.

RCPNTR = INTEGER (Given)
Pointer to RECORD(36) = CHARACTER * (80) (Given and Returned). The buffer to hold the current FITS record.

AUTO = LOGICAL (Given)
If true the processing should be done in automatic mode, where the user is not prompted for file names, since these are generated from %FILROO with the sub-file or group numbers appended.

PNNDF = CHARACTER * (*) (Given)
The name of the parameter by which the filename of the output NDF will be obtained.

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported are 'DISK' for a disk file, and 'TAPE' for standard magnetic tape.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

VMS = LOGICAL (Given)
If true, the operating system is VMS or RSX. If false, the operating system is assumed to be UNIX.

LENDIA = LOGICAL (Given)
If true, the machine uses Little Endian byte order (bytes swapped compared to FITS). LENDIA is ignored when VMS = .TRUE..

SIZE = INTEGER (Given)
The number of elements in the data array.

NDIM = INTEGER (Given)
Dimensionality of the NDF.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the NDF.

BPV = INTEGER (Given)
The number of bytes per data value.

FMTCNV = LOGICAL (Given)
If true, format conversion from the integer FITS data to the real output data array is required. This is ignored when BADPIX is false (which should be the case for IEEE floating-point data).

FMTIN = CHARACTER * (*) (Given)
The HDS format of the data in the FITS file. It will be ignored if there is no format conversion.

FMTOUT = CHARACTER * (*) (Given)
The destination HDS format of the data array in the output file.

IEEE = LOGICAL (Given)
If true the FITS data are in IEEE floating-point format.

BADPIX = LOGICAL (Given)
If true the data-blank was defined in the FITS header. It will be ignored if the data are in IEEE format.

BLANK = INTEGER (Given)
The data-blank value equivalent to the bad-pixel flag. It should be ignored if %BADPIX is false.

BSCALE = REAL (Given)
The scale factor of the FITS integer data for their conversion to the true floating-point values.

BZERO = REAL (Given)
The offset of the FITS integer data for their conversion to the true floating-point values.

DARRAY = LOGICAL (Given)
If true there is a data array present if the FITS file.

NONSDA = LOGICAL (Given)
If true the data array is not standard, i.e. in group format. It is ignored if %DARRAY is false.

GCOUNT = INTEGER (Given)
The number of groups in the file.

PCOUNT = INTEGER (Given)
The number of parameters per group in the file.

MXPARM = INTEGER (Given)
The maximum number of group parameters, and the dimension size for the various group arrays.

PTYPE(MXPARM) = CHARACTER * (*) (Given)
The type (descriptive name) of each group parameter.

PSCALE(MXPARM) = DOUBLE PRECISION (Given)
The scale factors of the group parameters so that the parameters may be converted to the true floating-point values.

PZERO(MXPARM) = DOUBLE PRECISION (Given)
The offsets of the group parameters so that the parameters may be converted to the true floating-point values.

FILROO = CHARACTER * (*) (Given)
The rootname of the output NDF. The suffix Gn, where n=%NG, is appended in group-format mode to generate the filename. Otherwise in automatic mode the NDF filename is the rootname.

LOGHDR = LOGICAL (Given)
If true there is a log file open and records of the output file names will be written to it.

FD = INTEGER (Given)
The file descriptor for the log file. It is ignored if %LOGHDR is false.

CFN = CHARACTER * (*) (Given)
The number on the tape of the FITS file being processed if MEDIUM is 'TAPE', or the input disk-FITS filename if MEDIUM is 'TAPE'.

SUBFIL = INTEGER (Given)
The number of the sub-file/extension within the current FITS file being processed.

GEXTND = LOGICAL (Given)
If true there may be extensions in the FITS sub-file.

NCARD = INTEGER (Given)
The number of header 80-character cards in the header. Note the size of the structure will not do because it will normally have unfilled elements at the end, because of the way the work space is obtained in quanta. It should be the sum of the headers and the group parameters.

SCARD = INTEGER (Given)
The number of the card from where the searches of the header will begin. This is needed because the headers make contain a dummy header prior to an extension.

NENCOD = INTEGER (Given)
The number of AST encodings supplied in ENCODS.

ENCODS(NENCOD) = CHARACTER * (*) (Given)
The user's preferred AST encodings. If NENCOD is zero, then this is ignored, and an intelligent guess is made as to which encoding to use. The encoding determines which FITS headers are used to create the NDF WCS component.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the tape/disk buffer.

ACTSIZ = INTEGER (Given and Returned)
The actual block size (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = 'DISK'.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted.

CURREC = LOGICAL (Given and Returned)
If true the current FITS record is to be used immediately, i.e. it has alrady been read from tape or disk into %RECORD.

NEXT = LOGICAL (Returned)
This qualifies the status. If true it instructs the calling routine to go to the next FITS sub-file, and if status is bad the calling routine should flush the error messages.

PARAMS(MXPARM * BPV) = BYTE (Returned)
Numerical values of parameters associated with a group-format array.

STATUS = INTEGER (Given and Returned)
Global status value.

FTS1_PHEAD
Processes the headers in a FITS file on tape or disk

Description:

This routine reads selected files off a FITS tape or disk and writes the header data into an HDS file, and they are reported to the user. The dimension of the HDS buffer array in the HDS structure is enlarged if the buffer is filled. The number of bytes of header cards in the last data block (of the current file) containing header data is returned. This is to enable other routines to know where the data array starts in that buffer. If this number equals the actual blocksize this means a new data block needs to be read to access the data array.

Invocation:

```
CALL FTS1_PHEAD( BFPNTR, RCPNTR, RECORD, MEDIUM, MD, LOC, BLKSIZ, :  MAXHDR, REPORT,
ACTSIZ, OFFSET, CURREC, HSTART, :  HDNUM, EXTEND, NHEADS, STATUS )
```

Arguments:

BFPNTR = INTEGER (Given)
Pointer to the BUFFER byte array

RCPNTR = INTEGER (Given)
Pointer to the RECORD byte array

RECORD(36) = CHARACTER * (80) (Given and Returned)
The buffer to hold the current FITS record. Note that this out of order for SGP/16, but it is needed here so that a mapped array may be passed. This is a temporary kludge until the FITS readers are redesigned.

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported is ' DISK' for a disk file.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

LOC = CHARACTER * (DAT__SZLOC) (Given)
The locator to the HDS object that will contain the header cards.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the data buffer.

MAXHDR = INTEGER (Given)
The maximum number of header sections in the current sub-file.

REPORT = LOGICAL (Given)
If true the header cards are reported to the user.

ACTSIZ = INTEGER (Given and Returned)
The actual block size (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = ' DISK' .

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted as the header plus any earlier sub files.

CURREC = LOGICAL (Given and Returned)
If true the current FITS record is to be used immediately, i.e. it has alrady been read from tape or disk into %RECORD. On exit it is true when a new record has been read to determine whether or not there is an extension, but no extension is found.

HSTART(MAXHDR) = INTEGER (Returned)
The number of the header card where each FITS header starts.

HDNUM(MAXHDR) = INTEGER (Returned)
The number of header records processed and stored in the output data structure, one per header section in the current sub-file.

EXTEND = LOGICAL (Returned)
If true there are extensions in the FITS sub-file.

NHEADS = INTEGER (Returned)
The number of header sections in the sub-file. This includes dummy FITS header sections.

STATUS = INTEGER (Given and Returned)
Global status value.

FTS1_PRVAL
Prints the value of a specified card in an array of FITS headers

Description:

This routine reports the value of a keyword in an array of FITS headers via the Messaging system. It also returns the value and comment assoiciated with the header. Hierarchical keywords are allowed. A bad status is returned if the desired keyword is not present in the header array, or the card index is out of bounds.

Invocation:

```
CALL FTS1_PRVAL( NCARD, HEADER, KEYWRD, OCCUR, CARD, SVALUE, COMENT, STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of cards in the FITS header array.

HEADER(NCARD) = CHARACTER * (80) (Given)
The array of FITS headers.

KEYWRD = CHARACTER * (*) (Given)
The keyword to search for in the array. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Comparisons are performed in uppercase and blanks are removed. Each keyword must be no longer than 8 characters.

OCCUR = INTEGER (Given)
The occurrence of the keyword to locate. Values less than 1 obtain the first occurrence.

CARD = INTEGER (Given)
The index number of card containing the desired keyword. Must be between 1 and NCARD.

SVALUE = CHARACTER * (*) (Returned)
The value. This should have length of at least 68 characters.

COMENT = CHARACTER * (*) (Returned)
The value. This should have length of at least 60 characters, although normally 47 characters suffice.

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

Notes:

The comments and values are extracted from a single card.

FTS1_PTKEY
Puts non-reserved keyword cards into the FITS extension of an NDF file

Description:

This subroutine inserts a number of non-reserved keyword cards into the FITS extension of an NDF file at the positions just before a specified keyword card. If a keyword card exists in the extension, the routine will move the card to the specified position. This provides a way to relocate the existing keyword card. If a keyword card does not exist in the extension, the routine will put the keyword card at the specified position and will add an equals sign to it, at column 9 for simple keywords or immediately after the keyword for compound ones. To those newly added keywords, a character value '{undefined}' is assigned. To write new values for these keywords, you should use subroutine FTS1_WKEYx.

Following keywords are regarded as reserved keywords: SIMPLE, BITPIX, NAXIS, NAXISn, EXTEND, PCOUNT, GCOUNT and XTENSION. Their order in the extension should be fixed and should not be changed by any routine. Therefore if any of them is included in the NAMES, it will be ignored.

This subroutine can not be used to insert comment cards, that is, the cards with keyword 'COMMENT' or 'HISTORY', etc.

Invocation:

```
CALL FTS1_PTKEY( FTSLOC, NKEY, NAMES, PSTNS, STATUS )
```

Arguments:

FTSLOC = CHARACTER * (*) (Given)
The locator to the FITS extension of the NDF.

NKEY = INTEGER (Given)
The number of keyword cards to be inserted into the FITS extension.

NAMES(NKEY) = CHARACTER * (*) (Given)
The names of the keywords to be inserted. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Each keyword must be no longer than 8 characters. When inserted, the lower case letters are converted to uppercase and blanks are removed.

PSTNS(NKEY) = CHARACTER * (*) (Given)
The names of the cards before which the corresponding new keyword cards are inserted. If any name in PSTNS does not exist in the original FITS card array or is blank, its corresponding new card will be inserted just before end-card or be appended to the original FITS card array when there is no end-card. If two or more new cards have the same PSTNS name, they will all be put before the PSTNS name in the same order as they are in NAMES.

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

Prior Requirements :

The FITS extension is mapped for update access. It therefore must have some values assigned before using the routine.

FTS1_QTYPE
Determines the data type of a FITS header value

Description:

This routine takes a FITS header card and determines the HDS data type of the value. If there is no value because there is no equals sign present or the keyword is HISTORY or COMMENT, the type is returned as 'COMMENT'.

Invocation:

```
CALL FTS1_QTYPE( CARD, TYPE, STATUS )
```

Arguments:

CARD = CHARACTER * (*) (Given)

The FITS header card. It should be 80 character long, but the routine might work with less depending on the length of the value and keyword.

TYPE = CHARACTER * (*) (Returned)

The HDS data type of the header's value. It is one of the following: '_INTEGER', '_REAL', '_DOUBLE', '_LOGICAL', '_CHAR', or 'COMMENT'. The length should be at least

STATUS = INTEGER (Given and Returned)

The global status.

FTS1_RDATA
Reads the data of a FITS file on disk or tape

Description:

This routine reads the byte stream in the data blocks from the FITS tape or disk file written in the simple format, and writes the data into an array. The bytes may be reversed for VAX/VMS.

Invocation:

```
CALL FTS1_RDATA ( MEDIUM, MD, SIZE, BPV, REVERS, BLKSIZ, ACTSIZ, :  BUFFER, OFFSET,
RECORD, RDISP, DARRAY, STATUS )
```

Arguments:

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported is ' DISK' for a disk file.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

SIZE = INTEGER (Given)
Number of elements in the data array.

BPV = INTEGER (Given)
The number of bytes per data value.

REVERS = LOGICAL (Given)
If true the FITS data bytes are to be reversed within each word (when BPV is 2) or each integer (when BPV is 4) etc. If BPV=1 this flags makes no difference. Normally, only 2' s complement integer data need be reversed. Floating-point data require adjacent bytes to be swapped.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the tape/disk buffer.

ACTSIZ = INTEGER (Given and Returned)
The actual block size (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = ' DISK' .

BUFFER(BLKSIZ) = BYTE (Given and Returned)
The buffer containing the block of data. This is only read when %OFFSET does not equal %ACTSIZ, i.e. there are some non-header data within it.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted.

RECORD(2880) = BYTE (Given and Returned)
The buffer to hold the current FITS record.

RDISP = INTEGER (Given and Returned)
The number of bytes in the current record already interpreted. If this displacement is equal to the record length then a new FITS record will be obtained. The displacement will be updated during processing of the group parameters and data, and therefore can have an arbitrary value between 0 and 2880. Do not modify this argument outside this routine once initialised.

DARRAY(SIZE * BPV) = BYTE (Returned)
The data array used to store the data read.

STATUS = INTEGER(Given and Returned)
Global status value.

FTS1_RFMOD
Reads a text file containing instructions for editing an NDF's FITS extension

Description:

This routines opens a text file and parses it to determine how to modify an NDF's FITS extension. Details of the format and its interpretation is given in the item called " File Format" . The routine returns the editing command, keyword, position, values, comment, and data type in arrays.

Invocation:

```
CALL FTS1_RFMOD( FD, MAXMOD, NWRITE, EDITS, KEYWDS, KEYPOS, KOCCUR, POCCUR, VALUES,
COMNTS, TYPES, STATUS )
```

Arguments:

FD = INTEGER (Given)
The FIO identifier of the text file containing the editing instructions.

MAXMOD = INTEGER (Given)
The maximum number of modifications.

NWRITE = INTEGER (Returned)
The number of modifications actually made.

EDITS(MAXMOD) = CHARACTER * (*) (Returned)
The editing commands. Thus need only be one character per element.

KEYWDS(MAXMOD) = CHARACTER * (*) (Returned)
The FITS keywords to be modified into FITS card array. The length should be at least 48 characters to allow for hierarchical keywords.

KEYPOS(MAXMOD) = CHARACTER * (*) (Returned)
The position keyword names. The new keywords are inserted immediately before each corresponding position keyword. The length should be at least 48 characters to allow for hierarchical keywords.

KOCCUR(MAXMOD) = INTEGER (Returned)
The occurrences of the KEYWDS keywords to use.

POCCUR(MAXMOD) = INTEGER (Returned)
The occurrences of the KEYPOS keywords to use.

VALUES(MAXMOD) = CHARACTER * (*) (Returned)
The values to be given to the KEYWDS keywords. The length should be at least 68 characters to allow for the maximum length of a value.

COMNTS(MAXMOD) = CHARACTER * (*) (Returned)
The comments of the NAME keywords to use. The length should be at least 68 characters to allow for the maximum length of a comment, but normally 50 should be adequate.

TYPES(MAXMOD) = CHARACTER * (*) (Returned)
The data types of the values to write. These will be one of the following: ' _CHAR', ' _DOUBLE', ' _INTEGER', ' _LOGICAL', ' _REAL' . In addition there are two special values: ' COMMENT' to indicate that the card is a comment (so strictly it has no type), and ' ' to indicate that the data type is unknown, as occurs for a value defined by a reference keyword. The length should be at least 8 characters.

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

File Format :

The file consists of a series of lines, one per editing instruction, although blank lines and lines beginning with a ! or # are treated as comments. Note that the order does matter, as the edits are performed in the order given.

The format is summarised below:

command keyword{[occur]}{(keyword{[occur]})} {value {comment}}

where braces indicate optional values, and occur is the occurrence of the keyword. In effect there are four fields delineated by spaces that define the edit operation, keyword, value and comment.

Field 1: This specifies the editing operation. Allowed values are Amend, Delete, Exist, Move, Null, Read, Write, and Update, and can be abbreviated to the initial letter. Delete removes a named keyword. Read causes the value of a named keyword to be displayed to standard output. Exist reports TRUE to standard output if the named keyword exists in the header, and FALSE if the keyword is not present. Move relocates a named keyword to be immediately before a second keyword. When this positional keyword is not supplied, it defaults to the END card, and if the END card is absent, the new location is at the end of the headers. Write creates a new card given a value and an optional comment. Its location uses the same rules as for the Move command. Update revises the value and/or the comment. If a secondary keyword is defined explicitly, the card may be relocated at the same time. Update requires that the keyword exists. Amend behaves as Write if the keyword in Field 2 is not already present, or as Update if the keyword exists. Null replaces the value of a named keyword with blanks.

Field 2: This specifies the keyword to edit, and optionally the position of that keyword in the header after the edit (for Move, Write, Update, and Amend edits). The new position in the header is immediately before a positional keyword, whose name is given in parentheses concatenated to the edit keyword. See " Field 1" for defaulting when the position parameter is not defined or is null.

Both the editing keyword and position keyword may be compound to handle hierarchical keywords. In this case the form is keyword1.keyword2.keyword3 etc. All keywords must be valid FITS keywords. This means they must be no more than 8 characters long, and the only permitted characters are uppercase alphabetic, numbers, hyphen, and underscore. Invalid keywords will be rejected.

Both the edit and position keyword may have an occurrence specified in brackets []. This enables editing of a keyword that is not the first occurrence of that keyword, or locate a edited keyword not at the first occurrence of the positional keyword. Note that it is not normal to have multiple occurrences of a keyword in a FITS header, unless it is blank, COMMENT or HISTORY. Any text other than a positive integer is interpreted as the first occurrence.

Use a null value (' ' or " ") if you want the card to be a comment with keyword other than COMMENT or HISTORY. As blank keywords are used for hierarchical keywords, to write a comment in a blank keyword you must give a null edit keyword. These have no keyword appears before the left parenthesis or bracket, such as (), [], [2], or (EPOCH).

Field 3: This specifies the value to assign to the edited keyword in the the Amend, Write, and Update operations, or the name of the new keyword in the Rename modification. If the keyword exists, the existing value or keyword is replaced, as appropriate. The data type used to store the value is inferred from the value itself. See topic " Value Data Types" .

For the Update and Write modifications there is a special value, $V, which means use the current value of the edited keyword, provided that keyword exists. This makes it possible to modify a comment, leaving the value unaltered. In addition $V(keyword) requests that the value of the keyword given between the parentheses be assigned to the keyword being edited.

The value field is ignored when the keyword is COMMENT, HISTORY or blank and the modification is an Update or Write.

Field 4: This specifies the comment to assign to the edited keyword for the Amend, Write, and Update operations. A leading ' /' should not be supplied.

There is a special value, $C, which means use the current comment of the edited keyword, provided that keyword exists. This makes it possible to modify a value, leaving the comment unaltered. In addition $C(keyword) requests that the comment of the keyword given between the parentheses be assigned to the edited keyword.

To obtain leading spaces before some commentary, use a quote (') or double quote (") as the first character of the comment. There is no need to terminate the comment with a trailing and matching quotation character. Also do not double quotes should one form part of the comment.

Value Data Types :

The data type of the value is determined as follows:

- Values enclosed in quotes (') or doubled quotes (") are strings. Note that numeric or logical string values must be quoted to prevent them being converted to a numeric or logical value in the FITS extension.
- Otherwise type conversions of the first word after the keywords are made to integer, double precision, and logical types in turn. If a conversion is successful, that becomes the data type. In the case of double precision, the type is set to real when the number of significant digits only warrants single precision. If all the conversions failed the value is deemed to be a string.

Examples of the File Format :

The best way to illustrate the options is by listing some example lines.

P AIRMASS This reports the value of keyword AIRMASS to standard output.

E FILTER This determines whether keyword FILTER exists and reports TRUE or FALSE to standard output.

D OFFSET This deletes the keyword OFFSET.

Delete OFFSET[2] This deletes any second occurrence of keyword OFFSET.

Rename OFFSET1[2] OFFSET2 This renames the second occurrence of keyword OFFSET1 to have keyword OFFSET2.

W AIRMASS 1.379 This writes a real value to new keyword AIRMASS, which will be located at the end of the FITS extension.

A AIRMASS 1.379 This writes a real value to keyword AIRMASS if it exists, otherwise it writes a real value to new keyword AIRMASS located at the end of the FITS extension.

N AIRMASS This blanks the value of the AIRMASS keyword, if it exists.

W FILTER(AIRMASS) Y This writes a logical true value to new keyword FILTER, which will be located just before the AIRMASS keyword, if it exists.

Write FILTER(AIRMASS) ' Y' As the preceding example except that this writes a character value " Y" .

W COMMENT(AIRMASS) . Following values apply to mid-observation This writes a COMMENT card immediately before the AIRMASS card, the comment being " Following values apply to mid-observation" .

W DROCOM(AIRMASS) ' ' Following values apply to mid-observation As the preceding example but this writes to a non-standard comment keyword called DROCOM. Note the need to supply a null value.

W (AIRMASS) ' ' Following values apply to mid-observation As the preceding example but this writes to a blank-keyword comment.

U OBSERVER " Dr. Peter O' Leary" Name of principal observer This updates the OBSERVER keyword with the string value " Dr. Peter O' Leary" , and comment " Name of principal observer" . Note that had the value been enclosed in single quotes ('), the apostrophe would need to be doubled.

M OFFSET This moves the keyword OFFSET to just before the END card.

Move OFFSET(SCALE) This moves the keyword OFFSET to just before the SCALE card.

Move OFFSET[2](COMMENT[3]) This moves the second occurrence of keyword OFFSET to just before the third COMMENT card.

FTS1_RGRDA
Reads the data of a FITS file in group format from disk or tape

Description:

This routine reads a data block from the FITS tape or disk file that has byte stream of data in the groups format, and writes the data into an array. The values of the parameters associated with the data array are also obtained. The bytes may be reversed for VAX/VMS.

Invocation:

```
CALL FTS1_RGRDA ( MEDIUM, MD, SIZE, BPV, REVERS, PCOUNT, BLKSIZ, :  ACTSIZ, BUFFER,
OFFSET, RECORD, RDISP, PARAM, :  DARRAY, STATUS )
```

Arguments:

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported is ' DISK' for a disk file.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

SIZE = INTEGER (Given)
Number of elements in the data array. Note this is not the same as the number of bytes.

BPV = INTEGER (Given)
The number of bytes per data value.

REVERS = LOGICAL (Given)
If true the FITS data bytes are to be reversed within each word (when BPV is 2) or each integer (when BPV is 4) etc. If BPV=1 this flags makes no difference. Normally, only 2' s complement integer data need be reversed. Floating-point data require adjacent bytes to be swapped. Note that the group parameters are by definition in 2' s complement integers, so are reversed regardless of the sense of this flag.

PCOUNT = INTEGER (Given)
The number of parameters associated with the data array.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the tape/disk buffer.

ACTSIZ = INTEGER (Given and Returned)
The actual block size (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = ' DISK' .

BUFFER(BLKSIZ) = BYTE (Given and Returned)
The buffer containing the block of data. This is only read when %OFFSET does not equal %ACTSIZ, i.e. there are some non-header data within it.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted.

RECORD(2880) = BYTE (Given and Returned)
The buffer to hold the current FITS record.

RDISP = INTEGER (Given and Returned)
The number of bytes in the current record already interpreted. If this displacement is equal to the record length then a new FITS record will be obtained. The displacement will be updated during processing of the group parameters and data, and therefore can have an arbitrary value between 0 and 2880. Do not modify this argument outside this routine once initialised.

PARAM(PCOUNT * BPV) = BYTE (Returned)
The parameters associated with the data array.

DARRAY(SIZE * BPV) = BYTE (Returned)
The data array.

STATUS = INTEGER (Given and Returned)
Global status value.

FTS1_RNAND
Replaces NaN values with bad values in a vector of 64-bit IEEE floating-point numbers

Description:

This routine replaces any IEEE not-a-number (NaN) values present in a vector of 64-bit IEEE-754 floating-point numbers to the standard _DOUBLE bad-pixel value. Also converted to the standard bad-pixel value are values who exponent is greater than maximum provided by the host machine.

Invocation:

```
CALL FTS1_RNAND( EL, BUF, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of IEEE numbers to be processed. An expression must not be given.

BUF(EL) = DOUBLE PRECISION (Given and Returned)
The IEEE numbers to be cleansed of NaN values.

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

References :

- " IEEE Standard for Binary Floating-Point Arithmetic" , ANSI/IEEE 754, 1985.

FTS1_RNANR
Replaces NaN values with bad values in a vector of 32-bit IEEE floating-point numbers

Description:

This routine replaces any IEEE not-a-number (NaN) values present in a vector of 32-bit IEEE-754 floating-point numbers to the standard _REAL bad-pixel value. Also converted to the standard bad-pixel value are values who exponent is greater than maximum provided by the host machine.

Invocation:

```
CALL FTS1_RNANR( EL, BUF, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of IEEE numbers to be processed. An expression must not be given.

BUF(EL) = REAL (Given and Returned)
The IEEE numbers to be cleansed of NaN values.

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

References :

- " IEEE Standard for Binary Floating-Point Arithmetic" , ANSI/IEEE 754, 1985.

FTS1_ROOTN
Creates the rootname for an NDF

Description:

This is a server routine for FITSIN. It packages up the operations required to define the rootname for NDFs in automatic mode. The rootname is the prefix followed by the file number underscore sub-file number (if present).

Invocation:

```
CALL FTS1_ROOTN( MEDIUM, FN, SUBFIL, PREFIX, FILROO, NCROOT, :  STATUS )
```

Arguments:

MEDIUM = CHARACTER * (*) (Given)

The medium containing the FITS file. Currently supported are 'DISK' for a disk file, and 'TAPE' for standard magnetic tape.

CFN = CHARACTER * (*) (Given)

The number on the tape of the FITS file being processed if MEDIUM is 'TAPE', or the input disk-FITS filename if MEDIUM is 'TAPE'.

SUBFIL = INTEGER (Given)

The number of the sub-file/extension within the current FITS file being processed.

PREFIX = CHARACTER * (*) (Given)

The prefix to be given to the file name of catalogues produced in automatic mode. It is ignored when %MEDIUM = 'DISK'.

FILROO = CHARACTER * (*) (Returned)

The root file name of NDFs (to be used in automatic mode).

NCROOT = INTEGER (Returned)

The length in characters of the rootname.

STATUS = INTEGER (Given and Returned)

The global status.

FTS1_RSTAB
Reads the table format from a FITS file and write to a text file

Description:

This routine reads the data blocks from a FITS table-format tape or disk file, and creates an ASCII file into which it writes the table.

Invocation:

```
CALL FTS1_RSTAB ( TABLE, MEDIUM, MD, PNTABL, TABNAM, AUTO, AXIS1, AXIS2, BLKSIZ, ACTSIZ,
BUFFER, OFFSET, CURREC, RECORD, STATUS )
```

Arguments:

TABLE = CHARACTER * (*) (Returned)
The work buffer to store a line of the table, i.e. must be AXIS1+1 bytes long.

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported is 'DISK' for a disk file.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

PNTABL = CHARACTER * (*) (Given)
The parameter name used to create and associate the table file.

TABNAM = CHARACTER * (*) (Given)
The suggested name of the table file, unless %AUTO is true when it is the actual name of the table file to be created.

AUTO = LOGICAL (Given)
If true the supplied file name is used to open the table file rather than obtaining the file name via the parameter system.

AXIS1 = INTEGER (Given)
Number of characters in a line of the table.

AXIS2 = INTEGER (Given)
The number of lines in the table.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the tape/disk buffer.

ACTSIZ = INTEGER (Given and Returned)
The actual block size (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = 'DISK'.

BUFFER(BLKSIZ) = BYTE (Given and Returned)
The buffer containing the block of data. This is only read when %OFFSET does not equal %ACTSIZ, i.e. there are some non-header data within it.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted.

CURREC = LOGICAL (Given and Returned)
If true the current FITS record is to be used immediately, i.e. it has alrady been read from tape or disk into %RECORD.

RECORD(2880) = BYTE (Given and Returned)
The buffer to hold the current FITS record.

STATUS = INTEGER (Given and Returned)
Global status value.

FTS1_SCOFB
Applies scale and zero to a REAL data vector, and substitutes magic values for blank FITS data

Description:

This routine applies scale and offset to a 1-d REAL array. Any pixels with the %BLANK value are substituted by the standard magic value. On output, pixel is input value times scale plus offset.

Invocation:

```
CALL FTS1_SCOFB ( BSCALE, BZERO, UNDEF, BLANK, SIZE, ARRAY, :  STATUS )
```

Arguments:

BSCALE = REAL (Given)
Scale factor to be applied to the array.

BZERO = REAL (Given)
Offset to be applied to the array.

UNDEF = LOGICAL (Given)
If true, testing and replacement of undefined data values is to occur. A blank value, as specified by %BLANK, is replaced by the standard magic value. If false, the value of %BLANK is ignored.

BLANK = INTEGER (Given)
Value of an undefined datum.

SIZE = INTEGER (Given)
Number of elements in the data array.

ARRAY(SIZE) = REAL (Given and Returned)
The data array to which scaling and offset is to be applied.

STATUS = INTEGER (Given)
Global status value.

Bugs:

None known.

FTS1_SCTAB
Creates an ASCII catalogue and description file in SCAR format from a FITS tape or disk file

Description:

This is a server routine for FITSIN/FITSDIN. It packages up the operations required to handle a FITS ASCII table file and turn it into an ASCII catalogue and SCAR description file. The names of the output files are recorded in the logfile.

Invocation:

```
CALL FTS1_SCTAB( HEADER, PNDSCF, PNTAB, MEDIUM, MD, NCARD, SCARD, :  NDIM, DIMS, LOGHDR,
FD, CFN, SUBFIL, PREFIX, :  AUTO, BLKSIZ, ACTSIZ, BFPNTR, OFFSET, CURREC, :  RCPNTR,
STATUS )
```

Arguments:

HEADER(NCARD) = CHARACTER * 80 (Given)
The FITS headers in 80-character records.

PNDSCF = CHARACTER * (*) (Given)
The name of the parameter by which the filename of the SCAR description file will be obtained.

PNTAB = CHARACTER * (*) (Given)
The name of the parameter by which the filename of the table will be obtained.

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported are 'DISK' for a disk file, and 'TAPE' for standard magnetic tape.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

NCARD = INTEGER (Given)
The number of card images in the buffer.

SCARD = INTEGER (Given)
The number of the card from where the searches of the header will begin. This is needed because the headers make contain a dummy header prior to an extension.

NDIM = INTEGER (Given)
Dimensionality of the table. At present only 2-d is supported.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the table.

LOGHDR = LOGICAL (Given)
If true there is a log file open and records of the output file names will be written to it.

FD = INTEGER (Given)
The file descriptor for the log file. It is ignored if %LOGHDR is false.

CFN = CHARACTER * (*) (Given)
The number on the tape of the FITS file being processed if MEDIUM is 'TAPE', or the input disk-FITS filename if MEDIUM is 'TAPE'.

SUBFIL = INTEGER (Given)
The number of the sub-file/extension within the current FITS file being processed.

PREFIX = CHARACTER * (*) (Given)
The prefix to be given to the file name of catalogues produced in automatic mode. It is ignored when %MEDIUM = ' DISK'.

AUTO = LOGICAL (Given)
If true the processing should be done in automatic mode, where the user is not prompted for file names, since these are generated from the prefix, file and sub-file numbers in the case where the medium is tape. The form is prefix_file or prefix_file_subfile if the subfile is greater than 1. %MEDIUM = ' DISK' the prefix is ignored.

BLKSIZ = CHARACTER * (*) (Given and Returned)
The maximum block size and dimension of %BUFFER.

ACTSIZ = CHARACTER * (*) (Given and Returned)
The actual block size on tape or disk (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = ' DISK'.

BFPNTR = INTEGER (Given)
Pointer to the buffer containing catalogue data, the buffer itself will be updated each time a tape block is read. If the offset equals the block size then the existing data in the buffer will not be used.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted as the header plus any earlier FITS files.

CURREC = LOGICAL (Given and Returned)
If true the current FITS record is to be used immediately, i.e. it has alrady been read from tape or disk into %RECORD.

RCPNTR = INTEGER (Given)
A pointer to the buffer to hold the current FITS record of 36 80-character card images.

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

Prior Requirements :

- The magnetic tape or disk file must already be associated, and any log file must be opened. [routine_prior_requirements]...

FTS1_SDSCF
Creates a SCAR description file from a FITS table-format headers

Description:

This routine searches for the mandatory FITS-Tables-format header cards stored in a buffer, and their values are returned, if they are present. Should an item be missing or have an unsupported value an error is reported, a bad status is set and the routine exits. This version supports mandatory descriptors that are not in the correct order.

There are two methods of opening the FACTS description file. The first associates the file with a parameter. Unfortunately, FIO filename parameters cannot be given dynamic values. Therefore, the name of the table in the header card images is presented, and the naming rule for an FACTS description file given. The user has to type in the file name, when prompted for the FACTS file to be created for ADC(/SCAR) usage. However, in the alternative, automatic mode, the supplied file name is used with the DSCF prefix to open the FACTS file without recourse to the parameter system. Information in the FITS headers is transferred to this description file. The mandatory FACTS parameters for a sequential disk file are written plus EPOCH, VERSION, NRECORDS and AUTHOR where these are known. These are followed by FACTS records describing the fields in the table, and the ENDFIELD record. Finally all the comment lines in the FITS header are copied, in order, to the FACTS file. Blank comment entries are inserted in the

The syntax of the field's format descriptor is checked to see that it is standard.

Invocation:

```
CALL FTS1_SDSCF( NCARD, HEADER, SCARD, PNDSCF, AUTO, TABNAM, :  DSCFNM, NDIM, AXIS,
STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the header.

HEADER(NCARD) = CHARACTER * (*) (Given)
The buffer containing the table-format FITS header.

SCARD = INTEGER (Given)
The number of the card from where the searches of the header will begin. This is needed because the headers make contain a dummy header prior to an extension.

PNDSCF = CHARACTER * (*) (Given)
The parameter name used to create and associate the FACTS description file. It is only used when %AUTO is false.

AUTO = LOGICAL (Given)
If true the supplied file name is used to open the file rather than obtaining the file name via the parameter system.

TABNAM = CHARACTER * (*) (Given and Returned)
The name of the table file. It is read if %AUTO is true and written when %AUTO is false.

DSCFNM = CHARACTER * (*) (Returned)
The name of the description file.

NDIM = INTEGER (Returned)
The number of active dimensions.

AXIS(DAT__MXDIM) = INTEGER (Returned)
The dimensions of the data array.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- The format specifier for each field is validated.

FTS1_SKIP
Skips over the data of a FITS file

Description:

This routine skips over the data in a FITS file by reading the data blocks from the FITS file. The file may be basic FITS, groups or an extension.

Invocation:

```
CALL FTS1_SKIP( MEDIUM, MD, SIZE, BPV, GCOUNT, PCOUNT, BLKSIZ, :  ACTSIZ, BUFFER, OFFSET,
RECORD, RDISP, STATUS )
```

Arguments:

MEDIUM = CHARACTER * (*) (Given)
The medium containing the FITS file. Currently supported is 'DISK' for a disk file.

MD = INTEGER (Given)
The tape or file descriptor depending on the value of %MEDIUM.

SIZE = INTEGER (Given)
The number of elements in the data array.

BPV = INTEGER (Given)
The number of bytes per data value.

GCOUNT = INTEGER (Given)
The number of groups in the file.

PCOUNT = INTEGER (Given)
The number of parameters per group in the file.

BLKSIZ = INTEGER (Given)
The maximum blocksize and dimension of the tape buffer.

ACTSIZ = INTEGER (Given and Returned)
The actual block size (a multiple of the FITS record length of 2880 bytes). It is only an input argument for %MEDIUM = 'DISK'.

BUFFER(BLKSIZ) = BYTE (Given and Returned)
The buffer containing the block of data. This is only read when %OFFSET does not equal %ACTSIZ, i.e. there are some non-header data within it.

OFFSET = INTEGER (Given and Returned)
The number of bytes in the current block already interpreted.

RDISP = INTEGER (Given and Returned)
The number of bytes in the current record already interpreted. If this displacement is equal to the record length then a new FITS record will be obtained. The displacement will be updated during processing of the group parameters and data, and therefore can have an arbitrary value between 0 and 2880. Do not modify this argument outside this routine once initialised.

RECORD(2880) = BYTE (Given and Returned)
The buffer to hold the current FITS record.

STATUS = INTEGER (Given and Returned)
Global status value.

FTS1_UKEYC
Writes the value of type CHARACTER to a keyword from a buffer of FITS-header card images

Description:

This routine searches a buffer containing the header card images from a FITS file for card with keyword %NAME; and, if found, override its original value or with the given character value %VALUE. The keyword can have a comment string leading by the character specified by %CMTBGN. In addition comment-type keywords—those with keywords COMMENT, HISTORY, and blank, or argument COMCAR is .TRUE.—may also have their comments revised by %COMNT (whereupon %VALUE is ignored). Selection of the desired card (especially important for commentary cards) can be controlled by the starting the search at card numbered %STCARD. The search ends when the next end of a header block, marked by the END keyword, is encountered or the buffer is exhausted. If the keyword is present %THERE is true, otherwise it is false. If the parameter is present more than once in the header, only the first occurence will be used.

The name may be compound to permit writing of hierarchical keywords.

The buffer of FITS header card image should be mapped in the 'UPDATE' mode.

Invocation:

```
CALL FTS1_UKEYC( NCARD, BUFFER, STCARD, NAME, VALUE, CMTBGN, COMNT, COMCAR, THERE, CARD,
STATUS )
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the buffer.

BUFFER(NCARD) = CHARACTER * (*) (Given and Returned)
The buffer containing the header card images.

STCARD = INTEGER (Given)
The number of the card from which to start search.

NAME = CHARACTER * (*) (Given)
The name of the keyword to be written a new value. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Comparisons are performed in uppercase and blanks are removed. Each keyword must be no longer than 8 characters. The total length must not exceed 48 characters. This is to allow for the value, and indentation into a blank-keyword card (as hierarchical keywords are not standard and so cannot be part of the standard keyword name).

VALUE = CHARACTER * (*) (Given)
The value to be used to override the original value of the keyword. It is ignored when the keyword is COMMENT, HISTORY, or blank.

CMTBGN = CHARACTER * (1) (Given)
The character which indicates the beginning of the comment string of to be appended to the keyword. Normally it is '/'. when it is blank, no comment will be appended to the keyword. It is ignored when the keyword is COMMENT, HISTORY, or blank.

COMNT = CHARACTER * (*) (Given)
The comment string of the keyword. It may be truncated at the end to put into the space left after writing keyword value for non-commentary keywords.

COMCAR = LOGICAL (Given)
If .TRUE., the supplied card is a comment and thus the value and comment delimiter are ignored, and just the keyword and comment string are used to generate the header card.

THERE = LOGICAL (Returned)
If true, the specified keyword is present.

CARD = INTEGER (Returned)
The number of the last continuation comment card whose contents are included in the returned string.

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

Notes:

- The comments are written from column 32 or higher if the value demands more than the customary 20 characters for the value. A comment may be omitted if the value is so long to leave no room.
- The BUFFER appears out of standard order because when this routine is called, BUFFER is expected to be a mapped array. This order prevents having to append several %VAL(length) arguments instead of just the one.

FTS1_UKEYx
Writes the value to the specified keyword from a buffer of FITS-header card images

Description:

This routine searches a buffer containing the header card images from a FITS file for card with keyword %NAME; and, if found, override its original value with the given value %VALUE. The keyword can have a comment string leading by the character specified by %CMTBGN. The search ends when the next end of a header block, marked by the END keyword, is encountered or the the buffer is exhausted. If the keyword is present %THERE is true, otherwise it is false. If the parameter is present more than once in the header, only the first occurence will be used.

The name may be compound to permit writing of hierarchical keywords.

The buffer of FITS header card image should be mapped in the 'UPDATE' mode.

Invocation:

```
CALL FTS1_UKEYx( NCARD, BUFFER, STCARD, NAME, VALUE, CMTBGN, COMNT, THERE, CARD, STATUS
)
```

Arguments:

NCARD = INTEGER (Given)
The number of card images in the buffer.

BUFFER(NCARD) = CHARACTER * (*) (Given and Returned)
The buffer containing the header card images.

STCARD = INTEGER (Given)
The number of the card from which to start search.

NAME = CHARACTER *(*) (Given)
The name of the keyword to be written a new value. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. Comparisons are performed in uppercase and blanks are removed. Each keyword must be no longer than 8 characters. The total length must not exceed 48 characters. This is to allow for the value, and indentation into a blank-keyword card (as hierarchical keywords are not standard and so cannot be part of the standard keyword name).

VALUE = ? (Given)
The value to be used to override the original value of the keyword.

CMTBGN = CHARACTER * (1) (Given)
The character which indicates the beginning of the comment string of to be appended to the keyword. Normally it is '/'. when it is blank, no comment will be appended to the keyword.

COMNT = CHARACTER * (*) (Given)
The comment string of the keyword. It may be truncated at the end to put into the space left after writing keyword value.

THERE = LOGICAL (Returned)
If true, the specified keyword is present.

CARD = INTEGER (Returned)
The number of the last continuation comment card whose contents are included in the returned string.

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

Notes:

- There is a routine for each of the data types logical, integer, 64-bit integer, real, and double precision: replace "x" in the routine name by L, I, K, R or D as appropriate.
- The comments are written from column 32 or higher if the value demands more than the customary 20 characters for the value. A comment may be omitted if the value is so long to leave no room.
- The BUFFER appears out of standard order because when this routine is called, BUFFER is expected to be a mapped array. This order prevents having to append several %VAL(length) arguments instead of just the one.

FTS1_VHEAD
Validates a FITS header card and reconstitutes to the standard

Description:

This routines takes a buffer containing text similar to a FITS header card and attempts to produce a correctly formatted FITS header card. The validation process performs the following checks the input buffer: a) the length of the input buffer is no more than 80 characters, otherwise it is truncated; b) the keyword only contains uppercase Latin alphabetic characters, numbers, underscore, and hyphen (this is a fatal error except for lowercase letters); c) value cards have an equals sign in column 9 and a space in column 10; d) quotes enclose character values; e) single quotes inside string values are doubled; f) character values are left justified to column 11 (retaining leading blanks) and contain at least 8 characters (padding with spaces if necessary); g) non-character values are right justified to column 30 for mandatory keywords, or when the fixed format is requested (unless the value is double-precision requiring more than 20 digits); h) the comment delimiter is in column 32 for the mandatory keywords or when the fixed format is requested for non-character values, or is at least two characters following the value otherwise; i) an equals sign in column 9 of a commentary card is replaced by a space, issuing a warning message at normal reporting level; and j) comments begin at least two columns after the end of the comment delimiter.

Some non-fatal errors—a), b), c), d), and i)—produce warning messages at message level MSG__NORM.

Invocation:

```
CALL FTS1_VHEAD( BUFFER, FIXED, CARD, STATUS )
```

Arguments:

BUFFER = CHARACTER * (*) (Given)
The FITS header ' card' to be validated.

FIXED = LOGICAL (Given)
If this is .TRUE., all values use the FITS fixed format with left-justified character strings starting two columns after

CARD = CHARACTER * (80) (Returned)
The validated FITS header ' card' .

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

FTS1_WCSAX
Re-creates AXIS structures from a WCS component FrameSet

Description:

This routine creates NDF Axis structures from an AXIS Frame in the supplied FrameSet. It looks for AXIS and PIXEL Frames in the supplied FrameSet. If either of these Frames is not found, it does nothing. Otherwise, it attempts to create AXIS structures in the NDF from the AXIS Frame in the FrameSet. The AXIS Centre, Label and Unit components are set.

Invocation:

```
CALL FTS1_WCSAX( INDF, FS, NDIM, STATUS )
```

Arguments:

INDF = INTEGER (Given)
The NDF identifier.

FS = INTEGER (Given)
An AST pointer for a FrameSet.

NDIM = INTEGER (Given)
The number of axes in the NDF.

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

FTS1_WCSDF
Sees if two FrameSets are different after being written to a FitsChan

Description:

The returned function value indicates if the two FrameSets are inconsistent with each other. The two supplied FrameSets are written to two FitsChans, using the specified encoding. All keyword values in the two FitsChans are then compared. If any keyword has a significantly different value in the two FitsChans, then the FrameSets are inconsistent, and a .TRUE value is returned. Otherwise, .FALSE. is returned.

Invocation:

```
RETURN = FTS1_WCSDF( ENCOD, FS1, FS2, STATUS )
```

Arguments:

ENCOD = CHARACTER * (*) (Given)
The encoding scheme to use when converting the supplied FrameSets into FITS header cards.

FS1 = INTEGER (Given)
An AST pointer to the first FrameSet.

FS2 = INTEGER (Given)
An AST pointer to the second FrameSet.

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

Notes:

- A value of .TRUE. is returned if an error has already occurred, or if this function should fail for any reason.

Function Value :

FTS1_WCSDF = LOGICAL (Returned) .TRUE. if the supplied FrameSets are inconsistent, and .FALSE. otherwise.

FTS1_WCSIM
Imports WCS information from the supplied FitsChan into the supplied NDF

Description:

An attempt is made to read an AST FrameSet (see SUN/210) from the supplied FitsChan. If succesful, it is added into the FrameSet representing the NDF's WCS component. This is done by connecting the base Frames of the two FrameSets with a UnitMap (on the assumption that they are equivalent). The modified FrameSet is then stored back in the NDF (the NDF library will automatically remove any PIXEL, GRID and AXIS Frames from the FrameSet as these are generated afresh each time NDF_GTWCS is called).

The supplied FitsChan may contain more than one description (or " encoding") of the FrameSet to be added to the NDF, each encoding using a different set of header cards. These encodings may not all be consistent with each other. For instance, if a Starlink application stores a FrameSet twice in a FITS header using FITS-WCS and AST native encodings, an IRAF application may then modify the FITS-WCS encoding without making equivalent modifications to the native encoding. In this case, we should use the FITS-WCS encoding in preference to the native encoding when reconstructing the NDFs WCS component. On the other hand, if the two encodings were still consistent, it would be preferable to use the native encoding since the FITS-WCS encoding may not give a complete description of the original FrameSet.

The choice of encoding has several stages:

o If the caller has supplied a list of preferred encodings in the ENCODS argument, then the first available encoding in this list is used. o If no preferred encodings are supplied, then a check is made to see if a native encoding is available. If there is no native encoding, then the default encoding supplied by AST is used. This will be a non-native encoding selected on the basis of the header cards available in the FitsChan. o If a native encoding is available, and is the only available encoding, then it is used. o If both native and non-native encodings are available, then the first non-native encoding to be found which is inconsistent with the native encoding is used. If all encodings are consistent, then the native encoding is used. The first inconsistent encoding is used on the assumption that software which modifies the native encoding (i.e. mainly Starlink software) will also modify the non-native encodings so that they remain consistent. Foreign software however (i.e. non-AST software) will probably not bother to modify the native encoding when the non-native encoding is modified.

Invocation:

```
CALL FTS1_WCSIM( FC, INDF, NENCOD, ENCODS, STATUS )
```

Arguments:

FC = INTEGER (Given)
An AST pointer to the FitsChan in which to store the NDF's WCS information.

INDF = INTEGER (Given)
The NDF identifier.

NENCOD = INTEGER (Given)
The number of encodings supplied in ENCODS.

ENCODS(NENCOD) = CHARACTER * (*) (Given)
The preferred encodings to use, in order of preference (most preferable first). Ignored if NENCOD is zero.

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

FTS1_WCSUT
Removes the spurious copy of the AXIS Frame which is left in WCS component when using non-Native encodings

Description:

This routine removes the Current Frame in the WCS component of an NDF if no value has been set for its Domain attribute, and if it corresponds closely to the AXIS Frame (i.e. if the Mapping from the Current Frame to the AXIS Frame is nearly a UnitMap).

When using non-Native encoding, the AXIS Frame written by NDF2FITS will not have any associated Domain value, and so will not be recognised by the NDF library as an AXIS Frame when it is read back in by FITS2NDF. The Frame will therefore be left in the WCS component of the NDF, even though it is in reality a copy of the AXIS Frame. This routine removes such Frames.

Invocation:

```
CALL FTS1_WCSUT( INDF, STATUS )
```

Arguments:

INDF = INTEGER (Given)
The NDF identifier.

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

FTS1_WKEYC
Writes a FITS-header card for a CHARACTER value or a comment

Description:

This routine writes a FITS-header card using the supplied keyword, value, comment, and comment delimiter. The card is either a character string value, or a comment card. The latter occurs when the keyword is blank, HISTORY or COMMENT, or the COMCAR argument is .TRUE.; in this case both the supplied value and comment delimiter are ignored. The name may be compound to permit writing of hierarchical keywords.

Invocation:

```
CALL FTS1_WKEYC( NAME, VALUE, CMTBGN, COMNT, COMCAR, HEADER, STATUS )
```

Arguments:

NAME = CHARACTER * (*) (Given)

The name of the keyword to be written. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. The value is converted to uppercase and blanks are removed before being used. Each keyword must be no longer than 8 characters. The total length must not exceed 48 characters. This is to allow for the value, and indentation into a blank-keyword card (as hierarchical keywords are not standard and so cannot be part of the standard keyword name).

VALUE = CHARACTER * (*) (Given)

The value of the keyword. It is ignored when the keyword is COMMENT, HISTORY, or blank, or COMCAR = .TRUE..

CMTBGN = CHARACTER * (1) (Given)

The character which indicates the beginning of the comment string of to be appended to the keyword. Normally it is '/'. when it is blank, no comment will be appended to the keyword. It is ignored when the keyword is COMMENT, HISTORY, or blank, or COMCAR = .TRUE..

COMNT = CHARACTER * (*) (Returned)

The comment string of the keyword. It may be truncated at the end to put into the space left after writing keyword value for non-commentary keywords.

COMCAR = LOGICAL (Given)

If .TRUE., the supplied card is a comment and thus the value and comment delimiter are ignored, and just the keyword and comment string are used to generate the header card.

HEADER = CHARACTER * (80) (Returned)

The created FITS-header card.

STATUS = INTEGER (Given and Returned)

The global status.

FTS1_WKEYx
Writes a FITS-header card

Description:

This routine writes a FITS-header card using the supplied keyword, value, comment, and comment delimiter. The name may be compound to permit writing of hierarchical keywords.

Invocation:

```
CALL FTS1_WKEYx( NAME, VALUE, CMTBGN, COMNT, HEADER, STATUS )
```

Arguments:

NAME = CHARACTER ∗(∗) (Given)

The name of the keyword to be written. This may be a compound name to handle hierarchical keywords, and it has the form keyword1.keyword2.keyword3 etc. The maximum number of keywords per FITS card is 20. The value is converted to uppercase and blanks are removed before being used. Each keyword must be no longer than 8 characters. The total length must not exceed 48 characters. This is to allow for the value, and indentation into a blank-keyword card (as hierarchical keywords are not standard and so cannot be part of the standard keyword name).

VALUE = ? (Given)

The value of the keyword.

CMTBGN = CHARACTER ∗ (1) (Given)

The character which indicates the beginning of the comment string of to be appended to the keyword. Normally it is ' /' . when it is blank, no comment will be appended to the keyword.

COMNT = CHARACTER ∗ (∗) (Returned)

The comment string of the keyword. It may be truncated at the end to put into the space left after writing keyword value.

HEADER = CHARACTER ∗ (80) (Returned)

The created FITS-header card.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- There is a routine for each of the data types logical, integer, 64-bit integer, real, and double precision: replace " x" in the routine name by L, I, K, R or D as appropriate.
- The comments are written from column 32 or higher if the value demands more than the customary 20 characters for the value. A comment may be omitted if the value is so long to leave no room.

IRA_ANNUL
Annuls an IRA identifier

Description:

This routine should be called when access to the astrometry information associated with an IRA identifier is no longer needed. It releases the resources used to store the astrometry information.

This routine attempts to execute even if STATUS is bad on entry. However, in this case no error report will be produced if this routine subsequently fails.

Invocation:

```
CALL IRA_ANNUL( IDA, STATUS )
```

Arguments:

IDA = INTEGER (Given)
The IRA identifier to be annulled.

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

IRA_CLOSE
Closes down the IRA astrometry package

Description:

This routine should be called once IRA facilities are no longer needed. It annuls any currently valid IRA identifiers, releasing any resources reserved by them. Once this routine has been called, any further use of IRA must be preceeded with a call to IRA_INIT.

This routine attempts to execute even if STATUS is set to a bad value on entry.

Invocation:

```
CALL IRA_CLOSE( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

IRA_CONVT
Converts sky co-ordinates from one system to another

Description:

This routine use SLALIB to convert a list of sky co-ordinates from one supported Sky Co-ordinate System (SCS) to any other supported system. It is assumed that the observations were made at the date given by the Julian epoch supplied. If the input and output co-ordinates are referred to different mean equinox, then precession is applied to convert the input co-ordinates to the output system. No correction for nutation is included. If any of the input co-ordinate values are equal to the Starlink " BAD" value (VAL__BADD) then the corresponding output values will both be set to the bad value.

Invocation:

```
CALL IRA_CONVT( NVAL, AIN, BIN, SCSIN, SCSOUT, EPOCH, AOUT, BOUT, STATUS )
```

Arguments:

NVAL = INTEGER (Given)
The number of sky co-ordinate pairs to be converted.

AIN(NVAL) = DOUBLE PRECISION (Given)
A list of sky longitude co-ordinate values to be converted, in radians.

BIN(NVAL) = DOUBLE PRECISION (Given)
A list of sky latitude co-ordinate values to be converted, in radians.

SCSIN = CHARACTER * (*) (Given)
A string holding the name of the sky co-ordinate system of the input list. Any unambiguous abbreviation will do. An optional equinox specifier may be included in the name (see ID2 section " Sky Coordinates").

SCSOUT = CHARACTER * (*) (Given)
A string holding the name of the sky co-ordinate system required for the output list. Any unambiguous abbreviation will do.An optional equinox specifier may be included in the name.

EPOCH = DOUBLE PRECISION (Given)
The Julian epoch at which the observations were made. When dealing with IRAS data, the global constant IRA__IRJEP should be specified. This constant is a Julian epoch suitable for all IRAS data.

AOUT(NVAL) = DOUBLE PRECISION (Returned)
The list of converted sky longitude co-ordinate values, in radians.

BOUT(NVAL) = DOUBLE PRECISION (Returned)
The list of converted sky latitude co-ordinate values, in radians.

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

IRA_CREAT
Create an identifier for specified astrometry information

Description:

The supplied astrometry information is stored in internal common blocks and an " IRA identifier" is returned which can be passed to other IRA routines to refer to the stored astrometry information. This identifier should be annulled when it is no longer required by calling IRA_ANNUL. In addition, a call to IRA_EXPRT may optionally be made to store the astrometry information in an NDF (see argument INDF).

The projection used is specified by the argument PROJ, and must be one of the supported projection types (see routine IRA_IPROJ). For more information on the available projections, see the ID/2 appendix " Projection Equations" . Each projection requires the values for several parameters to be supplied in argument P. These parameters have the same meaning for all projections (for further details see ID/2 appendix " Projection Equations"):

P(1): The longitude (in the Sky Coordinate System specified by argument SCS) of the reference point in radians.

P(2): The latitude (in the Sky Coordinate System specified by argument SCS) of the reference point in radians.

P(3): The first image coordinate (i.e. X value) of the reference point. Image coordinates are fractional values in which the centre of the pixel (1,1) has coordinates (0.5,0.5).

P(4): The second image coordinate (i.e. Y value) of the reference point.

P(5): The size along the X image axis, of a pixel centred at the reference point, in radians. Actual pixel size will vary over the image due to the distorting effect of the projection. The absolute value is used.

P(6): The size along the Y image axis, of a pixel centred at the reference point, in radians. The absolute value is used.

P(7): The position angle of the Y image axis, in radians. That is, the angle from north to the positive direction of the Y image axis, measured positive in the same sense as rotation from north to east. (Here " north" and " east" are defined by the value of SCS). The X image axis is 90 degrees west of the Y axis.

P(8): An angle through which the celestial sphere is to be rotated before doing the projection. The axis of the rotation is a radius passing through the reference point. The rotation is in an anti-clockwise sense when looking from the reference point towards the centre of the celestial sphere. The value should be in radians. Changing this angle does not change the orientation of the image axes with respect to north (which is set by p(7)).

Invocation:

```
CALL IRA_CREAT( PROJ, NP, P, SCS, EPOCH, INDF, IDA, STATUS )
```

Arguments:

PROJ = CHARACTER * (*) (Given)
The projection type (see routine IRA_IPROJ for a list of currently recognised values). Any unambiguous abbreviation can be given.

NP = INTEGER (Given)
The size of array P.

P(NP) = DOUBLE PRECISION (Given)
The parameter values required by the projection.

SCS = CHARACTER * (*) (Given)
The name of the Sky Coordinate System which the projection is to create, or an unambiguous abbreviation. See routine IRA_ISCS for a list of currently recognised values. See ID2 section " Sky Coordinates") for general information of Sky Coordinate Systems.

EPOCH = DOUBLE PRECISION (Given)
The Julian epoch at which the observations were made. A single mean epoch is sufficient to describe all IRAS observations. Such a value is contained in the IRA constant IRA__IRJEP.

INDF = INTEGER (Given)
The identifier for the NDF in which the astrometry information is to be stored. If an invalid NDF identifier is given (eg the value NDF__NOID) then the astrometry information is not stored in an NDF (but IDA can still be used to refer to the astrometry information).

IDA = INTEGER (Returned)
The returned IRA identifier.

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

IRA_CTOD1
Converts a single formatted sky co-ordinate value into a double-precision value

Description:

The input string is presumed to hold a sky co-ordinate value in character form. If NC is 1, the string is interpreted as a longitude value. If NC is 2, the string is interpreted as a latitude value. This routine reads the string and produces a double precision value holding the co-ordinate value in radians. The value is not shifted into the first order range (eg if an angular value equivalent to 3∗PI is given, the value 3∗PI will be returned, not 1∗PI). If the input string is blank the output value is set to the Starlink " BAD" value (VAL__BADD). Refer to IRA_CTOD for details of the allowed format for the input string.

Invocation:

```
CALL IRA_CTOD1( TEXT, SCS, NC, VALUE, STATUS )
```

Arguments:

TEXT = CHARACTER ∗ (∗) (Given)

The string containing the formatted version of the sky co-ordinate value. If this string is blank, VALUE is returned with the " BAD" value (VAL__BADD), but no error report is made.

SCS = CHARACTER ∗ (∗) (Given)

The sky co-ordinate system (see ID2 section " Sky Coordinates"). Any unambiguous abbreviation will do.

NC = INTEGER (Given)

Determines which sky co-ordinate is to be used. If a value of 1 is supplied, the string is interpreted as a longitude value (e.g. RA if an equatorial system is being used). If a value of 2 is supplied, the string is interpreted as a latitude value. Any other value results in an error being reported.

VALUE = DOUBLE PRECISION (Returned)

The numerical value of the sky co-ordinate represented by the string in TEXT. The value is in radians.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_CTOD
Converts formatted sky co-ordinate values into double-precision values

Description:

The input strings are presumed to hold sky co-ordinate values in character form. This routine reads the strings and produces double precision values holding the co-ordinate values, in radians.

Each input string can consist of a set of up to three " fields" . Each field starts with a numeric value (which can have a fractional part) terminated by a character string. This character string consists of an optional single character, called the terminator character, followed by an arbitrary number of spaces The terminator character (if present) must be one of the letters h,d,m,s or r. An h terminator indicates that the field value is in units of hours. A d terminator indicates that the field value is in units of degrees. An r terminator indicates that the field value is in units of radians. An m terminator indicates that the field value is in units of minutes. An s terminator indicates that the field value is in units of seconds. The interpretation of minutes and seconds depends on whether the value is a time value or an angle value. The longitude value for equatorial sky co-ordinate systems (RA) is expected to be a measure of time, all other co-ordinate values are expected to be a measure of angle. These defaults are overriden if the first field is a " degrees" , " hours" or " radians" field (as indicated by the presence of a d, h or r terminator character). The interpretation of fields with no terminator character depends on which field is being considered. If the first field has no terminator, it is assumed to be either a degrees or hours field. If the second or third field has no terminator, it is assumed to be a seconds field if the previous field was a minutes field, and a minutes field if the previous value was an hours or degrees field.

In addition, an input string may contain a single field with no terminator character in an " encoded" form. " Encoded" fields are identified by the fact that the field contains 5 or more digits to the left of the decimal point (including leading zeros if necessary). These fields are decoded into hours or degrees as follows: Any fractional part is taken as the fractional part of the seconds field, the tens and units digits are taken as the integer part of the seconds field, the hundreds and thousands digits are taken as the minutes fields, the remaining digits are taken as the degrees or hours field. Thus -12345.4 would be interpreted as (- 1 hour 23 mins 45.4 seconds) or (- 1 degree 23 mins 45.4 seconds). The same value could also be specified as

- 1 23 45.5, -1h 23m 45.5s (if it represents a time value), or
- 1d 23 45.5 (if it represents an angular value).

The supplied values must be in their first order ranges (i.e. 0 to 2.PI for longitude values and -PI/2 to +PI/2 for latitude values). Values outside these ranges cause an error to be reported, and the status value IRA__RANGE is returned). The exception to this is if the string is prefixed with a " $*$" character, in which case any numeric value may be supplied. In this case the supplied value is returned directly (e.g. if the string " $*$400D" is given, the radian equivalent of 400 degrees will be returned, not 40 (=400-360) degrees). If either of the input strings are blank the corresponding output value is set to the Starlink " BAD" value (VAL__BADD).

Invocation:

```
CALL IRA_CTOD( ATEXT, BTEXT, SCS, A, B, STATUS )
```

Arguments:

ATEXT = CHARACTER $*$ ($*$) (Given)

The string containing the formatted version of the longitude value. If this string is blank, A is returned with the " BAD" value (VAL__BADD), but no error is reported.

BTEXT = CHARACTER * (*) (Given)
The string containing the formatted version of the latitude value. If this string is blank, B is returned with the " BAD" value (VAL__BADD), but no error is reported.

SCS = CHARACTER * (*) (Given)
The sky co-ordinate system (see ID2 section " Sky Coordinates"). Any unambiguous abbreviation will do.

A = DOUBLE PRECISION (Returned)
The longitude value represented by the string ATEXT, in radians.

B = DOUBLE PRECISION (Returned)
The latitude value represented by the string BTEXT, in radians.

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

IRA_DTOC1
Convert a single floating=point sky co-ordinate value to character form

Description:

This routine creates a text string containing a formatted version of the given sky co-ordinate value. The value is assumed to be a longitude value if NC is 1, and a latitude if NC is 2. The formats of the output string are as described in routine IRA_DTOC. Longitude values are shifted into the range 0 - 2∗PI before being used. Latitude values are shifted into the range +/- PI before being used. An error is reported if a latitude value then has an absolute value greater than PI/2 (this differs from the behaviour of IRA_NORM which always reduces the latitude value to +/- PI/2).

Invocation:

```
CALL IRA_DTOC1( VALUE, SCS, NC, STYLE, TEXT, STATUS )
```

Arguments:

VALUE = DOUBLE PRECISION (Given)

The value of the sky co-ordinate to be formatted, in radians. If VALUE has the Starlink " BAD" value (VAL__BADD) then the output string TEXT is set blank.

SCS = CHARACTER ∗ (∗) (Given)

The sky co-ordinate system in use (see ID2 section " Sky Coordinates"). Any unambiguous abbreviation will do.

NC = INTEGER (Given)

Determines which sky co-ordinate is given. If a value of 1 is supplied, VALUE is interpreted as a longitude value (e.g. RA if an equatorial system is being used). If a value of 2 is supplied, VALUE is interpreted as a latitude value. Any other value causes an error to be reported.

STYLE = INTEGER (Given)

A value in the range 1 to 5 which specifies the style of output formatting required. In addition a value of zero can be specified which causes a default style to be used dependant on the value of SCS. See routine IRA_DTOC for a description of the styles and defaults.

TEXT = CHARACTER ∗ (∗) (Returned)

The string containing the formatted description of the sky co-ordinate value. The variable supplied for TEXT should have a declared length equal to the value of parameter IRA__SZFSC.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_DTOC
Converts a pair of double-precision sky co-ordinate values to character form

Description:

This routine creates a pair of text strings containing formatted versions of the given sky co-ordinate values. The exact format depends on the type of sky co-ordinate system in use and the value of STYLE (see the " Notes" section below). The input co-ordinate values are shifted into their first order ranges before being used (see IRA_NORM).

Invocation:

```
CALL IRA_DTOC( A, B, SCS, STYLE, ATEXT, BTEXT, STATUS )
```

Arguments:

A = DOUBLE PRECISION (Given)

The value of the sky longitude to be formatted, in radians. If A has the Starlink " BAD" value (VAL__BADD) then the output string ATEXT is set blank.

B = DOUBLE PRECISION (Given)

The value of the sky latitude to be formatted, in radians. If B has the " BAD" value then the output string BTEXT is set blank.

SCS = CHARACTER * (*) (Given)

The sky co-ordinate system in use (see ID2 section " Sky Coordinates"). Any unambiguous abbreviation will do.

STYLE = INTEGER (Given)

A value in the range 1 to 5 which specifies the style of output formatting required. Additionally, a value of zero can specified which causes a default style to be used dependant on the value of SCS. See the " Notes" section below for a description of the individual styles and defaults for each SCS.

ATEXT = CHARACTER * (*) (Returned)

The string containing the formatted description of the sky longitude value A. The variable supplied for ATEXT should have a declared length equal to the value of parameter IRA__SZFSC.

BTEXT = CHARACTER * (*) (Returned)

The string containing the formatted description of the sky latitude value B. The variable supplied for BTEXT should have a declared length equal to the value of parameter IRA__SZFSC.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- SCS = " EQUATORIAL" Default is style 2 (used if argument STYLE is zero on entry).

STYLE = 1: (a full description)

" RA = 12hrs 3m 0.02s" and " DEC = -33deg 23m 0.00s"

STYLE = 2: (a more brief form readable by IRA_CTOD)

" 12h 3m 0.02s" and " -33d 23m 0.00s"

STYLE = 3: (a very brief form readable by IRA_CTOD)

" 120300.00" and " -332300.00" (e.g. hhmmss.ss and ddmmss.ss)

STYLE = 4: (a brief form readable by IRA_CTOD)
" 12 03 0.02" and " -33 23 0.00"
STYLE = 5: (a brief form readable by IRA_CTOD)
" 12.050006" and " -33.383333" (e.g. fractional values in hours (RA) and degrees (DEC))

- SCS = " GALACTIC" Default is style 5 (used if argument STYLE is zero on entry).

STYLE = 1:
" l = 12deg 3m 0.02s" and " b = -33deg 23m 0.00s"
STYLE = 2:
" 12deg 3m 0.02s" and " -33d 23m 0.00s"
STYLE = 3:
" 0120300.00" and " -332300.00" (e.g. dddmmss.ss and ddmmss.ss)
STYLE = 4: (a brief form readable by IRA_CTOD)
" 12 03 0.02" and " -33 23 0.00"
STYLE = 5: (a brief form readable by IRA_CTOD)
" 12.050006" and " -33.383333" (e.g. fractional values in degrees)

- SCS = " ECLIPTIC" Default is style 5 (used if argument STYLE is zero on entry).

STYLE = 1:
" Lambda = 12deg 3m 0.02s" and " Beta = -33deg 23m 0.00s"
STYLE = 2:
" 12deg 3m 0.02s" and " -33d 23m 0.00s"
STYLE = 3:
" 0120300.00" and " -332300.00" (e.g. dddmmss.ss and ddmmss.ss)
STYLE = 4: (a brief form readable by IRA_CTOD)
" 12 03 0.02" and " -33 23 0.00"
STYLE = 5: (a brief form readable by IRA_CTOD)
" 12.050006" and " -33.383333" (e.g. fractional values in degrees)

IRA_EXPRT
Stores astrometry information in an NDF

Description:

An HDS structure is created containing the astrometry information identified by IDA. This " astrometry structure" is stored as a component of an extension within the NDF specified by INDF (any previous astrometry structure is over-written). The names of the NDF extension and the astrometry structure are set by a call to IRA_LOCAT. If no such call is made the names of the extension and astrometry structure retain the values set up in IRA_INIT (" IRAS" and " ASTROMETRY"). The astrometry structure has an HDS data type of IRAS_ASTROMETRY. The NDF extension must already exist before calling this routine.

Any existing astrometry structure is first deleted from the NDF (in which ever extension it was found) before creating the new one.

Invocation:

```
CALL IRA_EXPRT( IDA, INDF, STATUS )
```

Arguments:

IDA = INTEGER (Given)
An IRA identifier for the astrometry information.

INDF = INTEGER (Given)
The identifier for the NDF in which the astrometry information is to be stored.

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

IRA_FIND
Finds an astrometry structure within an NDF

Description:

A search is made for an astrometry structure within an NDF. If one is found, THERE is returned true. If one is not found, no error is reported but THERE is returned false. The name of the extension in which it was found is returned, together with the name of the astrometry structure, and a locator to the extension.

Invocation:

```
CALL IRA_FIND( INDF, THERE, XNAME, ASNAME, LOC, STATUS )
```

Arguments:

INDF = INTEGER (Given)
The NDF identifier.

THERE = LOGICAL (Returned)
True if an astrometry structure was found, false otherwise.

XNAME = CHARACTER * (*) (Returned)
The name of the NDF extension in which the astrometry structure was found. Blank if none found.

ASNAME = CHARACTER * (*) (Returned)
The name of a component of the NDF extension holding the astrometry information. Blank if none found.

LOC = CHARACTER * (*) (Returned)
A locator to the extension identified by XNAME. DAT__NOLOC if no astrometry structure is found.

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

IRA_GETCO
Obtains a pair of sky co-ordinates from the ADAM environment

Description:

The ADAM parameters specified by arguments APAR and BPAR are used to acquire values for the first and second sky co-ordinates respectively, in the sky co-ordinate system specified by argument SCS. The parameters are obtained as literal character strings and decoded into floating point values. See routine IRA_CTOD for a description of the allowed formats of the strings associated with these parameters. The input values of arguments A and B can optionally be supplied to the user as default parameter values. The parameter prompt strings contained in the application's interface file can be overridden by giving a non-blank value for argument PRMAPP. In this case, the prompts are formed by appending the value of PRMAPP to the co-ordinate descriptions returned by routine IRA_SCNAM. For instance, if PRMAPP = " of the field centre" , and an equatorial sky co-ordinate system is in use, then the prompt for APAR will be " Right Ascension of the field centre" , and the prompt for BPAR will be " Declination of the field centre" . Note, the total length of the prompt strings is limited to 80 characters. If PRMAPP is blank, then the current prompt strings are used (initially equal to the values in the interface file).

Invocation:

```
CALL IRA_GETCO( APAR, BPAR, PRMAPP, SCS, DEFLT, A, B, STATUS )
```

Arguments:

CHARACTER = APAR (Given)
The name of the ADAM parameter (type LITERAL) to use for the sky longitude value.

CHARACTER = BPAR (Given)
The name of the ADAM parameter (type LITERAL) to use for the sky latitude value.

PRMAPP = CHARACTER * (*) (Given)
A string to append to each axis description to form the parameter prompt strings. If this is blank then the current prompt strings are used (i.e. initially set to the values in the interface file).

SCS = CHARACTER * (*) (Given)
The name of the sky co-ordinate system to use. Any unambiguous abbreviation will do (see ID2 section " Sky Coordinates").

DEFLT = LOGICAL (Given)
True if the input values of A and B are to be communicated to the environment as run-time defaults for the parameters specified by APAR and BPAR. If A or B is " BAD" on entry (i.e. equal to VAL__BADD) then no default is set up for the corresponding parameter.

A = DOUBLE PRECISION (Given and Returned)
The value of the first sky co-ordinate. In radians.

B = DOUBLE PRECISION (Given and Returned)
The value of the second sky co-ordinate. In radians.

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

IRA_GETEQ
Extracts the epoch of the reference equinox from a string specifying a Sky Co-ordinate System

Description:

If, on entry, the argument SCS contains an explicit equinox specifier (see routine IRA_ISCS), the epoch contained within it is returned in argument EQU as a double precision value, and argument BJ is returned equal to the character " B" or " J" depending on whether the epoch is Besselian or Julian. If there is no equinox specifier in argument SCS on entry, then the default of B1950 is returned.

If the sky co-ordinate system specified by SCS is not referred to the equinox (e.g. GALACTIC) then EQU is returned equal to the Starlink " BAD" value VAL__BADD, and BJ is returned blank.

The argument NAME is returned holding the full (unabbreviated) name of the sky co-ordinate system without any equinox specifier. On exit, the argument SCS holds the full name plus an explicit equinox specifier (for systems which are referred to the equinox). Thus, if SCS contained " EQUAT" on entry, it would contain " EQUATORIAL(B1950)" on exit.

Invocation:

```
CALL IRA_GETEQ( SCS, EQU, BJ, NAME, STATUS )
```

Arguments:

SCS = CHARACTER * (*) (Given and Returned)

On entry this should contain an SCS name (or any unambiguous abbreviation), with or without an equinox specifier. On exit, it contains the full SCS name with an explicit equinox specifier (for those sky co-ordinate systems which are referred to the equinox). If no equinox specifier is present on entry, then a value of B1950 is used (if required). This variable should have a declared length given by the symbolic constant IRA__SZSCS.

EQU = DOUBLE PRECISION (Returned)

The epoch of the reference equinox. This is extracted from any explicit equinox specifier contained in SCS on entry. If there is no equinox specifier in SCS, a value of 1950.0 is returned. If the sky co-ordinate system is not referred to the equinox (e.g. GALACTIC) the Starlink " BAD" value (VAL__BADD) is returned, irrespective of any equinox specifier in SCS.

BJ = CHARACTER * (*) (Returned)

Returned holding either the character B or J. Indicates if argument EQU gives a Besselian or Julian epoch. Returned blank if the sky co-ordinate system is not referred to the equinox.

NAME = CHARACTER * (*) (Returned)

The full name of the sky co-ordinate system without any equinox specifier. This variable should have a declared length given by the symbolic constant IRA__SZSCS.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_GTCO1
Obtains a single sky co-ordinate value from the ADAM environment

Description:

The ADAM parameter specified by argument PARAM is used to acquire a longitude or latitude value in the requested sky co-ordinate system. Argument NC determines which is to be obtained. The string is decoded into a double precision number representing the sky position. See the documentation for IRA_CTOR for a description of the allowed formats. The input sky co-ordinate value can optionally be communicated to the environment as a dynamic default.

Invocation:

```
CALL IRA_GTCO1( PARAM, PROMPT, SCS, NC, DEFLT, VALUE, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the ADAM parameter used to get the sky co-ordinate value.

PROMPT = CHARACTER * (*) (Given)

A string to override the current prompt for the parameter. If this is blank, the prompt is left at its current value. The initial value for the prompt is defined in the interface file. Note, unlike routine IRA_GETCO, the axis name is not automatically included in the prompt.

SCS = CHARACTER * (*) (Given)

The sky co-ordinate system in use. Any unambiguous abbreviation will do (see ID2 section " Sky Coordinates").

NC = INTEGER (Given)

Determines which sky co-ordinate is to be returned. If a value of 1 is supplied, the string obtained for the parameter is interpreted as a longitude value (e.g. RA if an equatorial system is being used). If a value of 2 is supplied, the string is interpreted as a latitude value. Any other value causes an error to be reported.

DEFLT = LOGICAL (Given)

If true, then the value of VALUE on entry is communicated to the environment as a dynamic default. If false, or if VALUE is " BAD" on entry (i.e. equal to VAL__BADD), then no dynamic default is set up.

VALUE = DOUBLE PRECISION (Given and Returned)

The sky co-ordinate value. On input it contains the default value (in radians) to use if DEFLT is true. On exit it contains the decoded value obtained from the environment, in radians.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_GTSCS
Gets the full name of a Sky Co-ordinate System (with equinox specifier) from the environment

Description:

The ADAM parameter specified by argument SCSPAR is used to get a character string from the environment. A check is done to make sure that the string obtained represents a supported Sky co-ordinate System (SCS). The user may include an equinox specifier (see IRA_ISCS) in the text string to override the default reference equinox of B1950. If an illegal SCS name is entered the user is reprompted. If DEFLT is given true, the value of SCS on entry is used as a default for the parameter. The value of SCS is expanded (both on entry and exit) to a full SCS name (an abbreviation of the SCS may be supplied either by the calling routine or by the user instead of the full name).

Invocation:

```
CALL IRA_GTSCS( SCSPAR, DEFLT, SCS, STATUS )
```

Arguments:

SCSPAR = CHARACTER * (*) (Given)

The name of the ADAM parameter to use, which should be of type LITERAL.

DEFLT = LOGICAL (Given)

If true, then the value of argument SCS on entry is used (after expansion) as the run-time default for the parameter.

SCS = CHARACTER * (*) (Given and Returned)

On entry, specifies the default value for the parameter. On exit, contains the full version of the sky co-ordinate system entered by the user. The supplied variable should have a declared length given by symbolic constant IRA__SZSCS.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_IDEPOCH
Gets the epoch associated with an IRA identifier

Description:

This routine returns the epoch associated with an IRA identifier.

Invocation:

```
CALL IRA_IDEPOCH( IDA, EPOCH, STATUS )
```

Arguments:

IDA = INTEGER (Given)
The IRA identifier for the astrometry information.

EPOCH = DOUBLE PRECISION (Returned)
Returned holding the epoch associated with the IRA identifier.

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

IRA_IDPROJN
Gets the projection name associated with an IRA identifier

Description:

This routine returns the Sky Co-ordinate System associated with an IRA identifier.

Invocation:

```
CALL IRA_IDPROJN( IDA, PROJN, STATUS )
```

Arguments:

IDA = INTEGER (Given)

The IRA identifier for the astrometry information.

PROJN = CHARACTER * (*) (Returned)

On exit, contains the projection name associated with the IRA identifier. The supplied variable should have a declared length given by symbolic constant IRA__SZPRJ.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_IDPROJP
Get a projection parameter value associated with an IRA identifier

Description:

This routine returns the value of a projection parameter associated with an IRA identifier.

Invocation:

```
CALL IRA_IDPROJP( IDA, IPAR, PROJP, STATUS )
```

Arguments:

IDA = INTEGER (Given)
The IRA identifier for the astrometry information.

IPAR = INTEGER (Given)
The index of the required projection parameter, in the range 1 to IRA__MAXP.

PROJP = DOUBLE PRECISION (Returned)
Returned holding the value of the projection parameter.

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

IRA_IDSCS
Gets the Sky Co-ordinate System associated with an IRA identifier

Description:

This routine returns the Sky Co-ordinate System associated with an IRA identifier.

Invocation:

```
CALL IRA_IDSCS( IDA, SCS, STATUS )
```

Arguments:

IDA = INTEGER (Given)
The IRA identifier for the astrometry information.

SCS = CHARACTER * (*) (Returned)
On exit, contains the full version of the sky co-ordinate system associated with the IRA identifier. The supplied variable should have a declared length given by symbolic constant IRA__SZSCS.

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

IRA_INIT
Initialises the IRA astrometry package

Description:

This routine must be called before calling any other IRA routine which has an " IDA" argument. It is not necessary to call this routine before using routines such as IRA_DIST which do not have an " IDA" argument. This routine annulls any currently valid IRA identifiers, sets the NDF extension name in which the astrometry structure is located to " IRAS" , sets the name of the astrometry structure to " ASTROMETRY" , and resets graphics options to their default values.

Invocation:

```
CALL IRA_INIT( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

IRA_IPROJ
Returns a list of supported projection names

Description:

A string is returned containing the list of supported projection names and equivalent names. The names are separated by commas. The currently supported projections are:

GNOMONIC (or equivalently TANGENT_PLANE)

LAMBERT (or equivalently CYLINDRICAL)

AITOFF (or equivalently ALL_SKY)

ORTHOGRAPHIC

See ID/2 appendix " Projection Equations" for more details about the supported projections.

Invocation:

```
CALL IRA_IPROJ( LIST, STATUS )
```

Arguments:

LIST = CHARACTER * (*) (Returned)

The list of supported projections and equivalent names. The character variable supplied for this argument should have a declared size equal to the value of parameter IRA__SZPLS. If the supplied string is not long enough to hold all the names, a warning message is given, but no error status is returned. Each returned projection name has a maximum length given by symbolic constant IRA__SZPRJ.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_ISCS
Return a list of supported sky coordinate systems

Description:

A string is returned containing a list of names identifying the supported sky coordinate systems. The names in the output list are separated by commas.

By default, Equatorial and Ecliptic coordinates are referred to the mean equinox of Besselian epoch 1950.0. The calling application can override this default by appending a string known as an " equinox specifier" to the end of the SCS name (in fact all IRA routines will accept any unambiguous abbreviation of the SCS name). An equinox specifier consists of a year with upto 4 decimal places, preceded with the letter B or J to indicate a Besselian or Julian epoch, and enclosed in parentheses. The named coordinate system is then referred to the mean equinox of the epoch given in the equinox specifier. The following are examples of legal SCS values; EQUATORIAL(B1950), EQUAT(J2000), ECLIP, ECLIP(1983.2534), etc. If the date is not preceded with either B or J (as in the last example), a Besselian epoch is assumed if the date is less than 1984.0, and a Julian epoch is assumed otherwise.

The currently supported sky coordinate systems are:

EQUATORIAL

The longitude axis is Right Ascension, the latitude axis is Declination. Other legal names can be made by appending an equinox specifier (eg EQUATORIAL(B1983.5)). If no equinox specifier is added, the coordinates are referred to the mean equinox of Besselian epoch 1950.0. If the equinox is described by a Besselian epoch, the old FK4 Bessel-Newcomb system is used. If a Julian epoch is used, the new IAU 1976, FK5, Fricke system is used.

GALACTIC

The longitude axis is galactic longitude and the latitude axis is galactic latitude, given in the IAU 1958 galactic coordinate system.

ECLIPTIC

The longitude axis is ecliptic longitude and the latitude axis is ecliptic latitude. Other legal names can be made by appending an equinox specifier (eg ECLIPTIC(B1983.5)). If no equinox specifier is added, the coordinates are referred to the mean equinox of Besselian epoch 1950.0.

AZEL

The longitude axis is horizon azimuth and the latitude axis is horizon elevation.

All sky coordinate values supplied to, or returned from any IRA routine, are given in units of radians.

Invocation:

```
CALL IRA_ISCS( LIST, STATUS )
```

Arguments:

LIST = CHARACTER * (*) (Returned)

The list of supported sky coordinate system names. The character variable supplied for this argument should have a declared size equal to the value of parameter IRA__SZCLS. If the supplied string is not long enough to hold all the names, a warning message is given, but no error status is returned.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_LOCAT
Sets the location for new IRA astrometry structures

Description:

By default, IRA_CREAT and IRA_EXPRT store astrometry information in a component named " ASTROMETRY" within the " IRAS" NDF extension. These names may be changed if necessary by calling this routine. The supplied arguments give the name of the NDF extension and the component name to be used by all future calls to IRA_CREAT or IRA_EXPRT. It should be ensured that the extension exists before calling IRA_CREAT or IRA_EXPRT.

Invocation:

```
CALL IRA_LOCAT( XNAME, ASNAME, STATUS )
```

Arguments:

XNAME = CHARACTER * (*) (Given)

The name of NDF extension in which astrometry structures are to be created. If a blank value is supplied the current value is left unchanged.

ASNAME = CHARACTER * (*) (Given)

The name of a component of the NDF extension in which to store astrometry information. If a blank value is supplied the current value is left unchanged.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_NORM
Converts sky co-ordinate values to the equivalent first-order values

Description:

The given latitude value is shifted into the range +/- PI/2 (a shift of PI may be introduced in the longitude value to achieve this). The longitude value is then shifted into the range 0 to 2∗PI. If either A or B has the Starlink " BAD" value on entry (VAL__BADD) then both A and B are left unchanged on exit. Latitude values which are within 0.01 arcseconds of either pole are modified to put them exactly at the pole.

Invocation:

```
CALL IRA_NORM( A, B, STATUS )
```

Arguments:

A = DOUBLE PRECISION (Given and Returned)
The longitude, in radians. On exit, the value is shifted in to the range 0 to 2∗PI.

B = DOUBLE PRECISION (Given and Returned)
The latitude value, in radians. On exit, the value is shifted in to the range -PI/2 to +PI/2.

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

IRA_READ
Gets an identifier for astrometry information stored in an HDS astrometry structure

Description:

An attempt is made to read astrometry information from the supplied HDS object, assuming the object is an IRA astrometry structure. The astrometry information is copied into internal common blocks and an " IRA identifier" is returned which can be passed to other IRA routines to refer to the stored astrometry information. This identifier should be annulled when it is no longer required by calling IRA_ANNUL.

Invocation:

```
CALL IRA_READ( LOC, IDA, STATUS )
```

Arguments:

LOC = CHARACTER * (*) (Given)

An HDS locator to an astrometry structure. The constant IRA__HDSTY gives the HDS type required for this object.

IDA = INTEGER (Returned)

The IRA identifier which is used by other IRA routines to access the astrometry information.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_SETEQ
Encodes the epoch of a reference equinox within an SCS name

Description:

On entry, SCS contains the name of a Sky Co-ordinate System (or an unambiguous abbreviation), with or without an equinox specifier (see routine IRA_ISCS). On exit, SCS contains the full name of the Sky Co-ordinate System with an equinox specifier appended, determined by arguments EQU and BJ. Any old equinox specifier is first removed. If the Sky Co-ordinate System is not referred to the equinox (e.g. GALACTIC) then no equinox specifier is included in SCS on exit.

Invocation:

```
CALL IRA_SETEQ( EQU, BJ, SCS, STATUS )
```

Arguments:

EQU = DOUBLE PRECISION (Given)

The epoch of the reference equinox. After calling this routine, the sky co-ordinates described by SCS are referred to the mean equinox of the epoch given by EQU. If EQU has the Starlink " BAD" value (VAL__BADD) then no equinox specifier is included in SCS on exit.

BJ = CHARACTER $*$ ($*$) (Given)

Determines if the epoch specified by argument EQU is a Besselian or Julian epoch. BJ should have the value " B" or " J" . Any other value causes an error report (except that a blank value causes " B" to be used if EQU is less than 1984.0 and " J" otherwise).

SCS = CHARACTER $*$ ($*$) (Given and Returned)

On entry, SCS should contain an unambiguous abbreviation of a supported Sky Co-ordinate System (see routine IRA_ISCS), with or without an equinox specifier. On exit, SCS contains the full name of the Sky Co-ordinate System, appended with an equinox specifier determined by arguments EQU and BJ. If the Sky Co-ordinate System is not one that is referred to the equinox (e.g. GALACTIC) then no equinox specifier is included in SCS on exit. SCS should have a declared length equal to the symbolic constant IRA__SZSCS.

STATUS = INTEGER (Given and Returned)

The global status.

IRA_TRANS
Transforms co-ordinate data

Description:

Co-ordinate data are transformed from sky co-ordinates to image co-ordinates, or vice-versa, using the projection information identified by IDA. The direction of the transformation is determined by the argument FORWRD. If any input co-ordinate values are equal to the Starlink " BAD" value (VAL__BADD) then both the output values are set to the bad value.

Invocation:

```
CALL IRA_TRANS( NVAL, IN1, IN2, FORWRD, SCS, IDA, OUT1, OUT2, STATUS )
```

Arguments:

NVAL = INTEGER (Given)
The number of co-ordinate points to be transformed.

IN1(NVAL) = DOUBLE PRECISION (Given)
If FORWRD is true, then IN1 holds values of the first image co-ordinate (X), otherwise IN1 holds values of the sky longitude.

IN2(NVAL) = DOUBLE PRECISION (Given)
If FORWRD is true, then IN2 holds values of the second image co-ordinate (Y), otherwise IN2 holds values of the sky latitude.

FORWRD = LOGICAL (Given)
If true then the forward mapping is used from image co-ordinate to sky co-ordinate. Otherwise, the inverse mapping from sky co-ordinate to image co-ordinates is used.

SCS = CHARACTER * (*) (Given)
The name of the sky co-ordinate system in which sky co-ordinates are required (if FORWRD is true), or supplied (if FORWRD is false). Any unambiguous abbreviation will do. This need not be the same as the SCS identified by IDA. See ID2 section " Sky Coordinates" for more information on Sky Co-ordinate Systems. A blank value will cause the system associated with IDA to be used.

IDA = INTEGER (Given)
The IRA identifier for the astrometry information.

OUT1(NVAL) = DOUBLE PRECISION (Returned)
If FORWRD is true, then OUT1 holds values of the sky longitude corresponding to the image co-ordinates given in arrays IN1 and IN2. Otherwise, OUT1 holds values of the first image co-ordinate (X) corresponding to the input sky co-ordinates.

OUT2(NVAL) = DOUBLE PRECISION (Returned)
If FORWRD is true, then OUT2 holds values of the sky latitude corresponding to the image co-ordinates given in arrays IN1 and IN2. Otherwise, OUT2 holds values of the second image co-ordinate (Y) corresponding to the input sky co-ordinates.

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

IRA_WRITE
Writes astrometry information into an HDS object

Description:

The astrometry information identified by IDA is stored within the object located by argument LOC. The constant IRA__HDSTY gives the HDS type required for the object. The object must have this type and be empty.

Invocation:

```
CALL IRA_WRITE( IDA, LOC, STATUS )
```

Arguments:

IDA = INTEGER (Given)
An IRA identifier for the astrometry information.

LOC = CHARACTER * (*) (Given)
A locator to the object to which the astrometry information is to be written.

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

KPG1_ABSET
Separates a list of items into a character array

Description:

The routine separates a list of items into an character array and finds the minimum number of characters required to specify each item uniquely, and the length in characters of the longest item of the list.

Invocation:

```
CALL KPG1_ABSET( SEPAR, OPTS, ARRAY, NELM, MINCH, MAXNOC, STATUS )
```

Arguments:

SEPAR = CHARACTER * (*) (Given)
The separator that divides the list of items.

OPTS = CHARACTER * (*) (Given)
Contains a list of items separated by the delimiter SEPAR. It is limited to 132 characters.

ARRAY(*) = CHARACTER * (*) (Returned)
The list divided into an array of values. The list is in alphabetic order.

NELM = INTEGER (Returned)
The number of elements in the list, and stored in ARRAY.

MCH(*) = CHARACTER * (*) (Returned)
The minimum number of initial characters to identify each of the items in the list uniquely. In the special case when an option equals the start of another option, MCH for the first option equals the option's length (ignoring trailing blanks) in characters; MCH for the the second will be at least one character greater.

MINCH = INTEGER (Returned)
The minimum number of initial characters that can identify an item in the list uniquely. In other words, the minimum value of the elements of MCH.

MAXNOC = INTEGER (Returned)
The maximum length of an item in the array, ignoring trailing blanks.

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

Notes:

- The size and dimension of the character array must be sufficient to accommodate the length and number of strings.
- Error status SAI__ERROR is returned when the list of options is ambiguous.

KPG1_AGATC
Reports the character attributes of the current picture in the graphics database

Description:

This routine inquires the name, comment, and label (if one is present) of the current picture in the AGI graphics database and reports their values.

Invocation:

```
CALL KPG1_AGATC( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_AGFND
Selects the highest picture of a given name within the current AGI picture

Description:

This routine searches forwards through the AGI database for a picture of a given name that lies within the current picture, including the current picture itself. If one is found it becomes the new current picture. If it could not be found a bad status will be returned, and the current picture is unchanged.

Invocation:

```
CALL KPG1_AGFND( NAME, PICID, STATUS )
```

Arguments:

NAME = CHARACTER * (*) (Given)

The name of the picture to be searched for in the graphics database.

PICID = INTEGER (Returned)

The picture identifier of the most-recent picture named NAME.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_AGREF
Obtains a name of or a locator to an object referenced in the graphics database

Description:

This routine determines whether a given picture in the AGI graphics database has an object associated with it by reference. If it has, the name of the object or a locator to the object is returned with the desired access mode.

Invocation:

```
CALL KPG1_AGREF( PICID, ACCESS, THERE, NAME, STATUS )
```

Arguments:

PICID = INTEGER (Given)

The identifier of the picture with which a data object may be associated by reference.

ACCESS = CHARACTER * (*) (Given)

Access mode to the object: 'READ', 'WRITE' or 'UPDATE'.

THERE = LOGICAL (Returned)

If true the picture has an associated object and the returned name is meaningful.

NAME = CHARACTER * (*) (Returned)

The name of the data object referenced by picture PICID, or a locator to that object. It should be ignored if THERE is .FALSE.. The value should be tested to see if it a locator or a name. If it is a locator, it should be annulled by REF_ANNUL. A reasonable number of characters should be allowed to accommodate a name including its path and section.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

The code is a little convoluted because of a bug in AGI_GTREF which means that a long name cannot be passed to it, if the reference is via a locator.

KPG1_AINBx
Obtains for an axis the equivalent index co-ordinates given axis values

Description:

This routine determines the indices within an axis array for a series of axis values. It assumes that the array is monotonically increasing or decreasing, and is approximately linear. This routine may be used for arbitrary 1-d arrays in addition to axes, provided these criteria are met.

A Newton's approximation method is used comparing the actual value with a linear approximation. The upper and lower bounds used to define the increment are adjusted given the deviation from the linear axis. Once the value lies between adjacent array elements the nearer (by linear interpolation) becomes the required index.

Invocation:

```
CALL KPG1_AINBx( LBND, UBND, AXIS, EL, VALUE, INDEX, STATUS )
```

Arguments:

LBND = INTEGER (Given)
The lower bound of the axis array.

UBND = INTEGER (Given)
The upper bound of the axis array.

AXIS(LBND:UBND) = ? (Given)
The axis array.

EL = INTEGER (Given)
The number of values whose indices in the axis are to be found.

VALUE(EL) = ? (Given)
The axis-array values.

INDEX(EL) = ? (Returned)
The pixel indices of the values in the axis array. Notice that this is floating as fractional positions may be required. An index is set to the bad value when its input co-ordinate lies outside the range of co-ordinates in the axis or when the input co-ordinate is bad.

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

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The axis array and values, and the returned indices should have this data type as well.
- No error report is made and bad status is not set should any input co-ordinate lie outside the range of co-ordinates of the axis. Processing will continue through the list.

KPG1_AINDx
Obtains for an axis the equivalent index co-ordinates given axis values

Description:

This routine determines the indices within an axis array for a series of axis values. It assumes that the array is monotonically increasing or decreasing, and is approximately linear. This routine may be used for arbitrary 1-d arrays in addition to axes, provided these criteria are met.

A Newton's approximation method is used comparing the actual value with a linear approximation. The upper and lower bounds used to define the increment are adjusted given the deviation from the linear axis. Once the value lies between adjacent array elements the nearer (by linear interpolation) becomes the required index.

Invocation:

```
CALL KPG1_AINDx( LBND, UBND, AXIS, EL, VALUE, INDEX, STATUS )
```

Arguments:

LBND = INTEGER (Given)
The lower bound of the axis array.

UBND = INTEGER (Given)
The upper bound of the axis array.

AXIS(LBND:UBND) = ? (Given)
The axis array.

EL = INTEGER (Given)
The number of values whose indices in the axis are to be found.

VALUE(EL) = ? (Given)
The axis-array values.

INDEX(EL) = ? (Returned)
The pixel indices of the values in the axis array. Notice that this is floating as fractional positions may be required.

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

Notes:

- There is a routine for the data types real or double precision: replace "x" in the routine name by R or D respectively, as appropriate. The axis array and values, and the returned indices should have this data type as well.

KPG1_AKERx
Smooths a two-dimensional array using an arbitrary rectangular kernel

Description:

The routine smooths the array A using an arbitrary smoothing kernel, and returns the result in the array B.

Invocation:

```
CALL KPG1_AKERx( IBOX, JBOX, WEIGHT, SAMBAD, WLIM, NX, NY, BAD, VAR, A, B, BADOUT, STATUS
)
```

Arguments:

IBOX = INTEGER (Given)

Half-size, in pixels along the x axis, of the box over which the smoothing will be applied (the actual box used has an edge which is 2∗IBOX+1 pixels long). This defines the region within which the smoothing function is non-zero.

JBOX = INTEGER (Given)

Half-size, in pixels along the y axis, of the box over which the smoothing will be applied (the actual box used has an edge which is 2∗JBOX+1 pixels long). This defines the region within which the smoothing function is non-zero.

WEIGHT(2 ∗ IBOX + 1, 2 ∗ JBOX + 1) = ? (Given)

The weighting kernel. It need not be normalised to one, but this is strongly preferred since it reduces the chance of overflows.

SAMBAD = LOGICAL (Given)

If a .TRUE. value is given, then any " bad" pixels in the input array A will be propagated to the output array B (output values will be calculated for all other output pixels). If a .FALSE. value is given, then the WLIM argument is used to determine which output pixels will be bad (if any). This argument is not relevant if BAD is .FALSE..

WLIM = ? (Given)

The minimum weighting that can be used to compute a smoothed output pixel if SAMBAD is .FALSE.. Any output pixels falling short of the specified weight will be set to the bad value (invalid pixels carry no weight, others have Gaussian weights about the central pixel). The value must be greater than 0.0 and should be less than or equal to 1.0. This argument is not used if SAMBAD is .TRUE. or if BAD is .FALSE..

NX = INTEGER (Given)

First dimension of the 2-d array A.

NY = INTEGER (Given)

Second dimension of the 2-d array A.

BAD = LOGICAL (Given)

Whether it is necessary to check for bad pixels in the input array A.

VAR = LOGICAL (Given)

If a .FALSE. value is given for this argument, then smoothing will be performed as if the array supplied (A) is an array of data and the PSF will be used directly as specified. If a .TRUE. value is given, then smoothing will be applied as if the array is an array of variance values associated with an array of data; in this case, the effective PSF will be reduced in width by a factor 2 and the mean output values will be reduced to reflect the variance-reducing effect of smoothing.

A(NX, NY) = ? (Given)
Array containing the input image to be smoothed.

B(NX, NY) = ? (Returned)
Array containing the input image after smoothing.

BADOUT = LOGICAL (Returned)
Whether there are bad pixel values in the output array.

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

Notes:

- There are routines for processing double precision and real data. Replace " x" in the routine name by D or R as appropriate. The data types of the WLIM, A, B, and WEIGHT arguments must match the routine used.
- This routine should not be used for symmetric kernels because there are faster algorithms for handling these special cases.

KPG1_ALIGN
Align a pair of 2-dimensional arrays using a least squares fit

Description:

This routine aligns a pair of 2-dimensional arrays using a least squares fit. The two arrays must be the same size and are assumed to be connected by an affine transformation, and to be in near alignment.

Invocation:

```
CALL KPG1_ALIGN( NX, NY, IPIN, IPREF, VIN, VREF, IPVIN, IPVREF, FORM, IFAC, RFAC, IOFF,
ROFF, FITVAL, C, STATUS )
```

Arguments:

NX = INTEGER (Given)
The length of the first pixel axis in each array

NY = INTEGER (Given)
The length of the second pixel axis in each array

IPIN = INTEGER (Given)
Pointer to a _DOUBLE array with dimensions (NX,NY), holding the Data values associated with the IN array.

IPREF = INTEGER (Given)
Pointer to a _DOUBLE array with dimensions (NX,NY), holding the Data values associated with the REF array.

VIN = LOGICAL (Given)
If TRUE, use IPVIN as weights within the minimisation.

VREF = LOGICAL (Given)
If TRUE, use IPVREF as weights within the minimisation.

IPVIN = INTEGER (Given)
Pointer to a _DOUBLE array with dimensions (NX,NY), holding the Variances associated with the IN array. Only used if VIN is .TRUE.

IPVREF = INTEGER (Given)
Pointer to a _DOUBLE array with dimensions (NX,NY), holding the Variances associated with the REF array. Only used if VREF is .TRUE.

FORM = INTEGER (Given)
The form of the affine transformation to use:

- 0: Full unrestricted 6 coefficient fit
- 1: Shift, rotation and a common X/Y scale but no shear.
- 2: Shift and rotation but no scale or shear.
- 3: Shift but not rotation, scale or shear.

IFAC = DOUBLE PRECISION (Given)
A factor by which the input values should be multipled before being used. Idealy, this should result in them having a standard deviation of unity.

RFAC = DOUBLE PRECISION (Given)
A factor by which the reference values should be multipled before being used. Idealy, this should result in them having a standard deviation of unity.

IOFF = DOUBLE PRECISION (Given)
An offset to subtract from the scaled input values before being used. Idealy, this should result in them having a mean of zero.

ROFF = DOUBLE PRECISION (Given)
An offset to subtract from the scaled reference values before being Used. Idealy, this should result in them having a mean of zero.

FITVAL = LOGICAL (Given)
If TRUE, then the fit is extended to include the relative scale factor and zero point offset between the pixle values in the two arrays.

C = DOUBLE PRECISION($*$) (Returned)
The coefficients of the affine transformation:

- Xin = C1 + C2$*$Xref + C3$*$Yref
- Yin = C4 + C5$*$Xref + C6$*$Yref

(Xin,Yin) are grid coordinates in IN, and (Xref,Yref) are grid coordinates in REF. If FITVAL is FALSE, this array should have at least 6 elements. Otherwise it should have at least 8 elements. The last two elements give the relationship between the pixel values in the two arrays:

- in = C7$*$ref + C8

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

KPG1_ALSYS
Allows the user to change the AlignSystem attribute in a Frame

Description:

This routine obtains a value from the environment that is used to specify a new value for the AlignSystem attribute of a supplied Frame. This attribute determines the co-ordinate system in which the Frame will align with other similar Frames.

Invocation:

```
CALL KPG1_ALSYS( PARAM, FRM1, FRM2, AXIS, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the parameter to use.

FRM1 = INTEGER (Given)

A pointer to the AST Frame which is to have its AlignSystem value changed.

FRM2 = INTEGER (Given)

A pointer to an AST Frame. The System value from this Frame will be used as the AlignSystem value for FRM1 if the user supplied the value " Data" for the parameter.

AXIS = INTEGER (Given)

The index (one or more) of a single axis within FRM1 which is to have its AlsignSystem value changed, or zero. If zero is supplied, the new AlignSystem value is applied to the whole Frame. Supplying an axis index allows a single Frame within a CmpFrame to have its AlignSystem value changed. If FRM1 is not a CmpFrame, then the supplied value is ignored and a value of zero is assumed.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ARCOG
Allows the user to select an array component in a supplied NDF

Description:

This routine allows the user to select an NDF array component selected from the ones available in the supplied NDF. Any of 'Data', 'Quality', 'Variance', and 'Error' can be selected if they are present in the NDF. The user is re-prompted until a valid component name is obtained, or an error occurs.

Invocation:

```
CALL KPG1_ARCOG( PARAM, INDF, MCOMP, COMP, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The parameter to use.

INDF = INTEGER (Given)
The NDF identifier.

MCOMP = CHARACTER * (*) (Returned)
The name of the selected array component for use with NDF_MAP. The returned values are 'Data', 'Quality', 'Variance', and 'Error' for the array components DATA, QUALITY, VARIANCE, and ERROR respectively. Returned equal to 'Data' if an error occurs.

COMP = CHARACTER * (*) (Returned)
Equal to MCOMP except that 'Variance' is substituted in place of 'Error'. Only NDF_MAP accepts 'Error' as a component name. All other NDF routines require 'Variance' to be specified instead and so should use COMP instead of MCOMP.

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

KPG1_ARCOL
Forms a list of the available array components in an NDF

Description:

This routine takes a list of array components and checks whether or not each is defined within an NDF, and if it is, copy the component name to the output list.

Invocation:

```
CALL KPG1_ARCOL( NDF, INLIST, EXLIST, LENGTH, STATUS )
```

Arguments:

NDF = CHARACTER * (*) (Given)

The identifier of the NDF to be inspected for certain array components.

INLIST = CHARACTER * (*) (Given)

A comma-separated list of the array components whose presence is to be tested. Valid component names are 'Data', 'Quality', 'Variance', and 'Error'. The component names may be abbreviated.

EXLIST = CHARACTER * (*) (Returned)

A comma-separated list of the array components of INLIST that are defined in the NDF. Its length should be as long as the fully expanded list of components. The components are tested in the order supplied. Note that mixed case components are returned so that they may be used in reports to users. Thus the returned values are 'Data', 'Quality', 'Variance', and 'Error' for the array components DATA, QUALITY, VARIANCE, and ERROR respectively.

LENGTH = INTEGER (Returned)

The effective length of the output list. Returned equal to 1 if an error occurs.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ASAGD
Transforms AGI world co-ordinates to AGI data co-ordinates

Description:

This routine transforms AGI world co-ordinates to AGI data co-ordinates using the TRANSFORM structure associated with the current AGI picture. It is intended for use as a transformation routine by an AST IntraMap.

Invocation:

```
CALL KPG1_ASAGD( THIS, NPOINT, NCOORD_IN, INDIM, IN, FORWARD, NCOORD_OUT, OUTDIM, OUT,
STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- Bad input co-ordinates will be set to AST__BAD, but TRANSFORM uses VAL__BADD to mark bad values. We assume here that VAL__BADD and AST__BAD are the same.

KPG1_ASALN
Set Alignment attributes for a Frame to mimic a second frame

Description:

This routine determines how the " IWCS1" FrameSet will align with other FrameSets. It sets the following attributes of the current Frame of IWCS1:

- AlignSystem is set to the value of System in IWCS2
- AlignStdOfRest is set to the value of StdOfRest in IWCS2
- AlignSideBand is set to the value of SideBand in IWCS2
- AlignTimeScale is set to the value of TimeScale in IWCS2
- AlignOffset is set to 1 if SkyRefIs in IWCS2 is not " Ignored" .
- AlignSpecOffset is set to 1 if SpecOrigin in IWCS2 is non-zero.

Invocation:

```
CALL KPG1_ASALN( IWCS1, IWCS2, STATUS )
```

Arguments:

IWCS1 = INTEGER (Given)
An AST pointer to the FrameSet to be modified.

IWCS2 = INTEGER (Given)
An AST pointer to the FrameSet defining the required attribute values.

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

KPG1_ASAPA
Determines which pixel axis is most closely aligned with a WCS axis

Description:

This routine aligns a supplied WCS axis to the best-matching pixel axis of an NDF. It also returns pixel co-ordinates corresponding to supplied WCS-axis limits, and where possible a mapping with one input and one output that transforms current Frame co-ordinates into pixel co-ordinates.

It first attempts to split the mapping to derive a one-to-one mapping. Failing that it

Invocation:

```
CALL KPG1_ASAPA( INDF, FRM, MAP, IAXIS, AXLOW, AXHIGH, PAXIS, PXLOW, PXHIGH, MAP1D,
STATUS )
```

Arguments:

INDF = INTEGER (Given)
The identifier of the NDF containing a WCS component.

FRM = INTEGER (Given)
An AST pointer to the current Frame in the NDF.

MAP = INTEGER (Given)
The mapping from the PIXEL Frame to the current WCS Frame. This can also be a mapping from GRID to the current Frame. In which case the returned co-ordinates and mapping apply to GRID co-ordinates instead of PIXEL.

IAXIS = INTEGER (Given)
The index of the WCS axis in the current Frame to be aligned.

AXLOW = DOUBLE PRECISION (Given)
Lower bound of the WCS axis. If either AXLOW or AXHIGH is set to AST__BAD, then the whole axis range is used.

AXHIGH = DOUBLE PRECISION (Given)
Upper bound of the WCS axis. If either AXLOW or AXHIGH is set to AST__BAD, then the whole axis range is used.

PAXIS = INTEGER (Returned)
The index of the pixel most closely aligned with the supplied WCS axis.

PXLOW = DOUBLE PRECISION (Given)
Lower bound of the PIXEL axis corresponding to the AXLOW in the WCS axis.

PXHIGH = DOUBLE PRECISION (Given)
Upper bound of the PIXEL axis corresponding to the AXHIGH in the WCS axis.

MAP1D = INTEGER (Returned)
The mapping from the current WCS Frame to PIXEL whose input is purely along the chosen axis (IAXIS). It is set to AST__NULL if the supplied mapping (MAP) could not be split.

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

KPG1_ASCRV
Draws a polyline using AST

Description:

This routine draw a polyline between the supplied positions. These positions are presumed to refer to the (two-dimensional)) Current co-ordinate Frame in the supplied AST Plot.

If FAST is .TRUE., the supplied positions are transformed into graphics co-ordinates, and the polyline is then drawn as a series of straight line segments using PGPLOT directly. Drawing performed with FAST = .TRUE. is buffered to increase efficiency. Call this routine with N=0 to flush the buffer.

If FAST is .FALSE., AST_CURVE is used to draw the polyline as a series of geodesic curves in the Current Frame of the Plot. This will take into account any discontinuities in the Mapping from the Current Frame to the graphics co-ordinate Frame, but will be slower.

Invocation:

```
CALL KPG1_ASCRV( IPLOT, FAST, N, X, Y, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
A pointer to a Plot.

FAST = LOGICAL (Given)
Is faster plotting required?

N = INTEGER (Given)
The number of points in the polyline. Zero to flush the buffer when drawing in fast mode.

X(N) = REAL (Given)
The X co-ordinates at the polyline points, in the Current Frame of the Plot.

Y(N) = REAL (Given)
The Y co-ordinates at the polyline points, in the Current Frame of the Plot.

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

KPG1_ASDIS
Finds the distance between two points

Description:

This routine returns the distance between two positions in the supplied co-ordinate Frame. Either the geodesic or Euclidean distance can be returned.

Invocation:

```
RESULT = KPG1_ASDIS( FRAME, DIM, NAX, POS, I1, I2, GEO, STATUS )
```

Arguments:

FRAME = INTEGER (Given)
An AST pointer to the Frame. Only accessed if GEO is .TRUE.

DIM = INTEGER (Given)
The size of the first dimension of the POS array.

NAX = INTEGER (Given)
The number of axes in the Frame.

POS(DIM, NAX) = DOUBLE PRECISION (Given)
An array holding the co-ordinates at DIM positions within the supplied Frame.

I1 = INTEGER (Given)
The index of the first position, in the range 1 to DIM.

I2 = INTEGER (Given)
The index of the second position, in the range 1 to DIM.

GEO = LOGICAL (Given)
Is the geodesic distance required?

STATUS = INTEGER (Given)
Global status value.

Returned Value:

KPG1_ASDIS = DOUBLE PRECISION
The distance between the two points.

KPG1_ASDSV
Finds the distances between a set of points

Description:

This routine returns the distance to each point in a set of points, from the first point, measured along the path joining the points. Geodesic distances within the supplied Frame are used.

Invocation:

```
CALL KPG1_ASDSV( FRM, NP, NAX, POS, NORM, DIS, BAD, STATUS )
```

Arguments:

FRM = INTEGER (Given)
An AST pointer to the Frame.

NP = INTEGER (Given)
The size of the first dimension of the POS array.

NAX = INTEGER (Given)
The number of axes in the Frame.

POS(NP, NAX) = DOUBLE PRECISION (Given)
An array holding the co-ordinates at NP positions within the supplied Frame.

NORM = LOGICAL (Given)
If .TRUE., the returned distances are normalised to a maximum value of 1.0.

DIS(NP) = DOUBLE PRECISION (Returned)
The distance along the path to each position, starting at the first position.

BAD = LOGICAL (Returned)
Are there any AST__BAD values in the returned array?

STATUS = INTEGER (Given)
Global status value.

KPG1_ASFFR
Finds an Frame with a given Domain within a FrameSet

Description:

This routine finds the last Frame with a given Domain within a FrameSet, and returns its index. The Current Frame in the FrameSet is not changed.

The first and last component Frames within CmpFrames are included in the search (component Frames in the middle of a CmpFrame cannot be found as yet). If a matching Frame is found within a CmpFrame, then a copy of the matching Frame is appended to the FrameSet. The returned Frame index refers to this extracted component Frame, rather than the CmpFrame from which it was extracted.

Invocation:

```
CALL KPG1_ASFFR( TARGET, DOMAIN, IFRM, STATUS )
```

Arguments:

TARGET = INTEGER (Given)
An AST pointer for a FrameSet containing the Frames to be searched.

DOMAIN = CHARACTER * (*) (Given)
The Domain name to be searched for.

IFRM = INTEGER (Returned)
The index of the matching Frame within the returned FrameSet. Returned equal to AST__NOFRAME if no match was found, or if an error occurs.

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

KPG1_ASFGT
Creates a new Frame specified through the environment

Description:

This routine creates a new AST Frame with properties specified by the given environment parameter. The parameter value is interpreted as an HDS path containing a WCS FrameSet in the form created by KPG1_WWRT. If this is succesful, the current Frame of the FrameSet is returned. Otherwise, an attempt is made to interpret the parameter value as an NDF name. If the NDF is opened succesfully, its current WCS Frame is returned. If this fails, and the parameter value ends with " .FIT" , am attempt is made to interpret the parameter value as the name of a FITS file. If successful, the primary WCS Frame from the primary HDU headers is returned. If the above attempt fails, an attempt is made to interpret the parameter value as the name of a text file containing either an AST Frame dump, or a set of FITS headers.

If all the above fails, and the parameter value looks like an IRAS90 " Sky Co-ordinate System" (SCS) specification, then a SkyFrame is returned with the properties specified by the SCS. Otherwise, a simple Frame is returned with Domain set to the parameter value, the number of axes in the Frame being specified by another environment parameter.

Invocation:

```
CALL KPG1_ASFGT( PDOM, PDIM, PEP, FRM, NAX, STATUS )
```

Arguments:

PDOM = CHARACTER * (*) (Given)
Name of parameter to use to get Frame Domain.

PDIM = CHARACTER * (*) (Given)
Name of parameter to use to get number of Frame axes. Only accessed if the value obtained for PDOM is not an IRAS90 SCS.

PEP = CHARACTER * (*) (Given)
Name of parameter to use to get the epoch of observation. Only accessed if the value obtained for PDOM is an IRAS90 SCS.

FRM = INTEGER (Returned)
An AST pointer to the returned Frame. Returned equal to AST__NULL if an error occurs.

NAX = INTEGER (Returned)
The number of axes in the returned Frame. Returned equal to zero if an error occurs.

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

KPG1_ASFIL
Reads spatial positions from a text file

Description:

This routine obtains formatted positions from a text file specified by an environment parameter. The positions are assumed to represent axis values in the supplied Frame.

The file should contain one position per line. Each position is given by a set of strings delimited by comma, space or tab (the first gives the value for Axis 1, the second for Axis 2, etc.). The number of strings per line should equal the number of axes in the supplied Frame. The user can specify the columns to use using Parameter PARAM2.

The file may contain blank lines, and comment lines commencing with " !" or " #" .

Invocation:

```
CALL KPG1_ASFIL( PARAM1, PARAM2, FRM, NP, IPOUT, FNAME, STATUS )
```

Arguments:

PARAM1 = CHARACTER * (*) (Given)
The name of an environment parameter to use to get the file.

PARAM2 = CHARACTER * (*) (Given)
The name of an environment parameter to use to get the indices of the columns within the text file which are to be used. If blank, the file must contain a column for every axis in FRM, all of which are used in the order 1, 2, 3, etc.

FRM = INTEGER (Given)
A pointer to an AST Frame.

NP = INTEGER (Returned)
The number of positions read from the file.

IPOUT = INTEGER (Returned)
A pointer to an _DOUBLE array " COR(NP, *)" holding the obtained co-ordinates. The second dimension of the array is equal to the number of axes in the suppleid Frame. Should be released using PSX_FREE when no longer needed.

FNAME = CHARACTER * (*) (Returned)
The file' s name. Not accesed if the declared length of the supplied string is 1 character.

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

KPG1_ASFIX
Modifies the WCS FrameSet of an NDF to take account of re-gridding the pixel array

Description:

This routine copies the WCS FrameSet from NDF1 to NDF2, re-mapping the PIXEL Frame using the specified Mapping in the process. It should be used to set up the WCS FrameSet of a newly created output NDF which has been formed by applying a geometric transformation to an input NDF.

Invocation:

```
CALL KPG1_ASFIX( MAP, INDF1, INDF2, STATUS )
```

Arguments:

MAP = INTEGER (Given)
An AST Mapping from pixel co-ordinates in the INDF1 to pixel co-ordinates in INDF2.

INDF1 = INTEGER (Given)
The input NDF.

INDF2 = INTEGER (Given)
The output NDF. Any existing WCS FrameSet is discarded and replaced by a re-mapped copy of the WCS FrameSet from the input NDF.

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

KPG1_ASFRM
Sets the current Frame in a FrameSet to a Frame specified through the environment

Description:

This routine allows the user to specify a new Current Frame for a FrameSet using an environment parameter.

Invocation:

```
CALL KPG1_ASFRM( PARAM, EPARAM, IWCS, WCDOM, DCDOM, PROMPT, TOKEN, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
Name of parameter to use to get Frame description.

EPARAM = CHARACTER * (*) (Given)
Name of parameter to use to get Epoch value (if needed).

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

WCDOM = CHARACTER * (*) (Given)
The Domain name to use if the user requests " WORLD" co-ordinates. No translation of WORLD takes place if a blank value is supplied.

DCDOM = CHARACTER * (*) (Given)
The Domain name to use if the user requests " DATA" co-ordinates. No translation of DATA takes place if a blank value is supplied.

PROMPT = LOGICAL (Given)
An error is always reported if the requested Frame is not available in the FrameSet. PROMPT controls what happens after the error has been reported. If .TRUE., then the error is flushed, the parameter is cancelled, and the user is re-prompted for a new %PARAM value. Otherwise, the error is retained, and the routine exits with STATUS set to SAI__ERROR.

TOKEN = CHARACTER * (*) (Given)
A string containing an MSG message token reference (e.g. " ^FRED"). The value of the token is used within error messages and should describe the object (NDF, catalogue, etc.) from which the supplied FrameSet is derived. If the token reference string is blank it is ignored.

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

Notes:

- This routine may add a new co-ordinate Frame into the FrameSet.
- If the FrameSet contains more than one Frame with the requested Domain, then the last matching Frame in the FrameSet will be used (i.e. the one with highest index).

Environment Parameters

%PARAM = LITERAL (Read)

A string specifying the new co-ordinate Frame. If a null parameter value is supplied, then the error is annulled and the current Frame is left unchanged. The string can be one of the following:

- A Domain name such as SKY, AXIS, PIXEL, etc. The two " pseudo-domains" WORLD and DATA may be supplied and will be translated into the values supplied for arguments WCDOM and DCDOM respectively, so long as the FrameSet does not contain Frames with Domains WORLD or DATA.
- An integer value giving the index of the required Frame within the WCS component.
- An IRAS90 Sky Co-ordinate System (SCS) values such as EQUAT(J2000) (see SUN/163).

%EPARAM = _DOUBLE (Read)

If a celestial co-ordinate system is supplied (using parameter %PARAM) then an epoch value is needed to qualify it. This is the epoch at which the supplied sky positions were determined. It should be given as a decimal years value, with or without decimal places (" 1996.8" for example). Such values are interpreted as a Besselian epoch if less than 1984.0 and as a Julian epoch otherwise.

KPG1_ASGDP
Finds a position with good output co-ordinates within a given input region of a supplied Mapping

Description:

This routine finds a position which has good co-ordinates in the output Frame of the given Mapping, and is a significant distance from the origin of both input and output Frame. It returns both the input and output co-ordinates at this position. The position is constrained to lie within a specified box within the input Frame. The first point to be tested is the centre of the box. If this does not pass the tests described above, a set of 10000 points randomly distributed within the box is tested. An error is reported if no good position can be found.

Invocation:

```
CALL KPG1_ASGDP( MAP, NDIM1, NDIM2, LBND, UBND, INPOS, OUTPOS, STATUS )
```

Arguments:

MAP = INTEGER (Given)
The Mapping to use.

NDIM1 = INTEGER (Given)
The number of input co-ordinates.

NDIM2 = INTEGER (Given)
The number of output co-ordinates.

LBND(NDIM1) = DOUBLE PRECISION (Given)
The lower bounds of the test box, within the input Frame of the supplied Mapping.

UBND(NDIM1) = DOUBLE PRECISION (Given)
The upper bounds of the test box, within the input Frame of the supplied Mapping.

INPOS(NDIM1) = DOUBLE PRECISION (Returned)
The returned input co-ordinates at the selected position. There will be no AST__BAD values in this array.

OUTPOS(NDIM2) = DOUBLE PRECISION (Returned)
The returned output co-ordinates at the selected position. There will be no AST__BAD values in this array.

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

KPG1_ASGET
Gets an AST FrameSet from the WCS component of an NDF

Description:

This routine determines the axes to be used from an NDF and returns a FrameSet representing the WCS information in the NDF.

Each axis of the supplied NDF is checked to see if it is significant (i.e. has a size greater than 1). The index of each significant axis is returned in SDIM, and the bounds of the axis are returned in SLBND and SUBND. If EXACT is .TRUE., an error is reported if the number of significant axes is not exactly NDIM. This mode is intended for case where (say) the user has supplied a single plane from a three-dimensional data cube to an application that requires a two-dimensional array.

If EXACT is .FALSE. an error is only reported if the number of significant dimensions is higher than NDIM. If there are fewer than NDIM significant dimensions then the insignificant dimensions are used (starting from the lowest) to ensure that the required number of dimensions are returned. This mode is intended for cases where (say) the user supplies a one-dimensional data stream to an application that requires a two-dimensional array.

The GRID Frame (i.e. the Base Frame) obtained from the NDFs WCS component is modified so that it has NDIM axes corresponding to the axes returned in SDIM (the value 1.0 is used for the other axes).

Likewise, the PIXEL Frame obtained from the NDFs WCS component is modified so that it has NDIM axes corresponding to the axes returned in SDIM (the lower pixel bound is used for the other axes). The original PIXEL Frame is retained, but with Domain changed to NDF_PIXEL.

If TRIM is .TRUE., then the Current Frame obtained from the NDFs WCS component is also modified so that it has NDIM axes. If the original Current Frame has more than NDIM axes, then the axes to use are obtained from the environment using parameter USEAXIS. A new Current Frame is then made by picking these axes from the original Current Frame, assigning the value AST__BAD to the axes which have not been chosen.

If the original Current Frame has fewer than NDIM axes, then simple axes are added into the new Current Frame to make up a total of NDIM. These axes are given the value 1.0.

Various environment parameters may be used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between applications.

Invocation:

```
CALL KPG1_ASGET( INDF, NDIM, EXACT, TRIM, REQINV, SDIM, SLBND, SUBND, IWCS, STATUS
)
```

Arguments:

INDF = INTEGER (Given)
The identifier for the NDF.

NDIM = INTEGER (Given)
The number of dimensions required by the application.

EXACT = LOGICAL (Given)
Must the NDF have exactly NDIM significant axes? Otherwise it is acceptable for the NDIM axes to include some insignificant axes.

TRIM = LOGICAL (Given)
Should the returned FrameSet be trimmed to ensure that the Current Frame has NDIM axes? Otherwise, the Current Frame read from the NDF is not changed.

REQINV = LOGICAL (Given)
Is the inverse mapping (from Current Frame to Base Frame) required? If it is, an error is reported if the inverse mapping is not available. REQINV should be supplied .TRUE. in most cases.

SDIM(NDIM) = INTEGER (Returned)
The indices of the significant dimensions.

SLBND(NDIM) = INTEGER (Returned)
The lower pixel index bounds of the significant dimensions. These are stored in the same order as the indices in SDIM.

SUBND(NDIM) = INTEGER (Returned)
The upper pixel index bounds of the significant dimensions. These are stored in the same order as the indices in SDIM.

IWCS = INTEGER (Returned)
An AST pointer to the WCS FrameSet. Returned equal to AST__NULL if an error occurs. The Base Frame is an NDIM-dimensional GRID Domain.

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

Notes:

- If the Current Frame in the returned FrameSet has no Title, then the Title is set to the value of the NDF TITLE component (so long as the NDF TITLE is not blank or undefined).

Environment Parameters

USEAXIS = LITERAL (Read)
A set of NDIM axes to be selected from the Current Frame. Each axis can be specified either by giving its index within the Current Frame in the range 1 to the number of axes in the Frame, or by giving its symbol. This parameter is only accessed if TRIM is .TRUE. and the original Current Frame in the supplied NDF has too many axes. The dynamic default selects the axes with the same indices as the selected NDF axes. The value should be given as a GRP group expression, with default control characters.

KPG1_ASGFR
Reads a line of an AST Object description from a GRP group

Description:

This routine reads a line of an AST Object description from a GRP group, removing the first character if it is a " #" or a " !" . It then returns the line of text to the AST library using AST_PUTLINE. It is intended to be used as a source function with AST_CHANNEL.

Invocation:

```
CALL KPG1_ASGFR( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ASGFW
Writes a line of an AST Object description to a GRP group

Description:

This routine writes a line of an AST Object description to a GRP group, optionally prepending it with the a given string. It is intended to be used as a sink function with AST_CHANNEL.

Invocation:

```
CALL KPG1_ASGFW( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ASGRD
Draws a border or an annotated co-ordinate grid over an AST Plot

Description:

This routine call AST_BORDER to draw a border, or AST_GRID to draw an annotated co-ordinate Grid over the supplied Plot. The current pgplot viewport can optionally be extended prior to drawing the grid so that it covers a specified AGI picture. If the pgplot viewport is left matching the plotting area supplied when the Plot was created, then certain component of the grid (i.e. exterior tick marks), are clipped by pgplot. To avoid this, IPIC should normally be given as the AGI identifier for the FRAME picture containing the plot.

Invocation:

```
CALL KPG1_ASGRD( IPLOT, IPIC, GRID, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
An AST pointer to the Plot.

IPIC = INTEGER (Given)
An AGI identifier for the FRAME picture. Supply this as -1 if the current pgplot viewport is not to be changed.

GRID = LOGICAL (Read)
Draw a grid using AST_GRID? If .FALSE. then a border only is drawn (using AST_BORDER).

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

Notes:

- The PGPLOT interface to the AGI library should be opened before calling this routine.

KPG1_ASGRP
Reads spatial positions from a GRP group

Description:

This routine reads formatted positions from a GRP group. The positions are assumed to be in the supplied Frame. Each element in the group should contain 1 position per line. Each position is given by a set of strings delimited by comma, space or tab (the first gives the value for Axis 1, the second for Axis 2, etc.). The number of strings per element in the group should equal the number of axes in the Base Frame of the supplied FrameSet.

An error is reported if any unreadable elements are found.

Invocation:

```
CALL KPG1_ASGRP( PARAM, FRM, IGRP, NP, NAX, OUT, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of an environment parameter to use to get the indices of the columns within the text file which are to be used. If blank, the file must contain exactly NAX columns, all of which are used. If a null value is supplied, the dynamic default values will be used, which is [1,2,3... NAX].

FRM = INTEGER (Given)
A pointer to an AST Frame.

IGRP = INTEGER (Given)
A GRP identifier for the group to read.

NP = INTEGER (Given)
The number of positions which can be stored in the returned array.

NAX = INTEGER (Given)
The number of axes in the supplied Frame.

OUT(NP, NAX) = DOUBLE PRECISION (Returned)
The array to hold the returned co-ordinates.

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

KPG1_ASIRA
Creates an AST FrameSet from an IRAS90 astrometry structure

Description:

This routine creates an AST Frame describing the sky co-ordinates stored in an IRAS90 astrometry structure (see SUN/165), together with a Mapping from IRAS90 " image co-ordinates" to sky co-ordinates.

Invocation:

```
CALL KPG1_ASIRA( IDA, FRM, MAP, STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- This routine requires access to the IRA internal common blocks, and uses IRA internal subroutines.

KPG1_ASLOG
Takes log base 10 of each axis value (an AST IntraMap routine)

Description:

This routine transforms axis values by taking the logarithm (base 10). The inverse exponentiates the supplied axis values. It is intended for use as a transformation routine by an AST IntraMap.

Invocation:

```
CALL KPG1_ASLOG( THIS, NPOINT, NCOORD_IN, INDIM, IN, FORWARD, NCOORD_OUT, OUTDIM, OUT,
STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- Bad input co-ordinates will be set to AST__BAD, but TRANSFORM uses VAL__BADD to mark bad values. We assume here that VAL__BADD and AST__BAD are the same.

KPG1_ASMRG
Merges two FrameSets by aligning them in a common Frame

Description:

This routine merges two FrameSet by aligning them in a suitable common Frame. The Current Frame in the second FrameSet becomes the Current Frame in the merged FrameSet. The domain search order for finding a suitable Frame is:

1) The domain of the Current Frame in IWCS2, if not blank. 2) " SKY" 3) " SPECTRUM" 4) " PIXEL" 5) " GRID" 6) The domain specified by argument DOMAIN, if not blank. If DOMAIN is blank, " AGI_WORLD" is used. 7) Any other suitable Frame.

For each of these Domains, the current Frame is checked first. An error is reported if alignment is not possible, and a message identifying the alignment Frame is displayed if alignment is possible.

If either FrameSet contains a second Frame with the same Domain as the alignment Frame then a warning is issued.

If the above attempt to align the Frames directly using astConvert fails, then a further attempt is made if one of the two Frames is a SpecFrame or SkyFrame, and the other Frame is a CmpFrame. In this case, the CmpFrame is searched for a Frame that can be aligned with the SkyFrame or SpecFrame. If this is succesful, the other axes in the CmpFrame are fed bad values by the Mapping which connects the two Frames.

Invocation:

```
CALL KPG1_ASMRG( IWCS1, IWCS2, DOMAIN, QUIET, IND, STATUS )
```

Arguments:

IWCS1 = INTEGER (Given)
An AST pointer to the first FrameSet. This is modified by adding all the Frames from IWCS2 into it. The Current Frame on exit is inherited from IWCS2.

IWCS2 = INTEGER (Given)
An AST pointer to the second FrameSet. The Current and Base Frames are unchanged on exit.

DOMAIN = CHARACTER * (*) (Given)
A comma separated list of domains in which alignment of the FrameSets should be attempted if alignment is not possible in the Current Frame of the second FrameSet, or SKY, SPECTRUM, PIXEL or GRID.

QUIET = LOGICAL (Given)
Suppress the message identifying the alignment Frame?

IND = INTEGER (Given)
The alignment message is padded with IND leading spaces.

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

KPG1_ASNDF
Creates a FrameSet containing NDF-special Frames with given bounds

Description:

This function creates a FrameSet containing the NDF-special Frames, GRID, PIXEL, FRACTION and AXIS, appropriate to an NDF with the supplied dimensionality and pixel index bounds. Optionally, AXIS information can be propagated from a supplied NDF.

Invocation:

```
CALL KPG1_ASNDF( INDF, NDIM, DIM, LBND, UBND, IWCS, STATUS )
```

Arguments:

INDF = INTEGER (Given)

An NDF from which to propagate AXIS information. May be NDF__NOID, in which case the AXIS Frame in the returned FrameSet will describe the default AXIS coordinate system (i.e. pixel co-ordinates).

NDIM = INTEGER (Given)

The number of pixel axes in the modified FrameSet.

DIM(NDIM) = INTEGER (Given)

The indices within INDF corresponding to each of the required NDIM axes.

LBND(NDIM) = INTEGER (Given)

The lower pixel index bounds in the modified FrameSet.

UBND(NDIM) = INTEGER (Given)

The upper pixel index bounds in the modified FrameSet.

IWCS = INTEGER (Returned)

Pointer to a new FrameSet holding GRID, FRACTION, PIXEL and AXIS Frames describing the supplied NDF bounds, plus AXIS information from the supplied NDF.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ASOFF
Finds a position offset by a given distance from one position towards another position

Description:

This routine returns the co-ordinates of a position which is a given distance along a curve joining one position to another position. The curve can be either be the geodesic or Euclidean curve joining the two points.

Invocation:

```
CALL KPG1_ASOFF( FRAME, DIM, NAX, POS, I1, I2, GEO, DIS, OFFPOS, STATUS )
```

Arguments:

FRAME = INTEGER (Given)
An AST pointer to the Frame. Only accessed if GEO is .TRUE.

DIM = INTEGER (Given)
The size of the first dimension of the POS array.

NAX = INTEGER (Given)
The number of axes in the Frame.

POS(DIM, NAX) = DOUBLE PRECISION (Given)
An array holding the co-ordinates at DIM positions within the supplied Frame.

I1 = INTEGER (Given)
The index of the first position, in the range 1 to DIM.

I2 = INTEGER (Given)
The index of the second position, in the range 1 to DIM.

GEO = LOGICAL (Given)
Is the geodesic distance required?

DIS = DOUBLE PRECISION (GIVEN)
The distance to move away from position I1 towards position I2.

OFFPOS(NAX) = DOUBLE PRECISION (Returned)
The returned position.

STATUS = INTEGER (Given)
Global status value.

KPG1_ASPCL
Applies clipping to a Plot so that border and co-ordinate grid are restricted to the well bahaved regions

Description:

This routine imposes clipping on a Plot (using the AST_CLIP routine) that restricts drawing to the regions that seem well bahaved. Specifically, drawing is restricted to a rectangular region within the base Frame of the Plot that is determined by transforming the supplied PIXEL bounding box into the current Frame, then converting it to the base Frame.

Invocation:

```
CALL KPG1_ASPCL( IPLOT, LBND, UBND, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
An AST pointer to the Plot.

LBND($*$) = INTEGER (Given)
The lower pixel index bounds of the Plot in the PIXEL Frame.

UBND($*$) = INTEGER (Given)
The upper pixel index bounds of the Plot in the PIXEL Frame.

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

KPG1_ASPLG
Stores a given KeyMap for future use by KPG1_ASPLN

Description:

This routine stores the supplied AST KeyMap pointer in common so that subsequent calls to KPG1_ASPLN can use it. If KPG1_ASPLN has been registered with a Plot using AST_GRFSET, then it will be called by AST whenever AST needs to draw a line. It will then draw the required line and store a description of the line in the AST KeyMap supplied to this routine. Consequently, this routine should usually be called before calling AST_GRFSET.

Invocation:

```
CALL KPG1_ASPLG( KEYMAP, BLEDGE, X1, X2, Y1, Y2 )
```

Arguments:

KEYMAP = INTEGER (Given)
The KeyMap in which to store descriptions of the lines drawn by the AST Plot class.

BLEDGE = LOGICAL (Given)
If TRUE, then do not draw lines that touch an edge of the box specified by X1, X2, Y1 and Y2.

X1 = REAL (Given)
The X value at the left hand edge. Unused if BLEDGE Is .FALSE.

X2 = REAL (Given)
The X value at the right hand edge. Unused if BLEDGE Is .FALSE.

Y1 = REAL (Given)
The Y value at the bottom edge. Unused if BLEDGE Is .FALSE.

Y2 = REAL (Given)
The Y value at the top edge. Unused if BLEDGE Is .FALSE.

KPG1_ASPLN
Draws a line for an AST Plot and log it at the same time

Description:

This routine is intended to be registered with an AST Plot (using AST_GRFSET) so that it is subsequently used to draw all lines. When called by AST, it calls the underlying PGPLOT line drawing function to draw the polyline specified by N, X and Y, and then records the details of the drawn polyline in an AST KeyMap (previously specified by calling KPG1_ASPLG). This KeyMap can then be interogated in order to determine what lines were drawn. A new entry is added to the KeyMap each time this function is called. Each such entry is itself a KeyMap containing three entries with keys " N" , " X" and " Y" . The " N" entry is a scalar integer holding the corresponding N value, and the other two are single precision vector entries holding the X and Y values.

KPG1_ASPLG should be called prior to registering this function using AST_GRFSET.

Invocation:

```
RESULT = KPG1_ASPLN( GRFCON, N, X, Y )
```

Arguments:

GRFCON = INTEGER (Given)
A KeyMap containing information passed from the calling application.

N = INTEGER (Given)
The number of pointsin the polyline.

X(N) = REAL (Given)
The graphics X coord at each point on the polyline.

Y(N) = REAL (Given)
The graphics Y coord at each point on the polyline.

Notes:

- The PGPLOT interface to the AGI library should be opened before calling this routine.

KPG1_ASPLT
Creates an AST Plot covering the current PGPLOT viewport

Description:

This routine create a Plot covering the current PGPLOT viewport. The bounds of the PGPLOT window are changed if necessary to ensure that the PGPLOT world co-ordinate system corresponds to millimetres from the bottom-left corner of the view surface. This is the co-ordinate system used in the Base (GRAPHICS) Frame of the returned Plot.

Invocation:

```
CALL KPG1_ASPLT( IWCS, BOX, OPTS, IPLOT, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
The FrameSet to include in the Plot. May be AST__NULL.

BOX(4) = DOUBLE PRECISION (Given)
An array holding the bounds of the area within the Base Frame of IWCS which is to be mapped linearly onto the current PGPLOT viewport. The first pair of values should give the co-ordinates at the bottom-left corner of the plotting area and the second pair should give the co-ordinates at the top-right corner. The co-ordinate on the horizontal axis should be given first in each pair.

OPTS = CHARACTER * (*) (Given)
A set of Plot attribute settings to be used when creating the Plot.

IPLOT = INTEGER (Returned)
An AST pointer to the Plot. Returned equal to AST__NULL if an error occurs.

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

Notes:

- The PGPLOT interface to the AGI library should be opened before calling this routine.

KPG1_ASPRP
Propagates the WCS component from one NDF to another with the same number of axes, allowing for a linear mapping of the pixel co-ordinates

Description:

This routine copies the WCS FrameSet from INDF1, re-mapping the GRID Frame in the process so that pixel co-ordinates in the output NDF are related to pixel co-ordinates in the input NDF by the supplied linear transformation. The mapping from pixel co-ordinates in INDF1 (" PIX1") to the corresponding pixel co-ordinates in INDF2 (" PIX2") is:

PIX2 = MATRIX . PIX1 + OFFSET

For instance, for NDIM = 2:

X2 = MATRIX(1, 1).X1 + MATRIX(2, 1).Y1 + OFFSET(1) Y2 = MATRIX(1, 2).X1 + MATRIX(2, 2).Y1 + OFFSET(2)

Invocation:

```
CALL KPG1_ASPRP( NDIM, INDF1, INDF2, MATRIX, OFFSET, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
The number of dimensions. This should be the value returned by NDF_BOUND for INDF2.

INDF1 = INTEGER (Given)
An identifier for the source NDF. If this does not have NDIM pixel axes, a NDF section with NDIM axes will be obtained from the supplied NDF.

INDF2 = INTEGER (Given)
An identifier for the destination NDF. This must have NDIM pixel axes.

MATRIX(NDIM, NDIM) = DOUBLE PRECISION (Given)
The matrix connecting PIX1 and PIX2.

OFFSET(NDIM) = DOUBLE PRECISION (Given)
The offset vector for PIX2.

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

KPG1_ASPSY
Establishes synonyms for AST attribute names

Description:

This routine establishes synonyms for AST attribute names or attribute qualifiers (qualifiers are strings in parenthesise following the attribute name - the " graphical elements" used by some Plot attributes are examples of qualifiers). The routine KPG1_ASCHP is used to translated synonyms into values recognised by AST.

Note, substitutions for synonyms are performed in the order in which they are defined, with later substitutions potentially modifying the results of earlier substitutions. For this reason, synonyms for specific name/qualifer pairs (eg" FORMAT(VEC∗TOR)") should be defined ∗before∗ synonyms for qualifiers alone (eg " (VEC∗TOR)"). If these two examples were defined in the opposite order, then the qualifier in FORMAT(VEC) (i.e. " VEC") would get replaced by the translation supplied for synonym " (VEC∗TOR)" , so that the resulting string would then fail to match the synonym " FORMAT(VEC∗TOR)" and so would not result in the correct translation.

Invocation:

```
CALL KPG1_ASPSY( SYNON, TRAN, STATUS )
```

Arguments:

SYNON = CHARACTER ∗ (∗) (Given)

A synonym for an AST attribute name and/or qualifier. The supplied value is converted to upper case, and stripped of spaces. If the synonym contains just a qualifier (i.e. no attribute name), then KPG1_ASCHP will substitute the qualifier from the translation irespective of the attribute name. Minimum abbreviations for attribute qualifiers may be given by including an asterisk in the qualifier to mark the end of the minimum abbreviation. If a blank value is supplied, then the resources used to store the synonyms and translations are released.

TRAN = CHARACTER ∗ (∗) (Given)

The translation for the supplied synonym. This should be a legal AST attribute name/qualifier combination (but no check is made on this). The supplied value is converted to upper case, and stripped of spaces.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ASPTP
Puts a formatted AST position into a string

Description:

This routine puts a formatted position into a text string starting at a specified index within the string. Axis symbols may optionally be included. The axis values are separated by a specified string.

Invocation:

```
CALL KPG1_ASPTP( FRAME, NAX, POS, SYMBLS, SEP, TEXT, IAT, STATUS )
```

Arguments:

FRAME = INTEGER (Given)
The Frame in which the position is defined.

NAX = INTEGER (Given)
The number of axes in the Frame.

POS(NAX) = DOUBLE PRECISION (Given)
The position to format.

SYMBLS = LOGICAL (Given)
Are axis symbols to be included? If so, each axis value is formatted with a " symbol=value" string.

SEP = CHARACTER * (*) (Given)
The separator for axis values. Trailing spaces are significant.

TEXT = CHARACTER * (*) (Given and Returned)
The text to hold the formatted values.

IAT = INTEGER (Given and Returned)
On entry, the index of the last character before the point at which the text is to be placed. On exit, the index of the last character written by this routine.

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

KPG1_ASREF
Associates an NDF optionally from a reference name or locator

Description:

This routine obtains an NDF. There is a search path of sources for the NDF checked in the following order: a) command line, b) a reference locator or name, c) elsewhere in the parameter system (usually by prompting.) For a) and c) the NDF is obtained by association. For b) a locator or name is imported into the NDF_ system; an error will result if the locator does not point to a valid NDF. The role of this routine is to permit automatic processing of NDFs associated with database pictures. The command-line access allows the NDF reference stored with the last DATA picture to be overridden.

Invocation:

```
CALL KPG1_ASREF( PNNDF, MODE, GOTNAM, NAME, NDF, STATUS )
```

Arguments:

PNNDF = CHARACTER * (*) (Given)
The name of the Starlink parameter to be associated with the input NDF. May be blank, in which case no attempt will be made to obtain the NDF form the environment.

MODE = CHARACTER * (*) (Given)
Access mode to the NDF required: ' READ' or ' UPDATE' .

GOTNAM = LOGICAL (Given)
If .TRUE., a name or locator to a potential NDF has already been obtained and is supplied to this routine through the NAME argument. The name or locator may come from a reference in the graphics database.

NAME = CHARACTER * (*) (Given)
The name of the potential NDF, or a locator to it.

NDF = INTEGER (Returned)
The identifier to the NDF.

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

Prior Requirements :

- The parameter should not have been associated before calling this routine.

KPG1_ASREG
Registers all AST IntraMaps known by KAPPA

Description:

This routine registers all AST IntraMaps known to KAPPA. It should be called before any AST routine which may use an IntraMap (such as a transformation routine, plotting routine, read/write routine, etc.).

Invocation:

```
CALL KPG1_ASREG( STATUS )
```

Arguments:

STATUS = INTEGER (Given)

Global status value.

KPG1_ASRGG
Registers all graphical AST IntraMaps known by KAPPA

Description:

This routine registers all graphical AST IntraMaps known to KAPPA. It should be called before any AST routine which may use an IntraMap (such as a transformation routine, read/write routine, etc.).

Invocation:

```
CALL KPG1_ASRGG( STATUS )
```

Arguments:

STATUS = INTEGER (Given)
Global status value.

Notes:

Use KPG1_ASREG to register all IntraMaps (graphical and non-graphical). Use KPG1_ASRGN to register just non-graphical IntraMaps.

KPG1_ASRGN
Registers all non-graphical AST IntraMaps known by KAPPA

Description:

This routine registers all non-graphical AST IntraMaps known to KAPPA. It should be called before any AST routine which may use an IntraMap (such as a transformation routine, read/write routine, etc.).

Invocation:

```
CALL KPG1_ASRGN( STATUS )
```

Arguments:

STATUS = INTEGER (Given)

Global status value.

Notes:

Use KPG1_ASREG to register all IntraMaps (graphical and non-graphical). Use KPG1_ASRGG to register just graphical IntraMaps.

KPG1_ASSET
Allows the specification of attribute values for an AST Object

Description:

This routine allows the user to set attributes for an AST Object (see SUN/210). The value to which an attribute is set in the returned Object is determined as follows.

- If the user supplies a value for the attribute using the given parameter, then the Object is returned with the attribute set to the supplied value. The user supplies the attribute values as a GRP group expression in which each element is an AST attribute setting. The group expression may comprise persistent and/or temporary attributes, the former preceding the latter separated by a delimiting string (see argument PARAM). Persistent values are recorded in the parameter's current value and so can be re-used, whereas temporary attributes are not recorded and only apply to the current invocation of a task.
- Otherwise, if the Object already had an explicit value set for the attribute on entry (i.e. if AST_TEST returns .TRUE. for the attribute), then the value is unchanged on exit.
- Otherwise, a search is made for a default value for the attribute using the search path described below. If a default value is found for the attribute then the Object is returned with the attribute set to the default value. The use of these defaults can be suppressed by including the string " CLEAR" as the first element in the group of attribute values supplied for the specified environment parameter.

Defaults are specified as a group of attribute setting strings within a " defaults" text file. This file is found using the following search path:

1) If the environment variable <APP>_<PARAM> is defined (<APP> and <PARAM> in upper case), its value is taken to be the full path to the defaults file.

2) If <APP>_<PARAM> is not defined, the file $HOME/<app>_<param>.def is used (<app> and <param> in lower case).

3) If the file $HOME/<app>_<param>.def cannot be accessed, the file $KAPPA_DIR/<app>_<param>.def is used.

4) If the file $KAPPA_DIR/<app>_<param>.def cannot be accessed, the value of environment variable KAPPA_<PARAM> is taken to be the full path to the defaults file.

5) If KAPPA_<PARAM> is not defined, the file $HOME/kappa_<param>.def is used.

6) If the file $HOME/kappa_<param>.def cannot be accessed, the file $KAPPA_DIR/kappa_<param>.def is used.

Each attribute setting within a group of settings should be of the form " <name>=<value>" , where <name> is taken to be the name of a Object attribute (or synonym set by KPG1_ASPSY), and <value> is taken to be the value to assign to the attribute. These attributes are described in SUN/210. No error is reported if unrecognised attribute names or illegal attribute values are specified.

Before being used, the attribute settings are edited to replace any synonyms by their corresponding AST attribute names established by earlier calls to KPG1_ASPSY. Colour names are also replaced by corresponding PGPLOT colour indices.

Invocation:

```
CALL KPG1_ASSET( APP, PARAM, IOBJ, STATUS )
```

Arguments:

APP = CHARACTER * (*) (Given)
The name of the calling application in the form <package>_<application> (e.g. " KAPPA_DISPLAY"), for use in messages.

PARAM = CHARACTER * (*) (Given)
The name of the parameter to use for getting the group expression.

A non-alphanumeric prefix may prepend the actual parameter name. If present, this signifies the delimiter to separate persistent from temporary attributes, otherwise all attributes are regarded as persistent. The prefix may be one to three characters long. Underscore is regarded as alphanumeric. The standard delimiter is the plus sign.

The same non-alphanumeric delimiter may also appear as a suffix. Its presence indicates that this routine is being called more than once in an application for the same parameter. The suffix should appear in all but the last invocation.

IOBJ = INTEGER (Given and Returned)
An AST pointer to the Object to be modified. Returned unchanged if an error occurs.

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

Notes:

- Colour attribute values may be supplied in any form recognised by KPG1_PGCOL (e.g. colour name, MIN, MAX, integer index), and the nearest colour in the current KAPPA palette is used.
- If a null value is supplied for the parameter, the error is annulled and the Object is returned unchanged (except for any defaults obtained using the usual search path).

KPG1_ASSHR
Shrinks a Plot so that it covers an area which allows all annotation to fit within the specified area

Description:

This routine creates a new Plot covering the same window as the current PGPLOT window, but the window is shrunk in GRAPHICS space so that all the annotation produced by AST_GRID falls within the PGPLOT viewport which is current on entry. The sizes of annotations, gaps, etc. are shrunk if this is necessary in order to fit the annotations within the current PGPLOT viewport.

Invocation:

```
CALL KPG1_ASSHR( ASP, F, X1, X2, Y1, Y2, JUST, RJUST, IPLOT, OK, STATUS )
```

Arguments:

ASP = LOGICAL (Given)

The aspect ratio of the required plotting area (i.e. excluding annotation) after shrinking. If this is zero or negative, the largest possible area is used for the plotting area.

F = REAL (Given)

An amount by which to extend the margins left for annotation, expressed as a factor of the height or width of the plotting area. For instance, a value of 0.1 could be given to fit the annotation " comfortably" into the Plot. A value of 0.0 will result in the annotation being hard up against the edge of the plot.

X1 = REAL (Given)

The GRAPHICS X co-ordinate at the left edge of the area into which the annotation is to fit.

X2 = REAL (Given)

The GRAPHICS X co-ordinate at the right edge of the area into which the annotation is to fit.

Y1 = REAL (Given)

The GRAPHICS Y co-ordinate at the bottom edge of the area into which the annotation is to fit.

Y2 = REAL (Given)

The GRAPHICS Y co-ordinate at the top edge of the area into which the annotation is to fit.

JUST = CHARACTER*2 (Given)

Indicates the justification of the new plot within the specified area. 'BL', 'BC', 'BR', 'CL', 'CC', 'CR', 'TL', 'TC' or 'TR', where B is Bottom, C is Centre, T is Top, L is Left and R is Right. Only used if ASP > 0. Must be upper case. If either character is a space, then the corresponding value from RJUST is used instead. Other unrecognised values are treated as " C" .

RJUST(2) = REAL (Given)

Each element is used only if the corresponding element in JUST is a apce. The first element gives the fractional vertical position of the new plot: 0.0 means put the new plot as low as possible, 1.0 means put it as high as possible. The second element gives the fractional horizontal position of the new plot: 0.0 means put the new plot as far to the left as possible, 1.0 means put it as far to the right as possible.

IPLOT = INTEGER (Given and Returned)

The Plot. The supplied Plot is annulled and a new one is returned in its place. The new Plot contains all the Frames of the supplied Plot, but its Plot attributes are all cleared. It may be necessary for the caller to re-instate these attributes.

OK = LOGICAL (Returned)
Returned .FALSE. if there was insufficient room for the required Plot. No error is reported in this case, and IPLOT os returned unchanged.

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

KPG1_ASSIG
Ensures that the Current Frame from an NDF WCS FrameSet has no insignificant axes

Description:

This routine looks for insignificant axes in the Current Frame of the supplied WCS FrameSet (an axis is insignificant if all pixels within the NDF pixel array has the same position on the axis). If any insignificant axes are found, a new Frame is added to the FrameSet containing only the significant axes from the Current Frame. This new Frame is added into the FrameSet and becomes the new Current Frame. The PermMap which connects it to the original Current Frame assigns the correct constant values to the insignificant axes whn used in the inverse direction.

Invocation:

```
CALL KPG1_ASSIG( IWCS, NDIM, LBND, UBND, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
The WCS FrameSet from the NDF, as returned by KPG1_GTWCS.

NDIM = INTEGER (Given)
The number of dimensions in the NDF, as returned by NDF_BOUND.

LBND(NDIM) = INTEGER (Returned)
The lower pixel index bounds of the NDF, as returned by NDF_BOUND.

UBND(NDIM) = INTEGER (Returned)
The upper pixel index bounds of the NDF, as returned by NDF_BOUND.

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

KPG1_ASSIM
Simplifies a FrameSet

Description:

This routine simplifies a FrameSet by adding a copy of the Current Frame into it, using a simplified version of the Mapping from Base to Current Frame. The new Frame becomes the Current Frame.

This routine should be called before doing any plotting with a Plot (remember, a Plot is a FrameSet) in order to avoid the possiblity of intermediate Frames being used in which positions are undefined.

Care should be taken decding where to call this routine since the simplification process can be expensive. Do not call it within a deep nested loop!

Invocation:

```
CALL KPG1_ASSIM( IWCS, STATUS )
```

Arguments:

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

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

KPG1_ASSIR
Sets the attributes of a SkyFrame to match an IRAS90 SCS

Description:

This routine modifies the supplied skyframe so that it describes the celestial co-ordinate system given by the supplied IRAS90 " Sky Co-ordinate System" specified (see SUN/163). A .FALSE. function value is returned if the supplied string is not a valid IRAS90 SCS, but no error is reported.

Invocation:

```
RESULT = KPG1_ASSIR( FRM, SCS, PEP, STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Returned Value:

KPG1_ASSIR = LOGICAL

.TRUE. if the supplied SCS string was a valid IRAS90 Sky Co-ordinate System specifier, and ,FALSE. otherwise.

KPG1_ASSMP
Returns co-ordinates at evenly spaces positions along a given poly-line

Description:

This routine returns an a pointer to an array holding the co-ordinates at a set of NSAMP positions evenly spaced along a poly-line defined by NPOS " profile positions" .

Invocation:

```
CALL KPG1_ASSMP( FRAME, ARRDIM, NAX, NPOS, POS, GEO, NSAMP, SAMP, DELTA, STATUS )
```

Arguments:

FRAME = INTEGER (Given)

An AST pointer to the Frame in which the polyline is defined. This is used to define geodesic curves joining the supplied profile positions. Only accessed if GEO is .TRUE.

ARRDIM = INTEGER (Given)

The size of the first dimension of POS.

NAX = INTEGER (Given)

The number of axes in the Frame.

NPOS = INTEGER (Given)

The number of profile positions defining the poly-line. This must be less than or equal to ARRDIM.

POS(ARRDIM, NAX) = DOUBLE PRECISION (Given)

The profile positions. The first axis indexes the position number, and the first NPOS elements should be used. The second axis indexes the axis number. An error is reported if any invalid positions are supplied.

GEO = LOGICAL (Given)

Should the poly-line be constructed from geodesic curves in the supplied Frame? If not, the poly-line is made up of straight-line segments in the supplied Frame.

NSAMP = INTEGER (Given)

The number of samples required along the poly-line.

DELTA = DOUBLE PRECISION (Given and Returned)

The increment between samples. If not known, this should be set to zero. Returned holding the used increment.

SAMP(NSAMP, NAX) = DOUBLE PRECISION (Returned)

The array holding the sample positions.

STATUS = INTEGER (Given)

Global status value.

KPG1_ASSPL
Gets a set of one-dimensional Mappings for each axis in a FrameSet

Description:

This routine returns a set of AST Mapping pointers. Each Mapping has one input and one output. The I' th Mapping goes from Axis I in the Base Frame of the supplied FrameSet, to Axis I in the Current Frame of the FrameSet.

There should usually be a one-to-one correspondance between the axes in the Base and Current Frames in the FrameSet.

Invocation:

```
CALL KPG1_ASSPL( IWCS, MXAX, MAP, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
The AST pointer to the FrameSet.

MXAX = INTEGER (Given)
The maximum number of mappings to be returned. Un-used elements are returned holding AST__NULL.

MAP(MXAX) = INTEGER (Returned)
The Mapping pointers.

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

KPG1_ASSTS
Applies an attribute setting to a Plot

Description:

This routine applies the supplied attribute setting to the supplied Plot. The attribute setting should be of the form " name=value" where " name" is an AST attribute name or a synonym for an AST attribute name established by a call to KPG1_ASPSY, and " value" is the value to assign to the attribute. If the attribute name contains either COLOR or COLOUR then the value string is checked to see if it is the name of a colour, and if so, the corresponding colour index is used instead. The resulting setting, after translation of synonyms and colour names is applied to the supplied Plot. If the Plot already has a set value for the specified attribute, then the behaviour depends on OVER; if OVER is .TRUE. then the new attribute value over-writes the value already in the Plot; if OVER is .FALSE. then the supplied setting is ignored.

Invocation:

```
CALL KPG1_ASSTS( SETTNG, REPORT, OVER, IPLOT, BADAT, STATUS )
```

Arguments:

SETTING = CHARACTER * (*) (Given)
The attribute setting string.

REPORT = LOGICAL (Given)
Should an error be reported if the attribute setting string is not legal?

OVER = LOGICAL (Given)
Over-write existing attribute values in the Plot?

IPLOT = INTEGER (Given)
An AST pointer to the Plot to be modified.

BADAT = LOGICAL (Returned)
Was the setting string invalid? If so, an appropriate error message will have been reported (unless REPORT is .FALSE.).

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

Notes:

- Colour attribute values may be supplied in any form recognised by KPG1_PGCOL (e.g. colour name, MIN, MAX, integer index), and the nearest colour in the current KAPPA pallette is used.

KPG1_ASSTY
Checks for synonyms and colour names in AST attribute settings

Description:

This routine splits the supplied AST attribute setting string up into a name and a value, replacing synonyms for AST attribute names or qualifiers with the corresponding AST names and qualifiers, and replacing colour names within the attribute value with corresponding PGPLOT colour indices. Synonyms for AST attribute names or qualifiers are set up using KPG1_ASPSY.

Matching of attribute names and attribute qualifiers are performed separately. The names are matched first. An attempt to match any supplied attribute qualifier against a synonym is only made if the attribute names match, or if the synonym does not contain an attribute name. Synonyms may specify minimum abbreviations for attribute qualifiers by including an asterisk within the qualifier string. The asterisk marks the end of the minimum abbreviation.

Invocation:

```
CALL KPG1_ASSTY( SETTNG, IQUAL, NAME, VALUE, NQUAL, STATUS )
```

Arguments:

SETTNG = CHARACTER * (*) (Given)
The text to be checked, potetially containing synonyms and colour names.

IQUAL = INTEGER (Given)
If the attribute name includes one or more qualifiers (e.g. " colour(ticks,border)"), then the returned name includes the qualifier with index IQUAL (starting at one). The IQUAL value is ignored if the attribute name contains no qualifiers.

NAME = CHARACTER * (*) (Returned)
The corresponding AST attribute name (including at most one qualifier, as selected by IQUAL).

VALUE = CHARACTER * (*) (Returned)
The corresponding AST attribute value.

NQUAL = INTEGER (Returned)
The number of qualifers included in the attribute name within the supplied SETTNG string. May be zero.

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

KPG1_AST2H
Copies AST_ data to an HDS object

Description:

This routine copies a line of text representing AST_ data into a specified element of a one-dimensional character array. It is intended for use when writing AST_ data to an HDS object (i.e an HDS _CHAR array).

Invocation:

```
CALL KPG1_AST2H( DATA, ILINE, LINE, STATUS )
```

Arguments:

DATA(*) = CHARACTER * (*) (Given and Returned)
The character array into which the text is to be copied.

ILINE = INTEGER (Given)
The index of the element in DATA which is to receive the text (the contents of other elements are returned unchanged).

LINE = CHARACTER * (*) (Given)
The line of text to be inserted.

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

Notes:

This routine departs from the conventional argument order so as to accommodate the case where the DATA argument is a mapped HDS character array.

KPG_ASTCMN
Defines functional accessor/setter interface to KPG_AST common block

Description:

This file contains a set of functions that enable access to the KPG_AST common block from outside of the KPG library. Routines are only added on demand.

Invocation:

```
CALL KPG1_SETASTxxx or VALUE = KPG1_GETASTxxx
```

Notes:

Since the routines are only 3 lines long, they are all in a single file with minimal comment wrapper. Types of the argument for SET methods match the type in the common block.

Implementation Details :

The following routines are available: KPG1_SETASTLN(LN) - Set the ASTLN integer KPG1_SETASTGRP(GRP) - Set the ASTGRP identifier KPG1_SETASTGSP(CHAR) - Set ASTGSP character KPG1_SETASTDSB(DRDSB) - Set DRWDSB flag KPG1_SETASTFIT(FIT) - Set FIT flag KPG1_SETASTPLN(PLN) - KeyMap used to store details of lines drawn KPG1_SETASTBLE(BLE) - Are plot edges to be kept free of ticks etc? KPG1_SETASTX1(X1) - Left hand X value in Plot. KPG1_SETASTX2(X2) - Right hand X value in Plot. KPG1_SETASTY1(Y1) - Bottom Y value in Plot. KPG1_SETASTY2(Y2) - Top Y value in Plot. KPG1_GETASTNPS() KPG1_GETASTING() KPG1_GETASTOUG() KPG1_GETASTDSB() KPG1_GETASTFIT() KPG1_GETASTPLN() KPG1_GETASTBLE() KPG1_GETASTX1() KPG1_GETASTX2() KPG1_GETASTY1() KPG1_GETASTY2()

KPG1_ASTRM
Trims axes from the current Frame of a FrameSet

Description:

This routine ensures that the number of axes in the current Frame of the supplied FrameSet is the same as the number in the base Frame. If this is not the case on entry, a new Frame with the required number of axes is created and added into the FrameSet and becomes the new current Frame.

If the original current Frame has too few axes, the new Frame is a copy of the original current frame with extra simple axes added to the end. These extra axes are supplied a value of AST__BAD by the Mapping which connects the original current Frame to the new current Frame.

If the original current Frame has too many axes, the user is allowed to select a subset of the original axes using the environment parameter USEAXIS (see below). A new Frame is created by picking these selected axes from the original current Frame. This Frame is added into the FrameSet using a Mapping which has a forward transformation which simply drops the values for the unselected axes. The inverse transformation (from new to old Frame) attempts to assign usable values for the dropped axes if possible. This is only possible if the value for a dropped axis can be determined uniquely from the value of one of the selected axes. This may be the case for instance in a situation where (RA,wavelength) axes were selected from the (RA,Dec,Wavelength) axes describing a 2D longslit spectrum. The missing Dec value can probably be determined from the RA value because the relationship between RA and Dec is determined by the position and orientation of the slit on the sky. If it is not possible to determine the value for a dropped axis in this way, then what happens depends on whether or not any Frames can be found in the FrameSet that are Regions having a Domain name of " ROI<n>" , where " <n>" is a positive integer. If no such Frame can be found, AST__BAD is supplied for the dropped axis.

If the supplied current Frame has too many axes, and one or more ROI Regions are found in the FrameSet, then a new Frame is added into the FrameSet for each ROI Region found. These new Frames are all equivalent, containing axes from the supplied current Frame as specified by the USEAXIS parameter, and they are all connected to the supplied current Frame via a PermMap. Each PermMap has a forward transformation that simply drops the unwanted axis values. The inverse transformation of each PermMap supplies suitable values for the unwanted axes. These are determined by transforming the bounding box of the corresponding ROI Region into the original current Frame. The Domain name of the corresponding ROI Region is stored in the Ident attribute of the new Frame, and is also appended to the end of the Frame' s Domain. The new Frame corresponding to the lowest numbered ROI Region is left as the current Frame on exit.

Various environment parameters may be used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between applications.

Invocation:

```
CALL KPG1_ASTRM( IWCS, DEFAX, LBND, UBND, WORK, STATUS )
```

Arguments:

IWCS = INTEGER (Given)

The FrameSet to use. A new current Frame may be added to the FrameSet by this routine.

DEFAX($*$) = INTEGER (Given)

This array should have one element for each axis in the base Frame of the supplied FrameSet. The i' th value is the index within the original current Frame of the axis which is to be associated with the i' th base Frame axis by default. Only used if no better defaults can be found by splitting the FrameSet Mapping.

LBND(*) = INTEGER (Given)
The lower pixel bound on each pixel axis. Array length should be at least equal to the number of base Frame axes in IWCS.

UBND(*) = INTEGER (Given)
The upper pixel bound on each pixel axis. Array length should be at least equal to the number of base Frame axes in IWCS.

WORK(*) = INTEGER (Given)
Work space. It' s length should be at least twice as large as the largest pixel dimension implied by LBND and UBND.

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

Environment Parameters

USEAXIS = LITERAL (Read)
A set of NDIM axes to be selected from the current Frame. Each axis can be specified either by giving its index within the current Frame in the range 1 to the number of axes in the Frame, or by giving its symbol. This parameter is only accessed if the original current Frame in the supplied FrameSet has too many axes. The value should be given as a GRP group expression, with default control characters.

KPG1_ASTTL
Sets the Title attribute of a Plot

Description:

This routine sets the Title attribute of an AST Plot. The Title string can be obtained from a variety of places, and the following priority order is used:

The value of the Title attribute supplied by the user via the application's STYLE parameter is given top priority. If no Title was supplied then the Title component of the NDF is used. If this is blank, then the TItle attribute stored in the current Frame of the NDF's WCS FrameSet is used, if it has been set explicitly (i.e. is not just a default value). Otherwise, the name of the NDF is used.

Invocation:

```
CALL KPG1_ASTTL( IPLOT, IWCS, INDF, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
An AST pointer for the Plot. The Title attribute of the current Frame will be set on exit.

IWCS = INTEGER (Given)
An AST pointer for the WCS FrameSet obtained from the NDF.

INDF = INTEGER (Given)
An NDF identifier for the NDF.

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

KPG1_AVLUT
Associates, validates and maps an lookup table stored in an NDF

Description:

This routine associates for read access an NDF that is presumed to contain a lookup table in its data array. A series of validation checks are made: the array must be two-dimensional, the first dimension must be 3, the range of values must lie in the range 0.0–1.0. The last of these requires that the data array is mapped therefore for convenience and efficiency a pointer and length are returned. The lookup table mapped with type _REAL.

Invocation:

```
CALL KPG1_AVLUT( PNLUT, NDFL, PNTRI, EL, STATUS )
```

Arguments:

PNLUT = CHARACTER * (*) (Given)
The ADAM parameter to be associated with the NDF.

NDFL = INTEGER (Returned)
The identifier for the NDF containing the lookup table.

PNTRI(1) = INTEGER (Returned)
The pointer to the mapped NDF lookup table.

EL = INTEGER (Returned)
The length of the mapped NDF.

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

KPG1_AXANO
Generates an axis annotation from the NDF's axis label and units

Description:

This routine examines a nominated axis for a label and units. It creates a string of the form " label (units)" or " label" if the label but not the units are present. If neither are present a supplied default is returned instead.

Invocation:

```
CALL KPG1_AXANO( NDF, IAXIS, DEFAUL, AXSLAB, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

IAXIS = INTEGER (Given)
The number of the axis whose character components are to be used.

DEFAUL = CHARACTER * (*) (Given)
A default to return in AXSLAB should there be no axis label.

AXSLAB = CHARACTER * (*) (Returned)
The axis annotation.

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

Prior Requirements :

- The identifier should be associated with an NDF.

KPG1_AXBNx
Finds the bounds of an NDF's axis centre co-ordinates

Description:

This routine determines the lowest and highest centre co-ordinate for an NDF axis array component. Currently, it assumes that these will be the first and last elements.

Invocation:

```
CALL KPG1_AXBNx( EL, CENTRE, AXLCO, AXUCO, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the axis.

CENTRE(EL) = ? (Given)
The NDF axis centre co-ordinates.

AXLCO = ? (Returned)
The lowest centre co-ordinates for the axis.

AXUCO = ? (Returned)
The highest centre co-ordinate for the axis.

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

Notes:

- There is a routine for the data types real or double precision: replace "x" in the routine name by R or D respectively, as appropriate. The centre array and the returned values should have this data type as well.

KPG1_AXCOx
Obtains for an axis the axis indices given their values

Description:

This routine determines floating-point axis indices within an axis array for a series of pixel co-ordinates. It assumes that the array is monotonic and approximately linear, since it uses linear interpolation to derive the pixel indices. This routine may be used for arbitrary 1-d arrays in addition to axes, provided this criterion is met.

Invocation:

```
CALL KPG1_AXCOx( LBND, UBND, AXIS, EL, VALUE, INDEX, STATUS )
```

Arguments:

LBND = INTEGER (Given)
The lower bound of the axis array.

UBND = INTEGER (Given)
The upper bound of the axis array.

AXIS(LBND:UBND) = ? (Given)
The axis array.

EL = INTEGER (Given)
The number of indices whose values in the axis array are to be found.

VALUE(EL) = ? (Given)
The axis-array values.

INDEX(EL) = ? (Returned)
The pixel co-ordinates of the values in the axis array. Notice that this is in floating point as fractional positions may be returned. An index is set to the bad value when its input co-ordinate lies outside the range of co-ordinates in the axis.

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

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The axis array and co-ordinates, and the returned indices should have this data type as well.
- An error report is made and bad status returned should any input co-ordinate lie outside the range of co-ordinates of the axis. Processing will continue through the list.

KPG1_AXEXx
Calculates the extent of an NDF along an axis

Description:

This routine calculates the starting and ending positions of an NDF' s pixels along an axis, optionally taking account of the axis width and error values.

Invocation:

```
CALL KPG1_AXEXx( EL, CENTRE, USEERR, ERROR, USEWID, WIDTH, NSIGMA, ASTART, AEND, STATUS
)
```

Arguments:

EL = INTEGER (Given)
The number of elements in the axis arrays.

CENTRE(EL) = ? (Given)
The centres of the pixels on the axis.

USEERR = LOGICAL (Given)
Use the error array and NSIGMA in the calculation of the extent of the axis.

ERROR(EL) = ? (Given)
The errors of the pixel centres on the axis. It is only required (accessed) when USEERR = .TRUE..

USEWID = LOGICAL (Given)
Use the width array in the calculation of the extent of the axis.

WIDTH(EL) = ? (Given)
The widths of the pixels on the axis.

NSIGMA = ? (Given)
Number of multiples of the error to use in the calculation

ASTART = ? (Returned)
If the axis centre positions increase with NDF pixel index, this argument returns the axis position of the edge of the first pixel which has the lower co-ordinate. Otherwise it returns the axis position of the edge with the higher co-ordinate.

AEND = ? (Returned)
If the axis centre positions increase with NDF pixel index, this argument returns the axis position of the edge of the last pixel which has the higher co-ordinate. Otherwise it returns the axis position of the edge with the lower co-ordinate.

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

KPG1_AXGVx
Finds the first axis centre co-ordinate of an NDF above a threshold

Description:

This routine determines the element number and value of the first axis centre co-ordinate of an NDF above a given threshold. Currently, it assumes that the centre values increase or decrease monotonically from the first to the last elements.

A SAI__ERROR status is returned when no centre value exceeds the threshold.

Invocation:

```
CALL KPG1_AXGVx( EL, CENTRE, THRESH, ITH, VALTH, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the axis.

CENTRE(EL) = ? (Given)
The NDF axis centre co-ordinates.

THRESH = ? (Given)
The threshold.

ITH = INTEGER (Returned)
The index number of the element that first exceeds the threshold, or its negative when the co-ordinates decrease with increasing element number.

VALTH = ? (Returned)
The centre co-ordinate that first exceeds the threshold.

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

Notes:

- There is a routine for real and double-precision data types: replace "`x`" in the routine name by R or D respectively. The array and threshold supplied to the routine plus the co-ordinate that first exceeds the threshold must have the data type specified.

KPG1_AXLIx
Determines whether an array's values are equally spaced

Description:

This routine determines whether or not adjacent elements of a 1-d array have values that are equally spaced, i.e. it tests for linearity. It simply checks if the intervals between all successive pairs of elements are the same within the machine precision.

Invocation:

```
CALL KPG1_AXLIx( EL, ARRAY, LVAL, UVAL, LINEAR, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the array. It must be at least two.

ARRAY(EL) = ? (Given)
The array to be tested.

LVAL = ? (Returned)
Value of the first array element. If this is bad an estimated value is substituted when the array is linear.

UVAL = ? (Returned)
Value of the last array element. If this is bad an estimated value is substituted when the array is linear.

LINEAR = LOGICAL (Returned)
True if the array is linear.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for most numeric data types: replace "x" in the routine name by D, R, I, W, or UW as appropriate. The array (and the variables for the first and last array elements) supplied to the routine must have the data type specified.

KPG1_AXLVx
Finds the first axis centre co-ordinate of an NDF below a threshold

Description:

This routine determines the element number and value of the first axis centre co-ordinate of an NDF below a given threshold. Currently, it assumes that the centre values increase or decrease monotonically from the first to the last elements.

A SAI__ERROR status is returned when no centre value is less than the threshold.

Invocation:

```
CALL KPG1_AXLVx( EL, CENTRE, THRESH, ITH, VALTH, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the axis.

CENTRE(EL) = ? (Given)
The NDF axis centre co-ordinates.

THRESH = ? (Given)
The threshold.

ITH = INTEGER (Returned)
The index number of the element that first has a value below the threshold, or its negative when the co-ordinates decrease with increasing element number.

VALTH = ? (Returned)
The centre co-ordinate that first fails to exceed the threshold.

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

Notes:

- There is a routine for real and double-precision data types: replace "`x`" in the routine name by R or D respectively. The array and threshold supplied to the routine plus the co-ordinate that first fails to exceed the threshold must have the data type specified.

KPG1_AXRNG
Calculates the extent of an NDF along an axis

Description:

This routine calculates the starting and ending positions of an NDF's pixels along an axis, taking account of the axis width values.

Invocation:

```
CALL KPG1_AXRNG( EL, CENTRE, WIDTH, ASTART, AEND, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the axis arrays.

CENTRE(EL) = DOUBLE PRECISION (Given)
The centres of the pixels on the axis.

WIDTH(EL) = DOUBLE PRECISION (Given)
The widths of the pixels on the axis.

ASTART = DOUBLE PRECISION (Returned)
If the axis centre positions increase with NDF pixel index, this argument returns the axis position of the edge of the first pixel which has the lower co-ordinate. Otherwise it returns the axis position of the edge with the higher co-ordinate.

AEND = DOUBLE PRECISION (Returned)
If the axis centre positions increase with NDF pixel index, this argument returns the axis position of the edge of the last pixel which has the higher co-ordinate. Otherwise it returns the axis position of the edge with the lower co-ordinate.

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

KPG1_AXTYP
Determines the implementation type for NDF axis arrays

Description:

The routine returns the highest precision required to process the NDF axis component in all dimensions. ' _DOUBLE' is returned if any of the arrays have type ' _INTEGER' or ' DOUBLE'.

Invocation:

```
CALL KPG1_AXTYP( NDF, COMP, ATYPE, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

COMP = CHARACTER * (*) (Given)
The name of the axis array component whose implementation type is required: ' CENTRE', ' VARIANCE', or ' WIDTH'.

ATYPE = CHARACTER * (*) (Returned)
The implementation type of the axis arrays. It is in uppercase and is either ' _REAL' or ' _DOUBLE'.

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

Prior Requirements :

The NDF identifier must be valid and there must be an axis structure.

KPG1_AXVLx
Obtains the axis-array values given their corresponding pixel co-ordinates

Description:

This routine determines values within an axis array for a series of non-integer pixel co-ordinates. It assumes that the array is monotonic and approximately linear, since it uses linear interpolation to derive the axis values. This routine may be used for arbitrary one-dimensional arrays in addition to axes, provided this criterion is met.

Invocation:

```
CALL KPG1_AXVLx( LBND, UBND, AXIS, EL, PIXCO, DATCO, STATUS )
```

Arguments:

LBND = INTEGER (Given)
The lower bound of the axis array.

UBND = INTEGER (Given)
The upper bound of the axis array.

AXIS(LBND:UBND) = ? (Given)
The axis array.

EL = INTEGER (Given)
The number of pixel co-ordinates whose values in the axis array are to be found.

PIXCO(EL) = ? (Given)
The pixel co-ordinates of the values in the axis array.

DATCO(EL) = ? (Returned)
The axis-array values. A value is set to the bad value when its input co-ordinate lies outside the range of co-ordinates in the axis.

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

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The axis array and indices, and the returned values should have this data type as well.
- A pixel co-ordinate that lies within the lower-bound or upper-bound element of the axis array but not between elements, and hence cannot have interpolation (i.e. PIXCO is less than LBND
- 0.5 or greater than UBND + 0.5) return with the centre data co-ordinate of the lower-bound or upper-bound pixel respectively.
- An error report is made and bad status returned should any input index lie outside the range of the bounds of the axis. Processing will continue through the list.

KPG1_BADBX
Obtains an NDF section containing all good data in the supplied NDF

Description:

This routine finds the pixel bounding box that encloses all good data values in the DATA array of supplied NDF. It then either creates and returns an NDF section corresponding to this bounding box, or sets the pixel bounds of the supplied NDF to match the bounding box.

Invocation:

```
CALL KPG1_BADBX( INDF1, OPER, INDF2, NGOOD, STATUS )
```

Arguments:

INDF1 = INTEGER (Given)

The input NDF identifier. Note, if OPER is 1 or 2 then any mapped access to the NDF will be unmapped on exit. In addition, if OPER is 2 then any mapped access to NDFs that are located within an extension of the supplied NDF will be unmapped on exit. Also, for OPER 1 and 2, " UPDATE" access is required to the NDF, and (for OPER 2) any extension NDFs.

OPER = INTEGER (Given)

Indicates how the box should be used.

1 - the bounds of the supplied NDF will be modified to match the bounding box enclosing the good data.

2 - the bounds of the supplied NDF will be modified to match the bounding box enclosing the good data. In addition, any NDFs found within the MORE component of the supplied NDF, which have bounds equal to those of the supplied NDF, are changed to match the bounds of the bounding box.

If any other value is supplied for OPER, INDF2 will be returned holding an NDF identifier for a section of the supplied NDF matching the bounding box.

INDF2 = INTEGER (Returned)

An identifier for the smallest NDF section that contains all good DATA values in the the input NDF. Returned equal to NDF__NOID if OPER is 1, or if an error occurs.

NGOOD = INTEGER (Returned)

The number of good DATA values in the supplied NDF. Returned equal to zero if an error occurs.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_BBOXx
Determines the bounding box of selected pixels in an n-dimensional array

Description:

This routine compresses an n-dimensional array by integer factors along each dimension by summing the array values in a rectangular box. The output may be normalised to take account of any bad values that may be present.

Invocation:

```
CALL KPG1_BBOXx( NDIM, LBND, UBND, ARRAY, VALUE, EQUAL, BLBND, BUBND, NPASS, STATUS
)
```

Arguments:

NDIM = INTEGER (Given)
The dimensionality of the n-dimensional array.

LBND(NDIM) = INTEGER (Given)
The lower pixel index bounds of the input n-dimensional array.

UBND(NDIM) = INTEGER (Given)
The upper pixel index bounds of the input n-dimensional array.

ARRAY(*) = ? (Given)
The input n-dimensional data array.

VALUE = ? (Given)
The data value used to define the box. VAL__BADx is a legal value for this argument.

EQUAL = LOGICAL (Given)
If .TRUE., then the returned box encloses all pixels with value equal to VALUE. If .FALSE., then the returned box encloses all pixels with value not equal to VALUE.

BLBND(NDIM) = INTEGER (Returned)
The lower pixel index bounds of the required box.

BUBND(NDIM) = INTEGER (Returned)
The upper pixel index bounds of the required box.

NPASS = INTEGER (Returned)
The number of array elements that pass the test specified by VALUE and EQUAL. In general, the number of elements in the returned bounding box will be greater than NPASS.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by D or R as appropriate. The input and output data arrays plus a work space must have the data type specified.

Bugs:

{note_bugs_here}

KPG1_BILNR
Performs bi-linear interpolation checking for bad values

Description:

An imaginary pixel is centred on the supplied interpolation position. The area of the overlap between this imaginary pixel and each of the four surrounding real pixels is found, and used as the weight for the corresponding pixel value. The returned interpolated value is the weighted mean of the four surrounding pixel values. A bad value is returned if any of these four pixels lie outside the image or have a bad value.

The associated variance value is also returned if input variances are supplied.

Invocation:

```
CALL KPG1_BILNR( X, Y, XLO, XHI, YLO, YHI, USEVAR, DATA, VAR, INTERP, INTERV, STATUS
)
```

Arguments:

X = REAL (Given)
The pixel X co-ordinate at which to perform the interpolation.

Y = REAL (Given)
The pixel Y co-ordinate at which to perform the interpolation.

XLO = INTEGER (Given)
The lower bound on the array X axis.

XHI = INTEGER (Given)
The upper bound on the array X axis.

YLO = INTEGER (Given)
The lower bound on the array Y axis.

YHI = INTEGER (Given)
The upper bound on the array Y axis.

USEVAR = LOGICAL (Given)
Should a variance be returned?

DATA(XLO:XHI, YLO:YHI) = REAL (Given)
The data array to be interpolated.

VAR(XLO:XHI, YLO:YHI) = REAL (Given)
The variance array (only accessed if USEVAR is .TRUE.).

INTERP = REAL (Returned)
The interpolated data value.

INTERV = REAL (Returned)
The interpolated variance value.

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

KPG1_BL1Dx
Smooths a one-dimensional vector using a block-average filter

Description:

This routine smooths a one-dimensional vector using a block-average filter. Output pixels are set bad if all input pixels in the block are bad, or if the output pixel is in the first or last half block.

Invocation:

```
CALL KPG1_BL1Dx( N, IN, SIZE, OUT, STATUS )
```

Arguments:

N = INTEGER (Given)
The number of elements of the array to be smoothed.

IN(N) = ? (Given)
The input array.

SIZE = INTEGER (Given)
The size of the filter (in pixels).

OUT(N) = ? (Returned)
The output array.

STATUS = INTEGER (Given & Returned)
Global status.

Notes:

- There is a routine for real and double-precision data types: replace "x" in the routine name by R or D respectively. The input and output arrays supplied to the routine must have the data type specified.

KPG1_BLOCx
Smooths a two-dimensional image using a rectangular box filter

Description:

The routine smooths a two-dimensional image using a rectangular box filter; each pixel is replaced by the mean of those good neighbours which lie within a box of specified size.

Invocation:

```
CALL KPG1_BLOCx( BAD, SAMBAD, VAR, NX, NY, A, IBOXX, IBOXY, NLIM, B, BADOUT, ASUM, NSUM,
STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether it is necessary to check for bad pixels in the input image.

SAMBAD = LOGICAL (Given)
If a .TRUE. value is given for this argument, then bad input pixels will be propagated to the output image unchanged (a smoothed output value will be calculated for all other pixels). If a .FALSE. value is given, then the NLIM argument determines whether an output pixel is good or bad. The value of SAMBAD is not relevant if BAD is .FALSE..

VAR = LOGICAL (Given)
If a .FALSE. value is given for this argument, then the smoothing applied will be appropriate to a data image. If a .TRUE. value is given, then the smoothing will be appropriate to an image containing variance values. In the latter case the output values will be (on average) smaller than the input values to take account of the variance-reducing effect which smoothing produces.

NX = INTEGER (Given)
First dimension of the image to be smoothed.

NY = INTEGER (Given)
Second dimension of the image to be smoothed.

A(NX, NY) = ? (Given)
Input image to be smoothed.

IBOXX = INTEGER (Given)
Half-size of the smoothing box in pixels in the X direction (the actual size of the box used will be $2*\mathrm{IBOXX}+1$ pixels).

IBOXY = INTEGER (Given)
Half-size of the smoothing box in pixels in the Y direction (the actual size of the box used will be $2*\mathrm{IBOXY}+1$ pixels).

NLIM = INTEGER (Given)
Minimum number of good pixels which must be present in the smoothing box in order to calculate a smoothed output pixel. If this minimum number is not satisfied, then a bad output pixel will result. A value between 1 and the total number of pixels in the smoothing box should be supplied.

B(NX, NY) = ? (Returned)
The smoothed output image.

BADOUT = LOGICAL (Returned)
Whether bad pixels are present in the output image.

ASUM(NX) = ? (Returned)
Workspace for the pixel sums.

NSUM(NX) = INTEGER (Returned)
Workspace for counting good pixels.

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

Notes:

- There is a routine for processing single- and double-precision arrays; replace " x" in the routine name by R or D as appropriate. The data type of the A, B and ASUM arguments must match the routine used.

KPG1_BOR2x
Places a border of constant values at the edges of a two-dimensional array

Description:

This routine assigns a constant value to the edge pixels of a two-dimensional array. The width of the edge is adjustable along each axis, but it is the same for the leading and trailing edges along a given dimension. If the border is wider than the array a SAI__ERROR status is returned.

Invocation:

```
CALL KPG1_BOR2x( VALUE, BORWID, DIM1, DIM2, ARRAY, STATUS )
```

Arguments:

VALUE = ? (Given)
Value to be assigned to the borders of the array.

BORWID(2) = INTEGER (Given)
The width in pixels of the borders in each dimension, x then y. Each must be fewer than its corresponding dimension.

DIM1 = INTEGER (Given)
The first dimension of the array to have constant-valued peripheries.

DIM2 = INTEGER (Given)
The second dimension of the array to have constant-valued peripheries.

ARRAY(DIM1, DIM2) = ? (Given and Returned)
The array to have its borders set to a constant.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array and value to be substituted that are supplied to the routine must have the data type specified.

KPG1_CADDx
Adds a constant to each element of a vectorised array

Description:

The routine adds a constant to each element of a vectorised array to produce a new array. Bad value checking is performed if required.

Invocation:

```
CALL KPG1_CADDx( BAD, EL, A, CONST, B, NERR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = ? (Given)
Input array.

CONST = DOUBLE PRECISION (Given)
Constant to be added to each array element.

B(EL) = ? (Returned)
Output array.

NERR = INTEGER (Returned)
Number of numerical errors which occurred while processing the array.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.
- This routine will handle numerical errors (i.e. overflow) by assigning the appropriate " bad" value to affected output array elements. If the constant supplied cannot be converted to the data type of the arrays without overflow, then all elements of the output array will be assigned this bad value and NERR will return the value of EL.

KPG1_CCPRO
Gets a character component for an output NDF with optional propagation from another NDF

Description:

This routine uses the parameter sysyem to obtain a value for a selected character component of an output or updated NDF. If the null value is supplied, the character component is copied from an input NDF to the output NDF, unless the component is undefined, in the input, in which case it is left undefined in the output.

Invocation:

```
CALL KPG1_CCPRO( PNCOMP, COMP, NDFI, NDFO, STATUS )
```

Arguments:

PNCOMP = CHARACTER * (*) (Given)
The name of the ADAM parameter used to obtain the character component's value.

COMP = CHARACTER * (*) (Given)
The name of the character component. It must be 'TITLE', 'LABEL', or 'UNITS'.

NDFI = INTEGER (Given)
The identifier of the input NDF from which a character component is to be copied to the output NDF.

NDFO = INTEGER (Given)
The identifier of the output or updated NDF to which a character component is to be written.

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

KPG1_CEIL
Returns the smallest integer larger than or equal to a supplied value

Description:

This routine returns the smallest integer larger than or equal to a supplied value.

Invocation:

```
RESULT = KPG1_CEIL( VALUE )
```

Arguments:

VALUE = REAL (Given)

The value.

Function Value :

KPG1_CEIL = INTEGER The smallest integer larger than or equal to the supplied value.

KPG1_CGET
Obtains an NDF character component, removing escape sequences

Description:

The routine obtains the value of the specified character component of an NDF (i.e. the value of the LABEL, TITLE or UNITS component). It is identical to NDF_CGET except that any PGPLOT or AST escape sequences in the string are removed.

Invocation:

```
CALL KPG1_CGET( INDF, COMP, VALUE, STATUS )
```

Arguments:

INDF = INTEGER (Given)
NDF identifier.

COMP = CHARACTER * (*) (Given)
Name of the character component whose value is required: 'LABEL', 'TITLE' or 'UNITS'.

VALUE = CHARACTER * (*) (Given and Returned)
The component's value.

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

Notes:

- If the requested component is in an undefined state, then the VALUE argument will be returned unchanged. A suitable default should therefore be established before calling this routine.
- If the length of the VALUE argument is too short to accommodate the returned result without losing significant (non-blank) trailing characters, then this will be indicated by an appended ellipsis, i.e. '...'. No error will result.

KPG1_CH2PM
Creates an AST PolyMap describing a Starlink POLYNOMIAL HDS structure

Description:

This routine creates an AST PolyMap that implements the polynomial transformation described by a supplied Starlink POLYNOMIAL structure (see SGP/38).

Invocation:

```
CALL KPG1_CH2PM( LOC, POLYMAP, STATUS )
```

Arguments:

LOC = CHARACTER * (DAT__SZLOC) (Given)
An HDS locator for the POLYNOMIAL structure.

POLYMAP = INTEGER (Returned)
An AST pointer to the new PolyMap.

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

Notes:

- The returned PolyMap has a defined forward transformation (equivalent to the supplied POLYNOMIAL), but no inverse transformation.
- Both CHEBYSHEV and SIMPLE variants of the POLYNOMIAL structure are supported. But currently only 1- or 2- dimensional Chebyshev polynomials can be handled. An error is reported for Chebyshev polynomials of higher dimensionality.

KPG1_CHAXx
Checks for usable AXIS structures

Description:

This routine looks for monotonic AXIS structures within the specified axes of the supplied NDF. If all axes have such AXIS structures, then a flag (DATAVL) is returned .TRUE.. If any axis is non-monotonic, a warning message is issued, DATAVL is returned .FALSE., and a scale of 1.0 and offset of 0.0 are returned.

Each axis is then checked for linearity. If the axis is linear, then the corresponding scale and offset of the linear mapping from pixel to data co-ordinates are returned. Otherwise a warning message is issued and the returned scale and offset refer to a linear approximation to the axis co-ordinate system.

Invocation:

```
CALL KPG1_CHAXx( INDF, NDIM, DIM, DATAVL, SCALE, OFFSET, STATUS )
```

Arguments:

INDF = INTEGER (Given)
An identifier for the NDF.

NDIM = INTEGER (Given)
The number of axes.

DIM(NDIM) = INTEGER (Given)
The indices of the axes to be checked, in increasing order.

DATAVL = LOGICAL (Returned)
Returned .TRUE. if the NDF contains monotonic AXIS structures for all requested axes. Returned .FALSE. otherwise.

SCALE(NDIM) = ? (Returned)
The scale factors in the linear relationships between axis co-ordinates and pixel co-ordinates. Returned equal to 1.0 if DATVAL is returned .FALSE..

OFFSET(NDIM) = ? (Returned)
The offsets in the linear relationships between axis co-ordinates and pixel co-ordinates. Returned equal to 0.0 if DATVAL is returned .FALSE..

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

Notes:

- There is a routine for each of the real and double-precision data types: replace " x" in the routine name by D or R as appropriate. The SCALE and OFFSET arrays supplied to the routine must have the data type specified.
- The returned values of SCALE and OFFSET are such that:

DATA = SCALE(I) $*$ PIXEL $+$ OFFSET(I)

where PIXEL is a pixel co-ordinate for the I' th dimension listed in array DIM (i.e. dimension DIM(I)), and DATA is the corresponding axis co-ordinate.

KPG1_CHE2X
Evaluates a two-dimensional Chebyshev polynomial

Description:

This routine evaluates a two-dimensional Chebyshev polynomial for one or more arguments. It uses Clenshaw's recurrence relationship twice.

Invocation:

```
CALL KPG1_CHE2X( NPTS, XMIN, XMAX, X, YMIN, YMAX, Y, XDEG, YDEG, NCOEF, CC, NW, WORK,
EVAL, STATUS )
```

Arguments:

NPTS = INTEGER (Given)

The number of evaluations to perform at constant Y position.

XMIN = ? (Given)

The lower endpoint of the range of the fit along the first dimension. The Chebyshev series representation is in terms of a normalised variable, evaluated as (2x - (XMAX + XMIN)) / (XMAX - XMIN), where x is the original variable. XMIN must be less than XMAX.

XMAX = ? (Given)

The upper endpoint of the range of the fit along the second dimension. See XMIN.

X(NPTS) = ? (Given)

The co-ordinates along the first dimension for which the Chebyshev polynomial is to be evaluated.

YMIN = ? (Given)

The lower endpoint of the range of the fit along the first dimension. The Chebyshev series representation is in terms of a normalised variable, evaluated as (2y - (YMAX + YMIN)) / (YMAX - YMIN), where y is the original variable. YMIN must be less than YMAX.

YMAX = ? (Given)

The upper endpoint of the range of the fit along the second dimension. See YMIN.

Y = ? (Given)

The co-ordinate along the second dimension for which the Chebyshev polynomial is to be evaluated.

XDEG = INTEGER (Given)

The degree of the polynomial along the first dimension.

YDEG = INTEGER (Given)

The degree of the polynomial along the second dimension.

NCOEF = INTEGER (Given)

The number of coefficients. This must be at least the product of (XDEG+1) $*$ (YDEG+1).

CC(NCOEF) = ? (Given)

The Chebyshev coefficients. These should be the order such that CCij is in CC(i$*$(YDEG+1)+j+1) for i=0,XDEG; j=0,YDEG. In other words the opposite order to Fortran standard.

NW = INTEGER (Given)

The number of elements in the work array. It must be at least XDEG + 1.

WORK(NW) = ? (Returned)

Workspace.

EVAL(NPTS) = ? (Returned)

The evaluated polynomial for the supplied arguments. Should an argument lie beyond the range ([XMIN,XMAX], [YMIN,YMAX] the bad value is returned in the corresponding element of EVAL.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

There is a routine for the real and double precision data types: replace " x" in the routine name by R or D respectively. The XMIN, XMAX, X, YMIN, YMAX, Y, CC, WORK, and EVAL arguments supplied to the routine must have the data type specified.

KPG1_CHELx
Replaces the value of an array element with a specified value

Description:

This routine replaces a specified element of a vectorised array with a supplied value and returns the former value of the element.

Invocation:

```
CALL KPG1_CHELx( EL, CHEL, NEWVAL, ARRAY, OLDVAL, STATUS )
```

Arguments:

EL = INTEGER (Given)
The dimension of the array.

CHEL = INTEGER (Given)
The element number to be replaced. It must lie between 1 and EL, otherwise an error report is made.

NEWVAL = ? (Given)
The new value of the element.

ARRAY(EL) = ? (Given and Replaced)
The array whose value is to be replaced.

OLDVAL = ? (Returned)
The former value of the array element.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by I, R, or D as appropriate. Arguments ARRAY, NEWVAL and OLDVAL must have the data type specified.

KPG1_CHEPx
Evaluates a Chebyshev polynomial

Description:

This evaluates a Chebyshev polynomial for orders zero to NTERM-1 at a given normalised [-1,+1] co-ordinate. It uses a recurrence relationship to evaluate beyond the second term.

Invocation:

```
CALL KPG1_CHEPx( X, NTERM, T, STATUS )
```

Arguments:

X = ? (Given)

The normalised co-ordinate for which the Chebyshev polynomial is to be evaluated. It is assumed to lie in the range [-1,+1] having been normalised using the limits that created the coefficients.

NTERM = INTEGER (Given)

The number of terms in the Chebyshev polynomial. It equals the order plus one.

T(NTERM) = ? (Returned)

The evaluated Chebyshev polynomial for each term.

STATUS = INTEGER (Given and Returned)

The global status.

References :

- T. Hopkins & C.Phillips, 1988, " Numerical Methods in Practice" , Addison-Wesley, p.190-191. [routine_references]...

KPG1_CHEVx
Evaluates a one-dimensional Chebyshev polynomial

Description:

This routine evaluates a one-dimensional Chebyshev polynomial for one or more arguments. It uses Clenshaw's recurrence relationship.

Invocation:

```
CALL KPG1_CHEVx( XMIN, XMAX, NCOEF, CHCOEF, NPTS, X, EVAL, STATUS )
```

Arguments:

XMIN = ? (Given)

The lower endpoint of the range of the fit. The Chebyshev series representation is in terms of a normalised variable, evaluated as (2x - (XMAX + XMIN)) / (XMAX - XMIN), where x is the original variable. XMIN must be less than XMAX.

XMAX = ? (Given)

The upper endpoint of the range of the fit. See XMIN.

NCOEF = INTEGER (Given)

The number of coefficients. This must be at least the polynomial order plus one.

CC(NCOEF) = ? (Given)

The Chebyshev coefficients.

NPTS = INTEGER (Given)

The number of arguments for which the Chebyshev polynomial is to be evaluated.

X(NPTS) = ? (Given)

The arguments for which the Chebyshev polynomial is to be evaluated.

EVAL(NPTS) = ? (Returned)

The evaluated polynomial for the supplied arguments. Should an argument lie beyond the range [XMIN,XMAX], the bad value is returned in the corresponding element of EVAL.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

There is a routine for the real and double precision data types: replace " x" in the routine name by R or D respectively. The XMIN, XMAX, CC, X, and EVAL arguments supplied to the routine must have the data type specified.

KPG1_CHVAx
Replaces all occurrences of a value in an array with another value

Description:

This routine copies the input array to the output array, except where a specified value occurs, and this is replaced with a new value throughout the output array.

Invocation:

```
CALL KPG1_CHVAx( EL, INARR, OLDVAL, NEWVAL, OUTARR, NREP, STATUS )
```

Arguments:

EL = INTEGER (Given)
The dimension of the input and output arrays.

INARR(EL) = ? (Given)
The input array.

OLDVAL = ? (Given)
Value to be replaced.

NEWVAL = ? (Given)
New value to be substituted for the old value.

OUTARR(EL) = ? (Returned)
The output array containing the modified values.

NREP = INTEGER (Returned)
The number of replacements made.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays and values supplied to the routine must have the data type specified.

KPG1_CMADx
Compresses an n-dimensional array by summing in 'rectangular' boxes

Description:

This routine compresses an n-dimensional array by integer factors along each dimension by summing the array values in a rectangular box. The output may be normalised to take account of any bad values that may be present.

Invocation:

```
CALL KPG1_CMADx( NDIM, DIMS, INARR, COMPRS, NLIM, NORMAL, OUTARR, SUM, NUM, STATUS
)
```

Arguments:

NDIM = INTEGER (Given)

The dimensionality of the n-dimensional array. It must be greater than one. To handle a one-dimensional array, give it a second dummy dimension of 1.

DIMS(NDIM) = INTEGER (Given)

The dimensions of the input n-dimensional array.

INARR(*) = ? (Given)

The input n-dimensional data array.

COMPRS(NDIM) = REAL (Given)

The factors along each dimension by which the input array is compressed to form the output array.

NLIM = INTEGER (Given)

The minimum number of good input elements in a compression box that permits the corresponding output array element to be good. If fewer than NLIM pixels participated in the sum, the output pixel will be bad.

NORMAL = LOGICAL (Given)

If true, the summation is normalised to allow for bad data, i.e. the sum with each box is multiplied by the ratio of the total number of pixels in the box to the actual number included to yield the output pixel value. If false, the output array element just equates to the sum of good pixels within each box.

OUTARR(*) = ? (Write)

The compressed n-dimensional array. Its dimension I must be given by DIMS(I)/COMPRS(I).

SUM(*) = ? (Returned)

Workspace used for efficiency in computing the summations. This should have size at least equal to DIMS(1).

NUM(*) = INTEGER (Returned)

Workspace used to count the number of good elements in the input box. This should have size at least equal to DIMS(1).

STATUS = INTEGER (Given and Returned).

Global status value

Notes:

- There is a routine for the following numeric data types: replace "`x`" in the routine name by D or R as appropriate. The input and output data arrays plus a work space must have the data type specified.
- There is no protection against overflows when the absolute data values are very large.

Bugs:

{note_bugs_here}

KPG1_CMAVx
Compresses an n-dimensional array by averaging in 'rectangular' boxes

Description:

This routine compresses an n-dimensional array by integer factors along each dimension by averaging the array values in a rectangular box.

Invocation:

```
CALL KPG1_CMAVx( NDIM, DIMS, INARR, COMPRS, NLIM, OUTARR, SUM, NUM, STATUS )
```

Arguments:

NDIM = INTEGER (Given)

The dimensionality of the n-dimensional array. It must be greater than one. To handle a one-dimensional array, give it a second dummy dimension of 1.

DIMS(NDIM) = INTEGER (Given)

The dimensions of the input n-dimensional array.

INARR(*) = ? (Given)

The input n-dimensional data array.

COMPRS(NDIM) = REAL (Given)

The factors along each dimension by which the input array is compressed to form the output array.

NLIM = INTEGER (Given)

The minimum number of good input elements in a compression box that permits the corresponding output array element to be good. If fewer than NLIM pixels participated in the sum, the output pixel will be bad.

OUTARR(*) = ? (Write)

The compressed n-dimensional array. Its dimension I must be given by DIMS(I)/COMPRS(I).

SUM(*) = ? (Returned)

Workspace used for efficiency in computing the summations. This should have size at least equal to DIMS(1).

NUM(*) = INTEGER (Returned)

Workspace used to count the number of good elements in the input box. This should have size at least equal to DIMS(1).

STATUS = INTEGER (Given and Returned).

Global status value

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by D or R as appropriate. The input and output data arrays plus a work space must have the data type specified.
- There is no protection against overflows when the absolute data values are very large.

Bugs:

{note_bugs_here}

KPG1_CMPKx
Compresses an n-dimensional array by picking value at equally spaced intervals

Description:

This routine compresses an n-dimensional array by integer factors along each dimension by selecting the array values at equal intervals along each dimension. The intervals may be different in each dimension. The starting point may be defined. Bad values will just be copied like any other array value.

Invocation:

```
CALL KPG1_CMPKx( NDIM, DIMS, INARR, COMPRS, OFFSET, OUTARR, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
The dimensionality of the n-dimensional array.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the input n-dimensional array.

INARR($*$) = ? (Given)
The input n-dimensional data array.

COMPRS(NDIM) = REAL (Given)
The factors along each dimension by which the input array is compressed to form the output array.

OFFSET(NDIM) = REAL (Given)
The pixel indices of the first input-array element that is to be put into the output array. Subsequent selections are COMPRS pixels along, so these can be regarded as offsets of the selections. The Ith offset must lie in the range 1 to COMPRS(I).

OUTARR($*$) = ? (Write)
The compressed n-dimensional array. Its dimension I must be given by (DIMS(I) - OFFSET(I)) / COMPRS(I) + 1.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The input and output data arrays must have the data type specified.

Bugs:

{note_bugs_here}

KPG1_CMULD
Multiplies each element of a vectorised double precision array by a constant

Description:

The routine multiplies each element of a vectorised double precision array by a constant to produce a new double precision array. Bad value checking is performed if required.

Invocation:

```
CALL KPG1_CMULD( BAD, EL, A, CONST, B, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = DOUBLE PRECISION (Given)
Input array.

CONST = DOUBLE PRECISION (Given)
Constant by which each array element is to be multiplied.

B(EL) = DOUBLE PRECISION (Returned)
Output array.

NBAD = INTEGER (Returned)
Number of bad values in the output array B.

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

Notes:

- This routine is intended for processing double-precision data only. There is a related generic routine for processing other data types.
- This routine will handle numerical errors (i.e. overflow) by assigning the appropriate " bad" value to affected output array elements.

KPG1_CMULx
Multiplies each element of a vectorised array by a constant

Description:

The routine multiplies each element of a vectorised array by a constant to produce a new array. Bad value checking is performed if required.

Invocation:

```
CALL KPG1_CMULx( BAD, EL, A, CONST, B, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = ? (Given)
Input array.

CONST = DOUBLE PRECISION (Given)
Constant by which each array element is to be multiplied.

B(EL) = ? (Returned)
Output array.

NBAD = INTEGER (Returned)
Number of bad values in the output array B.

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

Notes:

- There is a routine for each numeric type except double precision: replace "x" in the routine name by R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.
- This routine will handle numerical errors (i.e. overflow) by assigning the appropriate "bad" value to affected output array elements.

KPG1_CMULK
Multiplies each element of a vectorised 64-bit integer array by a constant

Description:

The routine multiplies each element of a vectorised 64-bit- integer array by a constant to produce a new 64-bit-integer array. Bad value checking is performed if required.

Invocation:

```
CALL KPG1_CMULK( BAD, EL, A, CONST, B, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = INTEGER ∗ 8 (Given)
Input array.

CONST = DOUBLE PRECISION (Given)
Constant by which each array element is to be multiplied.

B(EL) = DOUBLE PRECISION (Returned)
Output array.

NBAD = INTEGER (Returned)
Number of bad values in the output array B.

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

Notes:

- This routine is intended for processing 64-bit integer data only. There is a related generic routine for processing other data types, except double precision which has its own routine like this one.
- This routine will handle numerical errors (i.e. overflow) by assigning the appropriate " bad" value to affected output array elements.

KPG1_CMVDx
Compresses n-dimensional data and variance arrays by summing in 'rectangular' boxes

Description:

This routine compresses an n-dimensional data array and its associated variance array by integer factors along each dimension by summing the arrays' values in a rectangular box. The output data array may be normalised to take account of any bad values that may be present.

Invocation:

```
CALL KPG1_CMVDx( NDIM, DIMS, INARR, INVAR, COMPRS, NLIM, NORMAL, OUTARR, OUTVAR, SUM,
NUM, STATUS )
```

Arguments:

NDIM = INTEGER (Given)

The dimensionality of the n-dimensional arrays. It must be greater than one. To handle one-dimensional arrays, give them a second dummy dimension of 1.

DIMS(NDIM) = INTEGER (Given)

The dimensions of the input n-dimensional arrays.

INARR(*) = ? (Given)

The input n-dimensional data array.

INVAR(*) = ? (Given)

The input n-dimensional variance array.

COMPRS(NDIM) = REAL (Given)

The factors along each dimension by which the input array is compressed to form the output array.

NLIM = INTEGER (Given)

The minimum number of good input elements in a compression box that permits the corresponding output array element to be good. If fewer than NLIM pixels participated in the sum, the output pixel will be bad.

NORMAL = LOGICAL (Given)

If .TRUE., the summation is normalised to allow for bad data, i.e. the sum with each box is multiplied by the ratio of the total number of pixels in the box to the actual number included to yield the output pixel value. If .FALSE., the output array element just equates to the sum of good pixels within each box.

OUTARR(*) = ? (Write)

The compressed n-dimensional data array. Its dimension I must be given by DIMS(I)/COMPRS(I).

OUTVAR(*) = ? (Write)

The compressed n-dimensional variance array. Its dimension I must be given by DIMS(I)/COMPRS(I).

SUM(*) = ? (Returned)

Workspace used for efficiency in computing the summations. This should have size at least equal to DIMS(1) * 2.

NUM(*) = INTEGER (Returned)

Workspace used to count the number of good elements in the input box. This should have size at least equal to DIMS(1) * 2.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace "`x`" in the routine name by D or R as appropriate. The input and output arrays plus a work space must have the data type specified.
- There is no protection against overflows when the absolute data values are very large.

Bugs:
{note_bugs_here}

KPG1_CMVVx
Compresses n-dimensional data and variance arrays by averaging in 'rectangular' boxes

Description:

This routine compresses an n-dimensional data array and its associated variance array by integer factors along each dimension by averaging the arrays' values in a rectangular box. The output data array may be variance-weighted average.

Invocation:

```
CALL KPG1_CMVVx( NDIM, DIMS, INARR, INVAR, COMPRS, NLIM, WEIGHT, OUTARR, OUTVAR, SUM,
NUM, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
The dimensionality of the n-dimensional arrays. It must be greater than one. To handle one-dimensional arrays, give them a second dummy dimension of 1.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the input n-dimensional arrays.

INARR(*) = ? (Given)
The input n-dimensional data array.

INVAR(*) = ? (Given)
The input n-dimensional variance array.

COMPRS(NDIM) = REAL (Given)
The factors along each dimension by which the input array is compressed to form the output array.

NLIM = INTEGER (Given)
The minimum number of good input elements in a compression box that permits the corresponding output array element to be good. If fewer than NLIM pixels participated in the sum, the output pixel will be bad.

WEIGHT = LOGICAL (Given)
If true, the summation is weighted by the corresponding variance. If false all the input good elements are given equal weight when forming the output data-array value.

OUTARR(*) = ? (Write)
The compressed n-dimensional data array. Its dimension I must be given by DIMS(I)/COMPRS(I).

OUTVAR(*) = ? (Write)
The compressed n-dimensional variance array. Its dimension I must be given by DIMS(I)/COMPRS(I).

SUM(*) = ? (Returned)
Workspace used for efficiency in computing the summations. This should have size at least equal to DIMS(1) * 2.

NUM(*) = ? (Returned)
Workspace used to count the number of good elements in the input box. This should have size at least equal to DIMS(1) * 2.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by D or R as appropriate. The input and output arrays plus a work space must have the data type specified.
- There is no protection against overflows when the absolute data values are very large.
- If the sum of variances in the weighted average is zero, the output element is bad.

Bugs:
{note_bugs_here}

KPG1_CNLIM
Parses a character string into integer bounds

Description:

This routine is used to parse a character specification which defines a range of integers. The specification is of the form:

Number[-number]

If the second number is specified then it is assumed that a sequence of numbers are to be parsed, e.g. 4-6 (or 6-4) specifies integers 4, 5, 6; and bounds 4 and 6 are returned. If only one number is specified, then both the returned bounds are set equal to it.

An asterisk may be used as a wildcard. Thus 5-∗ specifies all numbers upwards from 5 inclusive, the upper bound being given by the largest integer. ∗ indicates all integers, i.e. the bounds would be the smallest (non-bad) and largest integers. ∗-4 would specify a range from the smallest integer to 4 inclusive.

Invocation:

```
CALL KPG1_CNLIM( NOS, FIRST, LAST, STATUS )
```

Arguments:

NOS = CHARACTER ∗ (∗) (Given)
Number specification.

FIRST = INTEGER (Returned)
First integer in the sequence.

LAST = INTEGER (Returned)
Last integer in the sequence.

STATUS = INTEGER (Given and Returned)
Global status value.

KPG1_COLNM
Finds the named colour nearest to an RGB triple

Description:

This returns the name of a colour in the standard colour set that matches an input R-G-B colour, or failing that, that is nearest to the input colour. A city-block metric is used.

Invocation:

```
CALL KPG1_COLNM( R, G, B, NAME, STATUS )
```

Arguments:

R = REAL (Given)
The red intensity of the colour to be identified. It should be in the range 0.0 to 1.0.

G = REAL (Given)
The green intensity of the colour to be identified. It should be in the range 0.0 to 1.0.

B = REAL (Given)
The blue intensity of the colour to be identified. It should be in the range 0.0 to 1.0.

NAME = CHARACTER * (*) (Returned)
The name of the nearest colour in the named colour set to the input RGB colour. Note at least eighteen characters are required to avoid truncation.

STATUS = INTEGER (Returned)
The global status.

Notes:

- The actual R-G-B colours used will be constrained to lie in the range 0.0–1.0.

KPG1_COPYC
Copies a one-dimensional array of character strings

Description:

This routine copies a one-dimensional array of character strings from an inmput to an output array.

Invocation:

```
CALL KPG1_COPYC( NEL, IN, OUT, STATUS )
```

Arguments:

NEL = INTEGER (Given)
The length of the array.

IN = CHARACTER(NEL) * (*) (Given)
The input array.

OUT = CHARACTER(NEL) * (*) (Returned)
The output array.

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

KPG1_COPY
Copies an array of a given type to another array of the same type

Description:

This routine copies a one-dimensional array of numerical or character values from an input array to an output array.

Invocation:

```
CALL KPG1_COPY( TYPE, NEL, IPIN, IPOUT, STATUS )
```

Arguments:

TYPE = CHARACTER * (*) (Given)
The data type to be copied. Must be one of the HDS numeric types, _BYTE, _UBYTE, _WORD, _UWORD, _INTEGER, _INT64, _REAL or _DOUBLE, or " _CHAR*<N>)" where " <N>" is an integer giving the length of the character strings.

NEL = INTEGER (Given)
The number of elements in the vectorised arrays pointed to by IPIN and IPOUT.

IPIN = INTEGER (Given)
Pointer to the data to be copied.

IPOUT = INTEGER (Given and Returned)
Pointer to the array to contain the copied data.

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

Notes:

- Uses array pointers.

KPG1_CORRx
Calculates the correlation coefficient between two arrays

Description:

The routine calculates the sample Pearson correlation coefficient between the two supplied arrays.

Invocation:

```
CALL KPG1_CORR<T>( EL, X, Y, R, N, STATUS )
```

Arguments:

EL = INTEGER (Given)
Number of array elements to process.

X(EL) = ? (Given)
First input array.

Y(EL) = ? (Given)
First input array.

R = DOUBLE PRECISION (Returned)
The correlation coefficient.

N = INTEGER (Returned)
The number of points used to form the returned correlation coefficient.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.

KPG1_CPBDx
Copies bad pixels

Description:
This routine copies the bad pixels from one array into another.

Invocation:
```
CALL KPG1_CPBDx( N, D1, D2, OUT, STATUS )
```

Arguments:

N = INTEGER (Given)
Number of points.

D1(N) = ? (Given)
The original data values.

D2(N) = ? (Given)
An associated mask array.

OUT(N) = ? (Given)
The returned array. Equal to D1 if both D1 and D2 are good (bad if either D1 or D2 is bad).

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The input and output data arrays plus the mask array must have the data type specified.

KPG1_CPNTx
Creates a primitive NDF with a title via the parameter system

Description:

This routine packages a common series of NDF calls when creating a primitive NDF, whose name is obtained from the parameter system, and also obtaining its title from the parameter system. The need to handle an optional output file via the null character is catered. The data type of the NDF data array is the same as the supplied array. An NDF context is started and ended within this routine.

Invocation:

```
CALL KPG1_CPNTx( PNNDF, PNTIT, NDIM, DIMS, ARRAY, NULL, STATUS )
```

Arguments:

PNNDF = CHARACTER * (*) (Given)
The ADAM parameter name to obtain the name of the output NDF.

PNTIT = CHARACTER * (*) (Given)
The ADAM parameter name to obtain the title of the output NDF.

NDIM = INTEGER (Given)
The number of dimensions in the NDF's data array.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the NDF's data array.

ARRAY(*) = ? (Given)
The array to be placed in the primitive NDF's data array.

NULL = LOGICAL (Given)
If true a null value returned by either parameter GET is annulled.

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

Notes:

- There is a routine for each numeric data type: replace "_x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The data array supplied to the routine must have the data type specified.

KPG1_CPSTY
Copies the plotting style for an AST element to another AST element

Description:

This routine copies the AST attributes which determine the colour, style, width, font and size. If IN is non-blank, then the attributes are copied from the element given by IN, to the element given by OUT–the original values of these attributes for element OUT are returned in ATTRS. If IN is blank, then the values supplied in ATTRS are copied to the element given by OUT– the original values of these attributes for element OUT are again returned in ATTRS.

Invocation:

```
CALL KPG1_CPSTY( IPLOT, IN, OUT, ATTRS, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
The AST Plot.

IN = CHARACTER * (*) (Returned)
The name of the source element (e.g. " CURVES" , " AXIS1"), or blank.

OUT = CHARACTER * (*) (Returned)
The name of the destination element (e.g. " CURVES" , " AXIS1").

ATTRS(5) = REAL (Given and Returned)
The entry values are ignored unless IN is blank, in which case the entry values should be the colour, width, style, size and font attribute values to associate with element OUT. On exit, the original colour, width, style, size and font attribute values associated with element OUT.

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

KPG1_CROUT
Creates and returns locators to the top-level and data-array of an NDF-type structure

Description:

An NDF-type data structure, associated with the supplied parameter name is created. A locator for this structure is returned. A TITLE component, associated with another parameter name is created within the structure and a character value, is obtained from the parameter system and written to the TITLE component. A DATA_ARRAY component is created. If the supplied origins are all one, DATA_ARRAY is primitive, having the supplied dimensionality and dimensions, and is of type _REAL. Otherwise DATA_ARRAY is a simple ARRAY-type structure containing the primitive _REAL data array and origin information. The locator to the primitive data array is also returned.

Invocation:

```
CALL KPG1_CROUT( PARNAM, TLENAM, NDIM, DIMS, ORIGIN, LOCAT, DATLOC, DNAME, STATUS )
```

Arguments:

PARNAM = CHARACTER * (*) (Given)
Parameter name associated with the NDF-type structure to be created.

TLENAM = CHARACTER * (*) (Given)
Parameter name associated with the TITLE component created in the new NDF-type structure.

NDIM = INTEGER (Given)
Dimensionality of the data-array component of the new NDF-type structure.

DIMS(NDIM) = INTEGER (Given)
Dimensions of the data-array component of the new NDF-type structure.

ORIGIN(NDIM) = INTEGER (Given)
The origin information of the data array. If a primitive NDF is to be created these should all be set to 1. Note though if other ARRAY-type components—QUALITY and VARIANCE—are propagated, the origin data of ARRAY-type components must be consistent not to invalidate the NDF. Therefore use the values returned by KPG1_GETIM unless QUALITY and VARIANCE are not propagated.

LOCAT = CHARACTER * (*) (Returned)
Locator to the new NDF-type structure. If status is bad on exit this locator is annulled. If status is bad on exit this locator is annulled.

DATLOC = CHARACTER * (DAT__SZLOC) (Returned)
The locator to the structure containing the primitive form of the data array. If it is the top-level of the NDF structure, it will be a clone of LOCAT. Either way it will require annulment. If status is bad on exit this locator is annulled.

DNAME = CHARACTER * (DAT__SZNAM) (Returned)
The name of the data array as this will be needed for access, and it is different depending on its location. This argument is probably unnecessary, but it is here defensively, in case the origin is changed to or from the default.

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

Notes:

- This a stop-gap routine until the remainder of KAPPA IMAGE-format applications are converted to NDF. It enables both primitive (i.e. IMAGE-format) and simple NDFs to be processed.
- The maximum length of the TITLE component produced by this routine is 72 characters.

KPG1_CSHFT
Shifts the characters left or right in a string

Description:

This routine shifts all characters in a string by a given number of characters, padding with spaces at the ends.

Invocation:

```
CALL KPG1_CSHFT( N, TEXT )
```

Arguments:

N = INTEGER (Returned)

The number of characters to shift to the right. Negative values produce shifts to the left.

TEXT = CHARACTER * (*) (Given and Returned)

The text to be shifted.

KPG1_CSUBx
Subtracts a constant from each element of a vectorised array

Description:

The routine subtracts a constant from each element of a vectorised array to produce a new array. Bad value checking is performed if required.

Invocation:

```
CALL KPG1_CSUBx( BAD, EL, A, CONST, B, NERR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = ? (Given)
Input array.

CONST = DOUBLE PRECISION (Given)
Constant to be subtracted from each array element.

B(EL) = ? (Returned)
Output array.

NERR = INTEGER (Returned)
Number of numerical errors which occurred while processing the array.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.
- This routine will handle numerical errors (i.e. overflow) by assigning the appropriate " bad" value to affected output array elements. If the constant supplied cannot be converted to the data type of the arrays without overflow, then all elements of the output array will be assigned this bad value and NERR will return the value of EL.

KPG1_CTCKM
Copies values from catalogue columns to an AST KeyMap

Description:

This routine gets NEL values for a set of given CAT (see SUN/181) columns, derived from rows 1 to NEL of a given catalogue, selection, or index, and stores them in a KeyMap.

Invocation:

```
CALL KPG1_CTCKM( CI, NAX, GI, NEL, KEYMAP, STATUS )
```

Arguments:

CI = INTEGER (Given)
The CAT identifier for the catalogue, selection or index containing the required data.

NAX= INTEGER (Given)
The number of columns from which values are to be read.

GI(NAX) = INTEGER (Given)
The CAT identifiers for the column, expressions or parameters to be evaluated for rows 1 to NEL of the component identified by CI.

NEL = INTEGER (Given)
The number of rows to copy.

KEYMAP = INTEGER (Given)
The KeyMap. Each column value is stored as a scalar string with key " <colname>_<row index>" .

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

KPG1_CTCPC
Copies string values from a catalogue column to a GRP group

Description:

This routine gets NEL values for a given CAT (see SUN/181) column, derived from rows 1 to NEL of a given catalogue, selection, or index, and appends them to the end of the supplied GRP group (a new group is created if necessary).

Invocation:

```
CALL KPG1_CTCPC( CI, GI, NEL, IGRP, STATUS )
```

Arguments:

CI = INTEGER (Given)
The CAT identifier for the catalogue, selection or index containing the required data.

GI = INTEGER (Given)
The CAT identifier for the column, expression or parameter to be evaluated for rows 1 to NEL of the component identified by CI.

NEL = INTEGER (Given)
The number of rows to copy.

IGRP = INTEGER (Given and Returned)
The identiier for the GRP group in which the column values are to be stored. If this is supplied as GRP__NOID, then a new group is created. Otherwise the values are appended to the end of the supplied group.

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

KPG1_CTCPx
Copies values from catalogue columns to an array

Description:

This routine gets NEL values for a set of given CAT (see SUN/181) columns, derived from rows 1 to NEL of a given catalogue, selection, or index, and stores them in array OUT.

Invocation:

```
CALL KPG1_CTCPx( CI, NAX, GI, NEL, OUT, STATUS )
```

Arguments:

CI = INTEGER (Given)
The CAT identifier for the catalogue, selection or index containing the required data.

NAX= INTEGER (Given)
The number of columns from which values are to be read.

GI(NAX) = INTEGER (Given)
The CAT identifiers for the column, expressions or parameters to be evaluated for rows 1 to NEL of the component identified by CI. If any elements of this array are CAT__NOID, then the corresponding elements of OUT are filled with the row number.

NEL = INTEGER (Given)
The number of rows to copy.

OUT(NEL, NAX) = ? (Returned)
The returned values.

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

Notes:

- There is a routine for each of the data types integer, and double precision: replace " x" in the routine name by I, or D as appropriate. The output array from this routine should have the specified data type.

KPG1_D2W2x
Converts linear data co-ordinates to world co-ordinates

Description:

The co-efficients of the linear transformation from world co-ordinates to data co-ordinates are supplied in arguments SCALE and OFFSET. The inverse of this transformation is used to transform each supplied position from data to world co-ordinates.

Invocation:

```
CALL KPG1_D2W2x( SCALE, OFFSET, NPOINT, XP, YP, STATUS )
```

Arguments:

SCALE(2) = ? (Given)
The scale factors in the linear relationships between axis co-ordinates and pixel co-ordinates.

OFFSET(2) = ? (Given)
The offsets in the linear relationships between axis co-ordinates and pixel co-ordinates.

NPOINT = INTEGER (Given)
The number of points specified.

XP(NPOINT) = ? (Given and Returned)
On input the x data co-ordinate of each point. On return the x world co-ordinate of each point.

YP(NPOINT) = ? (Given and Returned)
On input the y data co-ordinate of each point. On return the y world co-ordinate of each point.

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

Notes:

- There is a routine for each of the real and double-precision data types: replace " x" in the routine name by D or R as appropriate. The SCALE, OFFSET, XP and YP arrays must have the data type specified.
- The supplied values of SCALE and OFFSET are such that:

DATA = SCALE(I) $*$ PIXEL $+$ OFFSET(I)

where PIXEL is a pixel co-ordinate for the I' th dimension, and DATA is the corresponding axis co-ordinate.

KPG1_DANOT
Generates an annotation from the NDF's label and units

Description:

This routine examines the NDF for a label and units. It creates a string of the form " label (units)" or " label" if the label but not the units are present. If neither are present a string comprising the array component followed by ' values in FILE' is created. Where " FILE" is the NDF filename. If the label is defined but is blank, this same default is used. Blank units are omitted. The units are squared for the variance component.

Invocation:

```
CALL KPG1_DANOT( NDF, COMP, DATLAB, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

COMP = CHARACTER * (*) (Given)
Name of the NDF array component: ' DATA', ' QUALITY', ' VARIANCE', or ' ERROR', though it is used literally and not checked to be a member of this set.

DATLAB = CHARACTER * (*) (Returned)
The composite annotation.

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

Prior Requirements :

- The identifier should be associated with an NDF.

KPG1_DARA*x*
Determines the data limits for an array using a variety of methods

Description:

This routine derives a data range applicable to the supplied array. It is intended to be used to specify the data limits for histogram plots and scaling data for image display. There is a variety of methods available to derive the limits, of which a subset of appropriate methods can be defined by argument METHOD.

The method is selected by the user through the literal parameter specified by PARAM. The supplied string should consist of up to three sub-strings or elements, separated by commas. In all but one method, the first element must specify the method to use. If supplied, the other two elements should be numerical values as described below (default values will be used if these elements are not provided). The following options are available.

- " Extended" – LOWER and UPPER are returned equal to the lowest and highest supplied data values, extended to give a margin of 2.5% of the total data range at each end.
- " Extended,10,5" – Like " Extended" , except the margins at the two ends are specified as a pair of numerical value in the second and third elements of the array. These values are percentages of the total data range. So, " Extended,10,5" includes a margin of 10% of the total data range in LOWER, and 5% in UPPER. If only one numerical value is given, the same value is used for both limits. If no value is given, both limits default to 2.5. " Range" is equivalent to " Extended,0,0" .
- " Faint" – The scaling is between the mean data value minus one standard deviation and the mean data value plus seven standard deviations. This is retained for historical reasons.
- " Limit" – Allows user to supply explicit lower and upper limiting values. For example, " Limit,10,20" would set the lower limit to 10 and its upper limit to 20. For compatibility reasons and for ease of use, this routine permits the omission of the method. Thus " 10,20" is equivalent to the previous example. The upper limit may be numerically smaller than the lower limit, as might be needed for display scaling. If fewer than two numerical values are supplied, the " Range" method is adopted. The limits must be within the allowed range for the data type of the ARRAY argument.
- " Percentiles,5,95" – The second and third elements of the array are interpreted as percentiles. For instance, " Perc,5,95" causes 5% of the data points to be below LOWER, and 10% to be above the UPPER. If only 1 value (p1) is supplied, the other one, p2, defaults to (100 - p1). If no values are supplied, p1 and p2 default to 5 and 95 respectively. Values must be in the range 0 to 100.
- " Range" – LOWER and UPPER are returned equal to the lowest and highest supplied array values.
- " Sigmas,2,3" – The second and third elements of the array are interpreted as multiples of the standard deviation of the data values. For instance, " S,2,3" causes the LOWER to be the mean of the data values, minus two sigma, and UPPER to be the mean plus three sigma. If only 1 value is supplied, the same value is used for both limits. If no values are supplied, both values default to 3.0.

The above method names can be abbreviated to one character.

If the parameter name is supplied as blank, then " Range" is assumed (i.e. LOWER and UPPER are chosen so that the encompasses the entire data range).

The extreme values are reported unless the limits parameter PARAM is specified on the command line. In this case extreme values are only calculated where necessary for the chosen method.

Invocation:

```
CALL KPG1_DARAx( PARAM, EL, ARRAY, METHDS, BAD, LOWER, UPPER, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the parameter to use to get the method for chosing the default limits for the axis. May be blank.

EL = INTEGER (Given)
The number of elements of the array to be analysed.

ARRAY(EL) = ? (Given)
The array for which the chosen statistics are required.

METHDS = CHARACTER * (*) (Given)
A comma-separated list of methods by which the data limits may be specified. Different tasks may require only a subset of the methods available. The allowed methods are " Faint" , " Extended" , " Limit" , " Percentiles" , " Range" , and " Sigmas" . See the description for details of these methods.

BAD = LOGICAL (Given and Returned)
True when bad pixels may be present.

LOWER = DOUBLE PRECISION (Returned)
The lower limit.

UPPER = DOUBLE PRECISION (Returned)
The upper limit.

STATUS = INTEGER(Given & Returned)
Global status.

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The ARRAY argument supplied must have the data type specified.

KPG1_DATCP
Gets an AST FrameSet from an NDF

Description:

Recursively copy an object into a component. This means that the complete object (including its components and its components' s components, etc.) is copied, not just the top level.

This routine is exactly like DAT_COPY except that it will copy array slices, which DAT_COPY will not.

Invocation:

```
CALL KPG1_DATCP( LOC1, LOC2, NAME, STATUS )
```

Arguments:

LOC1 = CHARACTER * (*) (Given)
Object locator.

LOC2 = CHARACTER * (*) (Given)
Structure locator.

NAME = CHARACTER * (*) (Given)
Component name.

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

KPG1_DAUNI
Generates a string containing the units of an NDF's data or variance component allowing for truncation

Description:

This routines generates a string containing the units of an NDF's data or variance component. If the length of the variable to store the string is shorter than the units field, the units string is truncated and an ellipsis is inserted. For the variance the data units are enclosed in parentheses and followed by '**2'. Again if the the length of the units string is too short, truncate the units within the parentheses and insert an ellipsis. A null string is returned if the UNITS component does not exist within the NDF.

Invocation:

```
CALL KPG1_DAUNI( NDF, COMP, UNITS, NCU, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier

COMP = CHARACTER * (*) (Given)
The NDF component: ' DATA', ' VARIANCE', ' QUALITY' or ' ERROR'. Any other component will result in a SAI__ERROR status being returned immediately.

UNITS = CHARACTER * (*) (Returned)
The string containing the units for the component, possibly truncated. It is recommended that string should be at least 20 characters long. If COMP = ' VARIANCE' the length must be no less than 9 characters; for COMP = ' DATA' the minimum is 4 characters; for COMP = ' ERROR' the minimum is 5 characters; otherwise the routine returns immediately with the SAI__ERROR status.

NCU = INTEGER (Returned)
The length of the units string in characters ignoring trailing blanks. Always returned equal to zero if COMP = ' QUALITY'.

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

KPG1_DCLIx
Determines a linear transformation from pixel to NDF-axis co-ordinates

Description:

This routine obtains a data co-ordinate system from the NDF axes, and determines whether all the axes are linear. If they are linear transformations between pixel co-ordinates and data co-ordinates are derived. A warning message is reported for each individual non-linear axis.

Invocation:

```
CALL KPG1_DCLIx( NDIM, NDF, DALBND, DAUBND, SCALE, OFFSET, LINEAR, STATUS )
```

Arguments:

NDIM = INTEGER (Given)

The number of contiguous axes. If it is negative, it indicates that not all the dimensions need be significant.

NDF = INTEGER (Given)

The identifier of the NDF whose axes are to provide the data co-ordinates. It must have at least NDIM dimensions, though this is not checked.

DALBND(NDIM) = ? (Returned)

The lower bounds of the data co-ordinates obtained from the NDF's axes. Note that this gives the data co-ordinate at the lower side of the array element, and not at its centre.

DAUBND(NDIM) = ? (Returned)

The upper bounds of the data co-ordinates obtained from the NDF's axes. Note that this gives the data co-ordinate at the upper side of the array element, and not at its centre.

SCALE(NDIM) = ? (Returned)

The scale factors in the linear transformations from world to data co-ordinates. They should be ignored if LINEAR is false.

OFFSET(NDIM) = ? (Returned)

The offsets in the linear transformations from world to data co-ordinates. They should be ignored if LINEAR is false.

LINEAR = LOGICAL (Returned)

If true the NDF axes are all linear and different from world co-ordinates.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- There is a routine for real and double-precision data types: replace "x" in the routine name by R or D respectively. The scale and offset, and the bounds returned by the routine must have the data type specified.
- This routine assumes monotonic axes.

KPG1_DEBUG
Determines whether the specified package should report debug diagnostics

Description:

This routine returns a logical flag indicating if a specified applications package should report debug diagnostics. This is the case if the environment variable <PACK>_DEBUG is defined (the value assigned to the environment variable is immaterial).

Invocation:

```
CALL KPG1_DEBUG( VERB, PACK, STATUS )
```

Arguments:

DEBUG = LOGICAL (Returned)
Should the package run in debug mode? Returned .FALSE, if an error has already occurred, or if this routine should fail for any reason.

PACK = CHARACTER * (*) (Given)
The name of the package (e.g. " KAPPA" , " POLPACK").

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

Notes:

- This routine attempts to execute even if STATUS is set to an error on entry.

KPG1_DIVx
Divides two vectorised arrays with optional variance information

Description:

The routine divides one vectorised array by another, with optional variance information. Bad value checking is also performed if required.

Invocation:

```
CALL KPG1_DIVx( BAD, VAR, EL, A, VA, B, VB, C, VC, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether it is necessary to check for bad values in the input arrays. The routine will execute more rapidly if this checking can be omitted.

VAR = LOGICAL (Given)
Whether associated variance information is to be processed.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = ? (Given)
First array of data, to be divided by the second array.

VA(EL) = ? (Given)
Variance values associated with the array A. Not used if VAR is set to .FALSE..

B(EL) = ? (Given)
Second array of data, to be divided into the first array.

VB(EL) = ? (Given)
Variance values associated with the array B. Not used if VAR is set to .FALSE..

C(EL) = ? (Returned)
Result of dividing array A by array B.

VC(EL) = ? (Returned)
Variance values associated with the array C. Not used if VAR is set to .FALSE..

NBAD = INTEGER (Returned)
Number of VAL__BADx values in the returned results array (C).

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

Notes:

- There is a routine for each of the data types real and double precision: replace " x" in the routine name by R or D as appropriate. The arrays passed to this routine should all have the specified data type.
- This routine will NOT handle numerical overflow.

KPG1_DNAG2R
Converts an NAG Hermitian Fourier transform array into an array usable by FFTPACK routine KPG1_DRFFTB

Description:

This subroutine modifies the supplied array of Fourier co-efficients (as produced by NAG subroutine C06FAF) so that an inverse FFT can be performed on them using FFTPACK routine KPG1_DRFFTB. The resulting inverse will have the same normalisation as the original data transformed using KPG1_DRFFTF.

This function is equivalent to PDA_NAG2R except that it uses work space for greater speed.

Invocation:

```
CALL KPG1_DNAG2R( NP, R, WORK )
```

Arguments:

NP = INTEGER (Given)
The size of the array.

R(NP) = DOUBLE PRECISION (Given and Returned)
The array holding the Fourier co-efficients. Supplied in NAG format and returned in FFTPACK format.

WORK(NP) = DOUBLE PRECISION (Given and Returned)
Work space.

Notes:

- A call to KPG1_DR2NAG followed by a call to KPG1_DNAG2R will result in the original data being divided by NP.

KPG1_DR2NAG
Converts an FFTPACK Hermitian Fourier transform array into the equivalent NAG array

Description:

This subroutine re-orders and normalises the supplied array of Fourier co-efficients (as produced by FFTPACK subroutine KPG1_DRFFTF) so that the returned array looks like the equivalent array returned by NAG routine C06FAF.

This function is equivalent to PDA_DR2NAG except that it uses work space for greater speed.

The real and imaginary co-efficients produced by KPG1_DRFFTF are numerically larger than the corresponding C06FAF co-efficients by a factor of SQRT(NP), and are ordered differently. Both routines return A0 (the zeroth real term, i.e. the DC level in the array) in element 1. KPG1_DRFFTF then has corresponding real and imaginary terms in adjacent elements, whereas C06FAF has all the real terms together, followed by all the imaginary terms (in reverse order):

KPG1_DRFFTF: A0, A1, B1, A2, B2, A3, B3, ... C06FAF: A0, A1, A2, A3, ..., ..., B3, B2, B1

The zeroth imaginary term (B0) always has the value zero and so is not stored in the array. Care has to be taken about the parity of the array size. If it is even, then there is one more real term than there is imaginary terms (excluding A0), i.e. if NP = 10, then the co-efficients are stored as follows:

KPG1_DRFFTF: A0, A1, B1, A2, B2, A3, B3, A4, B4, A5 C06FAF: A0, A1, A2, A3, A4, A5, B4, B3, B2, B1

If NP = 9, then the co-efficients are stored as follows:

KPG1_DRFFTF: A0, A1, B1, A2, B2, A3, B3, A4, B4 C06FAF: A0, A1, A2, A3, A4, B4, B3, B2, B1

Invocation:

```
CALL KPG1_DR2NAG( NP, R )
```

Arguments:

NP = INTEGER (Given)
The size of the array.

R(NP) = DOUBLE PRECISION (Given and Returned)
The array holding the Fourier co-efficients. Supplied in FFTPACK format and returned in NAG format.

WORK(NP) = DOUBLE PRECISION (Given and Returned)
Work space.

KPG1_DSFR1
Displays a textual description of the Current Frame in a FrameSet

Description:

This routine displays a textual description of the supplied AST Frame. The displayed information does not include any axis-specific details that are common to all classes of Frame (such as axis units, labels, etc.).

Invocation:

```
CALL KPG1_DSFR1( FRM, TEXT, NIND, FULL, STATUS )
```

Arguments:

FRM = INTEGER (Given)
An AST pointer to the Frame.

TEXT = CHARACTER * (*) (Given)
Text to display before the Frame description. May contain MSG tokens.

NIND = INTEGER (Given)
Number of spaces to display at the start of each line.

FULL = LOGICAL (Given)
Display full information?

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

KPG1_DSFRM
Displays a textual description of the Current Frame in a FrameSet

Description:

This routine displays a textual description of the Current Frame in the supplied AST FrameSet.

Invocation:

```
CALL KPG1_DSFRM( FSET, TEXT, LBND, UBND, FULL, STATUS )
```

Arguments:

FSET = INTEGER (Given)

An AST pointer to the FrameSet.

TEXT = CHARACTER * (*) (Given)

Text to display before the Frame description. May contain MSG tokens. This may be prefixed with a string that indicates further processing options. Currently the only such string recognised is " ndftrace:" , which causes the pixel scales to be written out to the output parameter FPIXSCALE. Any such string is not included in the displayed title.

LBND(*) = DOUBLE PRECISION (Given)

The lower bounds of a region within the base Frame over which the displayed nominal WCS axis scales are to be determined. Both UBND and LBND are ignored if the first element of either UBND or LBND is AST__BAD, in which case the displayed WCS axis scales are determined at coords (1,1,...) in the base Frame.

UBND(*) = DOUBLE PRECISION (Given)

The Upper bounds of a region within the base Frame. See LBND.

FULL = LOGICAL (Given)

Display full information?

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_DSSFM
Displays information about the storage form of an NDF array

Description:

This routine displays a textual description of the storage form of the specified NDF array.

Invocation:

```
CALL KPG1_DSSFM( INDF, COMP, REPORT, FORM, SCALE, ZERO, SCTYP, STATUS )
```

Arguments:

INDF = INTEGER (Given)
An identifier for the ndf.

COMP = CHARACTER * (*) (Given)
The name of the NDF component (DATA, QUALITY or VARIANCE) to be displayed.

REPORT = LOGICAL (Given)
If .TRUE. the storage form information is displayed on standard output.

FORM = CHARACTER * (*) (Returned)
The storage form.

SCALE = DOUBLE PRECISION (Returned)
The scale factor. This will be set to 1.0 if the storage form does not support scaled storage.

ZERO = DOUBLE PRECISION (Returned)
The zero offset. This will be set to 0.0 if the storage form does not support scaled storage.

SCTYP = CHARACTER * (*) (Returned)
The scaled data type. This will be set to `' '` if the storage form does not support scaled storage.

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

KPG1_DWSOx
Obtains for the scale and offset for a linear transformation from world to data co-ordinates

Description:

This routine determines a linear transformation between world (pixel) co-ordinates and data co-ordinates obtained from the axis structure. A linear equation is solved using the two boundary conditions for the lower and upper bounds in world and data co-ordinates.

Invocation:

```
CALL KPG1_DWSOx( LBND, UBND, AXIS, SCALE, OFFSET, STATUS )
```

Arguments:

LBND = INTEGER (Given)
The lower bound of the axis array.

UBND = INTEGER (Given)
The upper bound of the axis array.

AXIS(LBND:UBND) = ? (Given)
The axis array.

SCALE = ? (Returned)
The scale factor of the linear axis to transform from world co-ordinates to data co-ordinates.

OFFSET = ? (Returned)
The offset of the linear axis at pixel 0 to transform from world co-ordinates to data co-ordinates.

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

Notes:

- There is a routine for the data types real or double precision: replace `" x"` in the routine name by R or D respectively, as appropriate. The axis array, and the returned scale and offset should have this data type as well.

KPG1_ELGAU
Calculates the axis ratio, inclination and minor-axis width of a star image, given the Gaussian widths of marginal profiles at 45-degree intervals

Description:

The reciprocal of the square of the width varies approximately like the length of an ellipse diameter as the angle of projection varies. The routine calculates the required parameters assuming this relationship holds, then iterates, calculating the expected deviation from this law and subtracting it from the data before calculating a new estimate. The solution of the ellipse equations is analogous to using the Stokes parameters of linear polarization to find the ellipse parameters.

Invocation:

```
CALL KPG1_ELGAU( SIG, SIG0, AXISR, THETA, STATUS )
```

Arguments:

SIG(4) = REAL (Given)
The Gaussian widths of the marginal profiles of the star in directions at 0, 45, 90 and 135 degrees to the x axis.

SIG0 = REAL (Returned)
The width of the minor axis of the ellipse.

AXISR = REAL (Returned)
The axis ratio of the ellipse.

THETA = REAL (Returned)
The inclination of the major axis to the x axis in radians (x through y positive).

STATUS = INTEGER (Returned)
The global status.

Notes:

The routine assumes a Gaussian profile for the star.

KPG1_ELNMx
Writes a range of element numbers into an array

Description:

This routine writes numbers sequentially to a one-dimensional array, where the numbers are between defined integer limits and are stepped by 1 (or -1) from one pixel to the next. In other words the array value takes the element number plus an offset. Only the first EL elements can be accommodated in the output array, should the section be larger than this.

Invocation:

```
CALL KPG1_ELNMx( LBND, UBND, EL, ARRAY, STATUS )
```

Arguments:

LBND = INTEGER (Given)
The first value to be written to the output array. If this is larger than argument UBND, the values will decrease by 1 for each element.

UBND = INTEGER (Given)
The last value to be written to the output array, provided there are sufficient elements to accommodate it. If not only EL values will be stored.

EL = INTEGER (Given)
The number of elements in the output array.

ARRAY(EL) = ? (Returned)
The array into which the sub-array is copied.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The ARRAY argument must have the data type specified.

KPG1_ENV0x
Reads a numerical value from an environment variable

Description:

This routine reads a numerical value from a specified environment variable. No error occurs if the environment variable is not defined, or has a non-numeric value, and the supplied value is returned unchanged.

Invocation:

```
CALL KPG1_ENV0x( VARNAM, VALUE, STATUS )
```

Arguments:

VARNAM = CHARACTER * (*) (Given)
The environment variable to check.

VALUE = ? (Returned)
The numerical value.

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

KPG1_ENVDF
Sees if an environment variable is defined

Description:
This routine returns a logical flag indicating if the specified environment variable is defined.

Invocation:
```
CALL KPG1_ENVDF( VARNAM, DEF, STATUS )
```

Arguments:

VARNAM = CHARACTER * (*) (Given)
The environment variable to check.

DEF = LOGICAL (Returned)
Is the environment variable defined?

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

Notes:

- This routine attempts to execute even if STATUS is set to an error on entry.

KPG1_EXPOx
Takes the exponential to an arbitrary base of an array and its variance

Description:

This routine fills the output array pixels with the results of taking the exponential of the pixels of the input array to the base specified, i.e. New_value = Base ** Old_value. If the result is bigger than the maximum-allowed value then a bad pixel value is substituted in the output array.

If variance is present it is computed as: 2 New_Variance = Old_Variance * (New_value * log Base)

Invocation:

```
CALL KPG1_EXPOx( BAD, EL, INARR, VAR, INVAR, BASE, OUTARR, OUTVAR, NERR, NERRV, STATUS
)
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of elements in the input output arrays.

INARR(EL) = ? (Given)
Array containing input image data.

VAR = LOGICAL (Given)
If .TRUE., there is a variance array to process.

INVAR(EL) = ? (Given)
Array containing input variance data, when VAR = .TRUE..

BASE = DOUBLE PRECISION (Given)
Base of exponential to be used. It must be a positive number.

OUTARR(EL) = ? (Write)
Array containing results of processing the input data.

OUTVAR(EL) = ? (Write)
Array containing results of processing the input variance data, when VAR = .TRUE..

NERR = INTEGER (Returned)
Number of numerical errors which occurred while processing the data array.

NERRV = INTEGER (Returned)
Number of numerical errors which occurred while processing the variance array, when VAR = .TRUE..

STATUS = INTEGER (Given)
Global status value

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.

KPG1_FFRx
Reads free-format floating-point data from a string

Description:

A buffer containing free-format values separated by spaces or commas is decoded and the numerical values extracted. It packages up the equivalent SLALIB calls (SLA_FLOTIN and SLA_DFLTIN) in order to obtain an array of values with a single line of code, and provides standard error handling. See SUN/67 for more details of the parsing technique and limitations.

If the desired number of values cannot be decoded from the string an error status SAI__ERROR is set.

Invocation:

```
CALL KPG1_FFRx( STRING, NVAL, ORIGIN, VALUES, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)
The string containing the values to be decoded and extracted.

NVAL = INTEGER (Given)
The number of values to extract from the string.

ORIGIN = INTEGER (Given and Returned)
On input this is the column or pointer in the string from where the decoding commences. On output it is advanced to the next number or the end of the buffer if there are no more numbers.

VALUES(NVAL) = ? (Returned)
The decoded values. These are initialised to zero before decoiding commences.

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

Notes:

- There is a routine for floating-point numeric data types: replace " x" in the routine name by D or R as appropriate. The array returned from the routine must have the data type specified. [routine_notes]...

KPG1_FFTBD
Takes the inverse (Backward) FFT of a real image

Description:

The input array should hold a Fourier transform of a purely real image, in Hermitian format as produced by KPG1_FFTFD, or KPG1_HMLTD. The inverse FFT of this image is taken and returned in OUT.

Invocation:

```
CALL KPG1_FFTBD( M, N, IN, WORK, OUT, STATUS )
```

Arguments:

M = INTEGER (Given)
Number of columns in the input array.

N = INTEGER (Given)
Number of rows in the input array.

IN(M, N) = DOUBLE PRECISION (Given)
The input array (a Fourier transform of a real image, stored in Hermitian format).

WORK($*$) = DOUBLE PRECISION (Given)
Work space. This must be at least ($3*$MAX(M, N) $+ 15$) elements long.

OUT(M, N) = DOUBLE PRECISION (Returned)
The inverse FFT of the input array (a purely real image). Note, the same array can be specified for both input and output, in which case the supplied values are overwitten.

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

KPG1_FFTBR
Takes the inverse (Backward) FFT of a real image

Description:

The input array should hold a Fourier transform of a purely real image, in Hermitian format as produced by KPG1_FFTFR, or KPG1_HMLTR. The inverse FFT of this image is taken and returned in OUT.

Invocation:

```
CALL KPG1_FFTBR( M, N, IN, WORK, OUT, STATUS )
```

Arguments:

M = INTEGER (Given)
Number of columns in the input array.

N = INTEGER (Given)
Number of rows in the input array.

IN(M, N) = REAL (Given)
The input array (a Fourier transform of a real image, stored in Hermitian format).

WORK($*$) = REAL (Given)
Work space. This must be at least (3$*$MAX(M, N) + 15) elements long.

OUT(M, N) = REAL (Returned)
The inverse FFT of the input array (a purely real image). Note, the same array can be specified for both input and output, in which case the supplied values are overwitten.

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

KPG1_FFTFx
Takes the forward FFT of a real image

Description:

The Fourier transform of the input (purely real) image is taken and returned in OUT. The returned FT is stored in Hermitian format, in which the real and imaginary parts of the FT are combined into a single array. The FT can be inverted using KPG1_FFTBx, and two Hermitian FTs can be multipled together using routine KPG1_HMLTx.

Invocation:

```
CALL KPG1_FFTFx( M, N, IN, WORK, OUT, STATUS )
```

Arguments:

M = INTEGER (Given)
Number of columns in the input image.

N = INTEGER (Given)
Number of rows in the input image.

IN(M, N) = ? (Given)
The input image.

WORK($*$) = ? (Given)
Work space. This must be at least ($3*$MAX(M, N) $+ 15$) elements long.

OUT(M, N) = ? (Returned)
The FFT in Hermitian form. Note, the same array can be used for both input and output, in which case the supplied values will be over-written.

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

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The input and output data arrays plus a work space must have the data type specified.

Implementation Status:

Some of the generated code looks a bit odd because the names of the PDA subroutines do not match the standard type naming conventions required by GENERIC. Explicit IF statements are included but since these compare constants (after generic is run) the compiler will optimize the checks out of the final runtime. The real fix is to fix the names of the PDA routines (and associated KPG routine).

KPG1_FHDAT
Converts the NDF history date into a more-pleasing format

Description:

This makes a few minor modifications to the date string obtained from NDF history records to make it more like UNIX and astronomical style. Specifically two hyphens around the month are replaced by spaces, and the second and third letters of the month are made lowercase.

Invocation:

```
CALL KPG1_FHDAT( DATE, STATUS )
```

Arguments:

DATE = CHARACTER * (*) (Given and Returned)

On input the NDF format for a date and time, namely YYYY-MMM-DD HH:MM:SS.SSS. On exit, the KAPPA format for a date and time, namely YYYY Mmm DD HH:MM:SS.SSS

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_FIGRx
Sets each element in a n-dimensional array to its grid co-ordinate

Description:

This routine sets each pixel in an n-dimensional array to its grid co-ordinate.

Invocation:

```
CALL KPG1_FIGRx( NDIMS, DIMS, EL, ARRAY, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
Number of array dimensions.

DIMS(NDIM) = INTEGER (Given)
Array of dimension sizes for each array dimension.

EL = INTEGER (Given)
First dimension of the array of grid co-ordinates. It must be at least the total number of elements, i.e. the product of the DIMS dimensions.

ARRAY(EL, NDIM) = ? (Returned)
The array containing the grid co-ordinates of each element given by the supplied dimensions. The values are stored such that the first co-ordinate for all pixels is then followed by second co-ordinate for all pixels and so on.

STATUS = INTEGER (Given & Returned)
Global status value

Notes:

- There is a routine for double precision, real, and integer: replace " x" in the routine name by D, R, or I as appropriate. The ARRAY argument must have the data type specified.
- The maximum number of dimensions which can be handled by this routine is equal to the symbolic constant NDF__MXDIM.
- The ordering of the ARRAY elements is counterintuitive, but that' s what AST_TRANN uses.

KPG1_FILLx
Sets all elements in a vectorised array to a specified value

Description:
This routine sets all the pixels in a one-dimensional array to a specified value.

Invocation:
```
CALL KPG1_FILL<T>( VALUE, EL, ARRAY, STATUS )
```

Arguments:

VALUE = ? (Given)
Value to be substituted in every pixel.

EL = INTEGER (Given)
The dimension of the array to be filled with a constant.

ARRAY(EL) = ? (Returned)
The output array containing a single value.

STATUS = INTEGER (Given & Returned)
Global status value

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B, or UB as appropriate. The VALUE and ARRAY arguments must have the data type specified.

KPG1_FLASx
Flashes image into a cell array

Description:

This subroutine converts a two-dimensional array into an integer cell array without scaling it. However, the cell array is computed by modulo arithmetic between the lowest and highest colour indices, or the complement of this should a negative scaling be required. The image is normally inverted for output on an image display, though this can be overridden.

Invocation:

```
CALL KPG1_FLASx( BAD, DIM1, DIM2, INARR, LOWPEN, HIPEN, INVERT, :  POSTIV, OUTARR, STATUS
)
```

Arguments:

BAD = LOGICAL (Given)
If true bad pixels will be processed. This should not be set to false unless the input array contains no bad pixels.

DIM1 = INTEGER (Given)
The first dimension of the two-dimensional array.

DIM2 = INTEGER (Given)
The second dimension of the two-dimensional array.

INARR(DIM1, DIM2) = ? (Given)
The array of data to be flashed.

LOWPEN = INTEGER (Given)
The lowest colour index to be used in case smaller values have been reserved for annotation.

HIPEN = INTEGER (Given)
The highest colour index to be used. Normally this is the number of greyscale intensities available on the chosen device.

INVERT = LOGICAL (Given)
True if the image is to be inverted for display.

POSTIV = LOGICAL (Given)
True if the display of the image is to be positive.

OUTARR(DIM1, DIM2) = INTEGER (Returned)
The cell array.

STATUS = INTEGER (Given)
Value of the status on entering this subroutine.

Notes:

- There is a routine for five numeric data types: replace " x" in the routine name by B, D, I, R, or W as appropriate. The array supplied to the routine must have the data type specified.
- The array is normally inverted so that the image will appear the correct way round when displayed as the GKS routine to display the image inverts it.

KPG1_FLCOx
Obtains a list of co-ordinates from a text file

Description:

This routine obtains a list of n-dimensional co-ordinates for a series of positions. The text file should contain on each line a set of free-format co-ordinates that defines a position, except for comment lines denoted by a hash or shriek in column 1. The file is opened using the supplied parameter. A pass is made through the file to see how many records it contains. Dynamic workspace is reserved to contain the co-ordinates, and the file is then rewound. Another pass is made through the file to read the co-ordinate values which are then stored in the workspace. Pointers to the workspace are returned. This workspace should be released by calling PSX_FREE when it is no longer needed.

Invocation:

```
CALL KPG1_FLCOx( PNAME, NDIM, POSCOD, NPOINT, IPCO, LBND, UBND, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of the parameter with which to associate the text file.

NDIM = INTEGER (Given)
The number of co-ordinate dimensions to read from the text file.

POSCOD(NDIM) = INTEGER (Given)
The column numbers of the co-ordinate information in order x, y, z, etc. These must be positive.

NPOINT = INTEGER (Returned)
The number of co-ordinate sets specified in the text file.

IPCO = INTEGER (Returned)
A pointer to workspace of type <HTYPE> and size NDIM by NPOINT holding the NDIM co-ordinates of each point.

LBND(NDIM) = ? (Returned)
The lower bounds of the input data, i.e. the minimum value for each co-ordinate.

UBND(NDIM) = ? (Returned)
The upper bounds of the input data, i.e. the maximum value for each co-ordinate.

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

Notes:

- There is a routine for each of the real or double-precision data types: replace " x" in the routine name by R or D respectively, as appropriate. The returned co-ordinate array will have this type, and the supplied bounds arrays should also have the specified data type.
- The number of points read from the file is reported at the normal reporting level.

KPG1_FLIPx
Reverses the pixels in an n-dimensional array along a specified dimension

Description:

This routine reverses the order of the pixels in an n-dimensional array along a specified dimension. The pixel values are unchanged. The array may have any number of dimensions.

Invocation:

```
CALL KPG1_FLIPx( NDIM, DIM, DATIN, IDIM, DATOUT, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
Number of array dimensions.

DIM(NDIM) = INTEGER (Given)
Array of dimension sizes for each array dimension.

DATIN($*$) = ? (Given)
The input NDIM-dimensional array.

IDIM = INTEGER (Given)
Number of the array dimension along which the pixel values are to be reversed (in the range 1 to NDIM).

DATOUT($*$) = ? (Returned)
Output array, with the pixels reversed.

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

Notes:

- There is a routine for each standard numeric type; replace " x" in the array name by D, R, I, W, UW, B or UB as appropriate. The data type of the arrays supplied must match the particular routine used.

KPG1_FLOOR
Returns the largest integer smaller than or equal to a supplied value

Description:

This routine returns the largest integer smaller than or equal to a supplied value.

Invocation:

```
RESULT = KPG1_FLOOR( VALUE )
```

Arguments:

VALUE = REAL (Given)

The value.

Function Value :

KPG1_FLOOR = INTEGER The largest integer smaller than or equal to the supplied value.

KPG1_FLPTH
Gets the full path to a file given relative to a specified root directory

Description:

This routine returns the full path to a file specified relative to the $ROOT directory. The supplied file spec should be a UNIX-style file spec., but the returned full file path will be translated into the VMS equivalent if required.

Invocation:

```
CALL KPG1_FLPTH( INSTR, OUTSTR, NC, STATUS )
```

Arguments:

ROOT = CHARACTER * (*) (Given)

An environment variable which is to be translated to get the full path to the root directory, such as HOME, KAPPA_DIR. If a blank value is supplied, then the current working directory is used ($PWD on Unix and $PATH on VMS).

INSTR = CHARACTER * (*) (Given)

The file spec relative to the root directory. E.g. fred.dat, adam/display.sdf, etc.

OUTSTR = CHARACTER * (*) (Returned)

The full file path. For example /home/dsb/fred.dat (DISK$USER:[DSB]FRED.DAT on VMS), or /home/dsb/adam/display.sdf (DISK$USER:[DSB.ADAM]DISPLAY.SDF on VMS). Returned blank if an error occurs.

NC = INTEGER (Returned)

Number of characters used in OUTSTR. Returned equal to 0 if an error occurs.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_FMEDx
Derives the unweighted fast median of a vector

Description:

This routine derives the unweighte 'fast' median of an array. The method used is based on Wirth' s algorithm for selecting the *K*th value, which is very fast compared with a full sort. If there is an even number of elements, the median is computed from the mean of the two elements that straddle the median.

Invocation:

```
CALL KPG1_FMEDx( BAD, EL, ARRAY, POINT, MEDIAN, NGOOD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If BAD is .TRUE. there may be bad pixels present in the vector and so should be tested. If BAD is .FALSE. there is no testing for bad values.

EL = INTEGER (Given)
The number of elements within the array to sort.

ARRAY(EL) = ? (Given and Returned)
The array whose median is to be found. On exit the array is partially sorted.

POINT(EL) = INTEGER (Returned)
Workspace to hold pointers to the original positions of the data before extraction and conversion into the ARRAY array.

MEDIAN = ? (Returned)
The fast median of the array.

NGOOD = INTEGER (Returned)
The number of elements actually used after bad values have been excluded.

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

Notes:

- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array and MEDIAN arguments supplied to the routine must have the data type specified.

KPG1_FRACx
Finds values corresponding to specified fractions of an array' s ordered distribution

Description:

This routine finds the values at defined fractions of an array' s ordered distribution, such as percentiles. Thus to find the upper-quartile value, the fraction would be 0.75. Since it uses an histogram technique rather than sorting the whole array, for efficiency, the result may not be exactly correct. However, the histogram has a large number of bins (10000), combined with linear interpolation between bins in the routine reduce the error. The histogram extends between the minimum and maximum data values.

The routine also has an iterative method, whereby outliers, which compress the vast majority of data values into a few bins, are excluded from the histogram. Clipping occurs from both ends. A contiguous series of bins are removed until the largest or smallest fraction is encountered. Where the rejection of bins end, defines new limits, encompassing the vast majority of values. A new histogram is calculated using these revised limits. The excluded outliers are still counted in the evaluation of the fractions. The criterion for iteration may need tuning in the light of experience. At present it is when there are fewer than 4% non-zero bins.

The iteration can still fail to find accurate fractional values if smallest and largest fractions are close to 0 or 1 and correspond to extreme outliers. The routine recognises this state and determines the values for each outlier fraction separately, and then uses the next interior fraction as the limit. Then the routine proceeds with the clipping described above.

Invocation:

```
CALL KPG1_FRACx( EL, ARRAY, NFRAC, FRAC, BAD, CLFRAC, VALUES, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements of the array to be analysed.

ARRAY(EL) = ? (Given)
The array for which the chosen statistics are required.

NFRAC = INTEGER (Given)
Number of fractional positions.

FRAC(NFRAC) = REAL (Given and Returned)
Fractional positions in the histogram in the range 0.0–1.0. On exit they are arranged into ascending order.

BAD = LOGICAL (Given and Returned)
True when bad pixels may be present.

CLFRAC(NFRAC) = REAL (Returned)
The clipped fractional positions in the histogram in the range 0.0–1.0 after iterative clipping of the histogram.

VALUES(NFRAC) = DOUBLE PRECISION (Returned)
Values corresponding to the ordered fractional positions in the histogram.

STATUS = INTEGER (Given & Returned)
Global status.

Notes:

- There is a routine for each numerical data type: replace "x" in the routine name by B, D, I, R, W, UB or UW as appropriate. The array supplied to the routine must have the data type specified.
- For integer types the number of bins does not exceed the data range. The number of bins is reduced as clipping occurs.

Deficiencies :

- The iterative algorithm is not especially efficient; for data with a very wide range the iterations can be numerous. A sigma-clipping approach to remove the outliers might be better.
- The adjustment of the limiting fractions is done for each limit separately, thus involving a further pass through the array. At present it finds the more extreme outlier first by comparing the bin number of the limits with respect to the mean bin number.

KPG1_GAUFx
Fits a Gaussian to a one-dimensional array of data

Description:

This routine fits a Gaussian to a one-dimensional array of data. It finds the amplitude, centre, sigma and background levels by an iterative method.

Invocation:

```
CALL KPG1_GAUFx( DATA, LBND, UBND, NITER, TOLL, AMP, X0, :  SIGMA, BACK, STATUS )
```

Arguments:

DATA(LBND:UBND) = ? (Given)
The data array to be fitted by a Gaussian.

LBND = INTEGER (Given)
The lower bound of the data array.

UBND = INTEGER (Given)
The upper bound of the data array.

NITER = INTEGER (Given)
The maximum number of refining iterations.

TOLL = REAL (Given)
The accuracy criterion: absolute for the centre and relative for the amplitude and width. The accuracy of back is assessed as a fraction of the Gaussian amplitude.

AMP = REAL (Returned)
The Gaussian amplitude.

X0 = REAL (Returned)
The Gaussian centre.

SIGMA = REAL (Returned)
The 'sigma' of the Gaussian.

BACK = REAL (Returned)
The background signal level.

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

Notes:

There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B, UB as appropriate. The array supplied to the routine must have the data type specified.

KPG1_GAUSx
Smooths a two-dimensional array using a symmetrical Gaussian PSF

Description:

The routine smooths the array A using a symmetrical Gaussian point-spread function and returns the result in the array B.

Invocation:

```
CALL KPG1_GAUSx( SIGMA, IBOX, SAMBAD, WLIM, NX, NY, BAD, VAR, A, B, BADOUT, WEIGHT,
AMAR, WMAR, STATUS )
```

Arguments:

SIGMA = REAL (Given)

Standard deviation of the Gaussian to be used for smoothing.

IBOX = INTEGER (Given)

Half-size, in pixels, of the box over which the Gaussian smoothing profile will be applied (the actual box used has an edge which is 2∗IBOX+1 pixels long). This defines the region within which the point spread function is non-zero.

SAMBAD = LOGICAL (Given)

If a .TRUE. value is given, then any " bad" pixels in the input array A will be propagated to the output array B (output values will be calculated for all other output pixels). If a .FALSE. value is given, then the WLIM argument is used to determine which output pixels will be bad (if any). This argument is not relevant if BAD is .FALSE..

WLIM = ? (Given)

The minimum weighting that can be used to compute a smoothed output pixel if SAMBAD is .FALSE.. Any output pixels falling short of the specified weight will be set to the bad value (invalid pixels carry no weight, others have Gaussian weights about the central pixel). The value must be greater than 0.0 and should be less than or equal to 1.0. This argument is not used if SAMBAD is .TRUE. or if BAD is .FALSE..

NX = INTEGER (Given)

First dimension of the two-dimensional array A.

NY = INTEGER (Given)

Second dimension of the two-dimensional array A.

BAD = LOGICAL (Given)

Whether or not it is necessary to check for bad pixels in the input array A.

VAR = LOGICAL (Given)

If a .FALSE. value is given for this argument, then smoothing will be performed as if the array supplied (A) is an array of data and the PSF will be used directly as specified. If a .TRUE. value is given, then smoothing will be applied as if the array is an array of variance values associated with an array of data; in this case, the effective PSF will be reduced in width by a factor 2 and the mean output values will be reduced to reflect the variance-reducing effect of smoothing.

A(NX, NY) = ? (Given)

Array containing the input image to be smoothed.

B(NX, NY) = ? (Returned)

Array to receive the smoothed output image.

BADOUT = LOGICAL (Returned)
Whether there are bad pixel values in the output array.

WEIGHT(2 * IBOX + 1) = ? (Returned)
Workspace for the Gaussian weighting function.

AMAR(NX) = ? (Returned)
Workspace for the weighted sum of array values.

WMAR(NX) = ? (Returned)
Workspace for the sum of pixel weights.

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

Notes:

There are routines for processing double precision and real data. Replace " x" in the routine name by D or R as appropriate. The data types of the WLIM, A, B, WEIGHT, AMAR, and WMAR arguments must match the routine used.

KPG1_GAXLB
Obtains an axis annotation for an NDF axis

Description:

This routine obtains an axis annotation from the parameter system. The suggested default is of the form " label (units)" when there is both an axis label and units, or " label" if the label but not the units are present. If neither are present a supplied default is used instead. If a bad status, other than abort, is returned by the parameter system when getting the value, the error is annulled and the output annotation is the suggested default value.

Invocation:

```
CALL KPG1_GAXLB( NDF, IAXIS, PNAXL, DEFAUL, AXSLAB, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

IAXIS = INTEGER (Given)
The number of the axis whose character components are to be used.

PNAXL = CHARACTER * (*) (Given)
The name of the ADAM parameter from which the annotation will be obtained.

DEFAUL = CHARACTER * (*) (Given)
The suggested default when the NDF axis does not have a label, and the actual value returned when a null (!) value or any other non-abort bad status is obtained from the parameter system.

AXSLAB = CHARACTER * (*) (Returned)
The annotation obtained from the parameter system. The dynamic default is limited to 128 characters.

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

KPG1_GTCHV
Obtains a vector of choices from the environment

Description:

This routine gets up to MAXVAL strings from the user, selected from those supplied in OPTS. The indices of the supplied strings within OPTS are returned. The user supplies the strings in the form of a GRP group expression, using the default GRP control characters.

The user may supply an integer value instead of a string, in which case the integer is understood to be the index of the required string within OPTS. If the supplied list of strings contains the integer itself, then the integer is understood to be a string, not an index.

Invocation:

```
CALL KPG1_GCHMV( PARAM, NOPT, OPTS, MAXVAL, IDEF, NVAL, VALS, STATUS )
```

Arguments:

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

NOPT = INTEGER (Given)
The number of available options supplied in OPTS.

OPTS(NOPT) = CHARACTER * (*) (Given)
An array holding the options from which the user must choose. Leading and trailing white space is ignored. Blank options are not allowed.

MAXVAL = INTEGER (Given)
The maximum number of choices.

IDEF(MAXVAL) = INTEGER (Given)
The indices within OPTS of the default strings to use if a null (!) value is supplied for the parameter. If the first value is zero, a null parameter value results in a PAR__NULL status being returned.

NVAL = INTEGER (Given)
The number of choices supplied.

VALS(MAXVAL) = INTEGER (Returned)
The indices within OPTS of the selected options.

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

Notes:

- Case is insignificant when comparing supplied strings with available options.
- A dynamic default is set for the parameter before accessing it if IDEF supplied suitable defaults. The default consists of a comma-separated list of the default options.

KPG1_GDARE
Defines a region within the current PGPLOT window

Description:

This subroutine defines a two-dimensional region in the current PGPLOT window. The region is defined by the given position justification, and the linear fraction along each axis of the current PGPLOT window; and an aspect ratio. The linear fraction is applied first followed by the aspect-ratio constraint.

Invocation:

```
CALL KPG1_GDARE( JUST, FRACT, ASPECT, X1, X2, Y1, Y2, STATUS )
```

Arguments:

JUST = CHARACTER * (*) (Given)

Justification of the new region specified by a two-character code. The first character controls the vertical location, and may be T, B, or C to create the new region at the top, bottom, or in the centre respectively. The second defines the horizontal situation, and may be L, R, or C to define a new region to the left, right, or in the centre respectively. Thus a code of BR will make a new region in the bottom-right corner. The justification code need not be in uppercase.

FRACT(2) = REAL (Given)

The fractional size of the new region applied along each axis. So values of 0.5,0.5 would create a picture 0.25 the area of the current window.

ASPECT = REAL (Given)

The aspect ratio of the new region (x/y). If the value is negative, no aspect-ratio constraint is applied to define the new region.

X1 = REAL (Returned)

Lower x world co-ordinate in the current window of the new region.

X2 = REAL (Returned)

Upper x world co-ordinate in the current window of the new region.

Y1 = REAL (Returned)

Lower y world co-ordinate in the current window of the new region.

Y2 = REAL (Returned)

Upper y world co-ordinate in the current window of the new region.

STATUS = INTEGER (Given and Returned)

Global status value.

KPG1_GDBND
Gets the bounds of a new PGPLOT window from the environment

Description:

This routine defines a two-dimensional region in the current PGPLOT window, getting the bounds of the region from the environment.

Invocation:

```
CALL KPG1_GDBND( PNLOW, PNUPP, LBND, UBND, STATUS )
```

Arguments:

PNLOW = CHARACTER * (*) (Given)
Parameter name of lower-bound co-ordinates that defines a region.

PNUPP = CHARACTER * (*) (Given)
Parameter name of lower-bound co-ordinates that defines a region.

LBND(2) = REAL (Returned)
Co-ordinates of the lower bound that defines a region.

UBND(2) = REAL (Returned)
Co-ordinates of the upper bound that defines a region.

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

KPG1_GDFND
Selects the highest picture of a given name with WCS within the current AGI picture

Description:

This routine searches forwards through the AGI database for a picture of a given name that lies within the current picture and has an associated AST Plot structure. The the current picture itself is included in the search. If such a picture is found it becomes the new current picture. Otherwise, a bad status will be returned, and the current picture is unchanged.

This routine is like KPG1_AGFND except that KPG1_AGFDN does not require the returned picture to have an associated Plot.

Invocation:

```
CALL KPG1_GDFND( NAME, IPIC, STATUS )
```

Arguments:

NAME = CHARACTER * (*) (Given)
The name of the picture to be searched for in the graphics database.

IPIC = INTEGER (Returned)
The picture identifier of the most-recent picture named NAME.

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

KPG1_GDFNP
Selects the highest picture of a given name that embraces a position and has WCS

Description:

This routine returns an AGI identifier for the youngest picture that has the specified name, embraces the given position and lies within the bounds of the current picture. In this sense it is like AGI_RCLP. However, it also applies an extra requirement that the returned picture should have associated WCS. If no picture with associated WCS can be found, then it abandons this extra requirement and just returns the youngest picture encompassing the supplied position.

Invocation:

```
CALL KPG1_GDFNP( NAME, X, Y, IPIC, STATUS )
```

Arguments:

NAME = CHARACTER * (*) (Given)
The name of the picture to be searched for in the graphics database.

X = REAL (Given)
X position of test point

Y = REAL (Given)
Y position of test point

IPIC = INTEGER (Returned)
The picture identifier of the most-recent picture named NAME.

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

KPG1_GDGET
Gets the AST Plot associated with a graphics-database picture

Description:

This routine makes the specified graphics-database (AGI) picture current, creates a corresponding PGPLOT viewport and window, and returns a Plot associated with the picture.

On exit, the PGPLOT viewport corresponds to the area encompassed by the specified picture. The world co-ordinate bounds within this viewport are set so that the PGPLOT world co-ordinate system is millimetres from the bottom-left corner of the view surface. This corresponds to the Base (GRAPHICS) Frame in the returned Plot.

The returned Plot will normally be obtained from the MORE structure in the graphics database (where it was stored by a previous AST-based application). The Base Frame will be a GRAPHICS Frame, giving millimetres from the bottom-left corner of the view surface.

If no Plot is available in the database, then an initial PLOT is created containing a Base Frame with Domain GRAPHICS (giving millimetres from the bottom-left corner of the view surface), and a Current Frame corresponding to AGI world co-ordinates. The Frame to represent AGI world co-ordinates in the Plot may be supplied by the calling application (for instance, an application may supply a PIXEL Frame on the assumption that AGI world co-ordinates are pixel co-ordinates). If no such Frame is supplied then a simple Frame is used, with Domain set to AGI_WORLD.

A third Frame may optionally be added to the Plot representing AGI DATA co-ordinates. This Frame will have Domain AGI_DATA, and the Mapping from world to data co-ordinates will be given by the TRANSFORM structure stored with the picture in the database. If present, it will be the Current Frame in the Plot on exit. See " Usage" below for warnings about using this option.

Finally, some other Frames are added to the Plot representing various normalised co-ordinates:

BASEPIC: The co-ordinates of the bottom-left corner of the BASE picture are (0,0). The shorter dimension of the BASE picture has length 1.0, and the other axis has a length greater than 1.0.

NDC: Normalized device co-ordinates. The bottom-left corner of the screen is (0,0) and the top-right corner is (1,1).

CURPIC: The co-ordinates of the bottom-left corner of the current picture are (0,0). The shorter dimension of the current picture has length 1.0, and the other axis has a length greater than 1.0.

CURNDC: The co-ordinates of the bottom-left corner of the current picture are (0,0), and the top-right corner is (1,1).

If the Plot read from the database already contains any of these Frames then they are retained and no new Frame is added.

Invocation:

```
CALL KPG1_GDGET( IPIC, WCFRM, MKDATA, IPLOT, STATUS )
```

Arguments:

IPIC = INTEGER (Given)

The AGI identifier for the picture. If a value of -1 is supplied, the identifier for the current picture is used.

WCFRM = INTEGER (Given)

A pointer to an AST Frame which will be used to describe AGI world co-ordinates if the picture has no associated Plot. This argument is ignored if the picture already has a Plot stored with it in the database. If a null pointer (AST__NULL) is supplied, then a default Frame is used with Domain set to AGI_WORLD.

MKDATA = LOGICAL (Given)
Should a Frame with Domain AGI_DATA be included in the returned Plot to represent AGI DATA co-ordinates? This is ignored if the picture has a Plot already stored with it in the database.

IPLOT = INTEGER (Returned)
An AST pointer to the Plot. Returned equal to AST__NULL if an error occurs.

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

Notes:

- The PGPLOT interface to the AGI library should be opened before calling this routine.

KPG1_GDNEW
Creates a new DATA picture with ancillary pictures

Description:

This routine returns identifiers for a new DATA picture together with various ancillary pictures in the graphics database. On exit, the new DATA picture is the current AGI picture, and the PGPLOT viewport matches the DATA picture.

A FRAME picture is only created if it would contain something other than the DATA picture (this is assumed to be the case if any ancillary pictures are requested, or if non-zero margins are requested around the DATA picture). If created, the FRAME picture has the maximum possible size. The DATA picture is then created with a size which allows all the requested ancillary pictures to be created within the FRAME picture.

Various environment parameters may be used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between application.

Invocation:

```
CALL KPG1_GDNEW( COMMNT, MARGIN, NP, PNAME, PSIDE, PSIZE, SASPEC, BOX, IPICD, IPICF,
IPIC, STATUS )
```

Arguments:

COMMENT = CHARACTER * (*) (Given)
A comment to store with the new pictures added to the AGI database. This will usually by an indication of the application being run (egKAPPA_DISPLAY).

MARGIN(4) = REAL (Given)
The width of the borders to leave round the DATA picture, given as fractions of the corresponding dimension of the current picture. These should be supplied in the order bottom, right, top, left.

NP = INTEGER (Given)
The number of extra pictures to be included in the FRAME pictures (the DATA picture itself is not included in this list). Margins are left round the DATA picture with widths given by MARGIN. Any extra pictures are placed outside these margins, in positions described by PSIDE and PSIZE.

PNAME(NP) = CHARACTER * (*) (Given)
The names to store in the AGI database with the NP extra pictures.

PSIDE(NP) = CHARACTER * 1 (Given)
Each element of this array should be one of L, R, T or B. It indicates which side of the FRAME picture an extra picture is to be placed. For Left and Right, the extra picture occupies the full height of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the far Left or Right of all previously created pictures. For Top or Bottom, the extra picture occupies the full width of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the top or bottom of all previously created pictures. Ignored if NP is zero.

PSIZE(NP) = REAL (Given)
The size of each extra picture. For Left and Right pictures, this is the width of the picture, and the value is given as a fraction of the width of the current picture. For Top and Bottom pictures, it is the height of the picture, and it is given as a fraction of the height of the current picture. Ignored if NP is zero.

SASPEC = REAL (Given)
The aspect ratio with which a new DATA picture should be created. This is the height divided by the width of the DATA picture, assuming equal scales on each axis (egmetres). A value of zero

causes the DATA picture to have the aspect ratio which produces the largest picture. The actual value used will depend on the value supplied for the FILL parameter (see above).

BOX(4) = DOUBLE PRECISION (Given)
The world co-ordinate bounds to give to the DATA picture if a new DATA picture is created. These should normally be pixel co-ordinates. The (x,y) co-ordinates of the bottom left corner should be given in elements 1 and 2, and the (x,y) co-ordinates of the top right corner should be given in elements 3 and 4. If the box has zero area, then world co-ordinates are set to centimetres from the bottom left corner of the DATA picture.

IPICD = INTEGER (Returned)
An AGI identifier for the DATA picture.

IPICF = INTEGER (Returned)
An AGI identifier for the FRAME picture. The world co-ordinate system is inherited from the current picture on entry. If no FRAME picture was created, then an AGI identifier for the current picture is returned.

IPIC(NP) = INTEGER (Returned)
An array of AGI identifiers corresponding to the extra pictures requested in ZSIDE and PSIZE. The world co-ordinate system for each picture is inherited from the FRAME picture. The actual size of a picture may be less than the requested size if there is insufficient room left in the FRAME picture to give it its requested size. Identifiers for pictures which would have zero size (i.e. fall completely outside the FRAME picture) are returned equal to -1, but no error is reported.

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

Notes:

- Picture identifiers are returned equal to -1 if an error occurs, or if the picture cannot be created.

Environment Parameters

FILL = _LOGICAL (Read)
TRUE if the supplied aspect ratio (SASPEC) is to be ignored, creating the largest possible DATA picture within the current picture. When FILL is FALSE, the DATA picture is created with the supplied aspect ratio. Not accessed if argument SASPEC is supplied equal to zero (i.e. (SASPEC .EQ. 0.0) implies FILL=YES)

KPG1_GDOLD
Creates a new DATA picture with ancillary pictures aligned with an existing DATA picture

Description:

This routine creates a new DATA picture aligned with an existing DATA picture, together with any requested ancillary pictures. On exit, the new DATA picture is the current picture, and the current PGPLOT viewport corresponds to this picture.

A FRAME picture is only created if it would contain something other than the DATA picture (this is assumed to be the case if any ancillary pictures are requested, or if non-zero margins are requested around the DATA picture). Ancillary pictures are given their requested sizes except that they are clipped at the bounds of the original current picture. The FRAME picture is also clipped at the bounds of the original current picture.

Invocation:

```
CALL KPG1_GDOLD( MARGIN, COMMNT, NP, PNAME, PSIDE, PSIZE, IPICD, IPICD0, IPICF, IPIC,
STATUS )
```

Arguments:

MARGIN(4) = REAL (Given)

The width of the borders to leave round the DATA picture, given as fractions of the corresponding dimension of the current picture. These should be supplied in the order bottom, right, top, left. They may be negative.

COMMNT = CHARACTER * (*) (Given)

A comment to store with the new pictures added to the AGI database. This will usually by an indication of the application being run (e.g. KAPPA_DISPLAY).

NP = INTEGER (Given)

The number of extra pictures to be included in the FRAME picture (the DATA picture itself is not included in this list). Margins are left round the DATA picture with widths given by MARGIN. Any extra pictures are placed outside these margins, in positions described by PSIDE and PSIZE.

PNAME(NP) = CHARACTER * (*) (Given)

The names to store in the AGI database with the NP extra pictures.

PSIDE(NP) = CHARACTER * 1 (Given)

Each element of this array should be one of L, R, T or B. It indicates which side of the FRAME picture an extra picture is to be placed. For Left and Right, the extra picture occupies the full height of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the far Left or Right of all previously created pictures. For Top or Bottom, the extra picture occupies the full width of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the top or bottom of all previously created pictures. Ignored if NP is zero.

PSIZE(NP) = REAL (Given)

The size of each extra picture. For Left and Right pictures, this is the width of the picture, and the value is given as a fraction of the width of the current picture. For Top and Bottom pictures, it is the height of the picture, and it is given as a fraction of the height of the current picture. Ignored if NP is zero.

IPICD = INTEGER (Given)

An AGI identifier for the existing DATA picture.

IPICD0 = INTEGER (Returned)
An AGI identifier for the new DATA picture.

IPICF = INTEGER (Returned)
An AGI identifier for the FRAME picture. Thw world cooridnate system is inherited form the current picture on entry. If no FRAME picture was created, then an AGI identifier for the current picture is returned.

IPIC(NP) = INTEGER (Returned)
An array of AGI identifiers corresponding to the extra pictures requested in ZSIDE and PSIZE. The world co-ordinate system for each picture is inherited from the FRAME picture. The actual size of a picture may be less than the requested size if there is insufficient room left in the FRAME picture to give it its requested size. Identifiers for pictures which would have zero size (i.e. fall completely outside the FRAME picture) are returned equal to -1, but no error is reported.

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

Notes:

- Picture identifiers are returned equal to -1 if an error occurs, or if the picture cannot be created.

KPG1_GDPUT
Saves an AST Plot with a graphics-database picture

Description:

This routine saves the supplied AST Plot (see SUN/210) in the AGI database (see SUN/48) within the MORE structure of the specified picture. It can be retrieved if necessary using KPG1_GDGET (see the prologue of KPG1_GDGET for more information about using these two routines).

If the supplied Plot contains a " AGI Data" Frame with the Domain given by DDOM in which the axes are scaled and shifted versions of the axes of the AGI world co-ordinate Frame (specified by argument WDOM), then a TRANSFORM structure defining AGI Data co-ordinates is stored with the DATA picture. This is purely for the benefit of non-AST based applications which may use AGI Data co-ordinates (AST-based applications should always use the Plot stored with the picture in preference to the TRANSFORM structure stored in the AGI database).

Any Frames that are Regions and have a Domain beginning with " ROI" are deleted from the Plot before saving it. Also, any Frames that have an Ident value beginning with " ROI" are also deleted from the Frame (these may be added by KPG1_ASGET).

Invocation:

```
CALL KPG1_GDPUT( IPIC, WDOM, DDOM, IPLOT, STATUS )
```

Arguments:

IPIC = INTEGER (Given)
The AGI identifier for the picture. A value of -1 causes the Plot to be stored with the current picture.

WDOM = CHARACTER * (*) (Given)
Domain name of the co-ordinate Frame within IPLOT corresponding to AGI world co-ordinates. " AGI_WORLD" is used if a blank value is supplied.

DDOM = CHARACTER * (*) (Given)
Domain name of the co-ordinate Frame within IPLOT corresponding to AGI data co-ordinates. " AXIS" is used if a blank value is supplied.

IPLOT = INTEGER (Given)
An AST pointer to the Plot to be stored with the picture. If AST__NULL is supplied, any existing Plot stored with the picture is deleted.

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

Notes:

- The Base (GRAPHICS) Frame in the Plot should represent millimetres from the bottom left corner of the view surface.
- An error is reported if the Plot contains any Frames which have the Domain AGI_DATA.
- The PGPLOT interface to the AGI library should be opened before calling this routine.
- The Plot is stored in a component of the MORE structure named " AST_PLOT" and with type " WCS" .

KPG1_GDQPC
Returns the extent of the current picture

Description:

This routine makes the current PGPLOT bviewport and window match the current AGI picture, and returns the extent of the picture in AGI world co-ordinates and physical co-ordinates (metres).

Invocation:

```
CALL KPG1_GDQPC( X1, X2, Y1, Y2, XM, YM, STATUS )
```

Arguments:

X1 = REAL (Returned)
The X world co-ordinate of the bottom left corner.

Y1 = REAL (Returned)
The Y world co-ordinate of the bottom left corner.

X2 = REAL (Returned)
The X world co-ordinate of the top right corner.

Y2 = REAL (Returned)
The Y world co-ordinate of the top right corner.

XM = REAL (Returned)
The extent of the Y axis in metres.

YM = REAL (Returned)
The extent of the Y axis in metres.

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

Notes:

- The PGPLOT interface to the AGI library should be opened before calling this routine.
- The PGPLOT viewport corresponds to the current AGI picture on exit.

KPG1_GDWIN
Sets PGPLOT world co-ordinates to be the world co-ordinates of the specified AGI picture

Description:

This routine finds the bounds of the current PGPLOT viewport within the world-co-ordinate system of a specified AGI picture, and sets the PGPLOT window accordingly.

Invocation:

```
CALL KPG1_GDWIN( IPIC, STATUS )
```

Arguments:

IPIC = INTEGER (Given)

The AGI picture identifier. If -1, then the current picture is used.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_GETIM
Gets locators to an IMAGE-type structure and structure holding the data array for data input

Description:

Returns a locator to an IMAGE-type data-structure associated with a supplied parameter name, if a valid locator is not supplied. The DATA_ARRAY component is examined to determine whether it is primitive or a structure. Given the former, the second locator returned is the same as that to the IMAGE structure; on the other hand, DATA_ARRAY is tested for being a simple ARRAY, if it is not an error status is set, if it is the second locator returned points to the DATA_ARRAY structure. The simple ARRAY structure is searched for origin and bad-pixel information. Should the origin be not at 0 for each dimension, and/or the bad-pixel flag be set to false, warning messages are made.

It will not handle other NDF variants save report an error.

Invocation:

```
CALL KPG1_GETIM( PARNAM, LOCAT, DATLOC, DNAME, ORIGIN, STATUS )
```

Arguments:

PARNAM = CHARACTER * (*) (Given)

Parameter name associated with the input IMAGE-type structure. If this is blank, the supplied locator, LOCAT, will be assumed to be the locator to the top-level of an NDF (IMAGE-type) structure.

LOCAT = CHARACTER * (DAT__SZLOC) (Given & Returned)

On input when PARNAM is blank, there will be no association with a parameter; the locator is assumed to point at the top level of an NDF, and will be unchanged on exit.

If PARNAM is non blank, the input value of LOCAT is ignored. On exit LOCAT is the locator to the object associated with the given parameter name, unless status is bad, whereupon this locator is annulled.

DATLOC = CHARACTER * (DAT__SZLOC) (Returned)

The locator to the structure containing the primitive form of the data array. If it is the top-level of the NDF (IMAGE) structure, it will be a clone of LOCAT. Either way it will require annulment. If status is bad on exit this locator is annulled.

DNAME = CHARACTER * (DAT__SZNAM) (Returned)

The name of the data array as this will be needed for access, and it is different depending on its location.

ORIGIN(DAT__MXDIM) = INTEGER (Returned)

The origin of the data array for each dimension. It is set to 1 for non-existent dimensions. This argument returned so that if an output NDF is being created is will not be invalidated because of different origins in its ARRAY-type components.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- This a stop-gap routine until the remainder of KAPPA IMAGE-format applications are converted to NDF.

KPG1_GETOUTLINE
Retrieve an STC polygon describing the spatial extent of an NDF

Description:

If The NDF contains an OUTLINE extension, it is expected to be a character array containing an STC-S description of a polygon. If this is the case, the polygon is returned as the function value. Otherwise a NULL pointer is returned.

Invocation:

```
RESULT = KPG1_GETOUTLINE( Indf, STATUS )
```

Arguments:

INDF = INTEGER (Given)
Identifier for the NDF.

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

Returned Value:

A pointer to an AST Region, or AST__NULL if no Region can be created.

KPG1_GETYP
Obtains a valid HDS primitive data type via a parameter

Description:

This routine obtains and validates an HDS primitive data type, including the _CHAR∗n and LITERAL forms. Unamibguous abbreviations may be supplied. The user is reprompted up to four times if the value is not one of the allowed types.

Invocation:

```
CALL KPG1_GETYP( PARAM, HDSTYP, STATUS )
```

Arguments:

PARAM = CHARACTER ∗ (∗) (Given)
The parameter through which the value is to be obtained. The parameter data type should be LITERAL or _CHAR.

HDSTYP = CHARACTER ∗ (DAT__SZTYP) (Given)
The unabbreviated HDS primitive type obtained.

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

Notes:

- Error reports are made and flushed inside the loop if the _CHAR∗n has an invalid " n" value.
- Bad status is returned if no valid type has been obtained.
- _INT64 and _INTEGER require 5 characters for disambiguation. This is a change from when only _INTEGER was available.

KPG1_GHSTx
Calculates the histogram of an array of data

Description:

This routine calculates the truncated histogram of an array of data between defined limits and in a defined number of bins.

Invocation:

```
CALL KPG1_GHSTx( BAD, SIZE, ARRAY, WGTS, WEIGHT, NUMBIN, CUMUL, :  VALMAX, VALMIN, HIST,
STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If .TRUE., bad pixels will be processed. This should not be set to false unless the input array contains no bad pixels.

SIZE = INTEGER (Given)
The dimension of the array whose histogram is required.

ARRAY(SIZE) = ? (Given)
The input data array.

WGTS(SIZE) = DOUBLE PRECISION (Given)
The weight associed with each element of the input data array. Ignored if WEIGHT is zero.

WEIGHT = DOUBLE PRECISION (Given)
If non-zero, this is the increment in weight that contributes a single count to a histogram bin. This if WEIGHT is 0.1, and WGTS(10) is 0.232, the histogram bin containing the value of ARRAY(10) will be incremented by 2 (i.e. int(0.232/0.1). If WEIGHT is ZERO, then the WGTS array is ignored and every element of ARRAY is assigned a unit weight.

NUMBIN = INTEGER (Given)
Number of bins used in the histogram. For integer data types this should result in a bin width of at least one unit. So for example, byte data should never have more than 256 bins.

CUMUL = LOGICAL (Given)
Is a cumulative histogram required?

VALMAX = ? (Given and Returned)
Maximum data value included in the array. If this is supplied as VAL__BAD<T>, then the actual maximum value in the ARRAY is found and used, and returned on exit.

VALMIN = ? (Given and Returned)
Minimum data value included in the array. If this is supplied as VAL__BAD<T>, then the actual minimum value in the ARRAY is found and used, and returned on exit.

HIST(NUMBIN) = INTEGER (Returned)
Array containing the histogram.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- If VALMAX or VALMIN is supplied equal to VAL__BAD<T>, then the actual maximum and minimum values will be found (by searching the ARRAY), and used.

- There is a routine for all numeric data types: replace "`x`" in the routine name by B, D, R, I, UB, UW, or W as appropriate. The arguments ARRAY, VALMAX, and VALMIN must have the data type specified.

KPG1_GILST
Selects integers within a range of values

Description:

This routine selects integer numbers from LONUM to UPNUM. The routine gets a character string from the user and parses it to extract the specified numbers. The default selection is the all integers within the range LONUM to UPNUM.

Invocation:

```
CALL KPG1_GILST( LONUM, UPNUM, MAXLIN, PARAM, FLAG, NUMBER, NDISP, STATUS )
```

Arguments:

LONUM = INTEGER (Given)
The lower limit of the selection range.

UPNUM = INTEGER (Given)
The upper limit of the selection range.

MAXLIN = INTEGER (Given)
The max. number of numbers can be selected.

PARAM = CHARACTER∗(∗) (Given)
The name of the parameter used to get the number specification from the user. It can be a list, separated by commas, comprising any reasonable combination of the following formats:

ALL or ∗ - All integers between 1 and NUM

xx,yy,zz - A list of integers.

xx:yy - All integers between xx and yy inclusively. When xx is omitted the range begins from 1, when yy is omitted the range ends with UPNUM.

Any number can be specified more than once but it has the same effect as just entering it once.

FLAG(LONUM : UPNUM) = INTEGER (Returned)
A temporary flag array used to flag the number which has been selected.

NUMBER(MAXLIN) = INTEGER (Returned)
The selected numbers.

NDISP = INTEGER (Returned)
The number of selected numbers.

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

KPG1_GNDFP
Gets an NDF or NDF section with a specified number of significant dimensions, but also may select one further significant iteration dimension

Description:

The supplied parameter name PARNDF is associated with an NDF through the environment, and an identifier is obtained for the NDF using the specified access mode. Each axis of the NDF is checked to see if is significant (i.e. has a size greater than 1). The index of each significant axis is returned in SDIM; and the bounds of the axes are returned in LBND and UBND. If the number of significant axes is exactly one more than NDIM, one axis may be chosen for later iteration over 'planes' of NDIM dimensions. If there are fewer than NDIM significant dimensions then the insignificant dimensions are used (starting from the lowest) to ensure that the required number of dimensions are returned.

Invocation:

```
CALL KPG1_GNDFP( PARNDF, PARPAX, NDIM, MODE, INDF, SDIM, LBND, UBND, PERPAX, STATUS
)
```

Arguments:

PARNDF = CHARACTER * (*) (Given)
The parameter name to obtain the NDF.

PARPAX = CHARACTER * (*) (Given)
The parameter name to obtain the axis to iterate over the 'planes'.

NDIM = INTEGER (Given)
The number of dimensions required.

MODE = CHARACTER * (*) (Given)
The access mode required for the NDF.

INDF = INTEGER (Returned)
An identifier for the NDF.

SDIM(NDIM) = INTEGER (Returned)
The indices of the significant dimensions.

LBND(NDIM) = INTEGER (Returned)
The lower bounds of the NDF.

UBND(NDIM) = INTEGER (Returned)
The upper bounds of the NDF.

PERPAX = INTEGER (Returned)
The index of the axis perpendicular to the 'plane' if there is exactly one more significant axis than NDIM, otherwise set to zero.

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

KPG1_GNLBU
Obtains an axis annotation for NDF data or variance

Description:

This routine obtains an axis annotation from the parameter system. The suggested default has the form " label (units)" when the NDF has both a label and units, or " label" if there is a label but not units. When neither component is present the default is the component name follwed by " values" . The units are squared for the variance component. If a bad status, other than abort, is returned by the parameter system when getting the value, the error is annulled and the output annotation is the suggested default value.

Invocation:

```
CALL KPG1_GNLBU( NDF, PNLAB, COMP, AXSLAB, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

PNLAB = CHARACTER * (*) (Given)
The name of the ADAM parameter from which the annotation will be obtained.

COMP = CHARACTER * (*) (Given)
Name of the NDF array component: ' DATA', ' QUALITY', ' VARIANCE', or ' ERROR', though it is used literally and not checked to be a member of this set.

AXSLAB = CHARACTER * (*) (Returned)
The annotation obtained from the parameter system. The dynamic default is limited to 128 characters.

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

KPG1_GNTIT
Obtains a title using the NDF title as the suggested default

Description:

This routine obtains a title from the parameter system. The suggested default is title of the input NDF if it has one, otherwise a supplied default is used. If a bad status, other than abort, is returned by the parameter system when getting the value, the error is annulled and the output title is the default value.

Invocation:

```
CALL KPG1_GNTIT( NDF, PNTITL, DEFAUL, TITLE, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

PNTITL = CHARACTER * (*) (Given)
The name of the ADAM parameter from which the title will be obtained.

DEFAUL = CHARACTER * (*) (Given)
The suggested default when the NDF does not have a title, or the actual value returned when a null (!) value or any other non-abort bad status is obtained from the parameter system.

TITLE = CHARACTER * (*) (Returned)
The title obtained from the parameter system.

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

KPG1_GPCOL
Obtains the red, green and blue intensities of a colour by value or by name for the standard colour set

Description:

This routine obtains a colour via the ADAM parameter system. The colour may be either a named colour from the colour set; or red, green, and blue intensities separated by commas or spaces. If the named colour does not exist, or there is a problem extracting and converting the RGB values, or the RGB values are out of the range 0.0 to 1.0, an error is reported immediately and the user is prompted for another value.

Invocation:

```
CALL KPG1_GPCOL( PNCOL, RGBINT, STATUS )
```

Arguments:

PNCOL = CHARACTER * (*) (Given)

The name of the ADAM parameter used to obtain the colour. It must be of type _CHAR or LITERAL.

RGBINT(3) = REAL (Returned)

The red, green and blue intensities of the selected colour. They are normalised to the range 0.0 to 1.0.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_GRAPH
Draws a line graph

Description:

Opens a graphics device and draws a graph displaying a supplied set of points. Each point is defined by an X and Y value, plus an optional error bar. An AST Plot is returned so that the calling application can add further graphics to the plot if needed. When complete, the calling application should annul the Plot, and close the workstation:

CALL AST_ANNUL(IPLOT, STATUS) CALL AGP_DEASS(' DEVICE' , .FALSE., STATUS)

Various environment parameter names are used by this routine, to encourage uniformity in parameter naming, and behaviour. See KPG1_GRPHW for details.

Invocation:

```
CALL KPG1_GRAPH( N, X, Y, NSIGMA, YSIGMA, XLAB, YLAB, TTL, XSYM, YSYM, MODE, NULL, XL,
XR, YB, YT, APP, QUIET, LMODE, BSCALE, IPLOT, STATUS )
```

Arguments:

N = INTEGER (Given)
Number of points

X(N) = REAL (Given)
X value at each point. These are scaled by the value supplied in BSCALE(1) to generate the axis labels.

Y(N) = REAL (Given)
Y value at each point. These are scaled by the value supplied in BSCALE(2) to generate the axis labels.

NSIGMA = REAL (Given)
Controls the length of the vertical error bars. A value of zero suppresses error bars. Otherwise error bars are drawn which extend by from Y - NSIGMA∗YSIGMA to Y + NSIGMA∗YSIGMA.

YSIGMA(N) = REAL (Given)
The standard deviation associated with each Y value.

XLAB = CHARACTER ∗ (∗) (Given)
A default label for the X axis. Only used if the user does not supply an alternative. Trailing spaces are ignored. If a Plot is supplied via IPLOT, then the " Label(1)" attribute in the Plot is used as the default in preference to XLAB.

YLAB = CHARACTER ∗ (∗) (Given)
A default label for the Y axis. Only used if the user does not supply an alternative. Trailing spaces are ignored. If a Plot is supplied via IPLOT, then the " Label(2)" attribute in the Plot is used as the default in preference to YLAB.

TTL = CHARACTER ∗ (∗) (Given)
A default title for the plot. Only used if the user does not supply an alternative. If a Plot is supplied via IPLOT, then the " Title" attribute in the Plot is used as the default in preference to TTL.

XSYM = CHARACTER ∗ (∗) (Given)
The default symbol for the horizontal axis. Only used if the user does not supply an alternative. This will be stored with the Plot in the AGI database and (for instance) used by CURSOR as axis symbols when displaying the cursor positions on the screen. If a Plot is supplied via IPLOT, then the " Symbol(1)" attribute in the Plot is used as the default in preference to XSYM.

YSYM = CHARACTER * (*) (Given)
The default symbol for the horizontal axis. Only used if the user does not supply an alternative. This will be stored with the Plot in the AGI database and (for instance) used by CURSOR as axis symbols when displaying the cursor positions on the screen. If a Plot is supplied via IPLOT, then the " Symbol(2)" attribute in the Plot is used as the default in preference to XSYM.

MODE = INTEGER (Given)
Determines the way in which the data points are represented: 1 - A " staircase" histogram, in which each horizontal line is centred on the X position. Bad values are flanked by vertical lines drawn down to the lower edge of the viewport. 2 - The points are joined by straight lines. 3 - A marker is placed at each point (see MTYPE). 4 - (not used) 5 - A " chain" in which each point is marker by a marker and also join by straight lines to its neighbouring points. 6 - The same as Mode 1, except that bad values are not flanked by vertical lines drawn down to the lower edge of the viewport (a simple gap is left instead). 7 - The data points are not drawn.

NULL = LOGICAL (Given)
If .TRUE., then the user may supply a null (!) value for most of the parameters accessed by this routine to indicate that nothing is to be plotted. In this case, no error is returned. Otherwise, a PAR__NULL error status is returned if a null value is supplied.

XL = REAL (Given)
The default value for the XLEFT parameter. If VAL__BADR is supplied, the minimum of the X values is used (plus a small margin).

XR = REAL (Given)
The default value for the XRIGHT parameter. If VAL__BADR is supplied, the maximum of the X values is used (plus a small margin).

YB = REAL (Given)
The default value for the YBOT parameter. If VAL__BADR is supplied, the minimum of the low end of the Y error bars is used (plus a small margin).

YT = REAL (Given)
The default value for the YTOP parameter. If VAL__BADR is supplied, the maximum of the high end of the Y error bars is used (plus a small margin).

APP = CHARACTER * (*) (Given)
The name of the application in the form " <package>_<application>" . E.g. " KAPPA_NORMALIZE" .

QUIET = LOGICAL (Given)
If .FALSE., a message is displayed indicating the number of points which were plotted. If .TRUE., nothing is displayed on the alpha screen.

LMODE = LOGICAL (Given)
If .TRUE., then the user is given the chance to specify the default vertical bounds for the plot using parameter LMODE. If .FALSE., the supplied bounds (YB, YT) are used, and the eqivalent of " Extended" LMODE is used for any bounds which are not supplied.

BSCALE(2) = DOUBLE PRECISION (Given)
Scale factors which converts the supplied (x,y) values into the values to label on the plot. A similar BZERO argument could be added if there is ever a need.

IPLOT = INTEGER (Given and Returned)
On entry, this can either be AST_NULL or a pointer to a two-dimensional Frame. If AST__NULL, the supplied values for the XLAB, YLAB, TTL, XSYM and YSYM arguments are used without change. If a Frame is supplied, the Label, Title, Units and Symbol attributes of the Frame are used in preference to XLAB, YLAB, TTL, XSYM and YSYM (which are then ignored). Any supplied Frame pointer is annulled before returning, and a pointer to the Plot used to draw the axes is returned.

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

Notes:

- If an error occurs, or if no graphics is produced because the user supplied a null value for a parameter, IPLOT is returned equal to AST__NULL, and PGPLOT is shut down.

KPG1_GRLM1
Finds the default limits for a graph axis

Description:

This routine returns one or both default limits for a graph axis. Any limits which are supplied by the calling routine are used as supplied. The user is only allowed to over-ride limits which are supplied as VAL__BADR. The way in which these limits are chosen is specified by the user through the parameter specified by PARAM. This parameter can take the following values:

- " Range" – LIM1 and LIM2 are returned equal to the lowest and highest supplied data values (including error bars).
- " Extended" – LIM1 and LIM2 are returned equal to the lowest and highest supplied data values (including error bars), extended to give a margin of 2.5% of the total data range at each end.
- " Extended,10,5" – Like " Extended" , except the margins at the two ends are specified as a pair of numerical value in the second and third elements of the array. These values are percentages of the total data range. So, " Extended,10,5" includes a margin of 10% of the total data range in LIM1, and 5% in LIM2. If only one numerical value is given, the same value is used for both limits. If no value is given, both limits default to 2.5. " Range" is equivalent to " Extended,0,0" .
- " Percentiles,5,95" – The second and third elements of the array are interpreted as percentiles. For instance, " Perc,5,95" causes 5% of the data points (ignoring error bars) to be below LIM1, and 10% to be above the LIM2. If only 1 value (p1) is supplied, the other one, p2, defaults to (100 - p1). If no values are supplied, p1 and p2 default to 5 and 95.
- " Sigma,2,3" – The second and third elements of the array are interpreted as multiples of the standard deviation of the data values (ignoring error bars). For instance, " S,2,3" causes the LIM1 to be the mean of the data values, minus two sigma, and LIM2 to be the mean plus three sigma. If only 1 value is supplied, the same value is used for both limits. If no values are supplied, both values default to 3.0.

The above strings can be abbreviated to one character.

If the parameter name is supplied as blank, then " Extended" is assumed (i.e. LIM1 and LIM2 are chosen so that the axis encompasses the entire data range including error bars, with 2.5% margin at each end).

If only 1 limit is required (i.e. if LIM1 or LIM2 are supplied not equal to VAL__BADR), then only 1 numerical value can be supplied in the above limit descriptions.

Invocation:

```
CALL KPG1_GRLM1( PARAM, N, D1, D2, NSIGMA, SIGMA, LIM1, LIM2, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the parameter to use to get the method for chosing the default limits for the axis. May be blank.

N = INTEGER (Given)

No. of points

D1(N) = REAL (Given)

The central data value at each point.

D2(N) = REAL (Given)
An associated mask array. D1(I) is only used if both D1(I) and D2(I) are not equal to VAL__BADR.

NSIGMA = REAL (Given)
Controls the length of the error bars. The error bars are assumed to extended from D1(I) - NSIGMA∗SIGMA(I) to D1(I) + NSIGMA∗SIGMA(I). A value of zero suppresses error bars.

SIGMA(N) = REAL (Given)
The standard deviation associated with each Y value. Not accessed if NSIGMA is zero.

LIM1 = REAL (Given and Returned)
The chosen low data limit. Only returned if a value of VAL__BADR is supplied, Otherwise, the supplied value is returned unchanged.

LIM2 = REAL (Given and Returned)
The chosen high data limit. Only returned if a value of VAL__BADR is supplied, Otherwise, the supplied value is returned unchanged.

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

KPG1_GRLM2
Finds the default limits for a graph axis

Description:

This routine returns the default limits for a graph axis. The way in which the limits are chosen is specified by the user through the parameter specified by PARAM. This parameter can take the following values:

- " Range" – LIM1 and LIM2 are returned equal to the lowest and highest supplied data values (including error bars).
- " Extended" – LIM1 and LIM2 are returned equal to the lowest and highest supplied data values (including error bars), extended to give a margin of 2.5% of the total data range at each end.
- " Extended,10,5" – Like " Extended" , except the margins at the two ends are specified as a pair of numerical value in the second and third elements of the array. These values are percentages of the total data range. So, " Extended,10,5" includes a margin of 10% of the total data range in LIM1, and 5% in LIM2. If only one numerical value is given, the same value is used for both limits. If no value is given, both limits default to 2.5. " Range" is equivalent to " Extended,0,0" .
- " Percentiles,5,95" – The second and third elements of the array are interpreted as percentiles. For instance, " Perc,5,95" causes 5% of the data points (ignoring error bars) to be below LIM1, and 10% to be above the LIM2. If only 1 value (p1) is supplied, the other one, p2, defaults to (100 - p1). If no values are supplied, p1 and p2 default to 5 and 95.
- " Sigma,2,3" – The second and third elements of the array are interpreted as multiples of the standard deviation of the data values (ignoring error bars). For instance, " S,2,3" causes the LIM1 to be the mean of the data values, minus two sigma, and LIM2 to be the mean plus three sigma. If only 1 value is supplied, the same value is used for both limits. If no values are supplied, both values default to 3.0.

The above strings can be abbreviated to one character.

If the parameter name is supplied as blank, then " Extended" is assumed (i.e. LIM1 and LIM2 are chosen so that the axis encompasses the entire data range including error bars, with 2.5% margin at each end).

If only 1 limit is required (i.e. if LIM1 or LIM2 are supplied not equal to VAL__BADD), then only 1 numerical value can be supplied in the above limit descriptions.

Invocation:

```
CALL KPG1_GRLM2( PARAM, N, D1, D2, USEBAR, BAR, LIM1, LIM2, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the parameter to use to get the method for chosing the default limits for the axis. May be blank.

N = INTEGER (Given)

No. of points

D1(N) = DOUBLE PRECISION (Given)

The central data value at each point.

D2(N) = DOUBLE PRECISION (Given)
An associated mask array. D1(I) is only used if both D1(I) and D2(I) are not equal to VAL__BADD.

USEBAR = LOGICAL (Given)
Should BAR be used?

BAR(N,2) = DOUBLE PRECISION (Given)
The upper and lower ends of each error bar. Assumed equal to D1 if USEBAR is .FALSE. (i.e. no error bars).

LIM1 = DOUBLE PRECISION (Given and Returned)
The chosen low data limit. Only returned if a value of VAL__BADD is supplied, Otherwise, the supplied value is returned unchanged.

LIM2 = DOUBLE PRECISION (Given and Returned)
The chosen high data limit. Only returned if a value of VAL__BADD is supplied, Otherwise, the supplied value is returned unchanged.

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

KPG1_GRLM3
Finds the default limits for a graph axis (single precision)

Description:

This routine is exactly like KPG1_GRLM2,except that it uses single-precision data rather than double precision. Ideally, this should be handled using GENERIC, but this would involve renaming the long-standing KPG1_GRLM2 routine, which is not a good idea.

It returns the default limits for a graph axis. The way in which the limits are chosen is specified by the user through the parameter specified by PARAM. This parameter can take the following values:

- " Range" – LIM1 and LIM2 are returned equal to the lowest and highest supplied data values (including error bars).
- " Extended" – LIM1 and LIM2 are returned equal to the lowest and highest supplied data values (including error bars), extended to give a margin of 2.5% of the total data range at each end.
- " Extended,10,5" – Like " Extended" , except the margins at the two ends are specified as a pair of numerical value in the second and third elements of the array. These values are percentages of the total data range. So, " Extended,10,5" includes a margin of 10% of the total data range in LIM1, and 5% in LIM2. If only one numerical value is given, the same value is used for both limits. If no value is given, both limits default to 2.5. " Range" is equivalent to " Extended,0,0" .
- " Percentiles,5,95" – The second and third elements of the array are interpreted as percentiles. For instance, " Perc,5,95" causes 5% of the data points (ignoring error bars) to be below LIM1, and 10% to be above the LIM2. If only 1 value (p1) is supplied, the other one, p2, defaults to (100 - p1). If no values are supplied, p1 and p2 default to 5 and 95.
- " Sigma,2,3" – The second and third elements of the array are interpreted as multiples of the standard deviation of the data values (ignoring error bars). For instance, " S,2,3" causes the LIM1 to be the mean of the data values, minus two sigma, and LIM2 to be the mean plus three sigma. If only 1 value is supplied, the same value is used for both limits. If no values are supplied, both values default to 3.0.

The above strings can be abbreviated to one character.

If the parameter name is supplied as blank, then " Extended" is assumed (i.e. LIM1 and LIM2 are chosen so that the axis encompasses the entire data range including error bars, with 2.5% margin at each end).

If only 1 limit is required (i.e. if LIM1 or LIM2 are supplied not equal to VAL__BADR), then only 1 numerical value can be supplied in the above limit descriptions.

Invocation:

```
CALL KPG1_GRLM3( PARAM, N, D1, D2, USEBAR, BAR, LIM1, LIM2, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the parameter to use to get the method for chosing the default limits for the axis. May be blank.

N = INTEGER (Given)

No. of points

D1(N) = REAL (Given)
The central data value at each point.

D2(N) = REAL (Given)
An associated mask array. D1(I) is only used if both D1(I) and D2(I) are not equal to VAL__BADR.

USEBAR = LOGICAL (Given)
Should BAR be used?

BAR(N,2) = REAL (Given)
The upper and lower ends of each error bar. Assumed equal to D1 if USEBAR is .FALSE. (i.e. no error bars).

LIM1 = REAL (Given and Returned)
The chosen low data limit. Only returned if a value of VAL__BADR is supplied, Otherwise, the supplied value is returned unchanged.

LIM2 = REAL (Given and Returned)
The chosen high data limit. Only returned if a value of VAL__BADR is supplied, Otherwise, the supplied value is returned unchanged.

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

KPG1_GRPHW
Draws a line graph, using supplied work arrays

Description:

Opens a graphics device and draws a graph displaying a supplied set of points. Each point is defined by an X and Y value, plus an optional error bar. An AST Plot is returned so that the calling application can add further graphics to the plot if needed. When complete, the calling application should annul the Plot, and close the workstation:

CALL AST_ANNUL(IPLOT, STATUS) CALL KPG_PGCLS(' DEVICE' , .FALSE., STATUS)

Various environment parameters are used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between applications.

Invocation:

```
CALL KPG1_GRPHW( N, X, Y, NSIGMA, YSIGMA, XLAB, YLAB, TTL, XSYM, YSYM, MODE, NULL, XL,
XR, YB, YT, APP, QUIET, LMODE, BSCALE, DX, DY, DBAR, IPLOT, STATUS )
```

Arguments:

N = INTEGER (Given)
Nuumber of points

X(N) = REAL (Given)
X value at each point. These are scaled by the value supplied in BSCALE(1) to generate the axis labels.

Y(N) = REAL (Given)
Y value at each point. These are scaled by the value supplied in BSCALE(2) to generate the axis labels.

NSIGMA = REAL (Given)
Controls the length of the vertical error bars. A value of zero suppresses error bars. Otherwise error bars are drawn that extend from Y - NSIGMA∗YSIGMA to Y + NSIGMA∗YSIGMA.

YSIGMA(N) = REAL (Given)
The standard deviation associated with each Y value. Not accessed if NSIGMA is zero.

XLAB = CHARACTER ∗ (∗) (Given)
A default label for the X axis. Only used if the user does not supply an alternative. Trailing spaces are ignored. If a Plot is supplied via IPLOT, then the " Label(1)" attribute in the Plot is used as the default in preference to XLAB.

YLAB = CHARACTER ∗ (∗) (Given)
A default label for the Y axis. Only used if the user does not supply an alternative. Trailing spaces are ignored. If a Plot is supplied via IPLOT, then the " Label(2)" attribute in the Plot is used as the default in preference to YLAB.

TTL = CHARACTER ∗ (∗) (Given)
A default title for the plot. Only used if the user does not supply an alternative. If a Plot is supplied via IPLOT, then the " Title" attribute in the Plot is used as the default in preference to TTL.

XSYM = CHARACTER ∗ (∗) (Given)
The default symbol for the horizontal axis. Only used if the user does not supply an alternative. This will be stored with the Plot in the AGI database and (for instance) used by KAPPA:CURSOR as axis symbols when displaying the cursor positions on the screen. If a Plot is supplied via IPLOT, then the " Symbol(1)" attribute in the Plot is used as the default in preference to XSYM.

YSYM = CHARACTER * (*) (Given)
The default symbol for the horizontal axis. Only used if the user does not supply an alternative. This will be stored with the Plot in the AGI database and (for instance) used by KAPPA:CURSOR as axis symbols when displaying the cursor positions on the screen. If a Plot is supplied via IPLOT, then the " Symbol(2)" attribute in the Plot is used as the default in preference to YSYM.

MODE = INTEGER (Given)
Determines the way in which the data points are represented. 1 – A " staircase" histogram, in which each horizontal line is centred on the X position. Bad values are flanked by vertical lines drawn down to the lower edge of the viewport. 2 – The points are joined by straight lines. 3 – A marker is placed at each point. 4 – (not used) 5 – A " chain" in which each point is indicated by a marker and also join by straight lines to its neighbouring points. 6 – The same as Mode 1, except that bad values are not flanked by vertical lines drawn down to the lower edge of the viewport (a simple gap is left instead). 7 – The data points are not drawn.

NULL = LOGICAL (Given)
If .TRUE., then the user may supply a null (!) value for most of the parameters accessed by this routine to indicate that nothing is to be plotted. In this case, no error is returned. Otherwise, a PAR__NULL error status is returned if a null value is supplied.

XL = REAL (Given)
The default value for the XLEFT parameter. If VAL__BADR is supplied, the minimum of the X values is used (plus a small margin).

XR = REAL (Given)
The default value for the XRIGHT parameter. If VAL__BADR is supplied, the maximum of the X values is used (plus a small margin).

YB = REAL (Given)
The default value for the YBOT parameter. If VAL__BADR is supplied, the minimum of the low end of the Y error bars is used (plus a small margin).

YT = REAL (Given)
The default value for the YTOP parameter. If VAL__BADR is supplied, the maximum of the high end of the Y error bars is used (plus a small margin).

APP = CHARACTER * (*) (Given)
The name of the application in the form " <package>_<application>" , for instance " KAPPA_NORMALIZE" .

QUIET = LOGICAL (Given)
If .FALSE., a message is displayed indicating the number of points which were plotted. If .TRUE., nothing is displayed on the alpha screen.

LMODE = LOGICAL (Given)
If .TRUE., then the user is given the chance to specify the default vertical bounds for the plot using Parameter LMODE. If .FALSE., the supplied bounds (YB, YT) are used, and the equivalent of " Extended" LMODE is used for any bounds that are not supplied.

BSCALE(2) = DOUBLE PRECISION (Given)
Scale factors which converts the supplied (x,y) values into the values to label on the plot. A similar BZERO argument could be added if there is ever a need.

DX(N) = DOUBLE PRECISION (Given and Returned)
Work space.

DY(N) = DOUBLE PRECISION (Given and Returned)
Work space.

DBAR(N, 2) = DOUBLE PRECISION (Given and Returned)
Work space. Not accessed if NSIGMA is zero.

IPLOT = INTEGER (Given and Returned)
On entry, this can either be AST_NULL or a pointer to a two-dimensional Frame. If AST__NULL, the supplied values for the XLAB, YLAB, TTL, XSYM and YSYM arguments are used without change. If a Frame is supplied, the Label, Title, Units, and Symbol attributes of the Frame are used in preference to XLAB, YLAB, TTL, XSYM and YSYM (which are then ignored). Any supplied Frame pointer is annulled before returning, and a pointer to the Plot used to draw the axes is returned.

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

Notes:

- If an error occurs, or if no graphics is produced because the user supplied a null value for a parameter, IPLOT is returned equal to AST__NULL, and PGPLOT is shut down.

Environment Parameters

AXES = _LOGICAL (Read)
TRUE if annotated axes are to be produced.

CLEAR = _LOGICAL (Read)
TRUE if the graphics device is to be cleared on opening.

DEVICE = DEVICE (Read)
The plotting device.

LMODE = LITERAL (Read)
If the subroutine argument LMODE is .TRUE., then Parameter LMODE is used to specify how the default values for YBOT and YTOP are found. If argument LMODE is .FALSE. then " Extended" mode is always used. This parameter can take the following values:

- " Range" – The lowest and highest supplied data values are used (including error bars).
- " Extended" – The lowest and highest supplied data values are used (including error bars), extended to give a margin of 2.5% of the total data range at each end.
- " Extended,10,5" – Like " Extended" , except the margins at the two ends are specified as a pair of numerical value in the second and third elements of the array. These values are percentages of the total data range. So, " Extended,10,5" includes a margin of 10% of the total data range in YBOT, and 5% in YTOP. If only one numerical value is given, the same value is used for both limits. If no value is given, both limits default to 2.5. " Range" is equivalent to " Extended,0,0" .
- " Percentiles,5,95" – The second and third elements of the array are interpreted as percentiles. For instance, " Perc,5,95" causes 5% of the data points (ignoring error bars) to be below YBOT, and 10% to be above the YTOP. If only one value (p1) is supplied, the other one, p2, defaults to (100 - p1). If no values are supplied, p1 and p2 default to 5 and 95.
- " Sigma,2,3" – The second and third elements of the array are interpreted as multiples of the standard deviation of the data values (ignoring error bars). For instance, " S,2,3" causes the YBOT to be the mean of the data values, minus two sigma, and YTOP to be the mean plus three sigma. If only one value is supplied, the same value is used for both limits. If no values are supplied, both values default to 3.0.

The above strings can be abbreviated to one character.

MARGIN(4) = _REAL (Read)
The widths of the margins to leave for axis annotation, given as fractions of the corresponding dimension of the current picture. Four values may be given in the order bottom, right, top, left. If fewer than four values are given, extra values are used equal to the first supplied value. If these margins are too narrow any axis annotation may be clipped. The dynamic default is 0.15 (for all edges) if either annotated axes or a key are produced, and zero otherwise.

MARKER = _INTEGER (Read)
The PGPLOT marker type to use. Only accessed if MODE is 3 or 5.

STYLE = GROUP (Read)
A description of the plotting style required. The following synonyms for graphical elements may be used.

- " Err(Bars)" – Specifies colour, etc. for error bars. Size(errbars) scales the size of the serifs (i.e. a size value of 1.0 produces a default size).
- " Sym(bols)" – Specifies colour, etc. for markers (used in Modes 3 and 5).
- " Lin(es)" – Specifies colour, etc. for lines (used in Modes 1, 2, and 5).

TEMPSTYLE = GROUP (Read)
A description of plotting style required in addition to that define by STYLE. See STYLE for allowed synonyms. Unlike STYLE its values are not persistent between invocations.

XLEFT = _DOUBLE (Read)
The axis value to place at the left-hand end of the horizontal axis. The dynamic default is specified by argument XL. The value supplied may be greater than or less than the value supplied for XRIGHT.

XRIGHT = _DOUBLE (Read)
The axis value to place at the right-hand end of the horizontal axis. The dynamic default is specified by argument XR. The value supplied may be greater than or less than the value supplied for XLEFT.

YBOT = _DOUBLE (Read)
The axis value to place at the bottom end of the vertical axis. The dynamic default is specified by argument YB. The value supplied may be greater than or less than the value supplied for YTOP.

YTOP = _DOUBLE (Read)
The axis value to place at the top end of the vertical axis. The dynamic default is specified by argument YT. The value supplied may be greater than or less than the value supplied for YBOT.

KPG1_GTAXI
Selects Frame axes using an environment parameter

Description:

This routine returns the indices of selected axes in a supplied Frame. The axes to select are determined using the supplied environment parameter. Each axis can be specified either by giving its index within the Frame in the range 1 to the number of axes in the Frame, or by giving its symbol. Spectral, time and celestial longitude/latitude axes may also be specified using the options " SPEC" , " TIME" , " SKYLON" and " SKYLAT" . If the first value supplied in AXES is not zero, the supplied axes are used as the dynamic default for the parameter. The parameter value should be given as a GRP group expression, with default GRP control characters.

Invocation:

```
CALL KPG1_GTAXI( PARAM, FRAME, NAX, AXES, STATUS )
```

Arguments:

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

FRAME = INTEGER (Given)
An AST pointer for the Frame from which axes may be chosen.

NAX = INTEGER (Given)
The number of axes which must be selected.

AXES(NAX) = INTEGER (Given and Returned)
On entry, the axes to be specified as the dynamic default for the parameter (if AXES(1) is not zero). On exit, the indices of the selected axes. If AXES(1) is zero the supplied values are ignored and a PAR__NULL status value is returned if a null (!) value is supplied for the parameter. Otherwise, the PAR_NULL status is annulled if a null value is supplied, and the supplied axes are returned.

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

Notes:

- Case is insignificant when comparing supplied strings with available axis symbols.

KPG1_GTAXM
Selects Frame axes using an environment parameter

Description:

This routine returns the indices of selected axes in a supplied Frame. The axes to select are determined using the supplied environment parameter. Each axis can be specified either by giving its index within the Frame in the range 1 to the number of axes in the Frame, or by giving its symbol. Spectral, time and celestial longitude/latitude axes may also be specified using the options " SPEC" , " TIME" , " SKYLON" and " SKYLAT" . If the first value supplied in AXES is not zero, the supplied axes are used as the dynamic default for the parameter. The parameter value should be given as a GRP group expression, with default GRP control characters.

Invocation:

```
CALL KPG1_GTAXM( PARAM, FRAME, MAXAX, AXES, NAX, STATUS )
```

Arguments:

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

FRAME = INTEGER (Given)
An AST pointer for the Frame from which axes may be chosen.

MAXAX = INTEGER (Given)
The maximum number of axes that can be be selected.

AXES(NAX) = INTEGER (Given and Returned)
On entry, the axes to be specified as the dynamic default for the parameter (if AXES(1) is not zero). On exit, it holds the indices of the selected axes. If AXES(1) is zero the supplied values are ignored and a PAR__NULL status value is returned if a null (!) value is supplied for the parameter. Otherwise, the PAR_NULL status is annulled if a null value is supplied, and the supplied axes are returned.

NAX = INTEGER (Given)
The number of axes actually selected.

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

Notes:

- Case is insignificant when comparing supplied strings with available axis symbols.

KPG1_GTAXV
Gets one or more formatted axis values from the environment

Description:

This routine obtains one or more formatted values for a specified axis from the environment, using a specified parameter.

If the string supplied for the parameter consists of a single colon, then a description of the Current co-ordinate Frame is displayed, together with an indication of the format required for each axis value, and a new parameter value is then obtained.

Invocation:

```
CALL KPG1_GTAXV( PARAM, MXVAL, EXACT, FRAME, IAXIS, AXVAL, NVAL, STATUS )
```

Arguments:

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

MXVAL = INTEGER (Given)
The maximum number of values which can be supplied. It is not an error for less than MXVAL to be supplied. Must be no more than 20.

EXACT = LOGICAL(Given)
If .TRUE., then the user must supply exactly MXVAL values, and he is reprompted if less than MXVAL are given. If .FALSE. then the user can give between 1 and MXVAL values.

FRAME = INTEGER (Given)
A pointer to an AST Frame in which the axis lives.

IAXIS = INTEGER (Given)
The index of the axis within the Frame for which a value is required.

AXVAL(MXVAL) = DOUBLE PRECISION (Given and Returned)
On entry, holds the axis values to use as the dynamic default for the parameter. On exit, holds the supplied axis value. No dynamic default is used if any of the supplied values is AST__BAD.

NVAL = INTEGER (Returned)
The number of values obtained using the parameter and returned in AXVAL.

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

Notes:

- If a null (!) parameter value is supplied, the supplied value of AXVAL is returned, and the error is annulled if the AXVAL value is not equal to AST__BAD.
- AXVAL is left unchanged if an error has already occurred.
- AST__BAD is returned in AXVAL if an error occurs during this routine.

KPG1_GTCHV
Obtains a vector of choices from the environment

Description:

This routine gets NVAL strings from the user, selected from those supplied in OPTS. The indices of the supplied strings within OPTS are returned. The user supplies the strings in the form of a GRP group expression, using the default GRP control characters.

The user may supply an integer value instead of a string, in which case the integer is understood to be the index of the required string within OPTS. If the supplied list of strings contains the integer itself, then the integer is understood to be a string, not an index.

Invocation:

```
CALL KPG1_GTCHV( NOPT, OPTS, PARAM, NVAL, IDEF, VALS, STATUS )
```

Arguments:

NOPT = INTEGER (Given)
The number of available options supplied in OPTS.

OPTS(NOPT) = CHARACTER * (*) (Given)
An array holding the options from which the user must choose. Leading and trailing white space is ignored. Blank options are not allowed.

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

NVAL = INTEGER (Given)
The number of choices required. The user must supply exactly this number of choices.

IDEF(NVAL) = INTEGER (Given)
The indices within OPTS of the default strings to use if a null (!) value is supplied for the parameter. If the first value is zero, a null parameter value results in a PAR__NULL status being returned.

VALS(NVAL) = INTEGER (Returned)
The indices within OPTS of the selected options.

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

Notes:

- Case is insignificant when comparing supplied strings with available options.
- A dynamic default is set for the parameter before accessing it if IDEF supplied suitable defaults. The default consists of a comma-separated list of the default options.

KPG1_GTGPT
Obtains a group of strings from the environment including some that are transient

Description:

This routine obtains group of strings using the specified parameter, and returns them in a GRP group (see SUN/150).

The user specifies the strings by supplying a GRP group expression for the parameter value. If the final character in the supplied string is the group " flag character" (a minus sign by default), then the user is re-prompted for another group expression, and the strings specified by the second group expression are appended to the returned group. This continues until a group expression is supplied that does not end with the continuation character, or a null value is supplied (which is annulled).

The group comprises an optional persistent part that will be written as the parameter's current value, and an optional temporary component that only exists for the duration of the current application. The division is determined by the first appearance within the string supplied of a delimiter string or character (the DELIM argument). For instance, assume the delimiter is '+'. If the supplied value is 'smooth,box=5+filter=gauss', the persistent value would be 'smooth,box=5' and the temporary value 'filter=gauss'. Supplying merely the temporary value, such as '+filter=gauss' would return the existing current value of the parameter as the persistent component, and the temporary component is as before.

Normally, the " current value" for the parameter on exit would be the final group expression. If more than one group expression was supplied, then this will not represent the entire group. For this reason, this routine concatenates all the group expressions supplied, and stores the total group expression as the parameter value before exiting. It also stores the concatenated persistent components and writes this as the current value. Since a new value is stored for the parameter, the parameter should not be given an access mode of READ in the interface file.

If a null value is supplied the first time the parameter value is obtained, then the PAR__NULL status is returned, and SIZE is returned as zero (a valid GRP identifier is still returned however). If a null value is supplied on a subsequent access to the parameter, then the PAR__NULL status is annulled, and the group is returned containing any values obtained earlier.

Invocation:

```
CALL KPG1_GTGPT( PARAM, DELIM, LAST, IGRP, SIZE, STATUS )
```

Arguments:

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

DELIM = CHARACTER * (*) (Given)
The delimiter string used to indicator where the temporary part of a value begins. Any text before the delimiter behaves as if it were obtained using PAR_GET0C. After, and not including, the delimiter the value is deemed to be temporary and is never incorporated into the parameter's current value. The delimiter should not be an alphanumeric character or underscore.

LAST = LOGICAL (Given)
If this is the last or only invocation of this routine for a given parameter in a task, set this TRUE. For multiple invocations except the last, set this FALSE. Setting to FALSE causes the full concatenated expression supplied by the user to be stored in the parameter's current value, rather than just the persistent-attribute list a TRUE value generates needed when the task ends.

IGRP = INTEGER (Given and Returned)
The group to use. If this is GRP__NOID then a new group is created with default control characters and its identifier is returned. If the group already exists, its contents are discarded before adding new strings.

SIZE = INTEGER (Returned)
The total size of the returned group. Returned equal to zero if an error occurs.

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

Notes:

- This routine subverts the parameter system in that it uses an additional component in the parameter file to store the current value. This enables the current value to be accessed regardless of the value for the parameter supplied on the command line that would otherwise replace the previous current value with the full new value including the delimiter and temporary value.
- The additional component's name is the parameter name (up to 13 characters) followed by CV (for current value). In practice truncation should not be an issue.
- This routine should not be used with multiple parameters in the same task whose new current-value component in the parameter file would be the same. Again in practice this is extremely unlikely.
- It has not been proven to work if invoked more than twice for the same parameter, as there are no known occurrences of this in Starlink applications.

KPG1_GTGRP
Obtains a group of strings from the environment

Description:

This routine obtains group of strings using the specified parameter, and returns them in a GRP group (see SUN/150).

The user specifies the strings by supplying a GRP group expression for the parameter value. If the final character in the supplied string is the group " flag character" (a minus sign by default), then the user is re-prompted for another group expression, and the strings specified by the second group expression are appended to the returned group. This continues until a group expression is supplied which does not end with the continuation character, or a null value is supplied (which is annulled).

Normally, the " current value" for the parameter on exit would be the final group expression. If more than one group expression was supplied, then this will not represent the entire group. For this reason, this routine concatenates all the group expressions supplied, and stores the total group expression as the parameter value before exiting. Since a new value is stored for the parameter, the parameter should ne be given an access mode of READ in the interface file.

If a null value is supplied the first time the parameter value is obtained, then the PAR__NULL status is returned, and SIZE is returned as zero (a valid GRP identifier is still returned however). If a null value is supplied on a subsequent access to the parameter, then the PAR__NULL status is annulled, and the group is returned containing any values obtained earlier.

Invocation:

```
CALL KPG1_GTGRP( PARAM, IGRP, SIZE, STATUS )
```

Arguments:

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

IGRP = INTEGER (Given and Returned)
The group to use. If this is GRP__NOID then a new group is created with default control characters and its identifier is returned. If the group already exists, its contents are discarded before adding new strings.

SIZE = INTEGER (Returned)
The total size of the returned group. Returned equal to zero if an error occurs.

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

KPG1_GTMOR
Creates an HDS structure holding a user-supplied set of keyword=value strings

Description:

This routine returns a locator for a temporary HDS structure that holds a set of keyword values obtained from the environment.

Invocation:

```
CALL KPG1_GTMOR( PARAM, MORE, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)

The name of the environment parameter to use. The parameter is accessed as a group of text strings, using GRP. Each string should be of the form " keyword=value" . The keyword can be a single name or a dot-delimited heirarchy. The returned HDS object will contain a sinmilar heiracrhy.

MORE = CHARACTER * (DAT__SZLOC) (Returned)

A locator for the returned temporary HDS object. It will be a scalar structure with HDS type of " KPG1_GTMOR_TYPE" .

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_GTNDF
Gets an NDF or NDF section with a specified number of significant dimensions

Description:

The supplied parameter name is associated with an NDF through the environment, and an identifier is obtained for the NDF using the specified access mode. Each axis of the NDF is checked to see if is significant (i.e. has a size greater than 1). The index of each significant axis is returned in SDIM, and the bounds of the axis are returned in SLBND and SUBND. If EXACT is .TRUE., an error is reported if the number of significant axes is not exactly NDIM. If EXACT is .FALSE. an error is only reported if the number of significant dimensions is higher than NDIM. If there are less than NDIM significant dimensions then the insignificant dimensions are used (starting from the lowest) to ensure that the required number of dimensions are returned.

Invocation:

```
CALL KPG1_GTNDF( PARAM, NDIM, EXACT, MODE, INDF, SDIM, SLBND, SUBND, STATUS )
```

Arguments:

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

NDIM = INTEGER (Given)
The number of dimensions required.

EXACT = LOGICAL (Given)
This should be supplied .FALSE. if an NDF with less than NDIM significant dimensions can be used.

MODE = CHARACTER * (*) (Given)
The access mode required for the NDF.

INDF = INTEGER (Returned)
An identifier for the NDF.

SDIM(NDIM) = INTEGER (Returned)
The indices of the significant dimensions.

SLBND(NDIM) = INTEGER (Returned)
The lower bounds of the significant dimensions. These are stored in the same order as the indices in SDIM.

SUBND(NDIM) = INTEGER (Returned)
The upper bounds of the significant dimensions. These are stored in the same order as the indices in SDIM.

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

KPG1_GTOBJ
Gets an AST Object using an environment parameter

Description:

Gets an AST Object from an NDF, FITS file, HDS path or text file using an environment parameter.

First, attempt to interpret the parameter value as an HDS path. The HDS object must have a type of WCS, must be scalar, and must contain a single one-dimensional array component with name DATA and type _CHAR. This is the scheme used for HDS structures created by KPG1_WWRT.

If the above attempt fails, attempt to interpret the parameter value as an NDF name. If the NDF is opened succesfully, its WCS FrameSet is returned. If a FrameSet is required, its current Frame is returned if a Frame is required. Its base->current Mapping is returned if a Mapping is required. A Box covering the pixel grid and re-mapped into the current WCS Frame is returned if a Regin is required.

If the above attempt fails, and the parameter value ends with " .FIT" , attempt to interpret the parameter value as the name of a FITS file. Open the FITS file and attempt to obtained an AST FrameSet from the primary HDU headers.

If the above attempt fails, attempt to interpret the parameter value as the name of a text file containing either an AST object dump, or a set of FITS headers.

Invocation:

```
CALL KPG1_GTOBJ( PARAM, CLASS, ISA, IAST, STATUS )
```

Arguments:

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

CLASS = CHARACTER * (*) (Given)
The required class. Used in error reports (see ISA). If Objects of more than 1 class can be used, this should be supplied blank, and the calling routine should verify that the Object is usable.

ISA = EXTERNAL (Given)
A suitable AST " ISA.." function which returns .TRUE. if an Object is of a suitable class. This is ignored if CLASS is blank. Otherwise, an error is reported if th supplied Object is not of the required class.

IAST = INTEGER (Returned)
The AST Object, or AST__NULL.

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

Notes:

- If the group contains the AST dump of a Channel (of any class), then the Object returned via IAST will be the Channel itself. The exception to this is that if the " Begin " line at the start of the dump ends with the string " (Read)" , then the returned IAST Object will be the Object read from the Channel, rather than the Channel itself. For instance, if the group contains the AST dump of a FitsChan, and the first line of the dump is " Begin FitsChan(Read)" , then the returned IAST object will be the Object read from the FitsChan, rather than the FitsChan itself. This facility is only available for top level objects (e.g. FitsChans contained within FitsChans cannot be read in this way).

KPG1_GTPLR
Gets a spatial position from the environment as polar co-ordinates

Description:

This routine obtains a two-dimensional spatial position from the environment, using a specified parameter. The user supplies the position in polar co-ordinates about a supplied centre within the co-ordinate system of the Current Frame in the supplied WCS FrameSet. This FrameSet must have two axes, otherwise the routine will exit with an error.

To be acceptable, the supplied position must correspond to a valid position (on both axes) in the Base Frame of the supplied FrameSet. If a Frame is supplied instead of a FrameSet this restriction is not imposed, however polar co-ordinates cannot be supplied, only regular co-ordinates along both axes.

If the polar position supplied in argument CC on entry is valid (i.e. does not contain any AST__BAD values), then it is used as a dynamic default for the parameter. Otherwise, no dynamic default is used.

The parameter is accessed as a single literal string containing a space- or comma-separated list of radius and position-angle values. For SkyFrames the position angle is measured in degrees from North via East; for other Frames, it is anticlockwise from the origin defined by argument PAORIG. The allowed formats for the co-ordinates depends on the class of the Current Frame in the supplied FrameSet, and are described in SUN/210.

If the string supplied for the parameter consists of a single colon, then a description of the Current co-ordinate Frame is displayed, together with an indication of the format required for each axis value, and a new parameter value is then obtained.

Invocation:

```
CALL KPG1_GTPLR( PARAM, IWCS, NULL, POLE, PAORIG, CC, BC, STATUS )
```

Arguments:

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

IWCS = INTEGER (Given)
A pointer to an AST FrameSet. If a pointer to a Frame is supplied instead the routine will issue a warning that it is unable to handle polar co-ordinates and will expect spatial positions like routine KPG1_GTPOS. This can be aborted (!!) at the parameter prompt.

NULL = LOGICAL (Given)
If TRUE, a null (!) parameter value will result in the dynamic default value being used. If FALSE (or if there is no dynamic default), a null parameter value will result in a PAR__NULL error status.

PAORIG = DOUBLE PRECISION (Given)
This applies when the current Frame is not a SkyFrame. It specifies the origin for the position angle in degrees from the the first WCS axis. The normal convention is for this to be zero (i.e. from X in a Cartesian co-ordinate system) but another may be 90 for starting from up or Y.

POLE(2) = DOUBLE PRECISION (Given)
The position of the pole of the polar co-ordinates measured in the current co-ordinate Frame along each axis. If any of the co-ordinates are bad, the routine will issue a warning that it is unable to handle polar co-ordinates and will expect two spatial positions like routine KPG1_GTPOS. This can be aborted (!!) at the parameter prompt.

CC(2) = DOUBLE PRECISION (Given and Returned)
On entry, holds the position to use as the dynamic default for the parameter, in the Current Frame of the supplied FrameSet (or Frame). On exit, it holds the supplied position in the Current Frame. There should be one element for both axes.

BC(2) = DOUBLE PRECISION (Returned)
Returned holding the Base Frame position corresponding to the supplied Current Frame position. If a Frame is supplied for IWCS instead of a FrameSet, then BC will not be accessed. The returned values will be good on both axes.

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

Notes:

- CC and BC are left unchanged if an error has already occurred.
- Values of AST__BAD are returned in CC (and optionally BC) if an error occurs during this routine.

KPG1_GTPOS
Gets a spatial position from the environment

Description:

This routine obtains a spatial position from the environment, using a specified parameter. The user supplies the position in the co-ordinate system of the Current Frame in the supplied WCS FrameSet. To be acceptable, the supplied position must correspond to a valid position (on all axes) in the Base Frame of the supplied FrameSet. If a Frame is supplied instead of a FrameSet this restriction is not imposed.

If the position supplied in argument CC on entry is valid (i.e. does not contain any AST__BAD values) then it is used as a dynamic default for the parameter. Otherwise, no dynamic default is used.

The parameter is accessed as a single literal string containing a space or comma separated list of axis values. The allowed formats for the axis values depends on the class of the Current Frame in the supplied FrameSet, and are described in SUN/210.

If the string supplied for the parameter consists of a single colon, then a description of the Current co-ordinate Frame is displayed, together with an indication of the format required for each axis value, and a new parameter value is then obtained.

Invocation:

```
CALL KPG1_GTPOS( PARAM, IWCS, NULL, CC, BC, STATUS )
```

Arguments:

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

IWCS = INTEGER (Given)
A pointer to an AST Frame or FrameSet.

NULL = LOGICAL (Given)
If TRUE, a null (!) parameter value will result in the dynamic default value being used. If FALSE (or if there is no dynamic default), a null parameter value will result in a PAR__NULL error status.

CC(*) = DOUBLE PRECISION (Given and Returned)
On entry, holds the position to use as the dynamic default for the parameter, in the Current Frame of the supplied FrameSet (or Frame). On exit, holds the supplied position in the Current Frame. There should be one element for each axis in the Current Frame.

BC(*) = DOUBLE PRECISION (Returned)
Returned holding the Base Frame position corresponding to the supplied Current Frame position. If a Frame is supplied for IWCS instead of a FrameSet, then BC will not be accessed. The returned values will be good on all axes.

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

Notes:

- CC and BC are left unchanged if an error has already occurred.
- Values of AST__BAD are returned in CC (and optionally BC) if an error occurs during this routine.

KPG1_GTWCS
Gets an AST FrameSet from an NDF

Description:

This routine returns a FrameSet describing the WCS information in an NDF. If the NDF has no WCS component, any IRAS90 IRA structure is converted into a FrameSet and returned. If the NDF has no IRAS90 IRA structure, then an attempt is made to read a FrameSet from the FITS headers in the FITS extension. If the NDF has no FITS extension, then the default NDF FrameSet is returned.

Invocation:

```
CALL KPG1_GTWCS( INDF, IWCS, STATUS )
```

Arguments:

INDF = INTEGER (Given)
The identifier for the NDF.

IWCS = INTEGER (Returned)
An AST pointer to the WCS FrameSet. Returned equal to AST__NULL if an error occurs.

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

Notes:

- If a WCS FrameSet is created form an IRAS90 astrometry structure or a FITS extension, it will be stored in the supplied NDF if write access is available for the NDF.
- The preferred AST encodings to use when interpreting FITS headers can be specified as a comma-delimited string using environment variable KAPPA_ENCODINGS. If this variable is not defined, then the normal default encodings are used (see FITSDIN).

KPG1_H2AST
Copies AST_ data from an HDS object

Description:

This routine copies a line of text representing AST_ data from a specified element of a one-dimensional character array. It is intended for use when reading AST_ data from an HDS object (i.e an HDS _CHAR array).

Invocation:

```
CALL KPG1_H2AST( DATA, ILINE, LINE, STATUS )
```

Arguments:

DATA(*) = CHARACTER * (*) (Given)
The character array from which the text is to be copied.

ILINE = INTEGER (Given)
The index of the element in DATA which is to provide the text (the contents of other elements are ignored).

LINE = CHARACTER * (*) (Returned)
The line of text obtained.

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

KPG1_HCONx
Takes the complex conjugate of an Hermitian image

Description:

The complex conjugate of the supplied Hermitian image is returned. See routine KPG1_HMLTx for more information on Hermitian images.

Invocation:

```
CALL KPG1_HCONx( M, N, IN, STATUS )
```

Arguments:

M = INTEGER (Given)
Number of columns.

N = INTEGER (Given)
Number of rows.

IN(N, M) =? (Given and Returned)
On input it is the Hermitian image. On exit it holds the complex conjugate of the supplied array.

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

Notes:

- There is a routine for processing single- and double-precision arrays; replace " x" in the routine name by R or D as appropriate. The data type of the IN argument must match the routine used.

KPG1_HDSKY
Appends a primitive HDS object to an AST KeyMap

Description:

This function stores the vectorised data values in the supplied HDS object (which must be primitive) in the supplied KeyMap. The key for the KeyMap entry is the name of the HDS object. If the KeyMap already contains an entry with this name, then what happens is specified by OLD. Likewise, if the KeyMap does not already contain an entry with this name, then what happens is specified by NEW.

Invocation:

```
CALL KPG1_HDSKY( LOC, KEYMAP, OLD, NEW, STATUS )
```

Arguments:

LOC = CHARACTER * (DAT__SZLOC) (Given)
An HDS locator for a primitive scalar or array object.

KEYMAP = INTEGER (Given)
An AST pointer to an existing KeyMap.

OLD = INTEGER (Given)
Specifies what happens if the supplied KeyMap already contains an entry with the name of the supplied HDS object.

1 - Append the new vectorised array values read from the HDS object to the end of the values already in the KeyMap. The HDS values will be converted to the data type of the values already in the KeyMap (an error will be reported if this is not possible).

2 - Replace the existing KeyMap entry with a new entry holding the vectorised array values read from the HDS object.

3 - Do nothing. The KeyMap is returned unchanged, and no error is reported.

4 - Report an error. The KeyMap is returned unchanged, and an error is reported.

NEW = INTEGER (Given)
Specifies what happens if the supplied KeyMap does not already contain an entry with the name of the supplied HDS object.

1 - Create a new entry holding the vectorised array values read from the HDS object.

2 - Do nothing. The KeyMap is returned unchanged, and no error is reported.

3 - Report an error. The KeyMap is returned unchanged, and an error is reported.

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

Notes:

- An error is reported if the supplied HDS object is a structure.

KPG1_HMLTx
Multiplies two Hermitian images

Description:

Each input Hermitian image represents a COMPLEX image in which certain symmetries exist. These symmetries allow all the information stored in the COMPLEX image to be compressed into a single REAL image. This routine effectively unpacks the two REAL images given as input into two COMPLEX images, multiplies them together pixel-by-pixel to produce another COMPLEX image, and then packs the COMPLEX image back into a single Hermitian REAL image. In fact it is not necessary to do the actual unpacking and packing of these Hermitian images; this algorithm generates the output Hermitian image directly and thus saves time.

See the Notes for more details of Hermitian images and how they are multiplied.

Invocation:

```
CALL KPG1_HMLTx( M, N, IN1, IN2, OUT, STATUS )
```

Arguments:

M = INTEGER (Given)
Number of columns.

N = INTEGER (Given)
Number of rows.

IN1(N, M) = ? (Given)
The first input Hermitian image.

IN2(N, M) = ? (Given)
The second input Hermitian image.

OUT(N, M) = ? (Returned)
The product of the two input images, in Hermitian form.

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

Notes:

- There is a routine for processing single- and double-precision arrays; replace " x" in the routine name by R or D as appropriate. The data type of the IN1, IN2, and OUT arguments must match the routine used.
- Fourier transforms supplied in Hermitian form can be thought of as being derived as follows (see the NAG manual, introduction to Chapter C06 for more information on the storage of Hermitian FFTs):

1) Take the one-dimensional FFT of each row of the original image. 2) Stack all these one-dimensional FFTs into a pair of two-dimensional images the same size as the original image. One two-dimensional image (" A") holds the real part and the other (" B") holds the imaginary part. Each row of image A will have symmetry such that A(J,row) = A(M-J,row) (J goes from 0 to M-1), while each row of image B will have symmetry such that B(J,row) = -B(M-J,row). 3) Take the one-dimensional FFT of each column of image A. 4) Stack all these one-dimensional FFTs into a pair of two-dimensional images, image AA holds the real part of each FFT, and image BA holds the imaginary part. Each column of AA will have symmetry such that AA(column,K) = AA(column,N-K) (K goes from 0 to N-1), each column of BA will have symmetry such that BA(column,K) =

- BA(column,N-K). 5) Take the one-dimensional FFT of each column of image B. 6) Stack all these one-dimensional FFTs into a pair of two-dimensional images, image AB holds the real part of each FFT, and image BB holds the imaginary part. Each column of AB will have symmetry such that AB(column,K) = AB(column,N-K) (K goes from 0 to N), each column of BB will have symmetry such that BB(column,K) = -BB(column,N-K). 7) The resulting four images all have either positive or negative symmetry in both axes. The Hermitian FFT images supplied to this routine are made up of one quadrant from each image. The bottom-left quadrant is taken from AA, the bottom-right quadrant is taken from AB, the top-left quadrant is taken from BA and the top-right quadrant is taken from BB.

The product of two Hermitian FFTs is itself Hermitian and so can be described in a similar manner. If the first input FFT corresponds to the four images AA1, AB1, BA1 and BB1, and the second input FFT corresponds to the four images AA2, AB2, BA2, BB2, then the output is described by the four images AA, AB, BA and BB where:

AA = AA1∗AA2 + BB1∗BB2 - BA1∗BA2 - AB1∗AB2 BB = AA1∗BB2 + BB1∗AA2 + BA1∗AB2 + AB1∗BA2 BA = BA1∗AA2 - AB1∗BB2 + AA1∗BA2 - BB1∗AB2 AB = -BA1∗BB2 + AB1∗AA2 + AA1∗AB2 - BB1∗BA1

KPG1_HMSG
Assigns the name of an HDS object to a message token

Description:

The routine assigns the full name (including the file name) of an HDS object to a message token for use with the ERR_ and MSG_ routines (SUN/104). Appropriate syntax is used to represent file names which do not have the standard (.SDF) file type.

Invocation:

```
CALL KPG1_HMSG( TOKEN, LOC )
```

Arguments:

TOKEN = CHARACTER * (*) (Given)
Name of the message token.

LOC = CHARACTER * (*) (Given)
Locator to the HDS object.

Notes:

- This routine has no STATUS argument and does not perform normal error checking. If it should fail, then no value will be assigned to the message token and this will be apparent in the final message.

VAX-specific features used :

- This routine makes assumptions about the form of a VMS file name.

KPG1_HRCPx
Finds the reciprocal of a purely real Hermitian image

Description:

This routine replaces the supplied Hermitian image with one in which the real terms have been inverted (i.e. replaced by their reciprocal) and the imaginary terms have been set to zero.

To form 1/H where H is a general complex Hermitian image, the numerator and denominator are both multiplied by the complex conjugate of H (" H∗"). The denominator then becomes purely real and equal to the modulus squared of H. The reciprocal of the denominator is then taken using this routine and the resulting array is multiplied by H∗. Thus the steps are:

1) Call KPG1_HCONx to form H∗, the complex conjugate of H. 2) Call KPG1_HMLTx to multiply H by H∗. This gives a purely real image holding the squared modulus of H. 3) Call KPG1_HRCPx to take the reciprocal of H.H∗. 4) Call KPG1_HMLTx to multiply the reciprocal of H.H∗ by H∗.

Invocation:

```
CALL KPG1_HRCPx( M, N, BADVAL, D, STATUS )
```

Arguments:

M = INTEGER (Given)
The number of columns in the Hermitian image D.

N = INTEGER (Given)
The number of lines in the Hermitian image D.

BADVAL = ? (Given)
A value to return instead of the reciprocal of the real term, if any real term is zero.

D(M, N) = ? (Given and Returned)
On entry, a purely real Hermitian image. On exit, an Hermitian image representing the reciprocal of the supplied image (also purely real).

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

Notes:

- There is a routine for the double precision and real data types: replace " x" in the routine names by D or R as appropriate. The BADVAL and D arguments must have the data type specified.

KPG1_HSDSx
Tabulates an histogram

Description:
This routine reports an histogram to the user.

Invocation:
```
CALL KPG1_HSDSx( NUMBIN, HIST, HRMIN, HRMAX, STATUS )
```

Arguments:

NUMBIN = INTEGER (Given)
The number of bins in the histogram.

HIST(NUMBIN) = INTEGER (Given)
The array holding the histogram.

HRMIN = ? (Given)
The minimum data value that could be included within the histogram.

HRMAX = ? (Given)
The maximum data value that could be included within the histogram.

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

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by D or R as appropriate. The extreme values of the histogram must have the data type specified.

KPG1_HSECT
Copy a section from an HDS array to a new component

Description:

This routine creates a new HDS object holding a copy of a section of a supplied primitive HDS array. The section may extend outside the bounds of the supplied array, in which case the new areas will be filled with bad values (or spaces if the array holds character data).

Invocation:

```
CALL KPG1_HSECT( LOC1, NDIM, LBND, UBND, LOC2, NAME, STATUS )
```

Arguments:

LOC1 = CHARACTER∗(DAT__SZLOC) (Given)
An HDS locator for a primitive array object. This array has implicit lower bounds of (1,1,1,...) and upper bounds equal to its dimension sizes.

NDIM = INTEGER (Given)
The dimensionality of the required section. An error will be reported if this is not the same as the dimensionality of the supplied HDS array.

LBND(NDIM) = INTEGER (Given)
The lower bounds of the required array section.

UBND(NDIM) = INTEGER (Given)
The upper bounds of the required array section.

LOC2 = CHARACTER∗(DAT__SZLOC) (Given)
An HDS locator for a structure in which to create the new component holding a copy of the required array section.

NAME = CHARACTER∗(DAT__SZNAM) (Given)
The name of the new component to create within the " LOC2" structure. Any existing component with this name will be erased first.

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

KPG1_HSFLx
Writes an histogram to a Fortran formatted file

Description:

This routine writes an histogram to a Fortran ASCII file.

Invocation:

```
CALL KPG1_HSFLx( FD, NUMBIN, HIST, HRMIN, HRMAX, STATUS )
```

Arguments:

FD = INTEGER (Given)
Fortran file descriptor as provided by FIO.

NUMBIN = INTEGER (Given)
The number of bins in the histogram.

HIST(NUMBIN) = INTEGER (Given)
The array holding the histogram.

HRMIN = ? (Given)
The minimum data value that could be included within the histogram.

HRMAX = ? (Given)
The maximum data value that could be included within the histogram.

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

Notes:

- There is a routine for the following numeric data types: replace `" x"` in the routine name by D or R as appropriate. The extreme values of the histogram must have the data type specified.

Prior Requirements :

The Fortran file must all ready be opened and should have a record length of at least 52 characters.

KPG1_HSSTP
Calculates statistics from an histogram

Description:

This routine calculates certain statistical parameters for an histogram. The median is derived by linear interpolation within an histogram bin. The mode is estimated from 3∗median - 2∗mean, which is only valid for moderately skew distributions.

Invocation:

```
CALL KPG1_HSSTP( NUMBIN, HIST, VALMAX, VALMIN, SUM, MEAN, MEDIAN, MODE, STATUS )
```

Arguments:

NUMBIN = INTEGER (Given)
Number of bins in the histogram.

HIST(NUMBIN) = INTEGER (Given)
The histogram whose statistics are to be derived.

VALMAX = DOUBLE PRECISION (Given)
Maximum data value included in the histogram.

VALMIN = DOUBLE PRECISION (Given)
Minimum data value included in the histogram.

SUM = DOUBLE PRECISION (Returned)
Sum of all values in the histogram.

MEAN = DOUBLE PRECISION (Returned)
Mean of the histogram.

MEDIAN = DOUBLE PRECISION (Returned)
Median of the histogram.

MODE = DOUBLE PRECISION (Returned)
Mode of the histogram. The bad value (VAL__BADD) is returned if the calculated mode lies outside the data range of the histogram.

STATUS = INTEGER (Given)
Global status value.

References :

Moroney, M.J., 1957, " Facts from Figures" (Pelican) Goad, L.E. 1980, in " Applications of Digital Image Processing to Astronomy" , SPIE 264, 139.

KPG1_HSTAx
Computes simple ordered statistics for an array via an histogram

Description:

This routine computes simple ordered statistics for an array, namely: the number of valid pixels, the minimum and maximum pixel values (and their positions), the pixel sum, the mean, the mode, the median, and selected percentiles. For efficiency reasons the routine computes an histogram, rather than completely sorting the data. The accuracy of the statistics therefore depends inversely on the number of bins.

Invocation:

```
CALL KPG1_HSTAx( BAD, EL, DATA, NUMPER, PERCNT, NGOOD, IMIN, DMIN, IMAX, DMAX, SUM,
MEAN, MEDIAN, MODE, PERVAL, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether checks for bad pixels should be performed on the array being analysed.

EL = INTEGER (Given)
Number of pixels in the array.

DATA(EL) = ? (Given)
Array to be analysed.

NUMPER = INTEGER (Given)
Number of percentiles to evaluate.

PERCNT(NUMPER) = REAL (Given and Returned)
The array of percentiles to evaluate. They must in the range 0.0 to 100.0. If there are none to calculate, set NUMPER to 1 and pass the bad value in PERCNT(1). On exit these are placed in ascending order.

NGOOD = INTEGER (Returned)
Number of valid pixels in the array.

IMIN = INTEGER (Returned)
Index where the pixel with the lowest value was (first) found.

DMIN = DOUBLE PRECISION (Returned)
Minimum pixel value in the array.

IMAX = INTEGER (Returned)
Index where the pixel with the highest value was (first) found.

DMAX = DOUBLE PRECISION (Returned)
Maximum pixel value in the array.

SUM = DOUBLE PRECISION (Returned)
Sum of the valid pixels.

MEAN = DOUBLE PRECISION (Returned)
Mean of the valid pixels.

MEDIAN = DOUBLE PRECISION (Returned)
Median of the valid pixels.

MODE = DOUBLE PRECISION (Returned)
Mode of the valid pixels.

PERVAL(NUMPER) = DOUBLE PRECISION (Returned)
Percentile values of the valid pixels. These correspond to the ordered fractions returned in PERCNT.

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

Notes:

- There is a routine for each of the standard numeric types. Replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The data type of the array being analysed must match the particular routine used.
- If NGOOD is zero, then the values of all the derived statistics will be undefined and will be set to the " bad" value appropriate to their data type (except for the pixel sum, which will be zero).
- The median and percentiles are derived by linear interpolation within histogram bins, after clipping of outliers if the histogram is sparse. The mode is estimated from 3∗median - 2∗mean, which is only valid for moderately skew distributions.

References :

Moroney, M.J., 1957, " Facts from Figures" (Pelican) Goad, L.E. 1980, in " Applications of Digital Image Processing to Astronomy" , SPIE 264, 139.

Deficiencies :

- Does not compute the median and any percentiles in a single invocation of KPG1_FRACx.

KPG1_HSTFx
Finds values corresponding to specified fractions of an histogram

Description:

This routine finds the values at defined fractions of an histogram. Thus to find the lower-quartile value, the fraction would be 0.25. Since an histogram technique is used rather than sorting the whole array, the result may not be exactly correct. An histogram with a large number of bins, and the use of linear interpolation between bins in the routine reduce the error.

Invocation:

```
CALL KPG1_HSTFx( NUMBIN, HISTOG, MAXMUM, MINMUM, NFRAC, FRAC, VALUES, STATUS )
```

Arguments:

NUMBIN = INTEGER (Given)
The number of histogram bins. The larger the number of bins the greater the accuracy of the results. 1000 to 10000 is recommended.

HISTOG(NUMBIN) = INTEGER (Given)
The histogram.

MAXMUM = ? (Given)
The maximum value used to evaluate the histogram.

MINMUM = ? (Given)
The minimum value used to evaluate the histogram.

NFRAC = INTEGER (Given)
Number of fractional positions.

FRAC(NFRAC) = REAL (Given and Returned)
Fractional positions in the histogram in the range 0.0–1.0. On exit they are arranged into ascending order.

VALUES(NFRAC) = ? (Returned)
Values corresponding to the ordered fractional positions in the histogram.

STATUS = INTEGER (Given & Returned)
Global status value.

Notes:

- There is a routine for real and double-precision data types: replace "`x`" in the routine name by R or D as appropriate. The arguments MAXMUM, MINMUM, and VALUES must have the data type specified.

KPG1_HSTLO
Computes the values to display for a histogram

Description:

This routine find the set of (X,Y) values representing the centre of each bin in a given histogram. The logarithm of the supplied values are used if logarithmic axes are requested. In this case, zero or negative values result in VAL__BADR values being returned.

Invocation:

```
CALL KPG1_HSTLO( NHIST, HIST, HMIN, HMAX, XLOG, YLOG, XLOC, YLOC, STATUS )
```

Arguments:

NHIST = INTEGER (Given)
The number of bins in the histogram.

HIST(NHIST) = INTEGER (Given)
The histogram whose locus is to be found.

HMIN = REAL (Given)
The minimum data value that could be included within the histogram.

HMAX = REAL (Given)
The maximum data value that could be included within the histogram.

XLOG = LOGICAL (Given)
If .TRUE., logarithmic value are returned for the X axis.

YLOG = LOGICAL (Given)
If .TRUE., logarithmic value are returned for the Y axis.

XLOC(NHIST) = REAL(WRITE)
Work array for the x locus of the histogram.

YLOC(NHIST) = REAL(WRITE)
Work array for the y locus of the histogram.

STATUS = INTEGER (Given and Returned)
This is the status value on entry to this subroutine.

KPG1_HSTQx
Equalises the histogram of an array

Description:

This routine equalises (linearises) the histogram of an array. Thus approximately equal numbers of different values appear in output equalised version of the input array.

Invocation:

```
CALL KPG1_HSTQx( BAD, EL, INARR, NUMBIN, OLDHST, MAP, OUTARR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true there may be bad values within the input array. If false there are no bad values. It is used as an efficiency mechanism.

EL = INTEGER (Given)
The dimension of the arrays.

INARR(EL) = ? (Given)
The array to be equalised.

NUMBIN = INTEGER (Given)
Number of bins used in histogram. A moderately large number of bins is recommended so that there is little artifactual quantisation is introduced, say a few thousand except for byte data.

OLDHST(NUMBIN) = INTEGER (Returned)
The histogram of the input array, i.e. before equalisation.

MAP(NUMBIN) = INTEGER (Returned)
Key to transform the input histogram's bin number to the equalised bin number after histogram mapping.

OUTARR(EL) = ? (Returned)
The array containing the equalised values.

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to and returned from this routine must have the data type specified.

References :

Gonzalez, R.C. and Wintz, P., 1977, "Digital Image Processing" , Addison-Wesley, pp. 118–126.

KPG1_IMPRG
Propagates NDF information for IMAGE-format applications

Description:

Until the NDF access routines are fully implemented in KAPPA, this routine provides a means by which KAPPA applications using the IMAGE data format may correctly propagate an input NDF. Thus a pre-V1.0 KAPPA application will behave like a post-V1.0 one that only works on primitive DATA_ARRAYs, and does not process VARIANCE, QUALITY, or HISTORY.

See SGP/38 for more details of the NDF and its propagation rules.

Invocation:

```
CALL KPG1_IMPRG( INLOC, CLIST, OUTLOC, STATUS )
```

Arguments:

INLOC = CHARACTER * (*) (Read)
Locator to the input NDF.

CLIST = CHARACTER * (*) (Read)
A comma-separated list of the NDF components which are to be propagated from the input to the output NDF. By default, HISTORY, LABEL and all extensions are propagated. See below for further details.

OUTLOC = CHARACTER * (*) (Read)
Locator to the output NDF.

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

Component Propagation :

The template components whose values are to be propagated to initialise the new data structure are specified via the CLIST argument. Thus CLIST=' AXIS,QUALITY' would cause the new NDF to inherit its axes structures and QUALITY values (if available) from the input structure, in addition to those propagated by default.

KPG1_INCOx
Obtains a list of co-ordinates from the environment

Description:

The supplied parameter is used to get a list of co-ordinates from the environment. These co-ordinates are stored in dynamic workspace which is expanded as necessary so that any number of points can be given. The user indicates the end of the list of points by a null value. Pointers to the workspace arrays holding the co-ordinates are returned, and should be annulled when no longer needed by calling PSX_FREE.

Invocation:

```
CALL KPG1_INCOx( PNAME, NDIM, NPOINT, IPCO, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
Name of the parameter with which to associate the co-ordinates.

NDIM = INTEGER (Returned)
The number of co-ordinate dimensions. It must be positive and no more than DAT__MXDIM.

NPOINT = INTEGER (Returned)
The number of points specified.

IPCO = INTEGER (Returned)
A pointer to workspace of type <HTYPE> and size NDIM by NPOINT holding the co-ordinates of each point.

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

Notes:

- There is a routine for real or double-precision numeric data types: replace " x" in the routine name by D or R as appropriate. The returned co-ordinate array will have the specified data type.

KPG1_IS3x
Sorts an array of data into increasing order, also shuffling two ancillary arrays correspondingly

Description:

The routine uses an insert sort method. This has proven itself the quickest for sorting small sets of data lots of times, as in image stacking using ordered statistics. The method looks at the second value, compares this with the first if swaps if necessary, then it looks at the third, looks at the previous values swaps with the lowest (or not at all) and so on until all values have been passed. It is fast (for the case above) simply because of the very few lines of operation. The sort is extended to the ancillary data DDAT, and PDAT these maintain their correspondence with the ORDDAT dataset on exit.

Invocation:

```
CALL KPG1_IS3x( ORDDAT, DDAT, PDAT, NENT, STATUS )
```

Arguments:

ORDDAT(NENT) = ? (Given and Returned)
The data to order. On output it contains the data in increasing order.

DDAT(NENT) = ? (Given and Returned)
A list of data associated with ORDDAT which needs to retain its correspondence with the items in ORDDAT.

PDAT(NENT) = INTEGER (Given and Returned)
A list of data associated with ORDDAT which needs to retain its correspondence with the items in ORDDAT (probably pointers).

NENT = INTEGER (Given)
The number of entries in ORDDAT.

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

Notes:

- There is a routine for the double-precision and real floating-point data types: replace " x" in the routine name by D or R as appropriate. The ORDDAT and DDAT arrays must have the data type specified.

KPG1_ISCLx
Scales image between limits

Description:

The input two-dimensional array is scaled between lower and upper limits such that the lower limit and all values below it are set to a minimum value, the upper limit and all values above are set to a maximum value, and all valid values in between are assigned values using linear interpolation between the limits. The image may or may not be inverted. The sign of the scaling can be reversed. The scaled data are written to an integer output array.

Invocation:

```
CALL KPG1_ISCLx( BAD, DIM1, DIM2, INARR, INVERT, LOWER, UPPER, :  LOWLIM, UPPLIM, BADVAL,
OUTARR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true there will be no checking for bad pixels.

DIM1 = INTEGER (Given)
The first dimension of the two-dimensional arrays.

DIM2 = INTEGER (Given)
The second dimension of the two-dimensional arrays.

INARR(DIM1, DIM2) = ? (Given)
The original, unscaled image data.

LOWER = ? (Given)
The data value that specifies the lower image-scaling limit. This may be smaller than %UPPER to provide a negative scale factor.

UPPER = ? (Given)
The data value that specifies the upper image-scaling limit.

INVERT = LOGICAL (Given)
True if the image is to be inverted for display.

LOWLIM = INTEGER (Given)
The lowest value to appear in the scaled array for good input data.

UPPLIM = INTEGER (Given)
The highest value to appear in the scaled array for good input data.

BADVAL = INTEGER (Given)
The value to be assigned to bad pixels in the scaled array.

OUTARR(DIM1, DIM2) = INTEGER (Returned)
The scaled version of the image.

STATUS = INTEGER (Given)
Value of the status on entering this subroutine.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array supplied to the routine must have the data type specified.

KPG1_ISSCS
Extracts the epoch of the reference equinox from a string specifying an IRAS90 Sky Co-ordinate System

Description:

If, on entry, the argument SCS contains an explicit IRAS90 equinox specifier (see SUN/163), the epoch contained within it is returned in argument EQU as a double precision value, and argument BJ is returned equal to the character " B" or " J" depending on whether the epoch is Besselian or Julian. If there is no equinox specifier in argument SCS on entry, then the default of B1950 is returned.

If the sky co-ordinate system specified by SCS is not referred to the equinox (e.g.GALACTIC) then EQU is returned equal to the Starlink " BAD" value VAL__BADD, and BJ is returned blank.

The argument NAME is returned holding the full (unabbreviated) name of the sky co-ordinate system without any equinox specifier. On exit, the argument SCS holds the full name plus an explicit equinox specifier (for systems which are referred to the equinox). Thus, if SCS contained " EQUAT" on entry, it would contain " EQUATORIAL(B1950)" on exit.

Invocation:

```
RESULT = KPG1_ISSCS( SCS, EQU, BJ, NAME, STATUS )
```

Arguments:

SCS = CHARACTER * (*) (Given and Returned)

On entry this should contain an IRAS90 SCS name (or any unambiguous abbreviation), with or without an equinox specifier. On exit, it contains the full SCS name with an explicit equinox specifier (for those sky co-ordinate systems which are referred to the equinox). If no equinox specifier is present on entry, then a value of B1950 is used (if required). This variable should have a declared length of 25.

EQU = DOUBLE PRECISION (Returned)

The epoch of the reference equinox. This is extracted from any explicit equinox specifier contained in SCS on entry. If there is no equinox specifier in SCS, a value of 1950.0 is returned. If the sky co-ordinate system is not referred to the equinox (e.g. GALACTIC) the Starlink " BAD" value (VAL__BADD) is returned, irrespective of any equinox specifier in SCS.

BJ = CHARACTER * (*) (Returned)

Returned holding either the character B or J. Indicates if argument EQU gives a Besselian or Julian epoch. Returned blank if the sky co-ordinate system is not referred to the equinox.

NAME = CHARACTER * (*) (Returned)

The full name of the sky co-ordinate system without any equinox specifier. This variable should have a declared length of 25.

STATUS = INTEGER (Given and Returned)

The global status.

Function Value :

KPG1_ISSCS = LOGICAL Returned .TRUE. if the supplied string is a valid IRAS90 SCS specifier, and .FALSE. otherwise (no error is reported).

KPG1_KER1x
Smooths an n-dimensional array using a specified one-dimensional kernel

Description:

The routine smooths the array IN using a specified one-dimensional kernel and returns the result in the array OUT. The one-dimensional kernel can be used to smooth either a single axis or mutiple axes. If multiple axes are smoothed, the total smoothing kernel is equivalent to the product of the kernels used to smooth each individual axis.

Any output pixel that is contributed to by a bad pixel, or a pixel that is over the edge of the input data array, is set bad in the output.

Invocation:

```
CALL KPG1_KER1x( NDIM, DIMS, IN, NAX, AXES, SIZE, CENTRE, KERNEL, WORK, OUT, STATUS
)
```

Arguments:

NDIM = INTEGER (Given)
The number of pixel axes in the IN and OUT arrays.

DIMS(NDIM) = INTEGER (Given)
The length, in pixels, of each dimension of the IN and OUT arrays.

IN($*$) = ? (Given)
The input array, accessed as a one-dimensional vector.

NAX = INTEGER (Given)
The number of axes along which smoothing should be performed.

AXES(NAX) = INTEGER (Given)
The indices of the pixel axes along which smoothing should be performed. The first axis is Axis 1.

SIZE = INTEGER (Given)
The number of values in the kernel.

CENTRE = INTEGER (Given)
The index of the element within KERNEL that corresponds to the centre of the kernel.

KERNEL(SIZE) = ? (Given)
The kernel array. The values in this array are used as supplied

- they are not normalised to a total sum of unity.

WORK(SIZE) = ? (Given and Returned)
A work array.

OUT($*$) = ? (Returned)
The output array, accessed as a one-dimensional vector.

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

Notes:

- This routine can be used to smooth either DATA or VARIANCE arrays. If a DATA array is smoothed with a given kernel, the corresponding VARIANCE array should be smoothed with a kernel containing the squares of the values in the original kernel.
- There are routines for processing double precision and real data. Replace "`x`" in the routine name by D or R as appropriate. The data types of the IN, KERNEL and OUT arguments must match the routine used.

KPG1_KGODx
Sorts through a dataset and throw away bad values

Description:

This routine copies good data and its variance from input arrays to to output arrays, leaving behind bad values.

Invocation:

```
CALL KPG1_KGODx( VAR, NPTS, INARR, INVAR, NGOOD, OUTARR, OUTVAR, STATUS )
```

Arguments:

VAR = LOGICAL (Given)
If .TRUE., copy the variance array as well, else only copy the data array.

NPTS = INTEGER (Given)
The number of points in the input array.

INARR(NPTS) = ? (Given)
Input data array.

INVAR(NPTS) = ? (Given)
Input variance array.

NGOOD = INTEGER (Returned)
Number of good points in the input data.

INARR(NPTS) = ? (Returned)
Output data array.

INVAR(NPTS) = ? (Returned)
Output variance array.

STATUS = INTEGER (Given & Returned)
Global status value.

Notes:

- Data are still good even if variance is bad. Put another way, when the data is good, its corresponding variance is copied regardless of its value.
- There is a routine for all numeric data types: replace "`x`" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The arrays supplied to the routine must have the data type specified.

KPG1_KYGRP
Creates an GRP group holding keyword/value pairs read from an AST KeyMap

Description:

This function is the inverse of KPG1_KYMAP. It extracts the values from the supplied AST KeyMap and creates a set of " name=value" strings which it appends to a supplied group (or creates a new group). If the KeyMap contains nested KeyMaps, then the " name" associated with each primitive value stored in the returned group is a hierarchical list of component names separated by dots.

Invocation:

```
CALL KPG1_KYGRP( KEYMAP, IGRP, STATUS )
```

Arguments:

KEYMAP = INTEGER (Given)

An AST pointer to the existing KeyMap. Numerical entries which have bad values (VAL__BADI for integer entries or VAL__BADD for floating point entries) are not copied into the group.

IGRP = INTEGER (Returned)

A GRP identifier for the group to which to append the " name=value" strings read from the KeyMap. A new group is created if GRP__NOID is supplied.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_KYHDS
Copies values from an AST KeyMap to a primitive HDS object

Description:

This function fills a specified HDS object with primitive values read from a vector entry in an AST KeyMap. It is the inverse of KPG1_HDSKY. The HDS object must already exist and must be a primitive array or scalar. The values to store in the HDS object are read from the KeyMap entry that has a key equal to the name of the HDS object. The vector read from the KeyMap is interpreted as an N-dimension array, where N is the number of dimensions in the HDS object. Array slices can be re-arranged as they are copied from KeyMap to HDS object. The AXIS argument specifies which axis is being re-arranged. Each array slice is perpendicular to this axis. The KeyMap array and the HDS array are assumed to have the same dimensions on all other axes.

Invocation:

```
CALL KPG1_KYHDS( KEYMAP, MAP, AXIS, MODE, LOC, STATUS )
```

Arguments:

KEYMAP = INTEGER (Given)
An AST pointer to the KeyMap.

MAP($*$) = INTEGER (Given)
An array which indicates how to map slices in the KeyMap array onto slices in the HDS array. The length of the supplied array should be equal to the HDS array dimension specified by AXIS. Element J of this array says where the data for the J' th slice of the HDS array should come from, where J is the index along the axis specified by AXIS. The value of element J is a zero-based index along axis AXIS of the array read from the KeyMap.

AXIS = INTEGER (Given)
The index of the axis to be re-arranged. The first axis is axis 1.

MODE = INTEGER (Given)
Specifies what happens if the supplied KeyMap does not contain an entry with the name of the supplied HDS object.

1 - Report an error.

2 - Do nothing

LOC = CHARACTER $*$ (DAT__SZLOC) (Given)
An HDS locator for a primitive scalar or array object.

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

Notes:

- An error is reported if the supplied HDS object is a structure.

KPG1_KYMAP
Creates an AST KeyMap holding keyword/value pairs read from a GRP group

Description:

This function checks each non-comment, non-blank line in the supplied GRP group. An error is reported if any such lines do not have the form " keyword = value" , where the keyword name can be a hierarchical list of component names separated by dots. The returned KeyMap has an entry for each component name found at the start of any keyword name. The value associated with the entry will either be a primitive value (if the keyword name contained no other components) or another KeyMap (if the keyword name contained other components).

For example, consider a group containing the following lines:

gaussclumps.epsilon = (0.001,0.002) gaussclumps.contrast = 2.3 clumpfind.naxis = 2 clumpfind.deltat = 2.0 method = gaussclumps

The returned KeyMap will contain three entries with keys " gaussclumps" , " clumpfind" and " method" . The value associated with the " gaussclumps" entry will be another KeyMap containing keys " epsilon" (a primitive vector entry containing the values 0.001 and 0.002) and " contrast" (a primitive scalar entry with value " 2.3"). The value associated with the " clumpfind" entry will be another KeyMap containing keys " naxis" and " deltat" , which will have primitive scalar values " 2" and " 2.0" . The value associated with the " method" entry will be the primitive scalar value " gaussclumps" .

Assigning the value " <def>" (case insensitive) to a keyword has the effect of removing the keyword from the KeyMap. For example:

^global.lis method = <def>

reads keyword values from the file " global.lis" , and then ensures that the KeyMap does not contain a value for keyword " method" . The calling application should then usually use a default value for " method" .

Assigning the value " <undef>" (case insensitive) to a keyword has the effect of forcing the value to be undefined. This can be useful in defining defaults where the keymap is locked after being populated.

Invocation:

```
CALL KPG1_KYMAP( IGRP, KEYMAP, STATUS )
```

Arguments:

IGRP = INTEGER (Given)
A GRP identifier for the group of text strings to be analysed.

KEYMAP = INTEGER (Returned)
An AST pointer to the new KeyMap, or AST__NULL if an error occurs. A valid pointer to an empty KeyMap will be returned if the supplied group contains nothing but comments and blank lines.

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

Notes:

- Vector elements should be separated by commas and enclosed within parentheses (commas and closing parentheses can be included literally in a vector element by preceeding them with a backslash).

- Component names must contain only alphanumerical characters, underscores, plus and minus signs [a-zA-Z0-9_+\-]. White space within keywords is ignored.
- Any lower case characters contained in a component name will be translated to the upper case equivalent.
- If the last non-blank character in a value is a backslash (" \ "), the backslash will be reomved, together with any white space following it, and the entire next line will be appended to the value.

KPG1_LGTRN
Saves a transformation for a base-10 logarithmic plot in the AGI database

Description:

This routine defines the transformations between world and a log10-log10 or log10-linear world co-ordinate system, and saves the transformation in the AGI database with the current picture.

Invocation:

```
CALL KPG1_LGTRN( XLOG, YLOG, STATUS )
```

Arguments:

XLOG = LOGICAL (Given)

If true the x-axis is logarithmic in world co-ordinates, otherwise it is linear.

YLOG = LOGICAL (Given)

If true the y-axis is logarithmic in world co-ordinates, otherwise it is linear.

STATUS = INTEGER (Given and Returned)

The global status.

Prior Requirements :

- There must be a current AGI picture.

KPG1_LIKE
Creates a section from an NDF that matches the area of another NDF

Description:

This routine returns an identifier for a section of a supplied NDF (INDF1) that covers the same area as a second template NDF (INDF2). The area matched can be either the pixel area, or the WCS area.

The number of pixel axes in the input is not changed, even if the template has a different number of pixel axes.

Invocation:

```
CALL KPG1_LIKE( INDF1, INDF2, LIKWCS, INDF3, STATUS )
```

Arguments:

INDF1 = INTEGER (Given)
The first NDF, from which a section is to be copied.

INDF2 = INTEGER (Given)
The second NDF (the template).

LIKWCS = LOGICAL (Given)
If .TRUE., then the section selected from INDF1 matches the WCS bounding box of INDF2. If .FALSE., then the section selected from INDF1 matches the pixel index bounding box of INDF2. An error is reported if LIKWCS is .TRUE. and the two WCS FrameSets cannot be aligned.

INDF3 = INTEGER (Returned)
The required section of INDF1.

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

KPG1_LINTD
Obtains a linear transformation between two sets of x,y positions with least squared error

Description:

The co-efficients of a linear transformation are returned which maps the (XA,YA) positions to the corresponding (XB,YB) positions with least squares error. The type of fit can be specified as:

o A shift of origin only (IFIT 1)

o A shift and rotation (IFIT = 2)

o A shift, rotation and magnification (IFIT = 3)

o A shift, rotation, magnification and shear (IFIT = 4)

If the value of IFIT is too high for the supplied data, a lower value will be used and returned in IFIT. The returned coefficients are such that:

Fitted XB position = C(1) + C(2)∗XA + C(3)∗YA

Fitted YB position = C(4) + C(5)∗XA + C(6)∗YA

Invocation:

```
CALL KPG1_LINTD( N, XA, YA, XB, YB, IFIT, C, MAXERR, RMSERR, STATUS )
```

Arguments:

N = INTEGER (Given)
The number of supplied positions.

XA(N) = DOUBLE PRECISION (Given)
The X co-ordinates at the first set of positions.

YA(N) = DOUBLE PRECISION (Given)
The Y co-ordinates at the first set of positions.

XB(N) = DOUBLE PRECISION (Given)
The X co-ordinates at the second set of positions.

YB(N) = DOUBLE PRECISION (Given)
The Y co-ordinates at the second set of positions.

IFIT = INTEGER (Given and Returned)
The type of fit required. A lower value will be used (and returned) if a fit of the specified type could not be obtained.

C(6) = DOUBLE PRECISION (Returned)
The coefficients of the linear fit.

MAXERR = DOUBLE PRECISION (Returned)
The maximum error between the supplied (XB,YB) positions and the fitted (XB,YB) positions, in pixels.

RMSERR = DOUBLE PRECISION (Returned)
The RMS error between the supplied (XB,YB) positions and the fitted (XB,YB) positions, in pixels.

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

KPG1_LISTC
Lists a character array to an ASCII file or reports it to the user

Description:

This takes a character array and either writes the array to an open ASCII file or uses the message system to report it to the user.

Invocation:

```
CALL KPG1_LISTC( FD, EL, ARRAY, FILE, STATUS )
```

Arguments:

FD = INTEGER (Given)
The file descriptor for the log file. It is ignored if FILE is false.

EL = INTEGER (Given)
The number of character strings in the array.

ARRAY(EL) = CHARACTER * (*) (Given)
The array of character strings.

FILE = LOGICAL (Given)
If true the array is listed to the ASCII file otherwise the array is reported to the user.

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

Prior Requirements :

- The ASCII file must be opened.

KPG1_LITNx
Creates linear transformation expressions between two n-dimensional co-ordinate systems

Description:

This routine evaluates linear transformations between each of the dimensions of two co-ordinate systems of the same dimensionality. It substitutes the scale factors and offsets into chartacter expressions suitable for creating a TRANSFORM structure.

Invocation:

```
CALL KPG1_LITNx( NDIM, ILBND, IUBND, OLBND, OUBND, ITOO, OTOI, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
The number of dimensions in each co-ordinate system.

ILBND(NDIM) = ? (Given)
The lower bounds of the input co-ordinate system corresponding to points OLBND in the output co-ordinate system.

IUBND(NDIM) = ? (Given)
The upper bounds of the input co-ordinate system corresponding to points OUBND in the output co-ordinate system.

OLBND(NDIM) = ? (Given)
The lower bounds of the output co-ordinate system corresponding to points ILBND in the input co-ordinate system.

OUBND(NDIM) = ? (Given)
The upper bounds of the output co-ordinate system corresponding to points IUBND in the input co-ordinate system.

ITOO(NDIM) = CHARACTER $*$ ($*$) (Returned)
The transformation expressions to convert from the input to the output co-ordinate system. It should have a length at least of $26 + 2 * \text{VAL\_\_SZ<T>}$ characters.

OTOI(NDIM) = CHARACTER $*$ ($*$) (Returned)
The transformation expressions to convert from the output to the input co-ordinate system. It should have a length at least of $26 + 2 * \text{VAL\_\_SZ<T>}$ characters.

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

Notes:

- There is a routine for real and double-precision data types: replace "x" in the routine name by R or D respectively. The bounds of the co-ordinate systems supplied to the routine must have the data type specified.

KPG1_LITRx
Saves a transformation for a linear plot in the AGI database

Description:

This routine defines the transformations between world and a linear data co-ordinate system that has eithe, neither or both axes with reversed polarity (increasing right to left or top to bottom), and saves the transformation in the AGI database with the current picture.

Invocation:

```
CALL KPG1_LITRx( SCALE, OFFSET, STATUS )
```

Arguments:

SCALE(2) = ? (Given)

The scale factors of each linear axis to transform from world co-ordinates to data co-ordinates.

OFFSET(2) = ? (Given)

The offsets of each linear axis at pixel 0 to transform from world co-ordinates to data co-ordinates.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- There is a routine for real and double-precision data types: replace " x" in the routine name by R or D respectively. The scale and offset supplied to the routine must have the data type specified.

Prior Requirements :

- There must be a current AGI picture.

KPG1_LLTRx
Saves a transformation for a data co-ordinate linear or logarithmic plot in the AGI database

Description:

This routine defines the transformations between world and a log10-log10 or log10-linear data co-ordinate system, that has either, neither or both axes with reversed polarity (increasing right to left or top to bottom), and saves the transformation in the AGI database with the current picture.

Invocation:

```
CALL KPG1_LLTRx( XLOG, YLOG, SCALE, OFFSET, STATUS )
```

Arguments:

XLOG = LOGICAL (Given)
If true the x-axis is logarithmic in data co-ordinates, otherwise it is linear.

YLOG = LOGICAL (Given)
If true the y-axis is logarithmic in data co-ordinates, otherwise it is linear.

SCALE(2) = ? (Given)
The scale factors of each linear axis to transform from world co-ordinates to data co-ordinates.

OFFSET(2) = ? (Given)
The offsets of each linear axis at pixel 0 to transform from world co-ordinates to data co-ordinates.

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

Notes:

- There is a routine for real and double-precision data types: replace " x" in the routine name by R or D respectively. The scale and offset supplied to the routine must have the data type specified.

Prior Requirements :

- There must be a current AGI picture.

KPG1_LOCTx
Locates the centroid of a blob image feature in an n-dimensional array

Description:

This routine locates the centroid of a blob feature within a defined search area in an n-dimensional array about suggested starting co-ordinates, and returns the final centroid position. A blob is a series of connected pixels above or below the value of the surrounding background.

The routine forms marginal profiles within a search square. A separate background estimate is subtracted from each sample in each of the profiles, in order to ensure that he profiles have a roughly flat background. The background estimate used for each profile sample is the lower quartile of the array values which were included in the sample. If this quartile cannot be calculated (in the case of a one-dimensional array, for instance, each sample will have only one contribution and so the quartile cannot be found), the background estimate is taken from a straight line passing through the first and last points of the profile.

A background value for each whole profile is then found, and the centroid of all data above this background level is found. The whole process is repeated a number of times, using the previous centroid position as the new initial guess position. Iterations continue until the maximum number of iterations is reached, or the requested accuracy is met, or until one of several error conditions is met.

Invocation:

```
CALL KPG1_LOCTx( NDIM, LBND, UBND, ARRAY, INIT, SEARCH, POSTIV, MXSHFT, MAXITE, TOLER,
FINAL, SEL, WORK1, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
The dimensionality of the n-dimensional array. It must be greater than 1.

LBND(NDIM) = INTEGER (Given)
The lower bounds of the n-dimensional array.

UBND(NDIM) = INTEGER (Given)
The upper bounds of the n-dimensional array.

ARRAY(*) = ? (Given)
The input data array.

INIT(NDIM) = REAL (Given)
The co-ordinates of the initial estimate position.

SEARCH(NDIM) = INTEGER (Given)
Size of the search region to be used in pixels along each dimension. Each value must be odd and lie in the range 3–51.

POSTIV = LOGICAL (Given)
True if image features are positive above the background.

MXSHFT(NDIM) = REAL (Given)
Maximum shifts allowable from the initial position along each dimension.

MAXITE = INTEGER (Given)
Maximum number of iterations to be used. At least one iteration will be performed even if this is less than one.

TOLER = REAL (Given)
Accuracy required in the centroid position.

SEL = INTEGER (Given)
The number of elements in a search box (i.e. the product of the values in SEARCH).

FINAL(NDIM) = REAL (Returned)
The final co-ordinates of the centroid position.

WORK1(51, SEL, NDIM) = REAL (Returned)
A work array.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for each of the numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate.
- The lower and upper bounds must correspond to the dimension of the array which is passed by assumed size, and therefore the routine does not check for this.

Bugs:
{note_bugs_here}

KPG1_LOGAx
Takes the logarithm to an arbitrary base of an array and its variance

Description:

This routine fills the output array pixels with the results of taking the logarithm of the pixels of the input array to the base specified, i.e. New_value = Log Old_value. Base If the input pixel is negative, then a bad pixel value is used for the output value.

To find the logarithm to any base, the following algorithm is used, providing the base is positive and not equal to one:

New_Value = Log Old_Value / Log Base e e

If variance is present it is computed as: 2 New_Variance = Old_Variance / (Old_value $*$ log Base) e

Invocation:

```
CALL KPG1_LOGAx( BAD, EL, INARR, VAR, INVAR, BASE, OUTARR, OUTVAR, NERR, NERRV, STATUS
)
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input array.

EL = INTEGER (Given)
Number of elements in the input output arrays.

INARR(EL) = ? (Given)
Array containing input image data.

VAR = LOGICAL (Given)
If .TRUE., there is a variance array to process.

INVAR(EL) = ? (Given)
Array containing input variance data, when VAR = .TRUE..

BASE = DOUBLE PRECISION (Given)
Base of logarithm to be used. It must be a positive number.

OUTARR(EL) = ? (Write)
Array containing results of processing the input data.

OUTVAR(EL) = ? (Write)
Array containing results of processing the input variance data, when VAR = .TRUE..

NERR = INTEGER (Returned)
Number of numerical errors which occurred while processing the data array.

NERRV = INTEGER (Returned)
Number of numerical errors which occurred while processing the variance array, when VAR = .TRUE..

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.

KPG1_LSTAR
Writes a section of a two-dimensional array to a text file

Description:

This routine takes an input two-dimensional array and lists a specified section of that image, as defined by the x-and-y lower-and-upper bounds to a Fortran file. The file must either be already opened and specified by the input file descriptor, or be created in this routine and will be associated with the supplied parameter name. The first record in the file corresponds to the lowest index row.

Invocation:

```
CALL KPG1_LSTAR( DIM1, DIM2, ARRAY, XLOW, YLOW, XHIGH, YHIGH, OPENF, FDI, FILNAM, STATUS
)
```

Arguments:

DIM1 = INTEGER (Given)
The first dimension of the two-dimensional array.

DIM2 = INTEGER (Given)
The second dimension of the two-dimensional array.

ARRAY(DIM1, DIM2) = REAL (Given)
The two-dimensional array to be listed.

XLOW = INTEGER (Given)
x co-ord of lower-left corner of sub-array to be listed out.

YLOW = INTEGER (Given)
y co-ord of lower-left corner of sub-array to be listed out.

XHIGH = INTEGER (Given)
x co-ord of upper-right corner of sub-array to be listed out.

YHIGH = INTEGER (Given)
y co-ord of upper-right corner of sub-array to be listed out.

OPENF = LOGICAL (Given)
If true the Fortran file is to be associated with %FILNAM and opened. Otherwise the file is assumed to have been opened and has descriptor %FD.

FDI = INTEGER (Given)
The descriptor associated with the previously opened Fortran file.

FILNAM = CHARACTER * (*) (Given)
Parameter name of the file to be opened and to contain the listing output.

STATUS = INTEGER (Given and Returned)
Global status value

KPG1_LTGET
Obtains a locator to an array holding a colour table for the currently opened graphics device

Description:

This routine returns an HDS locator for a two-dimensional array holding the colour table to load into the currently open graphics device. The HDS object is searched for in an HDS container file in the users ADAM directory. The file is called " kappa.lut.sdf" and contains a LUT for different devices. The file should have been created by KPG1_LTSAV.

Each lut in the file is a _REAL array of shape (3,n) where n is the number of colours in the lut.

Each array has a name which identifies the graphics device to which it refers.

Invocation:

```
CALL KPG1_LTGET( PLOC, STATUS )
```

Arguments:

PLOC = CHARACTER * (DAT__SZLOC) (Returned)

The locator. returned equal to DAT__NOLOC if the colour table cannot be found, or if an error occurs.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- A graphics device must previously have been opened using PGPLOT.

KPG1_LTLOD
Loads the colour table for the currently open graphics device

Description:

This routine loads the colour table for the currently open graphics device from an HDS container file in the users ADAM directory. The file is called " kappa.lut.sdf" and contains a LUT for different devices. The file should have been created by KPG1_LTSAV. If the file does not exist, the colour table is set to a greyscale.

Each lut in the file is a _REAL array of shape (3,n) where n is the number of colours in the lut.

Each array has a name which identifies the graphics device to which it refers.

Invocation:

```
CALL KPG1_LTLOD( STATUS )
```

Arguments:

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

Notes:

- A graphics device must previously have been opened using PGPLOT.

KPG1_LTSAV
Saves the colour table for the currently open graphics device

Description:

This routine saves the colour table for the currently open graphics device in an HDS container file in the users ADAM directory. The file is called " kappa_lut.sdf" and contains LUTs for different devices. Each LUT is a _REAL array of shape (3,n) where n is the number of colours in the LUT. Each array has a name which identifies the graphics device to which it refers.

Invocation:

```
CALL KPG1_LTSAV( STATUS )
```

Arguments:

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

Notes:

- The HDS container file is created if it does not already exist.
- A graphics device must previously have been opened using PGPLOT.

KPG1_LUDCx
Performs an LU decomposition of a square matrix

Description:

This routine performs a decomposition of a square matrix into lower and upper triangular matrices using Crout' s method with partial pivoting. Implicit pivoting is also used to select the pivots.

Invocation:

```
CALL KPG1_LUDCx( N, EL, ARRAY, PINDEX, SCALE, EVEN, STATUS )
```

Arguments:

N = INTEGER (Given)

The number of rows and columns in the matrix to be decomposed.

EL = INTEGER (Given)

The size of the first dimension of the array as declared in the calling routine. This should be at least equal to N.

ARRAY(EL, N) = ? (Given and Returned)

On input this is the array to be decomposed into LU form. On exit it is the LU decomposed form of the rowwise-permuted input ARRAY, with the diagonals being part of the upper triangular matrix. The permutation is given by PINDEX.

SCALE(N) = ? (Returned)

Workspace to store the implicit scaling used to normalise the rows during implicit pivoting.

PINDEX(N) = INTEGER (Returned)

An index of the row permutations caused by the partial pivoting.

EVEN = LOGICAL (Returned)

If EVEN is .TRUE., there was an even number of row interchanges during the decomposition. If EVEN is .FALSE., there was an odd number.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The matrix and the workspace should have this data type as well.

KPG1_LUTIN
Transfers a lookup table between arrays that have different numbers of colour indices

Description:

This routine transfers a lookup table between arrays that may have different numbers of colour indices. When they are unequal the lookup table is expanded or contracted to fit the destination array. This may be achieved either by linear interpolation or by the nearest-neighbour method. (There are different sizes presumably because the lookup table was created on a device with a differently sized colour table.) If the arrays are of equal size the lookup table is merely copied between them.

Invocation:

```
CALL KPG1_LUTIN( INEL, INARR, OUTEL, NN, OUTARR, STATUS )
```

Arguments:

INEL = INTEGER (Given)
The number of colour indices in the lookup table.

INARR(3, 0:INEL-1) = REAL (Given)
The source lookup table. The first dimension is RGB. Values should lie in the range 0.0–1.0.

OUTEL = INTEGER (Given)
The number of colour indices available in the destination array. This is usually the number of colour indices available on the chosen graphics device.

NN = LOGICAL (Given)
If true, and the number of input and output colour indices are different, the nearest-neighbour method is used for assigning values in the output array. Otherwise linear interpolation is used. Nearest-neighbour preserves sharp edges in the lookup; linear interpolation is recommended for smoothly varying lookup tables.

OUTARR(3, 0:OUTEL-1) = REAL (Returned)
This is the array into which the table is put. The values will all lie in the range 0.0–1.0, even though the input may beyond this range. (Input values below 0.0 become 0.0 in OUTARR, and those above 1.0 become 1.0)

STATUS = INTEGER (Given)
Global status.

KPG1_LUTK2
Sets up the pixel colour indices which form a LUT key

Description:

This routine fills the COLS array with integer colour indices so that the array can be used as a colour table key. It can produce histogram or ramp keys.

Invocation:

```
CALL KPG1_LUTK2( FORM, CAXIS, LBND1, UBND1, LBND2, UBND2, HSTDAT, MAXPOP, LOG, COLS,
STATUS )
```

Arguments:

FORM = INTEGER (Given)
Indicates the form of key required: 0 - ramp, 1 - histogram.

CAXIS = INTEGER (Given)
The index of the array axis corresponding to colour index (1 or 2).

LBND1 = INTEGER (Given)
The lower bound on Axis 1.

UBND1 = INTEGER (Given)
The upper bound on Axis 1.

LBND2 = INTEGER (Given)
The lower bound on Axis 2.

UBND2 = INTEGER (Given)
The upper bound on Axis 2.

HSTDAT($*$) = INTEGER (Given)
A histogram of colour index counts. The length of this vector should be equal to the length of axis CAXIS.

MAXPOP = INTEGER (Given)
The maximum population in any cell of the histogram.

LOG = LOGICAL (Given)
If TRUE the histogram displays Log(count).

COLS(LBND1:UBND1, LBND2:UBND2) = INTEGER (Returned)
The returned array of colour indices.

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

KPG1_LUTK3
Produces a GRAPH colour table key for KPG1_LUTKY

Description:

This routine produces a colour table key consisting of one or three line plots (1 if the colour table is a greyscale, and 3 if it is not). The line plots are drawn using the supplied Plot.

Invocation:

```
CALL KPG1_LUTK3( IPLOT, PARAM, APP, LP, UP, X, RGB, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
The Plot to use.

PARAM = CHARACTER * (*) (Given)
The name of the style parameter to use (e.g. STYLE, KEYSTYLE). The appearance of the three curves can be set using attribute qualifiers (R), (G), and (B) (e.g. " width(r)=10").

APP = CHARACTER * (*) (Given)
The calling application, in the form " KAPPA_LUTVIEW" .

LP = INTEGER (Given)
The lowest PGPLOT colour index to copy.

UP = INTEGER (Given)
The highest PGPLOT colour index to copy.

X(LP:UP, 2) = DOUBLE PRECISION (Returned)
Work space to hold the X values

RGB(LP:UP, 3) = DOUBLE PRECISION (Returned)
Work space to hold the RGB intensities.

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

KPG1_LUTK4
Produces an AST Mapping from pen number to RGB intensity

Description:

This routine produces an AST Mapping which maps one-dimensional pen number into three-dimensional RGB intensity within the current PGPLOT graphics device.

Invocation:

```
CALL KPG1_LUTK4( LP, UP, WORK, MAP, STATUS )
```

Arguments:

LP = INTEGER (Given)
The lowest PGPLOT colour index to use.

UP = INTEGER (Given)
The highest PGPLOT colour index to use.

WORK(LP:UP, 3) = DOUBLE PRECISION (Returned)
Work space.

MAP = INTEGER (Returned)
The Mapping pointer.

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

KPG1_LUTKY
Draws a key showing a colour table

Description:

The key consists of a ramp of colour covering the specified range of colour indices, with annotated axes. Axis 1 is always the data-value axis (whether it is drawn vertically or horizontally). The whole plot (including annotation) is scaled to fit inside the picture specified by IPIC.

Invocation:

```
CALL KPG1_LUTKY( IPIC, PARAM, HIGH, LOW, LABEL, APP, LP, UP, F, GAP1, GAP2, JUST, RJUST,
NEL, COLDAT, STATUS )
```

Arguments:

IPIC = INTEGER (Given)
An AGI identifier for a picture in which the key is to be produced.

PARAM = CHARACTER * (*) (Given)
The name of the style parameter to use (e.g. STYLE, KEYSTYLE, etc.).

HIGH = REAL (Given)
The X-axis value (i.e. pixel value or pen number) corresponding to the colour index UP.

LOW = REAL (Given)
The X-axis value (i.e. pixel value or pen number) corresponding to the colour index LP.

LABEL = CHARACTER * (*) (Given)
The default label to put against the data values.

APP = CHARACTER * (*) (Given)
The calling application, in the form " KAPPA_LUTVIEW" .

LP = INTEGER (Given)
The smallest colour index to include in the display.

UP = INTEGER (Given)
The largest colour index to include in the display.

F = REAL (Given)
An amount by which to extend the margins left for annotation, expressed as a factor of the height or width of the plotting area. For instance, a value of 0.1 could be given to fit the annotation " comfortably" into the Plot. A value of 0.0 will result in the annotation being hard up against the edge of the plot.

GAP1 = REAL (Given)
A gap, in millimetres, to place between the bottom or right edge of the supplied picture, and the nearest edge of the colour ramp.

GAP2 = REAL (Given)
A gap, in millimetres, to place between the top or left edge of the supplied picture, and the nearest edge of the colour ramp.

JUST = CHARACTER*2 (Given)
Indicates the justification of the new plot within the specified area. ' BL' , ' BC' , ' BR' , ' CL' , ' CC' , ' CR' , ' TL' , ' TC' or ' TR' , where B is Bottom, C is Centre, T is Top, L is Left and R is Right. Must be upper case. If either character is a space, then the corresponding value from RJUST is used instead. Other unrecognised values are treated as " C" .

RJUST(2) = REAL (Given)
Each element is used only if the corresponding element in JUST is a apce. The first element gives the fractional vertical position of the new plot: 0.0 means put the new plot as low as possible, 1.0 means put it as high as possible. The second element gives the fractional horizontal position of the new plot: 0.0 means put the new plot as far to the left as possible, 1.0 means put it as far to the right as possible.

NEL = INTEGER (Given)
The size of the vector COLDAT. If this is zero, COLDAT is not accessed, and requested for keys in the form of a histogram are ignored (that is, keys are always produced in the form of a ramp).

COLDAT(NEL) = INTEGER (Given)
A vector of colour indices, one for each displayed data pixel. Only accessed if NEL is greater than zero and the STYLE parameter indicates that the key is to drawn in the form of a histogram.

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

KPG1_MANIx
Copies a data array with axis permutation and expansion

Description:

An input array is copied to an output array under control of a supplied AXES vector. Axes of the output array may be in a different order to those of the input array, and it may be required to expand along a new dimension or collapse along an existing combination (being replaced by the mean of all non-bad values).

Invocation:

```
CALL KPG1_MANIx( NDIMI, DIMI, IN, NDIMO, DIMO, AXES, COLOFF, EXPOFF, OUT, STATUS )
```

Arguments:

NDIMI = INTEGER (Given)
The dimensionality of the input array.

DIMI(NDIMI) = INTEGER (Given)
The shape of the input array.

IN(*) = ? (Given)
The input data array, vectorised. The number of elements is given by the product of the elements of DIMI.

NDIMO = INTEGER (Given)
The dimensionality of the output array (the number of elements in AXES).

DIMO(NDIMO) = INTEGER (Given)
The dimensions of the output array. Information about the extent of the expanded dimesions, as well as information implicit in DIMI and AXES, is held here.

AXES(NDIMO) = INTEGER (Given)
An array determining how the output array is copied from the input array. The I' th element determines the source of the I' th dimension of the output array. If it is zero, a new dimension will grow there. Otherwise, it gives the index of a dimension of the input array to copy.

COLOFF(*) = INTEGER (Returned)
Workspace. This array must have at least as many elements as the product of all the dimensions of the input array along which it is to be collapsed; that is the product of each element of DIMI whose index does not appear in AXES.

EXPOFF(*) = INTEGER (Returned)
Workspace. This array must have at least as many elements as the product of all the newly expanded dimensions in the output array; that is the product of each element of DIMO for which the corresponding element of AXES is zero.

OUT(*) = ? (Returned)
The output array, vectorised. The number of elements written is given by the product of the elements of NDIMO.

STATUS = INTEGER (Given)
The global status

Notes:

- There is a routine for all numeric data types: replace "x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The IN and OUT arguments must have the data type specified.
- The assumption that NDF__MXDIM = 7 is hard coded into this routine.
- A few arithmetic operations could be saved, at the expense of the code's clarity, by calculating the various array offsets at the same time as the next position vectors at various places in this routine. Since this routine is unlikely to be a computational bottleneck I have favoured clarity over optimal performance.

KPG1_MAP
Obtains mapped access to an array component of an NDF

Description:

This routine is a wrapper for NDF_MAP which obtains mapped access to an array component of an NDF, returning a pointer to the mapped values and a count of the number of elements mapped. At the moment this wrapper routine is a dummy which does nothing else.

Invocation:

```
CALL KPG1_MAP( INDF, COMP, TYPE, MMOD, PNTR, EL, STATUS )
```

Arguments:

INDF = INTEGER (Given)
NDF identifier.

COMP = CHARACTER * (*) (Given)
Name of the NDF array component to be mapped: 'DATA', 'QUALITY' or 'VARIANCE' (or 'ERROR').

TYPE = CHARACTER * (*) (Given)
Numeric type to be used for access (e.g. '_REAL').

MMOD = CHARACTER * (*) (Given)
Mapping mode for access to the array: 'READ', 'UPDATE' or 'WRITE', with an optional initialisation mode '/BAD' or '/ZERO' appended.

PNTR(*) = INTEGER (Returned)
Pointer(s) to the mapped values (see the Notes section).

EL = INTEGER (Returned)
Number of elements mapped.

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

KPG1_MDETx
Computes the determinant of a matrix

Description:

This routine calculates the determinant of an arbitrary n-by-n matrix. It LU decomposes the matrix and then forms the product of the diagonals by summing their logarithms to base 10. Logarithmic calculations are used to prevent overflows. A bad status is returned when the matrix is singular or the determinant is too large to store in the chosen data type's number representation.

Invocation:

```
CALL KPG1_MDETX( N, EL, ARRAY, WORK1, WORK2, DET, STATUS )
```

Arguments:

N = INTEGER (Given)
The number of rows and columns in the matrix whose determinant is to be derived.

EL = INTEGER (Given)
The size of the first dimension of the array as declared in the calling routine. This should be at least equal to N.

ARRAY(EL, N) = ? (Given and Returned)
On input this is the matrix whose determinant is to be found. Since the LU decomposition is performed in situ, the input values are lost. (On exit it is the LU decomposed form of the rowwise-permuted input ARRAY, with the diagonals being part of the upper triangular matrix.)

WORK1(N) = INTEGER (Returned)
Workspace for the LU decomposition.

WORK2(N) = ? (Returned)
Workspace for the LU decomposition.

DET = ? (Returned)
The determinant of matrix ARRAY. It is set to 1.0 when the matrix is singular.

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

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The matrix ARRAY and the WORK2 workspace should have this data type as well.

KPG1_MEANx
Finds the mean of the good data values in an array

Description:
Finds the mean of the good data values in an array.

Invocation:

```
CALL KPG1_MEANx( N, DATA, MEAN, STATUS )
```

Arguments:

N = INTEGER (Given)
Number of elements in the array.

DATA(N) = ? (Given)
The data array.

MEAN = ? (Returned)
The mean value. Set to VAL__BAD<T> if no good data are found.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each of the standard numeric types. Replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array and mean arguments supplied to the routine must have the data type specified.
- The summation is carried out in double precision.

KPG1_MEDUx
Derives the unweighted median of a vector

Description:

This routine derives the unweighted median of an array. If there are an even number of elements, the median is computed from the mean of the two elements that straddle the median.

Invocation:

```
CALL KPG1_MEDUx( BAD, EL, ARRAY, MEDIAN, NELUSE, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If BAD is .TRUE. there may be bad pixels present in the vector and so should be tested. If BAD is .FALSE. there is no testing for bad values.

EL = INTEGER (Given)
The number of elements within the array to sort.

ARRAY(EL) = ? (Given and Returned)
The array whose median is to be found. On exit the array is partially sorted.

MEDIAN = ? (Returned)
The median of the array.

NELUSE = INTEGER (Returned)
The number of elements actually used after bad values have been excluded.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array and MEDIAN arguments supplied to the routine must have the data type specified.

kpg1_memry
Returns memory currently being used by the current process

Description:

This subroutine returns the maximum resident set size for the current process.

Invocation:

```
CALL KPG1_MEMRY( MEM, STATUS )
```

Arguments:

MEM = INTEGER (Returned)

The memory currently being used by the current process, in KB.

STATUS = INTEGER (Given and Returned)

The inherited global status.

KPG1_MIXVx
Obtains from a parameter character values from either a menu of options or within a numeric range

Description:

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

o one of a supplied list of acceptable values, with unambiguous abbreviations accepted; or

o a numeric character string equivalent to a number, and the number must lie within a supplied range of acceptable values.

Invocation:

```
CALL KPG1_MIXVx( PARAM, MAXVAL, VMIN, VMAX, OPTS, VALUES, ACTVAL, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The name of the parameter. A GRP group expression should be supplied for the parameter.

MAXVAL = INTEGER (Given)
The maximum number of values required. A PAR__ERROR status is returned when the number of values requested is fewer 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 numerical data type: replace " x" in the routine name by D or R as appropriate. The data range arguments supplied to the routine must have the data type specified.

- There are two stages to identify or validate each character value obtained from the parameter. In the first the value is converted to the required data type. 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 routine adheres to the following rules. o All comparisons are performed in uppercase. Leading blanks are ignored. o 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. o 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 new values. This is not achieved through the MIN/MAX system.

KPG1_MKLUT
Creates a Mapping to connect two one-dimensional array of values

Description:

This routine creates a one-dimensional Mapping which translates an X into a Y value on the basis of supplied tables of corresponding X and Y. This is like an AST LutMap except that the LutMap class requires Y to be tabulated at equal X intervals, whereas this routine allows Y to be tabulated at arbitrary X intervals.

Invocation:

```
CALL KPG1_MKLUT( IX, IY, NPNT, NVAR, FRM, TABLE, MAP, STATUS )
```

Arguments:

IX = INTEGER (Given)
The index of the X values within the TABLE array.

IY = INTEGER (Given)
The index of the Y values within the TABLE array.

NPNT = INTEGER (Given)
The number of values supplied for each variable in the TABLE array.

NVAR = INTEGER (Given)
The number of variables described in the table. This will be at least 2 (for X and Y) but may be more.

FRM = INTEGER (Given)
If not AST__NULL, then this should be an AST pointer to a Frame with NVAR axes which will be used to normalise the axis values before creating the LutMap. No normalisation occurs if a value of AST__NULL is supplied.

TABLE(NPNT, NVAR) = DOUBLE PRECISION (Given and Returned)
The table containing corresponding X and Y values. The table can also contain values for other variables, which will be ignored. These will be normalised on exit using the AST Frame supplied by FRM.

MAP = INTEGER (Returned)
An AST pointer to the returned Mapping, or AST__NULL if no Mapping could be created.

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

Notes:

- It is only possible to create the Mapping if the tabulated X values are monotonic increasing or decreasing.
- The returned Mapping will have an inverse Transformation only if Y increases or decreases monotonically with X.

KPG1_MKPOS
Marks a position on a graphics device

Description:

This routine marks a position on a graphics device in various ways.

Invocation:

```
CALL KPG1_MKPOS( NAX, POS, IPLOT, CURR, MODE, MARKER, GEO, DONE, CLOSE, TEXT, JUST,
REGION, STATUS )
```

Arguments:

NAX = INTEGER (Given)

The number of co-ordinate values supplied in POS. This should be equal to the number of axes in the Frame specified by CURR.

POS(NAX) = DOUBLE PRECISION (Given)

The co-ordinates of the position, in the Frame specified by CURR. No marker is drawn if a AST__BAD or VAL__BADD value is supplied Ignored if DONE is .TRUE., and MODE is " POLY" , " CHAIN" or " BOX" .

IPLOT = INTEGER (Given)

An AST pointer to a Plot. The same Plot should be supplied for all points in a polygon, chain, or set of boxes.

CURR = LOGICAL (Given)

If .TRUE., then position supplied in POS refers to the Current Frame in IPLOT. Otherwise it refers to the Base Frame (which should be the GRAPHICS Frame).

MODE = CHARACTER * (*) (Given)

The type of marker to produce (case sensitive, no abbreviations):

- " NONE" – An immediate return is made without any graphics being drawn.
- " MARK" – Each position is marked by the symbol specified by argument MARKER.
- " POLY" – Causes each position to be joined by a line to the previous position marked. These lines may be geodesic (in the Current Frame of the Plot) or straight (on the screen), as specified by argument GEO.
- " CHAIN" – This is a combination of " Mark" and " Poly" . Each position is marked by a symbol and joined by a line to the previous position. Arguments MARKER, GEO, and CLOSE are used to specify the symbols and lines to use.
- " BOX" – An empty rectangle with edges parallel to the axes is drawn extending between the supplied position and the previous position.
- " VLINE" – A vertical line is drawn through the position covering the entire height of the Plot.
- " HLINE" – A horizontal line is drawn through the position covering the entire height of the Plot.
- " CROSS" – A full-screen cross-hair is drawn at the position, i.e. a combination of Vline and Hline.
- " TEXT" – The text string specified by argument TEXT is displayed, horizontally, and centred on the supplied position.

- " BLANK" – Nothing is drawn.
- " REGION" – The AST Region given by REGION is outlined.

MARKER = INTEGER (Given)
The PGPLOT marker type to use if MODE is " MARKER" or " CHAIN" .

GEO = LOGICAL (Given)
Should polygon and chain line segments be drawn as geodesic curves within the Current Frame of the Plot? If not they are drawn as simple straight lines within the Base Frame (GRAPHICS). The same value should be supplied for all points in a polygon or chain.

DONE = LOGICAL (Given)
Should be supplied .TRUE. when a polygon, chain or set of boxes has been completed. The contents of POS will be ignored in this case. DONE is ignored if MODE is not " PLOT" , " CHAIN" or " BOX" .

CLOSE = LOGICAL (Given)
If .TRUE., polygons and chains are closed by joining the first position to the last position (when DONE is supplied .TRUE.).

TEXT = CHARACTER * (*) (Given)
The text to display if MODE is " TEXT" . Trailing spaces are ignored.

JUST = CHARACTER * (*) (Given)
A string specifying the justification to be used when displaying the text supplied in TEXT (ignored if MODE is not " Text"). This should be a string of two characters; the first should be " B" , " C" or " T" , meaning bottom, centre or top. The second should be " L" , " C" or " R" , meaning left, centre or right. The text is displayed so that the position supplied in POS is at the specified point within the displayed text string.

REGION = INTEGER (Given)
A two-dimensional AST Region pointer. Only used if PLOT is " REGION" . In order to save time calling AST_CONVERT for every Region, the first Region to be plotted using this routine defines the co-ordinate frame for all subsequent Regions draw by later invocations of this routine. If in fact, later Regions may have a different co-ordinate frame, then this routine should be called with MODE=REGION and REGION=AST__NULL. This will cause the current Region co-ordinate frame to be forgotten so that the next non-NULL Region to be draw will define a new current co-ordinate frame for subsequent Regions.

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

KPG1_MMTHx
Returns the maximum and minimum values of an array within thresholds

Description:

This routine returns the maximum and minimum values of an input array, where it found the maximum and minimum, and the number of good and bad pixels in the array. The extreme values can be constrained to lie between two thresholds, where values outside the thresholds are ignored. So for example this routine might be used to find the smallest positive value or the largest negative value.

An error report is made if all the values are bad or lie outside of the thresholds.

Invocation:

```
CALL KPG1_MMTHx( BAD, EL, ARRAY, THRESH, NINVAL, MAXMUM, MINMUM, MAXPOS, MINPOS, STATUS
)
```

Arguments:

BAD = LOGICAL (Given)
If .TRUE., there may be bad pixels present in the array. If .FALSE., it is safe not to check for bad values.

EL = INTEGER (Given)
The dimension of the input array.

ARRAY(EL) = ? (Given)
Input array of data.

THRESH(2) = ? (Given)
The thresholds between which the extreme values are to be found (lower then upper). Values equal to the thresholds are excluded. To find the extreme values across the full range, set the thresholds to VAL__MINx and VAL__MAXx or use routine KPG_MXMNx.

NINVAL = INTEGER (Returned)
Number of bad pixels in the array.

MAXMUM = ? (Returned)
Maximum value found in the array.

MINMUM = ? (Returned)
Minimum value found in the array.

MAXPOS = INTEGER (Returned)
Index of the pixel where the maximum value is (first) found.

MINPOS = INTEGER (Returned)
Index of the pixel where the minimum value is (first) found.

STATUS = INTEGER (Given)
Global status value

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The ARRAY, THRESH, MAXMUM, and MINMUM arguments supplied to the routine must have the data type specified.

KPG1_MODEx
Estimates the mean of a number of normally distributed data values, some of which may be corrupt

Description:

The routine is based on maximising the likelihood function for a statistical model in which any of the data points has a constant probability of being corrupt. A weighted mean is chosen according to the deviation of each data point from the current estimate of the mean. The weights are derived from the relative probabilities of being valid or corrupt. A sequence of these iterations converges to a stationary point in the likelihood function. The routine approximates to a k-sigma clipping algorithm for a large number of data points and to a mode-estimating algorithm for fewer data points.

Invocation:

```
CALL KPG1_MODEx( X, NX, PBAD, NITER, TOLL, XMODE, SIGMA, :  STATUS )
```

Arguments:

X(NX) = ? (Given)
An array of data values.

NX = INTEGER (Given)
The number of data values.

PBAD = REAL (Given)
An estimate of the probability that any one data point will be corrupt. (This value is not critical.)

NITER = INTEGER (Given)
The maximum number of iterations required.

TOLL = REAL (Given)
The absolute accuracy required in the estimate of the mean. Iterations cease when two successive estimates differ by less than this amount.

XMODE = REAL (Returned)
The estimate of the uncorrupted mean.

SIGMA = REAL (Returned)
An estimate of the uncorrupted standard deviation of the data points.

STATUS = INTEGER (Returned)
The global status.

Notes:

There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B, UB as appropriate. The array supplied to the routine must have the data type specified.

KPG1_MONOx
Determines whether an array's values increase or decrease monotonically

Description:

This routine determines whether or not a vector of values increase or decrease monotonically. This is most useful for axes.

Invocation:

```
CALL KPG1_MONOx( BAD, EL, ARRAY, MONOTO, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If BAD=.TRUE., there may be bad values present in the array, and it instructs this routine to test for the presence of bad values.

EL = INTEGER (Given)
The number of elements in the array.

ARRAY(EL) = ? (Given)
The array to be tested.

MONOTO = LOGICAL (Returned)
If MONOTO is .TRUE., the array values are monotonic.

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

Notes:

- There is a routine for the each of the numeric data types. replace "x" in the routine name by B, D, I, R, W, UB, or UB as as appropriate. The array should have the data type specified.

KPG1_MTHEx
Evaluates a mathematical expression for a set of data and variance arrays

Description:

The routine evaluates a general mathematical expression giving the values in an output data array in terms of values in a set of input data arrays, each of the same size. The expression to be evaluated is specified by means of a mapping identifier obtained from the TRANSFORM system (see SUN/61). A set of input variance arrays is also supplied, and this routine uses these to obtain estimates of the variance in the evaluated results. These output variance estimates are obtained numerically by perturbing the input data values by appropriate amounts.

Invocation:

```
CALL KPG1_MTHEx( BAD, EL, N, DAT, VFLAG, VAR, IMAP, QUICK, MXBAT, WRK, RES, VRES, BADDR,
BADVR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether it is necessary to check for bad values in the input data or variance arrays.

EL = INTEGER (Given)
Number of elements in each input data array.

N = INTEGER (Given)
Number of input data arrays.

DAT(EL, N) = ? (Given)
Array containing each separate input data array stored in a separate row.

VFLAG(N) = LOGICAL (Given)
A set of logical flags indicating which of the input data arrays stored in the DAT array have associated variance values (stored in the VAR array). If the VFLAG value for a data array is .TRUE., then variance values exist, otherwise there is no associated variance array so a variance of zero will be assumed.

VAR(EL, $*$) = ? (Given)
Array containing variance estimates for the input data arrays. These values are stored in the rows of this array in the same order as the associated data values, but the total number of rows may be smaller than N due to the absence of values for those data arrays with a VFLAG value of .FALSE.. The declared second dimension size of the VAR array must be at least equal to the number of .TRUE. values in the VFLAG array.

IMAP = INTEGER (Given)
A TRANSFORM system identifier for an {N–>1} compiled mapping which defines the mathematical expression to be evaluated.

QUICK = LOGICAL (Given)
If this argument is set to .TRUE., then output variance estimates will be made by perturbing each input data array in turn, but in the positive direction only. If it is set to .FALSE., then each input data array will be perturbed in both directions and the maximum resulting output perturbation will be used to estimate the output variance. The former approach has the advantage of speed, but the latter gives more accurate results, especially for highly non-linear functions.

MXBAT = INTEGER (Given)
A positive integer defining the size of workspace supplied. The input data arrays will be processed in " batches" , each of which does not contain more than this number of elements. If this value is too low, then excessive time will be spent in looping and subroutine calls. If it is too high, then excessive page faulting may occur. A value of about 256 is normally adequate.

WRK(MXBAT $*$ (N $+$ 3)) = ? (Returned)
This array must be supplied as workspace.

RES(EL) = ? (Returned)
The output data array, containing the results of evaluating the mathematical expression.

VRES(EL) = ? (Returned)
The output variance estimates to accompany the results held in the RES array.

BADDR = LOGICAL (Returned)
Whether the returned array of results (RES) may contain bad values.

BADVR = LOGICAL (Returned)
Whether the returned array of variance estimates (VRES) may contain bad values.

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

Notes:

- There is a routine for processing both real and double precision data; replace " x" in the routine name by R or D as appropriate. The data types of the DAT, VAR, WRK, RES and RESV arrays should match the routine used.
- The algorithm used to estimate the output variance is general-purpose and will cope with any reasonable mathematical expression. However, note that it is likely to be less efficient than an algorithm written especially to estimate the variance for any particular expression whose form is known in advance.

Timing :

The time taken is approximately proportional to (i) N+1, where N is the number of input data arrays which have associated variance values, (ii) the size of each array and (iii) the time taken to evaluate the mathematical expression for each array element. In general, this last value will also tend to increase in approximate proportion to the number of input data arrays.

KPG1_MULx
Multiplies two vectorised arrays with optional variance information

Description:

The routine forms the product of two vectorised arrays, with optional variance information. Bad value checking is also performed if required.

Invocation:

```
CALL KPG1_MULx( BAD, VAR, EL, A, VA, B, VB, C, VC, NERR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether it is necessary to check for bad values in the input arrays. The routine will execute more rapidly if this checking can be omitted.

VAR = LOGICAL (Given)
Whether associated variance information is to be processed.

EL = INTEGER (Given)
Number of array elements to process.

A(EL) = ? (Given)
First array of data to be multiplied.

VA(EL) = ? (Given)
Variance values associated with the array A. Not used if VAR is set to .FALSE..

B(EL) = ? (Given)
Second array of data to be multipled.

VB(EL) = ? (Given)
Variance values associated with the array B. Not used if VAR is set to .FALSE..

C(EL) = ? (Returned)
Result of multiplying arrays A and B.

VC(EL) = ? (Returned)
Variance values associated with the array C. Not used if VAR is set to .FALSE..

NERR = INTEGER (Returned)
Number of numerical errors that occurred during the calculations.

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

Notes:

- There is a routine for each of the data types integer, real and double precision: replace "`x`" in the routine name by I, R or D as appropriate. The arrays passed to this routine should all have the specified data type.
- This routine will handle numerical overflow. If overflow occurs, then affected output array elements will be set to the "`bad`" value. A count of the numerical errors which occur is returned via the NERR argument.

KPG1_MVBDx
Moves bad data values to the end of an index of array values

Description:

This routine modifies the supplied index array by removing indices of all bad data values and shuffling indices for good data values down to fill the gaps.

Invocation:

```
CALL KPG1_MVBDx( ELA, ARRAY, ELI, INDX, NGOOD, STATUS )
```

Arguments:

ELA = INTEGER (Given)
The size of the data array.

ARRAY(ELA) = ? (Given)
The array containing the data values.

ELI = INTEGER (Given)
The size of the index array.

INDX(ELI) = INTEGER (Given and Returned)
Each element of this array is an index into the ARRAY array. On exit, the contents of INDX are re-arranged so that indices that refer to bad data values are removed and indices that refer to good data values are shuffled down to fill the gaps. Any indices which are outside the bounds of the data array are treated as if they referred to bad data values (i.e. they are removed).

NGOOD = INTEGER (Returned)
The number of indices within INDX that refer to good data values. On exit, the first NGOOD elements of INDX will contain the indices of the good data values within ARRAY. Any remaining values of INDX will retain their original values.

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

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate.

KPG1_MXMEx
Returns the maximum and minimum values between thresholds of an array including its errors

Description:

This routine returns the maximum and minimum values of an input array when combined with its associated error array. The extreme values can be constrained to lie between two thresholds, where values outside the thresholds are ignored. So for example this routine might be used to find the smallest positive value or the largest negative value. The number of multiples of each error that should be added to and subtracted from the primary array to find the extreme values is adjustable. The routine also returns where it found the maximum and minimum, and the number of elements where either or both of the array values is bad.

An error report is made if all the values are bad or lie outside of the thresholds.

Invocation:

```
CALL KPG1_MXMEx( BAD, EL, ARRAY, ERROR, NSIGMA, THRESH, NINVAL, MAXMUM, MINMUM, MAXPOS,
MINPOS, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If .TRUE., there may be bad pixels present in the array. If .FALSE., it is safe not to check for bad values.

EL = INTEGER (Given)
The dimension of the input array.

ARRAY(EL) = ? (Given)
Input array of data.

ERROR(EL) = ? (Given)
Error array associated with the data array.

NSIGMA = REAL (Given)
Number of multiples of the error.

THRESH(2) = ? (Given)
The thresholds between which the extreme values are to be found (lower then upper). Values equal to the thresholds are excluded. To find the extreme values across the full range, set the thresholds to VAL__MINx and VAL__MAXx.

NINVAL = INTEGER (Returned)
Number of bad pixels in the array.

MAXMUM = ? (Returned)
Maximum value found in the array.

MINMUM = ? (Returned)
Minimum value found in the array.

MAXPOS = INTEGER (Returned)
Index of the pixel where the maximum value is (first) found.

MINPOS = INTEGER (Returned)
Index of the pixel where the minimum value is (first) found.

STATUS = INTEGER (Given)
Global status value

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The ARRAY, ERROR, THRESH, MAXMUM, and MINMUM arguments supplied to the routine must have the data type specified.

KPG1_MXMNx
Returns the maximum and minimum values of an array

Description:

This routine returns the maximum and minimum values of an input array, where it found the maximum and minimum, and the number of good and bad pixels in the array.

Invocation:

```
CALL KPG1_MXMNx( BAD, EL, ARRAY, NINVAL, MAXMUM, MINMUM, MAXPOS, MINPOS, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true there may be bad pixels present in the array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The dimension of the input array.

ARRAY(EL) = ? (Given)
Input array of data.

NINVAL = INTEGER (Returned)
Number of bad pixels in the array.

MAXMUM = ? (Returned)
Maximum value found in the array.

MINMUM = ? (Returned)
Minimum value found in the array.

MAXPOS = INTEGER (Returned)
Index of the pixel where the maximum value is (first) found.

MINPOS = INTEGER (Returned)
Index of the pixel where the minimum value is (first) found.

STATUS = INTEGER (Given)
Global status value

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The ARRAY, MAXMUM, and MINMUM arguments supplied to the routine must have the data type specified.

KPG1_MXMNX
Returns the maximum and minimum values of an array

Description:

This routine returns the maximum and minimum values of an input array, where it found the maximum and minimum, and the number of good and bad pixels in the array.

Invocation:

```
CALL KPG1_MXMNX( TYPE, BAD, EL, IPNTR, NINVAL, MAXMUM, MINMUM, MAXPOS, MINPOS, STATUS
)
```

Arguments:

TYPE = CHARACTER * (*) (Given)
The HDS data type of the array.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The dimension of the input array.

IPNTR = INTEGER (Given)
A pointer to the data array.

NINVAL = INTEGER (Returned)
Number of bad pixels in the array.

MAXMUM = DOUBLE PRECISION (Returned)
Maximum value found in the array.

MINMUM = DOUBLE PRECISION (Returned)
Minimum value found in the array.

MAXPOS = INTEGER (Returned)
Index of the pixel where the maximum value is (first) found.

MINPOS = INTEGER (Returned)
Index of the pixel where the minimum value is (first) found.

STATUS = INTEGER (Given)
Global status value

Notes:

- This function simply wraps up the generic KPG1_MXMN<T> routines.

KPG1_NACVT
Converts an HDS object hierarchy to native data representation

Description:

The routine recursively descends an HDS object hierarchy, converting any primitive objects within it to have the appropriate native data representation, as provided by the host machine. This will minimise subsequent access time for this machine.

Invocation:

```
CALL KPG1_NACVT( LOC, STATUS )
```

Arguments:

LOC = CHARACTER * (DAT__SZLOC) (Given and Returned)

Locator for the object or structure whose contents are to be converted. If the object is primitive, then this locator may be replaced by a new one on output (as the object may need to be erased and re-created).

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_NAG2R
Converts an NAG Hermitian Fourier transform array into an array usable by FFTPACK routine KPG1_RFFTB

Description:

This subroutine modifies the supplied array of Fourier co-efficients (as produced by NAG subroutine C06FAE) so that an inverse FFT can be performed on them using FFTPACK routine KPG1_RFFTB. The resulting inverse will have the same normalisation as the original data transformed using KPG1_RFFTF.

This function is equivalent to PDA_NAG2R except that it uses work space for greater speed.

Invocation:

```
CALL KPG1_NAG2R( NP, R, WORK )
```

Arguments:

NP = INTEGER (Given)
The size of the array.

R(NP) = REAL (Given and Returned)
The array holding the Fourier co-efficients. Supplied in NAG format and returned in FFTPACK format.

WORK(NP) = REAL (Given and Returned)
Work space.

Notes:

- A call to KPG1_R2NAG followed by a call to KPG1_NAG2R will result in the original data being divided by NP.

KPG1_NAGTC
Swaps argument order when getting a mapped character array from an HDS object

Description:

This is just a dummy routine to swap the argument order when obtaining the value of a mapped character array from an HDS object. It is needed for Unix systems.

Invocation:

```
CALL KPG1_NAGTC( CVALUE, LOC, NDIM, DIM, STATUS )
```

Arguments:

CVALUE = CHARACTER * (*) (Returned)
The character value.

LOC = CHARACTER * (DAT__SZLOC) (Given)
The locator of the object whose value is required.

NDIM = INTEGER (Given)
The number of dimensions.

DIM(*) = INTEGER (Given)
The dimensions of the character object.

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

KPG1_NAPTC
Swaps argument order when putting a mapped character array into an HDS object

Description:

This is just a dummy routine to swap the argument order when putting the value of a mapped character array into HDS object. It is needed for Unix systems.

Invocation:

```
CALL KPG1_NAPTC( CVALUE, LOC, NDIM, DIM, STATUS )
```

Arguments:

CVALUE = CHARACTER * (*) (Given)
The character value.

LOC = CHARACTER * (DAT__SZLOC) (Given)
The locator of the object to have value CVALUE.

NDIM = INTEGER (Given)
The number of dimensions.

DIM(*) = INTEGER (Given)
The dimensions of the character object.

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

KPG1_NBADx
Finds the number of bad values in an array

Description:
Finds the number of bad values in a one-diimensional array.

Invocation:

```
CALL KPG1_NBADx( N, DATA, NBAD, STATUS )
```

Arguments:

N = INTEGER (Given)
Number of elements in the array.

DATA(N) = ? (Given)
The data array.

NBAD = INTEGER (Returned)
The number of bad values in the data.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for the double-precision and real floating-point data types: replace " x" in the routine name by D or R as appropriate. The input data array must have the data type specified.

KPG1_NDFNM
Returns the name of an NDF without a directory path (Unix only)

Description:

Gets the full path to the supplied NDF, then removes any directory path from the start, and return the resulting string.

Note, Unix file names are assumed.

Invocation:

```
CALL KPG1_NDFNM( INDF, NAME, NMLEN, STATUS )
```

Arguments:

INDF = INTEGER (Given)
The NDF.

NAME = CHARACTER * (*) (Returned)
The NDF name without directory path.

NMLEN = INTEGER (Returned)
The used length of NAME.

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

KPG1_NMCOL
Finds the RGB intensities of a named colour

Description:

Given the name of a colour this routine searches the standard colour set for it, and if it exists returns its R-G-B intensities. An error is returned if the named colour is not in the colour set. All comparisons are performed in uppercase with the blanks removed.

Invocation:

```
CALL KPG1_NMCOL( NAME, R, G, B, STATUS )
```

Arguments:

NAME = CHARACTER * (*) (Given)

The name of the nearest colour in the named colour set to the input RGB colour. Note at least eighteen characters are required to avoid truncation. This string may also be an HTML code of the form " #aabbcc" (or " @aabbcc" - for use in contexts where " #" is a comment character, e.g. KAPPA style files) where a, b and c are hexadecimal digits, and " aa" , " bb" and " cc" give red, blue and green intensities normalised to a maximum of " ff" (256).

R = REAL (Returned)

The red intensity of the named colour to be identified. It is in the range 0.0 to 1.0.

G = REAL (Returned)

The green intensity of the named colour to be identified. It is in the range 0.0 to 1.0.

B = REAL (Returned)

The blue intensity of the named colour to be identified. It is in the range 0.0 to 1.0.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_NOISx
Adds random Normal noise to a one-dimensional array

Description:

This routine takes a one-dimensional array and adds noise randomly to each element in the array. The random number at each element is drawn from a Normal distribution whose spread is given by the corresponding variance array at that element.

Invocation:

```
CALL KPG1_NOISx( BAD, EL, VARNCE, ARRAY, STATUS )
```

Arguments:

BAD = INTEGER (Given)
If true there may be bad pixels in input arrays, and so there will be bad-pixel testing.

EL = INTEGER (Given)
The number of elements in the arrays.

VARNCE(EL) = ? (Given)
The Normal variance array corresponding to the data array.

ARRAY(EL) = ? (Given and Returned)
The data array to which random errors are to be applied.

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

Notes:

- There is a routine for all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW or W as appropriate. The arrays supplied to the routine must have the data type specified.
- All arithmetic is performed in double precision.

KPG_NTHMx
Returns the n smallest values in an array

Description:

This routine takes an array and returns a stack containing the n smallest values in that array.

Bad pixels are processed by the magic-value method.

Invocation:

```
CALL KPG1_NTHMx( BAD, ARRAY, DIMS, N, STACK, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
If true bad-pixel testing will be performed. It should be true if there may be bad pixels present.

ARRAY (DIMS) = ? (Given)
Input array of data values.

DIMS = INTEGER (Given)
Dimension of the input array.

N = INTEGER (Given)
The number of values to be stored in stack, i.e. specifies n in " n-th smallest value."

STACK (N) = ? (Returned)
Ordered n smallest values in the input array. The first element is the n-th smallest, the second is the (n-1)-th smallest and so on until the n-th element is the minimum value.

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The ARRAY and STACK arguments supplied must have the data type specified.

KPG1_NUMBx
Counts the number of elements with values or absolute values above or below a limit

Description:

This routine returns the number of points in the input array that have a value or absolute value greater than or less than the input value.

Invocation:

```
CALL KPG1_NUMBx( BAD, VABS, ABOVE, EL, INARR, VALUE, NUMBER, STATUS )
```

Arguments:

BAD = LOGICAL (Given)

If true testing for bad pixels will be made. This should not be set to false unless there definitely no bad values within the input array.

VABS = LOGICAL (Given)

If true the comparison is performed with the absolute value of each array element.

ABOVE = LOGICAL (Given)

If true the criterion tests for array values greater than the limit; if false the criterion tests for array values less than the limit.

EL = INTEGER (Given)

The dimension of the input array.

INARR(EL) = ? (Given)

The input data array

VALUE = ? (Given)

Value to test each array value against.

NUMBER = INTEGER (Returned)

The number of elements of the input array greater than the specified value when VABS = FALSE, or the number of elements of the input array whose absolute values are greater than the specified value when VABS = TRUE.

STATUS = INTEGER (Given)

Global status value.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array and comparison value supplied to the routine must have the data type specified.

KPG1_NUMFL
Counts the number of lines in a text file

Description:

This routine counts the number of non-comment, comment, and blank lines in a text file. Comment lines are those beginning, with any of the characters passed in the COMENT argument (normally ! and #). Here ' beginning' means the first non-blank character.

Invocation:

```
CALL KPG1_NUMFL( FD, COMENT, NLINES, NCOMS, NBLANK, STATUS )
```

Arguments:

FD = INTEGER (Given)
Fortran file identifier.

COMENT = CHARACTER * (*) (Given)
A comma-separated list of comment characters. If any line of the file begins with one of these, the line is treated as a comment line.

NLINES = INTEGER (Given)
The number of non-comment lines in the file.

NCOMS = INTEGER (Given)
The number of comment lines in the file.

NCOMS = INTEGER (Given)
The number of blank lines in the file.

STATUS = INTEGER ({status_access_mode})
The global status.

Notes:

- The file is rewound, but not closed on exit.

Prior Requirements :

- The Fortran text file must already be opened.

KPG1_OPGRD
Gets the parameters of an optimal projection for a given set of sky positions

Description:

This routine calculates the parameters of a tangent-plane projection that gives an optimal representation of a set of supplied sky positions. The projection parameters are the normal FITS CRPIX1/2, CRVAL1/2, CDELT1/2 and CROTA2 keywords. They are chosen in order to maximise the number of sky positions that fall close to the centre of a pixel.

Invocation:

```
CALL KPG1_OPGRD( NPOS, POS, WEST, PAR, RDIAM, STATUS )
```

Arguments:

NPOS = INTEGER (Given)

The number of sky positions.

POS(2, NPOS) = DOUBLE PRECISION (Given)

The sky positions. These should be (longitude,latitude) values, in radians, in any celestial co-ordinate system.

WEST = LOGICAL (Given)

If .TRUE., then it is assumed that the X grid axis increases westwards (assuming north is parallel to +ve Y). This is the case for most celestial co-ordinate systems such as (RA,Dec) etc. If .FALSE., then it is assumed that the X grid axis increases eastwards (assuming north is parallel to +ve Y). This is the case for a few systems such as (az,el) and geographic (longitude,latitude).

PAR(7) = DOUBLE PRECISION (Given and Returned)

The projection parameters. Each parameter that is supplied with a value of AST__BAD on entry will be replaced on exit with the optimal value. Non-bad supplied values will be left unchanged on exit. The supplied values will also be left unchanged if optimal values cannot be determined. They are stored in the order CRPIX1, CRPIX2, CRVAL1, CRVAL2, CDELT1, CDELT2, CROTA2. CRPIX1 and CRPIX2 are in units of pixels. All the other projection parameters will be in units of radians, and refer to the celestial co-ordinate system in which the POS values are supplied. CROTA2 is the angle from the Y axis to celestial north, measured north through east (no matter the value of WEST). Returned pixel sizes are rounded to the nearest tenth of an arcsecond.

RDIAM = DOUBLE PRECISION (Returned)

The diameter of the circle that just encloses all the supplied sky positions, in radians.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_ORVAR
Returns the variances and covariances of the order statistics from n to 1, assuming an initially normal distribution

Description:

The routine returns the variances and covariances of the order statistics, assuming an initial (pre-ordered) normal distribution of mean 0 and standard deviation 1. The routine returns all variance/covariances in an array with the terms vectorised - that is following on after each row. This uses the symmetric nature of the matrix to compress the data storage, but remember to double the covariance components if summing in quadrature. The variances

- covariances are returned for all statistics from n to 1. The special case of n = 1 returns the variance of 2/pi (median).

Invocation:

```
CALL KPG1_ORVAR( NSET, NBIG, PP, VEC, STATUS )
```

Arguments:

NSET = INTEGER (Given)
Number of members in ordered set.

NBIG = INTEGER (Given)
Maximum number of entries in covariance array row. equal to NSET∗(NSET+1)/2).

PP(NSET) = DOUBLE PRECISION (Given)
Workspace for storing expected values of order statistics.

VEC(NBIG, NSET) = DOUBLE PRECISION (Returned)
The upper triangles of the nset by nset variance-covariance matrix packed by columns. Each triangle is packed into a single row. For each row element Vij is stored in VEC(i+j∗(j-1)/2), for 1<=i<=j<=nset.

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

Notes:

- Data is returned as above to save on repeated calls (which are too slow). To get the actual variance of the data of order n you need to sum all the variances and twice the covariances and use these to modify the actual variance of the (unordered) data.
- It is assumed that NSET cannot be any larger than MXVAL.

KPG1_PACOL
Obtains a marker colour

Description:

This routine obtains a colour index. A string is obtained from the parameter system. The interpretation of this string provides a number of ways to specify the colour index requested. The options are as follows.

- 'MAX' – The maximum (non-reserved) colour index, i.e. the highest colour index used for the display of an image.
- 'MIN' – The minimum non-reserved colour index, i.e. the lowest colour index used for the display of an image.
- An integer – The actual colour index. It is constrained between 0 and the highest colour index.
- A named colour – Uses the named colour from the palette, and if it is not present, the nearest colour from the palette is selected.
- An HTML code – Has the form " #aabbcc" (or " @aabbcc" - for use in contexts where " #" is a comment character, e.g. KAPPA style files) where a, b and c are hexadecimal digits, and " aa" , " bb" and " cc" give red, blue, and green intensities normalised to a maximum of " ff" (256).

Invocation:

```
CALL KPG1_PACOL( PNCOL, LP, UP, COLIND, STATUS )
```

Arguments:

PNCOL = CHARACTER * (*) (Given)
The name of the ADAM parameter to obtain the marker colour. It should have type LITERAL.

LP = INTEGER (Given)
The lowest non-reserved colour index.

UP = INTEGER (Given)
The highest non-reserved colour index.

COLIND = INTEGER (Returned)
The colour index of the selected colour.

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

Prior Requirements :

- A PGPLOT image-display workstation must be open and active.

KPG1_PASTx
Pastes an array on to another

Description:

This routine 'pastes' an array on to another array. Values of the underlying array are replaced by values in the pasted array, except that bad values may also be made transparent, in other words the original base array appears through the bad values. Origin information is used to situate the paste operation.

Invocation:

```
CALL KPG1_PASTx( TRANSP, BAD, OFFSET, IDIMS, ELI, INARR, ODIMS, ELO, BASE, STATUS )
```

Arguments:

TRANSP = LOGICAL (Given)

Whether bad pixels are transparent or not. If true, bad pixels in the pasted array will not be pasted into the revised array. This is ignored if BAD is false.

BAD = LOGICAL (Given)

If true then there may be bad values in the pasted array. If false then there are definitely no bad values in the pasted array.

OFFSET(NDF__MXDIM) = INTEGER (Given)

The offset in each dimension of the pasted array's origin with respect to the origin of the base array.

IDIMS(NDF__MXDIM) = INTEGER (Given)

The dimensions of the array to be pasted. Unused dimensions up to NDF__MXDIM should be set to one.

ELI = INTEGER (Given)

The number of elements in the array to be pasted.

INARR(ELI) = ? (Given)

The array that will be pasted on to the base array.

ODIMS(NDF__MXDIM) = INTEGER (Given)

The dimensions of the base array. Unused dimensions up to NDF__MXDIM should be set to one.

ELO = INTEGER (Given)

The number of elements in the base array.

BASE(ELO) = ? (Given and Returned)

The base array on to which the input array will be pasted.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- This routine works in n-D, where n is 1 to 7. Even if the array has actually less dimensions there is negligible loss of efficiency to supply dummy (=1) higher dimensions.
- There is a routine for each numeric data type: replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The base and paste arrays supplied to the routine must have the data type specified.

KPG1_PGCLR
Clears current PGPLOT viewport

Description:

This routine clears the current PGPLOT viewport if possible. If it is not possible (egfor a printer) it does nothing.

Invocation:

```
CALL KPG1_PGCLR( STATUS )
```

Arguments:

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

Prior Requirements :

- A PGPLOT image-display workstation must be open and active.

KPG1_PGCLS
Closes down the AGI database and PGPLOT workstation

Description:

This routine closes the graphics data base and PGPLOT workstation previously opened by KPG1_PGOPN. It does nothing if PGPLOT is not currently active.

Invocation:

```
CALL KPG1_PGCLS( PNAME, SAVCUR, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)

The name of the parameter to use.

SAVCUR = LOGICAL (Given)

If .TRUE., then the current AGI picture is retained as the current picture. If .FALSE., the picture which was current when the database was opened is re-instated.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- This routine attempts to execute even if an error has already occurred (but the pallette will not be saved if an error has already occurred).

KPG1_PGCOL
Obtains a marker colour, given a colour specification

Description:

This routine obtains a PGPLOT colour index to be used in an image display from a supplied string. The interpretation of this string provides a number of ways to specify the colour index requested. The options are as follows.

- 'MAX' – The maximum (non-reserved) colour index, i.e. the highest colour index used for the display of an image.
- 'MIN' – The minimum non-reserved colour index, i.e. the lowest colour index used for the display of an image.
- An integer – The actual colour index. It is constrained between 0 and the highest colour index.
- A named colour – Uses the named colour from the palette, and if it is not present, the nearest colour from the palette is selected.
- An HTML code – Has the form " #aabbcc" (or " @aabbcc" - for use in contexts where " #" is a comment character, e.g. KAPPA style files) where a, b and c are hexadecimal digits, and " aa" , " bb" and " cc" give red, blue, and green intensities normalised to a maximum of " ff" (256).

An error is reported if the string does not conform to any of these formats.

Invocation:

```
CALL KPG1_PGCOL( COL, LP, UP, COLIND, STATUS )
```

Arguments:

COL = CHARACTER * (*) (Given)
The string specifying the required colour.

LP = INTEGER (Given)
The lowest non-reserved colour index.

UP = INTEGER (Given)
The highest non-reserved colour index.

COLIND = INTEGER (Returned)
The colour index of the selected colour.

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

Prior Requirements :

- A PGPLOT image-display workstation must be open and active.

KPG1_PGCUR
Uses the cursor to get a set of points

Description:

This routine uses the PGPLOT cursor to get a set of positions in the world co-ordinate system of the current PGPLOT window. If a position is given outside the box specified by X1, X2, Y1, Y2, then a warning is issued and the position is ignored. The allowable box can extend outside the PGPLOT viewport (extrapolated world co-ordinates are returned for positions outside the viewport).

The routine returns when any one of the following occurs:

1) The maximum number of positions have been given (see MAXPNT). 2) The right mouse button, " X" or " ." is pressed (but only if KEYS contains " X" or " ."). The cursor position is not returned. 3) The key/button specified by EXACT is pressed. The cursor position IS returned.

Invocation:

```
CALL KPG1_PGCUR( INFO, MESS, NACT, ACTDES, KEYS, X1, X2, Y1, Y2, EXACT, X0, Y0, MAXPNT,
RBMODE, LINE, BOX, MARK, IPLOT, X, Y, ACT, NPNT, STATUS )
```

Arguments:

INFO = LOGICAL (Given)
Display information describing the available actions before getting the first position?

MESS = CHARACTER * (*) (Given)
The purpose for using the cursor. Eg " select 2 points" . May be blank.

NACT = INTEGER (Given)
The number of available actions. Ignored if INFO is .FALSE.

ACTDES(NACT) = CHARACTER * (*) (Given)
Short descriptions of each action. Ignored if INFO is .FALSE. Examples: " select a point" , " exit" , " mark a star" . etc.

KEYS = CHARACTER * (*) (Given)
A string of NACT unique characters. These are the keyboard keys which must be pressed to select the corresponding action. Note, case is insignificant, but trailing spaces are significant. The left, middle and right mouse buttons are represented by the upper case characters A, D and X respectively (this is imposed by PGPLOT). A dot (" .") is considered equivalent to an X (i.e. the right mouse button). In addition, due to problems in the GKS version of PGPLOT, a space (" ") is considered equivalent to an A (i.e. left mouse button).

The " X" and " ." keys (or equivalently the right mouse button) are special in that (if they are included in KEYS) they cause the routine to exit without adding the cursor position to the list of returned positions. If KEYS includes neither " X" nor " ." , then presses of " X" , " ." or the right mouse button are ignored.

X1 = REAL (Given)
World co-ord X at lower-left corner of region in which positons may be entered.

X2 = REAL (Given)
World co-ord X at upper-right corner of region in which positons may be entered. If X1 and X2 are equal then no restrictions are placed on the region in which positions may be given.

Y1 = REAL (Given)
World co-ord Y at lower-left corner of region in which positons may be entered.

Y2 = REAL (Given)
World co-ord Y at upper-right corner of region in which positons may be entered. If Y1 and Y2 are equal then no restrictions are placed on the region in which positions may be given.

EXACT = REAL (Given)
The index of an exit action. If the corresponding key/button press is made, then the cursor position is added to the list of returned positions and the routine then exits. Zero can be supplied if this facility is not required. Note, if KEYS contains " X" or " ." then the routine also exits (WITHOUT adding the cursor position to the returned list) if " X" , " ." or the right mouse button is pressed.

X0 = REAL (Given)
The X world co-ordinate of the initial cursor position. Ignored if VAL__BADR.

Y0 = REAL (Given)
The Y world co-ordinate of the initial cursor position. Ignored if VAL__BADR.

MAXPNT = INTEGER (Given)
The maximum number of positions which can be given by the user before exiting.

RBMODE = INTEGER (Given)
The form of the rubber band which connects the cursor to the previous position. Rubber bands are not available when using the GKS version of PGPLOT: 0 - do not use a rubber band. 1 - use a straight-line rubber band. 2 - use a horizontal box rubber band.

LINE = INTEGER (Given)
Specifies lines to be drawn as follows:

- 1: Join adjacent points and do not close the polygon. 0: Do not draw any lines. 1: Join adjacent points and close the polygon. 2: Draw a vertical line between y1 and y2 (or the height of the window if y1=y2). 3: Draw a horizontal line between x1 and x2 (or the width of the window if x1=x2). The plotting attributes are specified by the CURVES(...) attributes of the supplied Plot (see IPLOT).

BOX = INTEGER (Given)
If non-zero then a horizontal box is drawn between each position. The plotting attributes are specified by the BORDER(...) attributes of the supplied Plot (see IPLOT).

MARK = INTEGER (Given)
If -31 or larger, then a marker is drawn at each position. The type of marker is given by the specific value (see PGPLOT routine PGPT). The plotting attributes are specified by the MARKERS(...) attributes of the supplied Plot (see IPLOT).

IPLOT = INTEGER (Given)
Defines the plotting styles for any graphics (see LINE, BOX and MARK). If AST__NULL is supplied, ther current PGPLOT attributes are used for all graphics.

X(MAXPNT) = REAL (Returned)
Elements 1 to NPNT hold the selected X positions.

Y(MAXPNT) = REAL (Returned)
Elements 1 to NPNT hold the selected X positions.

ACT(MAXPNT) = INTEGER (Returned)
Elements 1 to NPNT hold the indices of the actions for each selected point. In range 1 to NACT.

NPNT = INTEGER (Returned)
The number of positions given by the user (this includes the position at which the action specified by EXACT was pressed - if it was. It does not include the position at which " X" , " ." or the right mouse button was pressed).

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

KPG1_PGCUT
Cuts a section out of the current PGPLOT window

Description:

This routine sets the PGPLOT viewport so that it covers a specified section of the current PGPLOT window. The world co-ordinate bounds of the corresponding window are set to the supplied bounds.

Invocation:

```
CALL KPG1_PGCUT( X1, X2, Y1, Y2, STATUS )
```

Arguments:

X1 = REAL (Given)
The X world co-ordinate at the bottom left corner of the section of the viewport to be cut.

X2 = REAL (Given)
The X world co-ordinate at the top right corner of the section of the viewport to be cut.

Y1 = REAL (Given)
The Y world co-ordinate at the bottom left corner of the section of the viewport to be cut.

Y2 = REAL (Given)
The Y world co-ordinate at the top right corner of the section of the viewport to be cut.

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

KPG1_PGESC
Removes PGPLOT escape sequences from a text string

Description:

This routine removes PGPLOT escape sequences form a text string. Any " \" characters in the string are removed, together with the character following each " \" .

Invocation:

```
CALL KPG1_PGESC( TEXT, STATUS )
```

Arguments:

TEXT = CHARACTER * (*) (Given and Returned)
The text string.

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

KPG1_PGHNM
Returns an HDS name describing the current PGPLOT graphics device

Description:

This routine returns a string which can be used as an HDS component name. The string describes the current PGPLOT graphics device. It is intended for use in identifying resources related to the graphics device (such as palette and colour table), stored within HDS files.

For most devices, the returned name is simply the PGPLOT device type. For GWM windows, the returned string includes the name of the gwm window (so that each window can have separate resources).

Invocation:

```
CALL KPG1_PGHNM( NAME, STATUS )
```

Arguments:

NAME = CHARACTER * (DAT__SZNAM) (Returned)
The returned string.

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

Notes:

- A graphics device must previously have been opened using PGPLOT.

KPG1_PGLOC
Locates a component of an HDS structure relating to the currently opened PGPLOT device

Description:

LOC1 is an locator for an HDS structure which contains components relating to one or more PGPLOT devices. These components, for instance, may contain the colour palette or colour table to be used with the corresponding PGPLOT device. This routine searches the structure for a component relating to the currently opened PGPLOT device, and returns a locator for it if found. If not found, and if the currently opened PGPLOT device is a GWM window, a search is made for a component relating to a GWM window with a different name. If no such device is found, (or if an error occurs) DAT__NOLOC is returned.

For instance, if the currently opened graphics device is " x2windows" (i.e. " xwindows2/GWM"), a search is made first for a component called AGI_3801_2. If this is not found, a search is made for a component with a name corresponding to any /GWM device (e.g. AGI_3800_1 which corresponds to " xwindows/GWM" , or one of the other xwindows sevices).

The component names used are the same as the names uses for the device within the AGI database (e.g. " AGI_3801_2").

Invocation:

```
CALL KPG1_PGLOC( LOC1, LOC2, STATUS )
```

Arguments:

LOC1 = CHARACTER * (*) (Given)
A locator for the object to be searched.

LOC2 = CHARACTER * (*) (Returned)
A locator for the found component.

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

Notes:

- A PGPLOT device must previously have been opened using AGI.

KPG1_PGLUT
Uses an array of colour representations to set up the PGPLOT colour table

Description:

This routine stores new colour representations for a range of PGPLOT colour indices. A list of NCOL colour representations is supplied. These are normalized so that the highest value produces full colour intensity. The first colour (1) is assigned to PGPLOT colour index LO, and the last (NCOL) is assigned to PGPLOT colour index HI. The colour representations for the PGPLOT colour indices between LO and HI are formed by interpolation amonst the supplied NCOL colours, using either nearest-neighbour or linear interpolation.

Invocation:

```
CALL KPG1_PGLUT( NCOL, COLS, LO, HI, NN, STATUS )
```

Arguments:

NCOL = INTEGER (Given)
The number of colour indices in the supplied colour table.

COLS(3, NCOL) = REAL (Given)
The lookup table. The first dimension is RGB. Values should lie in the range 0.0–1.0.

LO = INTEGER (Given)
The lowest PGPLOT colour index to use.

HI = INTEGER (Given)
The highest PGPLOT colour index to use.

NN = LOGICAL (Given)
If true, and the number of input and output colour indices are different, the nearest-neighbour method is used for assigning values in the output array. Otherwise linear interpolation is used. Nearest-neighbour preserves sharp edges in the lookup; linear interpolation is recommended for smoothly varying lookup tables.

STATUS = INTEGER (Given)
Global status.

KPG1_PGOPN
Opens the AGI database and activate a PGPLOT workstation

Description:

This routine opens the graphics database and activates a PGPLOT workstation selected using the specified parameter. The user's palette and colour table is then re-instated, over-riding the those established by PGPLOT.

The device should normally be shut down using KPG1_PGCLS.

Invocation:

```
CALL KPG1_PGOPN( PNAME, MODE, IPIC, STATUS )
```

Arguments:

PNAME = CHARACTER * (*) (Given)
The name of the parameter to use.

MODE = CHARACTER * (*) (Given)
The AGI access mode; " WRITE" or " UPDATE" . Write causes the current picture to be cleared (the contents of the database are unaffected).

IPIC = INTEGER (Returned)
An AGI identifier for the current picture.

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

KPG1_PGPIX
Displays an image using PGPLOT

Description:

This routine uses PGPIXL to display an array of colour indices as a rectangular image. The area occupied by the array of colour indices is specified within a nominated Domain. Two opposite corners of this area are transformed into the Base Frame of the Plot (which should correspond to the current PGPLOT world co-ordinate sysytem), and the array of colour indices is drawn between these two transformed positions.

Invocation:

```
CALL KPG1_PGPIX( IPLOT, DOMAIN, LBND, UBND, NX, NY, COLI, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
A pointer to an AST Plot. The Base Frame in this Plot should correspond to the world co-ordinates in the current PGPLOT window.

DOMAIN = CHARACTER * (*) (Given)
The Domain in which the co-ordinates supplied in BOX are defined.

LBND(2) = REAL (Given)
The lower bounds of the area covered by the supplied array of colour indices, in the Domain given by DOMAIN.

UBND(2) = REAL (Given)
The upper bounds of the area covered by the supplied array of colour indices, in the Domain given by DOMAIN.

NX = INTEGER
Number of columns in the array of colour indices.

NY = INTEGER
Number of rows in the array of colour indices.

COLI(NX, NY) = INTEGER (Given)
The array of colour indices.

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

KPG1_PGSHT
Sets the PGPLOT character size in world co-ordinates

Description:

This routine sets the PGPLOT character size to a specified value in world co-ordinates. It mimics SGS_SHTX in so far as this is possible.

Note SGS and PGPLOT behave differently if the scales on the X and Y axes are not equal. SGS keeps the character size constant in world oordinates, so absolute character size will be different for vertical and horizontal text. On the other hand, PGPLOT keeps the absolute character size fixed, resulting in the characters size in world co-ordinates varying for horizontal and vertical text. This routine sets the size for horizontal text. If the axis scales are not equal, vertical text will have have a different size (in world co-ordinates).

Invocation:

```
CALL KPG1_PGSHT( HGT, STATUS )
```

Arguments:

HGT = REAL (Given)
The required character height, in world co-ordinates.

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

KPG1_PGSTY
Establishes values for graphics attributes

Description:

This routine establishes current PGPLOT attributes so that they correspond to the values for a named graphics element stored in the supplied Plot (see SUN/210). Only attributes which have values explicitly set in the Plot are changed. If no value has been set for a Plot attribute, the corresponding PGPLOT attribute is left unchanged.

If SET is supplied .TRUE., then the PGPLOT attributes for the specified graphics element are extracted from the supplied Plot and made active. The previously active values are returned in ATTRS. If SET is supplied .FALSE., the values supplied in ATTRS are made current (in this case ATTRS is returned unchanged).

Invocation:

```
CALL KPG1_PGSTY( IPLOT, ELEM, SET, ATTRS, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
An AST pointer to a Plot.

ELEM = CHARACTER * (*) (Given)
The name of an AST graphics element.

SET = LOGICAL (Given)
Should the Plot values be made curent? Otherwise the values in ATTRS are made current.

ATTS(*) = DOUBLE PRECISION (Given and Returned)
On entry, the attribute values to set if SET is .FALSE., These should have been obtained from a previous call to this routine. On exit, the attribute values current on entry to this routine are returned (unless SET is .FALSE. in which case the supplied values are returned unchanged). This array should have at least 5 elements.

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

KPG1_PGTXT
Draws text using PGPLOT and return concatenation point

Description:

This routine plots the supplied text at a given angle using PGPLOT, putting the bottom left corner of the text at the supplied position. The position at which another string must be drawn to concatenate it with the string just drawn is returned.

Invocation:

```
CALL KPG1_PGTXT( ANGLE, TEXT, X, Y, STATUS )
```

Arguments:

ANGLE = INTEGER (Given)

The angle, in degrees, that the baseline is to make with the horizontal, increasing anti-clockwise (0.0 is horizontal).

TEXT = CHARACTER * (*) (Given)

The text to draw. The returned values of X and Y leave room for any trailing spaces, UNLESS THE ENTIRE STRING IS BLANK, IN WHICH CASE X AND Y ARE RETURNED UNCHANGED.

X = REAL (Given and Returned)

The X position for the bottom-left corner of the string. On exit, it is the X position at the bottom of the a string to be concatenated to the one just drawn.

Y = REAL (Given and Returned)

The Y position for the bottom-left corner of the string. On exit, it is the Y position at the bottom of the a string to be concatenated to the one just drawn.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_PIXSC
Determines formatted pixel scales at a given grid position

Description:

This routine determines the scales of the WCS axes in the current Frame of the supplied FrameSet, and formated them into a string. For a specified WCS axis, the returned scale is the WCS axis increment produced by moving a distance of one grid pixel away from the supplied " AT" position, along the WCS axis.

Invocation:

```
CALL KPG1_PIXSC( IWCS, AT, PIXSC, VALUE, UNIT, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
The FrameSet.

AT($*$) = DOUBLE PRECISION (Given)
The position in GRID co-ordinates at which the pixel scales are to be determined. Note, the pixel scales may vary across the data array if the WCS Mappings are non-linear. The array should have one element for each GRID axis.

PIXSC($*$) = DOUBLE PRECISION (Returned)
The returned pixel scales. Note, the pixel scale for both celestial longitude and latitude axes are returned as an arc-distance in radians. The array should have one element for each WCS axis.

VALUE($*$) = CHARACTER($*$) (Returned)
The formatted pixel scales. Celestial axes are formatted as arc-seconds using a " G15.6" format. Time values are also formatted using G15.6 (the Format attribute in the current WCS Frame is ignored, since it may produce a calendar date), in what ever units are indicated in the current Frame. Other types of axes (including spectral axes) are formatted using the axis Format attribute in the current WCS Frame. The array should have one element for each WCS axis. Each element of the array should be at least 15 characters long. The returned text is left justified.

UNIT($*$) = CHARACTER($*$) (Returned)
Units strings that describe the values returned in VALUES. The array should have one element for each WCS axis.

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

KPG1_PL2GE
Reads two-dimensional polynomial information from a POLYNOMIAL structure

Description:

This routine reads information describing a two-dimensional polynomial surface from a standard Starlink POLYNOMIAL structure, as defined in SGP/38. All floating-point information within the structure is returned as DOUBLE PRECISION.

It is assumed the calling programme needs a one-dimensional array of coefficients, where COEFF((IX-1)∗NYPAR + IY) contains the coefficient for the (IX,IY)th term (with NYPAR being the total number of Y terms). Such a one-dimensional array is used by the NAG routines and defined in the NAG manual (see Chapter E02 on " Curve and Surface fitting").

This routine will convert read the two-dimensional coefficient array from the POLYNOMIAL structure (in the format described in SGP/38) and load a flipped version of this into the required one-dimensional array. If there is a variance array present, the VARPRE flag is set .TRUE., and the variances returned in VARIAN.

The routine will also read the TMIN and TMAX arrays from the structure and return XMIN, XMAX, YMIN and YMAX. Note that these items are compulsory when VARNT=' CHEBYSHEV' but optional when VARNT=' SIMPLE' . In the latter case an attempt will be made to read TMIN and TMAX from the structure, but their absence will not be regarded as an error. The returned parameter LIMITS will indicate if these items have been read successfully.

Invocation:

```
CALL KPG1_PL2GE( LOC, MXPAR, VARNT, NXPAR, NYPAR, LIMITS, XMIN, XMAX, YMIN, YMAX, COEFF,
VARPRE, VARIAN, WORK, STATUS )
```

Arguments:

LOC = CHARACTER∗(DAT__SZLOC) (Given)
Locator to the existing POLYNOMIAL structure.

MXPAR = INTEGER (Given)
The maximum number of parameters in either x or y. The declared size of the coefficient arrays given to this routine is assumed to be MXPAR ∗ MXPAR (see below).

VARNT = CHARACTER∗(∗) (Returned)
Variant of the polynomial (' CHEBYSHEV' or ' SIMPLE'). This item should be at least CHARACTER∗9.

NXPAR = INTEGER (Returned)
Number of x parameters (= order of polynomial in x direction + 1).

NYPAR = INTEGER (Returned)
Number of y parameters (= order of polynomial in Y direction + 1).

LIMITS = LOGICAL (Returned)
When VARNT=' SIMPLE' this logical flag indicates whether any TMIN and TMAX limits have been read from the polynomial structure and returned in the next four arguments (in which case LIMITS will be .TRUE.).

XMIN = DOUBLE PRECISION (Returned)
Minimum value along x axis for which polynomial is valid.

XMAX = DOUBLE PRECISION (Returned)
Maximum value along x axis for which polynomial is valid.

YMIN = DOUBLE PRECISION (Returned)
Minimum value along y axis for which polynomial is valid.

YMAX = DOUBLE PRECISION (Returned)
Maximum value along y axis for which polynomial is valid.

COEFF(MXPAR $*$ MXPAR) = DOUBLE PRECISION (Returned)
Array of polynomial coefficients, in the format used by NAG routines (see KPS1_FSPF2).

VARPRE = LOGICAL (Returned)
Whether or not there are coefficient variances present in the POLYNOMIAL structure.

VARIAN(MXPAR $*$ MXPAR) = DOUBLE PRECISION (Returned)
Array of polynomial coefficient variances, in the format used by NAG routines. The values are only assigned when VARPRE is .TRUE..

WORK(MXPAR, MXPAR) = DOUBLE PRECISION (Returned)
Work array containing the two-dimensional array of coefficients as read from the POLYNOMIAL structure.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- Uses the magic-value method for bad or undefined pixels.

KPG1_PL2PU
Writes two-dimensional polynomial information to a POLYNOMIAL structure

Description:

This routine writes information describing a two-dimensional polynomial surface to a standard Starlink POLYNOMIAL structure, as defined in SGP/38. An empty POLYNOMIAL structure should already have been created. All floating point components within the structure are written as DOUBLE PRECISION.

It is assumed the calling programme has a one-dimensional array of coefficients, where COEFF((IX - 1) * NYPAR + IY) contains the coefficient for the (IX,IY)th term (with NYPAR being the total number of Y terms). Such a one-dimensional array is used by the NAG routines and defined in the NAG manual (see Chapter E02 on " Curve and Surface fitting").

This routine will convert the coefficient array to two-dimensional, flip it around and store it in the POLYNOMIAL structure so that DATA_ARRAY(IX,IY) contains the coefficient for the (IX,IY)th term (see SGP/38).

The routine will also load the TMIN and TMAX arrays with XMIN, XMAX, YMIN and YMAX. Note that these are compulsory when VARNT=' CHEBYSHEV' but optional when VARNT=' SIMPLE' . In the latter case the logical parameter LIMITS will be used to decide whether to write the limits. All the components have type _DOUBLE.

Invocation:

```
CALL KPG1_PL2PU( LOC, VARNT, NXPAR, NYPAR, LIMITS, XMIN, XMAX, YMIN, YMAX, COEFF, VARIAN,
WORK, STATUS )
```

Arguments:

LOC = CHARACTER * (DAT__SZLOC) (Given)
Locator to existing but empty POLYNOMIAL structure.

VARNT = CHARACTER * (*) (Given)
Variant of the polynomial (' CHEBYSHEV' or ' SIMPLE'). (This is written but not checked).

NXPAR = INTEGER (Given)
Number of x parameters (= order of polynomial in x direction + 1)

NYPAR = INTEGER (Given)
Number of y parameters (= order of polynomial in y direction + 1)

LIMITS = LOGICAL (Given)
When VARNT=' SIMPLE' this logical flag may be used to control whether the TMIN and TMAX limits are written to the polynomial structure (based on the next 4 arguments). Setting LIMITS=.TRUE. will cause the limits to be written. This parameter is ignored when VARNT=' CHEBYSHEV' , as the limits then are compulsory.

XMIN = DOUBLE PRECISION (Given)
Minimum value along x axis for which polynomial is valid.

XMAX = DOUBLE PRECISION (Given)
Maximum value along x axis for which polynomial is valid.

YMIN = DOUBLE PRECISION (Given)
Minimum value along y axis for which polynomial is valid.

YMAX = DOUBLE PRECISION (Given)
Maximum value along y axis for which polynomial is valid.

COEFF(NXPAR ∗ NYPAR) = DOUBLE PRECISION (Given)
Array of polynomial coefficients, in the format used by NAG routines.

VARIAN(NXPAR ∗ NYPAR) = DOUBLE PRECISION (Given)
Array of variances of polynomial coefficients, in the format used by NAG routines.

WORK(NXPAR, NYPAR) = DOUBLE PRECISION (Returned)
Work array used to flip the polynomial coefficients. On exit it will contain the two-dimensional array of coefficients written to the POLYNOMIAL structure.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- Uses the magic-value method for bad or undefined pixels.

KPG1_PLCIP
Finds the nearest colour in the palette to a named colour (PGPLOT version of KPG1_PALCI)

Description:

This routine finds the PGPLOT colour index within the palette of a named colour. The required colour must be in the standard colour set, and SAI__ERROR status is returned if it is not. If the named colour is not present the index of the colour nearest to the requested colour is returned. A city-block metric is used.

A close colour is only accepted if the close colour is not equal to the background colour. This avoids pens becoming invisible.

Invocation:

```
CALL KPG1_PLCIP( COLOUR, COLIND, STATUS )
```

Arguments:

COLOUR = CHARACTER * (*) (Given)

The name of the colour whose colour index is required. Note at least eighteen characters are required to avoid truncation. The name may be abbreviated. If there is any ambiguity, the first match (in alphabetical order) is selected.

COLIND = INTEGER (Returned)

The colour index within the palette of the colour or its nearest equivalent.

STATUS = INTEGER (Given and Returned)

The global status.

Prior Requirements :

- A PGPLOT image-display workstation must be open and active.

KPG1_PLGET
Gets the colour palette for the currently open graphics device from a supplied array

Description:

This routine gets the colour palette from a supplied array and loads it into the colour table of the currently open graphics device.

Invocation:

```
CALL KPG1_PLGET( CI1, CI2, ARRAY, STATUS )
```

Arguments:

CI1 = INTEGER (Given)
The lowest colour index to change.

CI2 = INTEGER (Given)
The highest colour index to change.

ARRAY(3, 0 : CI2) = REAL (Given)
The array containing the colour palette to load.

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

Notes:

- A graphics device must previously have been opened using PGPLOT.

KPG1_PLLOD
Loads the colour palette for the currently open graphics device

Description:

This routine loads the colour palette for the currently open graphics device from an HDS container file in the users ADAM directory. The file is called " kappa.palette.sdf" and contains a palette for different devices. The file should have been created by KPG1_PLSAV. If the file does not exist, the current colour table is left unchanged.

Each palette in the file is a _REAL array of shape (3,n) where n is the number of colours in the palette. The first colour (Index 1 in the array) is the background colour and is usually refered to as colour index zero. Therefore the highest colour index in the array is (n-1). Each array has a name which identifies the graphics device to which it refers. Each array has a name which identifies the graphics device to which it refers.

Invocation:

```
CALL KPG1_PLLOD( STATUS )
```

Arguments:

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

Notes:

- A graphics device must previously have been opened using PGPLOT.

KPG1_PLOTA
Opens the graphics device and see if there is an existing DATA picture with which the new DATA picture could be aligned

Description:

This routine opens a graphics device, clearing it or not as specified by the user. If it is not cleared an attempt is made to find a DATA picture within the current picture. If found it becomes the current AGI picture, the corresponding PGPLOT viewport is established as the current PGPLOT viewport, and an AST Plot is returned for it in which the Base (GRAPHICS) Frame corresponds to PGPLOT world co-ordinates. If no DATA picture is found, or if the device was not cleared on opening, the current AGI picture and PGPLOT viewport are unchanged on exit and no Plot is returned.

Various environment parameters are used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between applications.

Invocation:

```
CALL KPG1_PLOTA( IWCS, STAT, DOMAIN, IPIC0, IPICD, IPLOT, STATUS )
```

Arguments:

IWCS = INTEGER (Given)

An AST pointer to a FrameSet. This may be AST__NULL. The Current and Base Frames are unchanged on exit. If an existing DATA picture was created by a non-AST application it will not have a Plot stored with it. A default Plot will be created in this case, containing two Frames; a GRAPHICS Frame corresponding to millimetres from the bottom-left corner of the view surface, and a Frame corresponding to AGI world co-ordinates. The AGI database does not contain any information describing world co-ordinates and so such information must be supplied by the calling application, on some assumption such as " AGI world co-ordinates are PIXEL co-ordinates" . This information is provided through a FrameSet (IWCS) and a Domain name (DOMAIN). DOMAIN specifies the Domain in which AGI world co-ordinates are assumed to live. A FrameSet may then be supplied using argument IWCS containing a Frame with the same Domain which will be used to describe AGI world co-ordinates in the returned Plot (a default two-dimensional Frame with the specified Domain is used if no FrameSet is supplied or if it does not contain a Frame with the specified Domain). If DOMAIN is supplied Blank, a default two-dimensional Frame with Domain AGI_WORLD will be used to describe AGI world co-ordinates.

If the existing DATA picture was created by an AST application, it will have a Plot stored with it which means that AGI world co-ordinates will be ignored. Consequently, the values supplied for IWCS and DOMAIN will also be ignored.

STAT = CHARACTER * (*) (Given)

Determines whether or not the new DATA picture creted by the calling application is to be aligned with an existing DATA picture.

- " NEW" – no attempt is made to align the new DATA picture with an existing DATA picture, even if the CLEAR parameter is given a FALSE value.
- " OLD" – the new DATA picture is always aligned with an existing DATA picture. The CLEAR parameter is not accessed (it is assumed to have a FALSE value) and an error is reported if no DATA picture is available.
- " UNKNOWN" – If CLEAR is given a FALSE value then the new DATA picture is aligned with any existing DATA picture, but no error is reported if no DATA picture exists.

DOMAIN = CHARACTER * (*) (Given)
The Domain for AGI world co-ordinates. Only used if a FrameSet is supplied (IWCS). If a blank value is supplied then " AGI_WORLD" will be used. See description of argument IWCS above for more details.

IPIC0 = INTEGER (Returned)
An AGI identifier for the original current picture.

IPICD = INTEGER (Returned)
An AGI identifier for the existing DATA picture. Returned equal to -1 if there is no existing DATA picture or if the device was cleared on opening.

IPLOT = INTEGER (Returned)
An AST pointer to a Plot associated with an existing DATA picture. Returned equal to AST__NULL if an error occurs, or if there is no existing DATA picture, or if the device was cleared on opening.

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

Notes:

- Close down AGI and PGPLOT using KPG1_PGCLS.

Environment Parameters

CLEAR = _LOGICAL (Read)
TRUE if the graphics device is to be cleared on opening. See argument STAT.

DEVICE = DEVICE (Read)
The plotting device.

KPG1_PLOT
Prepares for graphics output

Description:

This routine opens a graphics device and prepares for graphical output using PGPLOT within a new DATA picture. Optional ancillary pictures around the DATA picture may also be created. A FRAME picture enclosing the DATA picture and any ancillary pictures is created if any ancillary pictures or non-zero margins were requested.

An AST Plot is returned which allows plotting within the DATA picture. The Base (GRAPHICS) Frame in this Plot corresponds to millimetres from the botttom left corner of the view surface. The Current Frame in the Plot is inherited from the supplied FrameSet (IWCS).

If there is an existing DATA picture on the device, then the new DATA picture can optionally be aligned with the existing DATA picture. In this case, the returned Plot is formed by adding any supplied FrameSet (IWCS) into the Plot stored with the existing DATA picture in the AGI database (a default Plot is used if the database does not contain a Plot). The FrameSet is added into the Plot by aligning them in the Current Frame of the FrameSet if possible. If this is not possible, they are aligned in the world co-ordinate system of the picture (stored in the AGI database). The Domain of the AGI world co-ordinate system is not stored in the database and must be supplied by the calling application using argument DOMAIN (this will usually be " PIXEL"). If alignment is not possible in AGI world co-ordinates, then they are aligned in the GRID domain. If this is also not possible, they are aligned in any suitable Frame.

On exit, the current PGPLOT viewport corresponds to area occupied by the new DATA picture. The bounds of the PGPLOT window produce a world co-ordinate system within the viewport corresponding to the Base Frame in the returned Plot (i.e. millimetres from the bottom-left corner of the view surface - NOT pixel co-ordinates). Note, this is different from the world co-ordinate system stored in the AGI database with the new DATA picture.

If the returned Plot contains an AXIS Frame in which the axes are scaled and shifted versions of the axes of the AGI world co-ordinate Frame (specified by argument DOMAIN), then a TRANSFORM structure is stored with the new DATA picture that defines AGI Data co-ordinates. This is purely for the benefit of non-AST based applications which may use AGI Data co-ordinates (AST-based applications should always use the Plot stored with the picture in preference to the TRANSFORM structure stored in the AGI database).

Various environment parameters are used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between applications.

Invocation:

```
CALL KPG1_PLOT( IWCS, STAT, APP, DATREF, MARGIN, NP, PNAME, PSIDE, PSIZE, ASPECT, DOMAIN,
BOX, IPICD, IPICF, IPIC, IPLOT, NFRM, ALIGN, STATUS )
```

Arguments:

IWCS = INTEGER (Given)

An AST pointer to a FrameSet. This may be AST__NULL. The Current and Base Frames are unchanged on exit.

STAT = CHARACTER * (*) (Given)

Determines whether or not the new DATA picture should be aligned with an existing DATA picture.

- " NEW" – no attempt is made to align the new DATA picture with an existing DATA picture, even if the CLEAR parameter is given a FALSE value.

- " OLD" – the new DATA picture is always aligned with an existing DATA picture. The CLEAR parameter is not accessed (it is assumed to have a FALSE value) and an error is reported if no DATA picture is available.
- " UNKNOWN" – If CLEAR is given a FALSE value then the new DATA picture is aligned with any existing DATA picture, but no error is reported if no DATA picture exists.

APP = CHARACTER * (*) (Given)
The name of the calling application, in the form <package>_<application> (e.g. " KAPPA_DISPLAY" , " POLPACK_POLPLOT" , etc.). The supplied string is stored as a comment with all new AGI pictures.

DATREF = CHARACTER * (*) (Given)
A string describing the source of the data being displayed, which will be stored in the AGI database with the new DATA picture. It is ignore if blank.

MARGIN(4) = REAL (Given)
The width of the borders to leave around the new DATA picture, given as fractions of the corresponding dimension of the DATA picture. These should be supplied in the order bottom, right, top, left.

NP = INTEGER (Given)
The number of extra pictures to be included in the FRAME pictures (the DATA picture itself is not included in this list). Margins are left around the DATA picture with widths given by MARGIN. Any extra pictures are placed outside these margins, in positions described by PSIDE and PSIZE.

PNAME(NP) = CHARACTER * (*) (Given)
The names to store in the AGI database with the NP extra pictures.

PSIDE(NP) = CHARACTER * 1 (Given)
Each element of this array should be one of L, R, T or B. It indicates which side of the FRAME picture an extra picture is to be placed. For Left and Right, the extra picture occupies the full height of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the far Left or Right of all previously created pictures. For Top or Bottom, the extra picture occupies the full width of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the top or bottom of all previously created pictures. Ignored if NP is zero.

PSIZE(NP) = REAL (Given)
The size of each extra picture. For Left and Right pictures, this is the width of the picture, and the value is given as a fraction of the width of the DATA picture. For Top and Bottom pictures, it is the height of the picture, and it is given as a fraction of the height of the DATA picture. Ignored if NP is zero.

ASPECT = REAL (Given)
The aspect ratio for the DATA picture. This is the height of the DATA picture (in millimetres) divided by the width of the DATA picture (also in millimetres). The new DATA picture is created with this aspect ratio unless the FILL parameter is given a TRUE value, in which case the aspect ratio is adjusted to get the largest DATA picture that can be created within the current picture. If a value of zero is supplied, then the largest DATA picture is used irrespective of FILL (which is then not accessed).

DOMAIN = CHARACTER * (*) (Given)
The Domain name corresponding to the AGI world co-ordinates. If a blank value is supplied then " AGI_WORLD" will be used.

BOX(4) = DOUBLE PRECISION (Given)
The co-ordinates to be assigned to the bottom-left, and top-right corners of the DATA picture in the AGI database (the co-ordinate system in defined by argument DOMAIN). Only used if the new DATA picture is NOT being aligned with an existing DATA picture. Supplied in the order XLEFT, YBOTTOM, XRIGHT, YTOP. Note, the supplied bounds are stored in the AGI database, but do not

effect the PGPLOT window on exit, which always has a world co-ordinate system of millimetres from the bottom-left corner of the view surface. If the supplied box has zero area, then world co-ordinates for the DATA picture in the AGI database will be centimetres from the bottom-left corner of the DATA picture.

IPICD = INTEGER (Returned)
An AGI identifier for the new DATA picture.

IPICF = INTEGER (Returned)
An AGI identifier for the new FRAME picture. World co-ordinate system is inherited from the current picture on entry. If no FRAME picture is created then an identifier for the current picture on entry is returned.

IPIC(NP) = INTEGER (Returned)
An array of AGI identifiers corresponding to the extra pictures requested in PSIDE and PSIZE. The world co-ordinate system for each picture is inherited from the FRAME picture. The actual size of a picture may be less than the requested size if there is insufficient room left in the FRAME picture to give it its requested size. Identifiers for pictures which would have zero size (i.e. fall completely outside the FRAME picture) are returned equal to -1, but no error is reported.

IPLOT = INTEGER (Returned)
An AST pointer to the new Plot. Returned equal to AST__NULL if an error occurs.

NFRM = INTEGER (Returned)
A Frame with index I in the supplied FrameSet (IWCS) will have index (I + NFRM) in the returned Plot (IPLOT). Returned equal to zero if an error occurs.

ALIGN = LOGICAL (Returned)
Was the new DATA picture aligned with an existing DATA picture?

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

Notes:

- Picture identifiers are returned equal to -1 if an error occurs, or if any pictures cannot be created.
- Close down AGI and PGPLOT using: CALL KPG1_PGCLS('DEVICE', .FALSE., STATUS)

Environment Parameters

CLEAR = _LOGICAL (Read)
TRUE if the graphics device is to be cleared on opening. See argument STAT.

DEVICE = DEVICE (Read)
The plotting device.

FILL = _LOGICAL (Read)
TRUE if the supplied aspect ratio is to be ignored, creating the largest possible DATA picture within the current picture. When FILL is FALSE, the DATA picture is created with the supplied aspect ratio. Only used when creating a new DATA picture.

STYLE = GROUP (Read)
A description of the plotting style required. This the name of a text file containing an AST attribute setting on each line, of the form " name=value" , where " name" is an AST Plot attribute name (or synonym recognised by this application–see KPG1_ASPSY), and " value" is the value to assign to the attribute.

KPG1_PLOTN
Creates a new DATA picture and ensure there is an AST Plot for it

Description:

This routine createa an AST Plot describing a given DATA picture. If a FrameSet is supplied (IWCS), then it is merged with the Plot. An error is reported if the FrameSet and Plot cannot be aligned.

On exit, the current PGPLOT viewport corresponds to area occupied by the DATA picture. The bounds of the PGPLOT window produce a world co-ordinate system within the viewport corresponding to the Base Frame in the returned Plot (i.e. millimetres from the bottom-left corner of the view surface–NOT pixel co-ordinates). Note, this is different to the world co-ordinate system stored in the AGI database with the DATA picture.

Invocation:

```
CALL KPG1_PLOTN( IWCS, DOMAIN, IPICD, IPLOT, NFRM, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
An AST pointer to a FrameSet. This may be AST__NULL. The Current and Base Frames are unchanged on exit.

DOMAIN = CHARACTER * (*) (Given)
The Domain in which AGI world co-ordinates within the specified DATA picture live. Alignment of the Plot and FrameSet will be attampted in this Domain if it is not possible in the Current Frame of the FrameSet.

IPICD = INTEGER (Given)
An AGI identifier for the DATA picture.

QUIET = LOGICAL (Given)
Suppress message identifying the Domain in which alignment occurs?

IPLOT = INTEGER (Returned)
An AST pointer to the new Plot. Returned equal to AST__NULL if an error occurs.

NFRM = INTEGER (Returned)
A Frame with index I in the supplied FrameSet (IWCS) will have index (I + NFRM) in the returned Plot (IPLOT). Returned equal to zero if an error occurs.

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

KPG1_PLOTP
Creates a new DATA picture, with optionally ancillary pictures

Description:

This routine createa a new DATA picture, together with optional ancillary pictures around the DATA picture. The new DATA picture can be aligned with an existing DATA picture. A FRAME picture enclosing the DATA picture and any ancillary pictures is created if any ancillary pictures or non-zero margins were requested.

On exit, the current PGPLOT viewport corresponds to area occupied by the new DATA picture. The bounds of the PGPLOT window produce a world co-ordinate system within the viewport corresponding to millimetres from the bottom-left corner of the view surface. Note, this is different to the world co-ordinate system stored in the AGI database with the new DATA picture.

Various environment parameters are used to obtain options, etc. The names of these parameters are hard-wired into this subroutine in order to ensure conformity between applications.

Invocation:

```
CALL KPG1_PLOTP( IPICD, APP, MARGIN, NP, PNAME, PSIDE, PSIZE, ASPECT, BOX, IPICD0, IPICF,
IPIC, STATUS )
```

Arguments:

IPICD = INTEGER (Given)

The AGI identifier for an existing DATA picture. If this is supplied equal to -1, the size and extent of the new DATA picture are determined by BOX. Otherwise, the size and extent of the new DATA picture are set equal to the existing DATA picture.

APP = CHARACTER * (*) (Given)

The name of the calling application in the form <package>_<application> (eg" KAPPA_DISPLAY").

MARGIN(4) = REAL (Given)

The width of the borders to leave round the new DATA picture, given as fractions of the corresponding dimension of the DATA picture. These should be supplied in the order bottom, right, top, left.

NP = INTEGER (Given)

The number of extra pictures to be included in the FRAME pictures (the DATA picture itself is not included in this list). Margins are left round the DATA picture with widths given by MARGIN. Any extra pictures are placed outside these margins, in positions described by PSIDE and PSIZE.

PNAME(NP) = CHARACTER * (*) (Given)

The names to store in the AGI database with the NP extra pictures.

PSIDE(NP) = CHARACTER * 1 (Given)

Each element of this array should be one of L, R, T or B. It indicates which side of the FRAME picture an extra picture is to be placed. For Left and Right, the extra picture occupies the full height of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the far Left or Right of all previously created pictures. For Top or Bottom, the extra picture occupies the full width of the DATA picture, margins, and any previously created extra pictures. The picture is placed at the top or bottom of all previously created pictures. Ignored if NP is zero.

PSIZE(NP) = REAL (Given)

The size of each extra picture. For Left and Right pictures, this is the width of the picture, and the value is given as a fraction of the width of the DATA picture. For Top and Bottom pictures, it is the

height of the picture, and it is given as a fraction of the height of the DATA picture. Ignored if NP is zero.

ASPECT = REAL (Given)
The aspect ratio for the DATA picture. This is the height of the DATA picture (in millimetres) divided by the width of the DATA picture (also in millimetres). The new DATA picture is created with this aspect ratio unless the FILL parameter is given a TRUE value, in which case the aspect ratio is adjusted to get the largest DATA picture which can be created within the current picture. If a value of zero is supplied, then the largest DATA picture is used irrespective of FILL (which is then not accessed).

BOX(4) = DOUBLE PRECISION (Given)
The co-ordinates to be assigned to the bottom-left, and top-right corners of the DATA picture in the AGI database (the co-ordinate system in defined by argument DOMAIN). Only used if the new DATA picture is NOT being aligned with an existing DATA picture. Supplied in the order XLEFT, YBOTTOM, XRIGHT, YTOP. Note, the supplied bounds are stored in the AGI database, but do not effect the PGPLOT window on exit, which always has a world co-ordinate system of millimetres from the bottom-left corner of the view surface. If the supplied box has zero area, then world co-ordinates for the DATA picture in the AGI database will be centimetres from the bottom-left corner of the DATA picture.

IPICD0 = INTEGER (Returned)
An AGI identifier for the new DATA picture.

IPICF = INTEGER (Returned)
An AGI identifier for the new FRAME picture. World co-ordinate system is inherited from the current picture on entry. If no FRAME picture is created then an identifier for the current picture on entry is returned.

IPIC(NP) = INTEGER (Returned)
An array of AGI identifiers corresponding to the extra pictures requested in ZSIDE and PSIZE. The world co-ordinate system for each picture is inherited from the FRAME picture. The actual size of a picture may be less than the requested size if there is insufficient room left in the FRAME picture to give it its requested size. Identifiers for pictures which would have zero size (i.e. fall completely outside the FRAME picture) are returned equal to -1, but no error is reported.

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

Notes:

- Picture identifiers are returned equal to -1 if the picture cannot be created (egdue to lack of room within the current picture).

Environment Parameters

FILL = _LOGICAL (Read)
TRUE if the supplied aspect ratio is to be ignored, creating the largest possible DATA picture within the current picture. When FILL is FALSE, the DATA picture is created with the supplied aspect ratio. Only used when creating a new DATA picture.

KPG1_PLOTS
Saves an AST Plot with an AGI DATA picture

Description:

This routine saves a specified Plot and data reference in the AGI database with a specified DATA picture. A specified Frame can be made current before saving the Plot.

If the supplied Plot contains a " AGI Data" Frame with the Domain given by DDOM in which the axes are scaled and shifted versions of the axes of the AGI world co-ordinate Frame (specified by argument WDOM), then a TRANSFORM structure defining AGI Data co-ordinates is stored with the DATA picture. This is purely for the benefit of non-AST based applications which may use AGI Data co-ordinates (AST-based applications should always use the Plot stored with the picture in preference to the TRANSFORM structure stored in the AGI database).

Invocation:

```
CALL KPG1_PLOTS( IPLOT, IPICD, DATREF, ICURR, WDOM, DDOM, STATUS )
```

Arguments:

IPLOT = INTEGER (Given)
An AST pointer to the Plot. The current Frame is unchanged on exit (even if a Frame is specified using ICURR).

IPICD = INTEGER (Given)
An AGI identifier for the DATA picture.

DATREF = CHARACTER * (*) (Given)
A data reference to store with the picture.

ICURR = INTEGER (Returned)
The index of a Frame to make current before storing the Plot. The supplied current Frame is used if ICURR is AST__NOFRAME.

WDOM = CHARACTER * (*) (Given)
The Domain name of the Frame correspondoing to AGI world co-ordinates.

DDOM = CHARACTER * (*) (Given)
Domain name of the co-ordinate Frame within IPLOT corresponding to AGI data co-ordinates. " AXIS" is used if a blank value is supplied.

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

KPG1_PLPUT
Puts a section of the current colour palette into the supplied array

Description:

This routine puts a specified section of the colour palette for the currently opened graphics device into the supplied array. Other elements of the array are left unchanged.

Invocation:

```
CALL KPG1_PLPUT( CI1, CI2, LBND, UBND, ARRAY, STATUS )
```

Arguments:

CI1 = INTEGER (Given)
The lowest colour index to change in the array.

CI2 = INTEGER (Given)
The highest colour index to change in the array.

LBND = INTEGER (Given)
The lower bound of the second axis of ARRAY.

UBND = INTEGER (Given)
The upper bound of the second axis of ARRAY.

ARRAY(3, LBND : UBND) = REAL (Given and Returned)
The array to receive the palette.

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

Notes:

- A graphics device must previously have been opened using PGPLOT.

KPG1_PLSAV
Saves the colour palette for the currently open graphics device

Description:

This routine saves a section of the colour palette for the currently open graphics device in an HDS container file in the users ADAM directory. The file is called " kappa_palette.sdf" and contains palettes for different devices. Each palette is a _REAL array of shape (3,n) where n is the number of colours in the palette. The first colour (i.e. the first element in the array) is the background colour and is refered to as colour index zero. Therefore the highest colour index in the array is (n-1). Each array has a name which identifies the graphics device to which it refers.

If a palette already exists for the device in the HDS container file, then the values stored in the HDS palette for the range of colour indices specified by CI1 and CI2 are modified to reflect the current colour table, and values in the HDS palette for other colour indices are left unchanged. If no HDS palette already exists, then an entire palette array is created and initally filled with values of

- 1. These indicate that no value has yet been specified for the colour index, and allows the default value to be used. This default may depend on the graphics device. For instance, the default for pen 1 will be white on an Xwindow but black on a printer.

Invocation:

```
CALL KPG1_PLSAV( CI1, CI2, RESET, STATUS )
```

Arguments:

CI1 = INTEGER (Given)
The lowest colour index to save. Greater than or equal to zero. Zero is the background colour.

CI2 = INTEGER (Given)
The highest colour index to save. If a value less than CI1 is given, then the highest available colour index is used.

RESET = LOGICAL (Given)
Should all pens outside the range given by CI1 and CI2 be reset to their default (unspecified) values in the HDS palette? If not, their current values are retained.

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

Notes:

- The HDS container file is created if it does not already exist.
- A graphics device must previously have been opened using SGS/GKS.

KPG1_PLTLN
Produces a graphical representation of a set of points in two dimensions

Description:

This routine produces a graphical representation of a set of points in two-dimensional space (e.g. a data value and a position, or two data values). Errors in both data values may be represented by error bars. No annotated axes are drawn. The calling routine should do this if required by passing the supplied Plot (IPLOT) to routine KPG1_ASGRD.

PGPLOT should be active, and the viewport should correspond to the DATA picture in which the plot is to be drawn. PGPLOT world co-ordinates within the viewport should be GRAPHICS co-ordinates (millimetres from the bottom-left corner of the view surface).

The Plotting style is accessed using one or more environment parameters specified by PARAM, and may include the following synonyms for graphical elements: " Err(Bars)" - Specifies colour, etc. for error bars. Size(errbars) scales the size of the serifs used if ERSHAP=1 (i.e. a size value of 1.0 produces a default size). " Sym(bols)" - Specifies colour, etc. for markers (used in Modes 3 and 5). " Lin(es)" - Specifies colour, etc. for lines (used in Modes 1, 2, 4 and 5).

Invocation:

```
CALL KPG1_PLTLN( N, ILO, IHI, X, Y, XERROR, YERROR, XBAR, YBAR, XSTEP, PARAM, IPLOT,
MODE, MTYPE, ERSHAP, FREQ, APP, STATUS )
```

Arguments:

N = INTEGER (Given)
Number of points to be plotted.

ILO = INTEGER (Given)
The index of the first grid point to be used.

IHI = INTEGER (Given)
The index of the last grid point to be used.

X(N) = DOUBLE PRECISION (Given)
The X value at each point, in PGPLOT world co-ordinate (i.e. millimetres from the bottom-left corner of the view surface).

Y(N) = DOUBLE PRECISION (Given)
The Y value at each point, in PGPLOT world co-ordinate (i.e. millimetres from the bottom-left corner of the view surface).

XERROR = LOGICAL (Given)
Display X error bars?

YERROR = LOGICAL (Given)
Display Y error bars?

XBAR(N, 2) = DOUBLE PRECISION (Given)
Row 1 contains the lower limit and Row 2 contains the upper limit for each horizontal error bar, in PGPLOT world co-ordinate (i.e. millimetres from the bottom left corner of the view surface). Only accessed if XERROR is .TRUE.

YBAR(N, 2) = DOUBLE PRECISION (Given)
Row 1 contains the lower limit and Row 2 contains the upper limit for each vertical error bar, in PGPLOT world co-ordinate (i.e. millimetres from the bottom-left corner of the view surface). Only accessed if YERROR is .TRUE.

XSTEP(N, 2) = DOUBLE PRECISION (Given)
Row 1 contains the lower limit and Row 2 contains the upper limit for each horizontal step, in PGPLOT world co-ordinate (i.e. millimetres from the bottom-left corner of the view surface). Only accessed if MODE is 4.

PARAM = CHARACTER * (*) (Given)
The names of one or more style parameters to be used when obtaining the plotting style. If more than one parameter is supplied, the list should be separated by commas. The supplied parameters will be used in the order supplied, with later parameters allowing attributes obtained via earlier parameters to be assigned new values.

IPLOT = INTEGER (Given)
An AST Plot which can be used to do the drawing. The Base Frame should be GRAPHICS co-ordinates (millimetres from the bottom- left corner of the PGPLOT view surface). The Current Frame should be the Frame in which annotation is required.

MODE = INTEGER (Given)
Determines the way in which the data points are represented. The options are as follows. 1 - A " staircase" histogram, in which each horizontal line is centred on the X position. Bad values are flanked by vertical lines drawn down to the lower edge of the viewport. 2 - The points are joined by straight lines. 3 - A marker is placed at each point (see MTYPE). 4 - Mark each point with a horizontal line of width given by XW. 5 - A " chain" in which each point is marker by a marker and also join by straight lines to its neighbouring points. 6 - The same as Mode 1, except that bad values are not flanked by vertical lines drawn down to the lower edge of the viewport (a simple gap is left instead). 7 - The data points are not drawn.

MTYPE = INTEGER (Given)
The PGPLOT marker type to use if MODE is 3 or 5.

ERSHAP = INTEGER (Given)
Determines the way in which error bars are drawn: 1 - X and Y errors are represented by horizontal and vertical lines respectively. Serifs are drawn at the ends of each line. The size of these sreifs is controlled by the size(errbar) plotting attribute. 2 - A cross is drawn joining the corners of the box encompassing the X and Y errors. 3 - A Diamond is drawn joining the ends of the horizontal and vertical error bars which would have been drawn if ERSHAP had been 1. These will all produce the same result (i.e. a single straight line) if errors are available only on one axis (see XERROR and YERROR). Not accessed if XERROR and YERROR are both .FALSE.

FREQ = INTEGER (Given)
The frequency at which errors are to be plotted. A value of 1 means " plot errors for every point" , 2 means " plot errors for every second point" , etc. Not accessed if XERROR and YERROR are both .FALSE.

APP = CHARACTER * (*) (Given)
The name of the calling application in the form <package>_<application> (eg " KAPPA_DISPLAY").

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

KPG1_POISx
Takes values and returns them with Poisson noise added

Description:

This routine adds or subtracts pseudo-random Poissonian (shot) noise to a series of values. It uses a Box-Mueller algorithm to generate a fairly good normal distribution.

Invocation:

```
CALL KPG1_POISx( EL, VALUES, SEED, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of values to which to add Poisson noise.

VALUES(EL) = ? (Given and Returned)
On input these are the value to which noise is to be applied. On return the values have the noise applied.

SEED = REAL (Given & Returned)
The seed for random number generator; updated by the random-number generator.

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array supplied to the routine must have the data type specified.
- There is no checking for overflows.
- Bad values remain bad in the returned array.

KPG1_POWx
Raises each element of a vectorised array to a specified power

Description:

The routine raises each element of a vectorised array to a specified power. Any associated variance values are also modified appropriately. For general non-integer powers the result is calculated as EXP(power∗LOG(data)). This enables bad output pixels to be detected by checking the magnitude of LOG(data). For integer powers, the result is calcualted using the Fortran " ∗∗" operator, which is more efficient.

Invocation:

```
CALL KPG1_POWx( BAD, VAR, POWER, EL, DIN, VIN, DOUT, VOUT, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input arrays.

VAR = LOGICAL (Given)
Have associated variances been supplied?

POWER = DOUBLE PRECISION (Given)
The required power.

EL = INTEGER (Given)
Number of array elements to process.

DIN(EL) = ? (Given)
Input data array.

VIN(EL) = ? (Given)
Input variance array. Only accessed if VAR is .TRUE.

DOUT(EL) = ? (Given)
Output data array.

VOUT(EL) = ? (Given)
Output variance array. Only accessed if VAR is .TRUE.

NBAD(2) = INTEGER (Returned)
Element 1 has the number of bad values stored in DOUT, and Element 2 has the number of bad values stored in VOUT.

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

Notes:

- There is a routine for processing both real and double precision data; replace " x" in the routine name by R or D as appropriate. The data types of the DIN, VIN, DOUT, and VOUT arrays should match the routine used.

KPG1_PQVID
Tests whether the current graphics device has suitable image-display characteristics

Description:

This routine determines whether the current graphics device is an image display with suitable characteristics. The checks are performed in the following order: the class of the device is checked against a supplied list of acceptable classes; the presence of certain attributes, given in a supplied list are checked in the order colour, input and open with reset; and finally a minimum number of colour indices must be exceeded.

Invocation:

```
CALL KPG1_PQVID( PNDEV, CLASS, CRITER, MININT, UP, STATUS )
```

Arguments:

PNDEV = CHARACTER * (*) (Given)

The ADAM parameter associated with the current graphics workstation. It is used to generate error messages.

CLASS = CHARACTER * (*) (Given)

A list of acceptable workstation classes as defined by GNS, each separated by a comma. See SUN/57 for a list. Note, as GNS does not yet support PGPLOT, no checks are mode on GNS class when using PGPLOT. Class checks are ignored if a blank string is supplied.

CRITER = CHARACTER * (*) (Given)

A list of criteria that the workstation must pass in order to be an acceptable image display. The options are: ' COLOUR' if colour must be available, ' CURSOR' if a cursor must be available on the workstation. Non-standard criteria will be ignored.

MININT = INTEGER (Given)

Minimum number of intensities or colour indices required. The routine ensures that there are at least the specified number.

UP = INTEGER (Returned)

The highest available colour index available.

STATUS = INTEGER (Given and Returned)

The global status.

Prior Requirements :

- A PGPLOT workstation must be open.

KPG1_PRCVT
Converts an HDS primitive to a native data representation

Description:

The routine converts a primitive HDS object so that it is stored using the appropriate native data representation provided by the host machine.

Invocation:

```
CALL KPG1_PRCVT( LOC, STATUS )
```

Arguments:

LOC = CHARACTER * (DAT__SZLOC) (Given and Returned)

Locator for the object to be converted. This may be modified on exit (as the original object may have to be erased and re-created).

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

This routine returns without action if the object supplied is not primitive, or if it does not need conversion (i.e. if it has already been converted or was created on the current machine). In this case the LOC argument will be returned unchanged.

KPG1_PRNTH
Locates the outer-most pair of parenthesis in a string

Description:

The routine returns the indices of the first opening parenthesis and the last closing parenthesis in the supplied string. Both are returned equal to zero if either parenthesis is not found.

Invocation:

```
CALL KPG1_PRNTH( STRING, OP, CL STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)
String to be searched.

OP = INTEGER (Returned)
Position within STRING at which the first occurence of " (" is located.

CL = INTEGER (Returned)
Position within STRING at which the last occurence of ")" is located.

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

KPG1_PROWx
Puts values into a row of a two-dimensional array

Description:

The routine enters values into a specified row of a two-dimensional array, the values being supplied in a separate one-dimensional array whose size matches the row size.

Invocation:

```
CALL KPG1_PROWx( EL, ROW, IROW, ARRAY, STATUS )
```

Arguments:

EL = INTEGER (Given)
Number of elements in a single row of the two-dimensional array.

ROW(EL) = ? (Given)
Array of values to be inserted into the row.

IROW = INTEGER (Given)
The row number in the two-dimensional array into which the values are to be inserted.

ARRAY(EL, $*$) = ? (Given and Returned)
The two-dimensional array which is to recieve the new values. The declared second dimension size of this array must not be fewer than IROW. The values in other rows of this array are not altered.

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

Notes:

- There is a routine for each numeric data type. Replace " x" in the routine name by B, UB, W, UW, I, R or D as appropriate. The data type of the ROW and ARRAY arrays should match the routine being used.

KPG1_PRSAx
Extracts a list of numerical values from a string

Description:

A supplied string is search for words (separated by spaces, tabs or commas), which are extracted and converted to the required numerical data type. The numerical values and their number are returned.

Invocation:

```
CALL KPG1_PRSAx( BUFFER, MAXVAL, VALUES, NVAL, STATUS )
```

Arguments:

BUFFER = CHARACTER * (*) (Given)
The string containing a list of numerical values.

MAXVAL = INTEGER (Given)
The maximum number of values that can be read from the buffer and stored.

VALUES(MAXVAL) = ? (Returned)
The numeric values extracted from the string.

NVAL = INTEGER (Returned)
The actual number of values extracted from the string.

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

Notes:

- An appropriate bad value is returned for any word equal to " BAD" (case insensitive).
- There is a routine for four numeric data type: replace " x" in the routine name by D, I, K, or R as appropriate. The array returned from the routine must have the data type specified. The limitation on other integer types is because there are no conversion routines between them and character in the CHR library.

KPG1_PSEED
Sets the PDA Random number seed

Description:

This routine sets the seed for the PDA random number routines to a non-repeatable value, and must be called prior to using any PDA random number routine. The seed is only set once in each process, and is set to a number which combines the process id and the current time.

The process id is included because the " time" system call (implemented by PSX_TIME) returns the time in seconds. On modern machines it is possible for an application to be called several times each second, resulting in the same seed being used each time if the seed is based only on the time.

Alternatively, a fixed seed can be used by assigning the seed value to environment variable STAR_SEED.

Invocation:

```
CALL KPG1_PSEED( STATUS )
```

Arguments:

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

KPG1_PSFSx
Finds the approximate size of a two-dimensional PSF

Description:

Marginal profiles are formed of the absolute PSF values along both axes. For each axis the maximum and minimum values in the corresponding profile are found. The first and last point at which each profile reaches a specified fraction of its total range is found, and the difference returned as the corresponding PSF size. N.B., it is assumed that the input PSF contains no bad pixels.

Invocation:

```
CALL KPG1_PSFSx( PSF, NPIX, NLIN, WORK, NPW, NLW, FRACT, ILEVEL, XSIZE, YSIZE, STATUS
)
```

Arguments:

PSF(NPIX, NLIN) = ? (Given)
The PSF image.

NPIX = INTEGER (Given)
Number of pixels per line in the PSF image.

NLIN = INTEGER (Given)
Number of lines in the PSF image.

WORK(NPW, NLW) = ? (Given)
Work space.

NPW = INTEGER (Given)
Number of elements per line in the work array. This should be at least equal to the maximum of NPIX and NLIN.

NLW = INTEGER (Given)
Number of lines in the work array. This should be at least 2.

ILEVEL = INTEGER (Given)
The user information level. If ILEVEL is 2 or more, then the user is told what the calculated sizes are.

FRACT = REAL (Given)
The fraction of the PSF peak amplitude at which the PSF size is determined. It should be positive and less than 0.5. If it is outside this range one sixteenth is used.

XSIZE = INTEGER (Returned)
The width in x of the PSF, in units of pixels.

YSIZE = INTEGER (Returned)
The width in y of the PSF, in units of lines.

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

Notes:

- There is a routine for processing single- and double-precision arrays; replace " x" in the routine name by R or D as appropriate. The data type of the PSF and WORK arguments must match the routine used.

KPG1_PVERS
Parses a package version string

Description:

This routine splits the supplied package version string (e.g. " V0.13-6") into 3 integers corresponding to the major version number, minor version number and revision number. The leading " V" can be omitted, and trailing fields can be ommitted (they default to zero).

Invocation:

```
CALL KPG1_PVERS( TEXT, MAJ, MIN, REV, STATUS )
```

Arguments:

TEXT = CHARACTER * (*) (Given and Returned)
The version string. Blanks are removed and it is converted to upper case on exit.

MAJ = INTEGER (Returned)
The major version number.

MIN = INTEGER (Returned)
The minor version number.

REV = INTEGER (Returned)
The revision number.

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

KPG1_PX2AX
Converts a pixel's indices into WCS or axis co-ordinates

Description:

This routine converts the pixel indices of an NDF pixel into the WCS co-ordinate values of the pixel's centre. If a WCS FrameSet is not available, then the pixel indices are converted into the axis co-ordinate system. If an axis co-ordinate system is not defined for the NDF, then the pixel co-ordinate system will be used instead.

Invocation:

```
CALL KPG1_PX2AX( NDIM, PX, INDF, AX, STATUS )
```

Arguments:

NDIM = INTEGER (Given)
Number of NDF dimensions.

PX(NDIM) = INTEGER (Given)
Indices of the NDF's pixel.

INDF = INTEGER (Given)
NDF identifier.

AX(*) = DOUBLE PRECISION (Returned)
WCS or axis co-ordinate values for the pixel's centre. There should be enough elements in this array for all the current frame WCS axes (which need not be the same as NDIM).

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

Notes:

This routine is simplified by handling only a single pixel. It will not be efficient enough to handle arrays of pixels.

KPG1_PXDPx
Expands an n-dimensional array by pixel duplication

Description:

This routine expands an input array by pixel duplication along each dimension. The duplication factors may be different for each dimension.

Invocation:

```
CALL KPG1_PXDPx( IDIMS, INARR, EXPAND, USEMSK, MASK, ODIMS, OUTARR, STATUS )
```

Arguments:

IDIMS(NDF__MXDIM) = INTEGER (Given)
The dimensions of the input array. Unused dimensions up to NDF__MXDIM should be set to one.

INARR(*) = ? (Given)
The input data array that is to be enlarged by duplication.

EXPAND(NDF__MXDIM) = INTEGER (Given)
The linear expansion factor applied along each dimension. Factors for unused dimensions up to NDF__MXDIM should be set to one.

USEMSK = LOGICAL (Given)
Should the mask array supplied by argument MASK be used?

MASK(*) = REAL (Given)
Only accessed if USEMSK is .TRUE. The MASK array corresponds to a single expansion block in the output array, and so should have a size equal to the product of the values supplied in the EXPAND argument. If USEMSK is .TRUE., the value stored for each output pixel equals the product of the input pixel which is being duplicated, and the corresponding mask pixel (that is, the mask pixel that is at the same position as the output position is within its current block of output pixels). If USEMSK is .FALSE. each output pixel value is simply equal to the input pixel value that is being duplicated. This is equivalent to using a mask that is filled with the value 1.0. The supplied mask array may contain VAL__BADR values in which case the corresponding output pixel will be set to VAL__BAD<T>.

ODIMS(NDF__MXDIM) = INTEGER (Given)
The dimensions of the expanded array. Unused dimensions up to NDF__MXDIM should be set to one.

OUTARR(*) = ? (Returned)
The expanded array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- This routine works in n-D, where n is 1 to 7. Even if the array has actually less dimensions there is negligible loss of efficiency to supply dummy (=1) higher dimensions.
- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The base and paste arrays supplied to the routine must have the data type specified.

KPG1_PXSCL
Determines pixel scales at a given grid position

Description:

This routine determines the scales of the WCS axes in the current Frame of the supplied FrameSet. For a specified WCS axis, the returned scale is the WCS axis increment produced by moving a distance of one grid pixel away from the supplied " AT" position, along the WCS axis.

Invocation:

```
CALL KPG1_PXSCL( IWCS, AT, PXSCL, STATUS )
```

Arguments:

IWCS = INTEGER (Given)

The FrameSet.

AT($*$) = DOUBLE PRECISION (Given)

The position in GRID co-ordinates at which the pixel scales are to be determined. Note, the pixel scales may vary across the data array if the WCS Mappings are non-linear. The array should have one element for each GRID axis.

PXSCL($*$) = DOUBLE PRECISION (Returned)

The returned pixel scales. Note, the pixel scale for both celestial longitude and latitude axes are returned as an arc-distance in radians. The array should have one element for each WCS axis.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_QNTLx
Finds a quantile in a (possibly weighted) set of data

Description:

The routine calculates the value of a specified quantile in a set of data values, which may be weighted. In concept (although not in practice) it sorts the supplied data values into ascending order along with their associated positive weights, if supplied. It then finds a quantile Q such that the sum of the weights associated with all data values less than Q is a specified fraction FRACT of the sum of all the weights supplied. If no weights are supplied, then each data value is assigned unit weight. There are two main applications of this algorithm:

a) To find specified quantiles of a distribution of data values for statistical purposes. In this case, the weights may optionally be used to represent the number of times each data value occurs. In such cases, it may be useful to regard the distribution as continuous, and therefore to interpolate linearly between data values when obtaining the result.

b) Alternatively, the values may represent residuals from some fitted function. In this case, by setting FRACT to 0.5, the " weighted median residual" may be found. This has the property that if it is subtracted from all the original residuals, then the weighted sum of the absolute values of the corrected residuals will be minimised. Thus, it may be used as the basis for iteratively finding an 'L1' fit. In such cases, the required result will be equal to one of the data values (or may lie mid-way between two of them) and interpolation between values is not normally required.

Invocation:

```
CALL KPG1_QNTLx( USEWT, INTERP, FRACT, EL, X, W, IP, Q, STATUS )
```

Arguments:

USEWT = LOGICAL (Given)
Whether or not the data have associated weights.

INTERP = LOGICAL (Given)
Whether or not interpolation between data values should be performed when obtaining the result.

FRACT = ? (Given)
The fraction specifying the required quantile, in the range 0.0 to 1.0.

EL = INTEGER (Given)
Number of data values.

X(*) = ? (Given)
Array of data values.

W(*) = ? (Given)
Array of associated positive weights (if required). This argument will only be referenced if USEWT is .TRUE..

IP(EL) = INTEGER (Given and Returned)
On entry, an array of pointers identifying which elements of X (and W if supplied) are to be considered. On exit, these pointers will have been permuted to access the specified data elements in an order which is more nearly sorted than before (although in general it will not represent a complete sort of the data).

Q = ? (Returned)
The value of the requested quantile.

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

Notes:

- There are versions of this routine for processing both REAL and DOUBLE PRECISION data; replace the " x" in the routine name by R or D as appropriate. The types of the FACT, X, W and Q arguments should match the routine being used.
- This routine is optimised for use when the number of data values is large. In general, only a partial sort of the data will be performed, so this routine will perform better than most other methods of finding quantiles, which typically require a complete sort.
- The order in which the input pointers are supplied in the array IP is arbitrary, but there will often be an efficiency advantage in supplying them so that they access the data in nearly-sorted order. Thus, re-supplying the array of pointers generated by a previous invocation of this routine (for the same or similar data) may be worthwhile.

Timing :

Details of the asymptotic time required to execute the original SELECT algorithm are not altogether clear from the published papers. It appears that this algorithm may have better average performance than other methods and the time required may approximate to EL $*$ LOG(MIN(K, EL - K + 1)) where K is the rank of the largest data value which is smaller than the quantile being sought. However, Sedgewick (see References) indicates that such algorithms should, in general, complete in time proportional to EL, so the above formula may be incorrect. When using weighted data, the time will be multiplied by a further factor reflecting the non-linearity of the cumulative weight versus rank function and the difficulty of inverting it.

References :

- Comm. of the ACM, vol 18, no. 3 (March 1975), p165.
- Also see page 173.
- In addition, see the algorithm assessment by T. Brown in Collected Algorithms of the ACM, (algorithm no. 489).
- Sedgwick, R., 1988, " Algorithms" (Addison-Wesley).

KPG1_QSRTX
Sorts a vector via the Quicksort algorithm

Description:

This routine sorts a vector in situ between an upper and lower bounds using the Quicksort algorithm.

Invocation:

```
CALL KPG1_QSRTx( EL, LOW, HIGH, ARRAY, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the array that is to be sorted.

LOW = INTEGER (Given)
The lower bound within the array, below which the array elements will not be sorted. It should be less than the upper bound and must be within the array. In the latter case an error will result and the routine will not sort the array.

HIGH = INTEGER (Given)
The upper bound within the array, above which the array elements will not be sorted. It should be greater than the lower bound and must be within the array. In the latter case an error will result and the routine will not sort the array.

ARRAY(EL) = ? (Given and Returned)
The array to be sorted.

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

References :

- Sedgwick, R., 1988, " Algorithms" (Addison-Wesley).

Timing :

For N elements to be sorted the timing goes as NlnN.

Implementation Status:

- There is a routine for each of the data types integer, real, double precision, and character: replace " x" in the routine nam by I, R, D, or C respectively as appropriate.
- If the maximum bound is less than the minimum, the bounds are swapped.

KPG1_QUOTE
Quote a supplied string

Description:

This routine returns a new string holding a copy of the supplied string within single quotes. Any single quotes within the supplied string are escaped using a backslash.

Invocation:

```
CALL KPG1_QUOTE( IN, OUT, STATUS )
```

Arguments:

IN = CHARACTER*(*) (Given)
The input string. Any trailing spaces are included in the returned quoted string.

OUT = CHARACTER*(*) (Returned)
The quoted string.

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

Notes:

- An error is reported if the output string is not large enough to hold the quoted string.

KPG1_R2NAG
Converts an FFTPACK Hermitian Fourier transform array into the equivalent NAG array

Description:

This subroutine re-orders and normalises the supplied array of Fourier co-efficients (as produced by FFTPACK subroutine KPG1_RFFTF) so that the returned array looks like the equivalent array returned by NAG routine C06FAE.

This function is equivalent to PDA_R2NAG except that it uses work space for greater speed.

The real and imaginary co-efficients produced by KPG1_RFFTF are numerically larger than the corresponding C06FAE co-efficients by a factor of SQRT(NP), and are ordered differently. Both routines return A0 (the zeroth real term, i.e. the DC level in the array) in element 1. KPG1_RFFTF then has corresponding real and imaginary terms in adjacent elements, whereas C06FAE has all the real terms together, followed by all the imaginary terms (in reverse order):

KPG1_RFFTF : A0, A1, B1, A2, B2, A3, B3, ... C06FAE: A0, A1, A2, A3, ..., ..., B3, B2, B1

The zeroth imaginary term (B0) always has the value zero and so is not stored in the array. Care has to be taken about the parity of the array size. If it is even, then there is one more real term than there is imaginary terms (excluding A0), i.e. if NP = 10, then the co-efficients are stored as follows:

KPG1_RFFTF : A0, A1, B1, A2, B2, A3, B3, A4, B4, A5 C06FAE: A0, A1, A2, A3, A4, A5, B4, B3, B2, B1

If NP = 9, then the co-efficients are stored as follows:

KPG1_RFFTF : A0, A1, B1, A2, B2, A3, B3, A4, B4 C06FAE: A0, A1, A2, A3, A4, B4, B3, B2, B1

Invocation:

```
CALL KPG1_R2NAG( NP, R, WORK )
```

Arguments:

NP = INTEGER (Given)
The size of the array.

R(NP) = REAL (Given and Returned)
The array holding the Fourier co-efficients. Supplied in FFTPACK format, returned in NAG format.

WORK(NP) = REAL (Given and Returned)
Work space.

KPG1_RCATW
Attempts to read an AST Object from a catalogue

Description:

This routine attempts to read an AST Object from the textual information stored with the supplied catalogue (see SUN/181). Reading of the textual information in the catalogue commences at the current line (i.e. access to the textual information is not reset before reading commences).

AST Objects can be written to a catalogue using routine KPG1_WCATW.

Invocation:

```
CALL KPG1_RCATW( CI, IAST, STATUS )
```

Arguments:

CI = INTEGER (Given)
A CAT identifier (see SUN/181) for the supplied catalogue.

IAST = INTEGER (Returned)
An AST pointer to the returned Object. AST__NULL is returned if an error occurs.

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

KPG1_RDAST
Reads AST_ data as text from an HDS object

Description:

This is a service routine to be provided as a " source" routine for the AST_CHANNEL function. It reads data from an HDS object (in response to reading from an AST_ Channel) and delivers it to the AST_ library for interpretation.

This routine has only a STATUS argument, so it communicates with other KPG routines via global variables stored in the KPG_AST common blocks. These are described below under " Global Variables used as Arguments" .

Invocation:

```
CALL KPG1_RDAST
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Global Variables used as Arguments :

ASTLC = CHARACTER ∗ (DAT__SZLOC) (Given) A locator for the HDS object which holds the data. This must be a one-dimensional _CHAR array, whose size and character string length will be determined via this locator. ASTLN = INTEGER (Given and Returned) This must initially be set to the value 1, to indicate that data will be read starting at the first element of the HDS array (note the routine will not operate correctly unless 1 is the initial value - you cannot start reading at another point in the array if you have previously read from a different array). On exit it will be incremented by the number of elements used to obtain data, so that it identifies the first element to be used on the next invocation. ASTPT = INTEGER (Given) A pointer to the contents of the HDS object, mapped in ’ READ’ mode.

KPG1_RDCAT
Reads a set of positions with labels from a CAT catalogue

Description:

This routine is equivalent to KPG1_RDTAB except that the values of an arbitrary set of columns (including character-valued columns) can be returned within an AST KeyMap (see KEYMAP).

See KPG1_RDTAB for further information.

Invocation:

```
CALL KPG1_RDCAT( PARAM, CURFRM, KEYMAP, LABS, IWCS, NPOS, NAX, IPPOS, IPID, TITLE, NAME,
STATUS )
```

Arguments:

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

CURFRM = LOGICAL (Given)
If .TRUE. the positions read from the catalogue are Mapped into the Current Frame of the associated FrameSet before being returned. Otherwise, they are returned in the Base Frame.

KEYMAP = INTEGER (Given)
An AST pointer to an existing KeyMap, or AST__NULL. If a KeyMap is supplied, it should contain a vector valued entry called " COLNAMES" containing the names of one or more catalogue columns to be returned in the KeyMap. On exit, the KeyMap will contain the column values within a set of scalar entries. Each such entry will have a key of the form " <colname>_<row number>" (" _1" for the first row). An error will be reported if the catalogue does not contain the requested columns.

LABS = INTEGER (Given and Returned)
A GRP identifier for a group containing the values in the LABEL column. If the catalogue contains a LABEL column, then its values are appended to the end of the supplied group. If LABS holds GRP__NOID on entry, then a new GRP group is created and its identifier returned in LABS, but only if the catalogue contains a LABEL column (otherwise the supplied value of GRP__NOID is retained on exit).

IWCS = INTEGER (Returned)
An AST pointer to the FrameSet read from the catalogue.

NPOS = INTEGER (Returned)
The number of positions returned.

NAX = INTEGER (Returned)
The number of axes in the Frame requested by CURFRM.

IPPOS = INTEGER (Returned)
A pointer to a two-dimensional DOUBLE PRECISION array holding the returned positions. Element (I,J) of this array gives axis J for position I. The first axis will have NPOS elements, and the second will have NAX elements. Should be released using PSX_FREE when no longer needed.

IPID = INTEGER (Returned)
A pointer to a one-dimensional INTEGER array holding the integer identifiers for the returned positions. The array will have NPOS elements. Should be released using PSX_FREE when no longer needed.

TITLE = CHARACTER * (*) (Returned)
The value of the TITLE parameter in the supplied catalogue. Returned blank if there is no TITLE parameter.

NAME = CHARACTER * (*) (Returned)
The file spec of the catalogue containing the positions list. Not accessed if the declared length is 1.

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

KPG1_RDLST
Reads a set of positions from a CAT catalogue

Description:

This routine reads a FrameSet, and a set of positions with associated integer identifiers from a CAT catalogue. The FrameSet should be stored as an AST Dump in the textual information associated with the catalogue. Such catalogues can be created using KPG1_WRLST. If the catalogue does not contain a FrameSet, then a default FrameSet will be used if possible. If the catalogue contains floating point columns named RA and DEC, then the default FrameSet contains a single SkyFrame (Epoch and Equinox are set from the EPOCH and EQUINOX catalogue parameters - if they exist). Otherwise, if the catalogue contains floating point columns named X and Y, then the default FrameSet contains a single two-dimensional Frame with axis symbols X and Y, and Domain GRID. If there is also a column named Z, then the Frame will be three-dimensional, with a Z axis.

However the FrameSet is obtained, it is assumed that the columns containing the axis values have CAT names equal to the Symbol attribute of the corresponding AST Axis. The catalogue columns from which to read the axis values are chosen by matching column names with Axis Symbols (only columns containing floating point values are considered). Frames are checked in the following order: the Base Frame, the Current Frame, all other Frames in order of increasing Frame index. An error is reported if no Frame has a set of corresponding columns.

It is assumed that position identifiers are stored in an integer column with name PIDENT. If no such column is found, the returned position identifiers start at 1 and increase monotonically.

The routine KPG1_RDTAB is like KPG1_RDLST, but provides an extra option to return a group of position labels read from a LABEL column in the catalogue.

Invocation:

```
CALL KPG1_RDLST( PARAM, CURFRM, IWCS, NPOS, NAX, IPPOS, IPID, TITLE, NAME, STATUS )
```

Arguments:

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

CURFRM = LOGICAL (Given)
If .TRUE. the positions read from the catalogue are Mapped into the Current Frame of the associated FrameSet before being returned. Otherwise, they are returned in the Base Frame.

IWCS = INTEGER (Returned)
An AST pointer to the FrameSet read from the catalogue.

NPOS = INTEGER (Returned)
The number of positions returned.

NAX = INTEGER (Returned)
The number of axes in the Frame requested by CURFRM.

IPPOS = INTEGER (Returned)
A pointer to a two-dimensional DOUBLE PRECISION array holding the returned positions. Element (I,J) of this array gives axis J for position I. The first axis will have NPOS elements, and the second will have NAX elements. Should be released using PSX_FREE when no longer needed.

IPID = INTEGER (Returned)
A pointer to a one-dimensional INTEGER array holding the integer identifiers for the returned positions. The array will have NPOS elements. Should be released using PSX_FREE when no longer needed.

TITLE = CHARACTER * (*) (Returned)
The value of the TITLE parameter in the supplied catalogue. Returned blank if there is no TITLE parameter.

NAME = CHARACTER * (*) (Returned)
The file spec of the catalogue containing the positions list. Not accessed if the declared length is 1.

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

KPG1_RDTAB
Reads a set of positions with labels from a CAT catalogue

Description:

This routine is equivalent to KPG1_RDLST except that any labels stored with the positions in the catalogue are returned in a GRP group (see LABS). The labels must be stored in a column called " LABEL" . Catalogues containing such labels are written by KPG1_WRTAB.

See KPG1_RDLST for further information.

Invocation:

```
CALL KPG1_RDTAB( PARAM, CURFRM, LABS, IWCS, NPOS, NAX, IPPOS, IPID, TITLE, NAME, STATUS
)
```

Arguments:

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

CURFRM = LOGICAL (Given)
If .TRUE. the positions read from the catalogue are Mapped into the Current Frame of the associated FrameSet before being returned. Otherwise, they are returned in the Base Frame.

LABS = INTEGER (Given and Returned)
A GRP identifier for a group containing the values in the LABEL column. If the catalogue contains a LABEL column, then its values are appended to the end of the supplied group. If LABS holds GRP__NOID on entry, then a new GRP group is created and its identifier returned in LABS, but only if the catalogue contains a LABEL column (otherwise the supplied value of GRP__NOID is retained on exit).

IWCS = INTEGER (Returned)
An AST pointer to the FrameSet read from the catalogue.

NPOS = INTEGER (Returned)
The number of positions returned.

NAX = INTEGER (Returned)
The number of axes in the Frame requested by CURFRM.

IPPOS = INTEGER (Returned)
A pointer to a two-dimensional DOUBLE PRECISION array holding the returned positions. Element (I,J) of this array gives axis J for position I. The first axis will have NPOS elements, and the second will have NAX elements. Should be released using PSX_FREE when no longer needed.

IPID = INTEGER (Returned)
A pointer to a one-dimensional INTEGER array holding the integer identifiers for the returned positions. The array will have NPOS elements. Should be released using PSX_FREE when no longer needed.

TITLE = CHARACTER * (*) (Returned)
The value of the TITLE parameter in the supplied catalogue. Returned blank if there is no TITLE parameter.

NAME = CHARACTER * (*) (Returned)
The file spec of the catalogue containing the positions list. Not accessed if the declared length is 1.

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

KPG1_REPRT
Reports a MSG message to user and also optionally write it to a file

Description:

This routine dislays the supplied message using MSG_OUT (unless QUIET is .TRUE), and (if LOG is .TRUE.) writes out the same text to the file identified by FD.

Invocation:

```
CALL KPG1_REPRT( MESS, QUIET, LOG, FD, STATUS )
```

Arguments:

MESS = CHARACTER * (*) (Given)
The message, which may contain MSG tokens.

QUIET = LOGICAL (Given)
Supress screen output?

LOG = LOGICAL (Given)
Write the text to a file?

FD = INTEGER (Given)
An FIO identifier for a file. If LOG is .TRUE., then the message is written to this file.

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

KPG1_RETRx
Retrieves a value from an array

Description:
The value stored at a given index within the supplied array is returned.

Invocation:
```
CALL KPG1_RETRx( EL, INDEX, DATA, VALUE, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the array.

INDEX = INTEGER (Given)
The index within the array of the required value.

DATA(EL) = ? (Given)
The input array.

VALUE = ? (Returned)
The returned value.

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

Notes:

- There is a routine for all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The VALUE and DATA arguments must have the data type specified.

KPG1_RETVx
Retrieves values specified by index from a vector

Description:

This routine returns values from a vector at supplied indices. This is intended for accessing a few values from a mapped array.

Invocation:

```
CALL KPG1_RETVx( EL, ARRAY, NVAL, INDEX, VALUES, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the array.

ARRAY(EL) = ? (Given)
The input array.

NVAL = INTEGER (Given)
The number of element values to return.

INDEX(NVAL) = INTEGER (Given)
The indices of the array elements to be returned.

VALUES(NVAL) = ? (Returned)
The value of the Xth array element.

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

Implementation Status:

- There is a routine for each of the numerics data types and logical: replace "x" in the routine name by D, R, I, K, W, UW, B, UB, L as appropriate. The array supplied to the routine must have the data type specified.
- Bad status is returned if the supplied indices lie beyond the array bounds.

KPG1_RFCOx
Reads co-ordinate data from a text free-format file

Description:

This routine reads co-ordinate information and from a text free-format file and converts it to floating-point values. The file is like a relational catalogue with one record per related set of co-ordinates, however, exact column alignment is not necessary. The converted input data are copied to an array. The array uses a new line for each record of the file, its columns storing the given positional data in the order x, y, z etc. If the end of file is not reached a flag is returned.

Invocation:

```
CALL KPG1_RFCOx( FD, DIM1, DIM2, POSCOD, COUNT, CODATA, LBND, :  UBND, CMPLET, STATUS
)
```

Arguments:

FD = INTEGER (Given)
Fortran file identifier.

DIM1 = INTEGER (Given)
The first dimension of the output array.

DIM2 = INTEGER (Given)
The second dimension of the output array.

POSCOD(DIM1) = INTEGER (Given)
The column numbers of the co-ordinate information in order x, y, z, etc. These must be positive.

COUNT = INTEGER (Given and Returned)
The number of the line in the output array to be written first. If the file has been completely read this becomes the number of data sets stored in the array (i.e. one less). It should be initialised externally the first time this routine is called, but not subsequently.

CODATA(DIM1, DIM2) = ? (Given and Returned)
The array to store the co-ordinates read from the text file. It should be initialised externally the first time this routine is called, but not subsequently.

LBND(DIM1) = ? (Given and Returned)
The lower bounds of the input data, i.e. the minimum value for each co-ordinate. It should be initialised externally the first time this routine is called, but not subsequently.

UBND(DIM1) = ? (Given and Returned)
The upper bounds of the input data, i.e. the maximum value for each co-ordinate. It should be initialised externally the first time this routine is called, but not subsequently.

CMPLET = LOGICAL (Returned)
If .TRUE., the text file has been completely read.

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

Notes:

- There is a routine for real and double-precision data types: replace "x" in the routine name by R or D respectively. The routine arguments CODATA, LBND and UBND must have the data type specified. _ The maximum permitted line length in the file is 256 characters. A line with a hash or shriek in column one is treated as a comment.

- The file is not closed on exit.
- The record number is not initialised and so this routine reads from the current line in the file. Hence this routine can be called repeatedly as the the output array is expanded to accommodate the input data.
- The numerical values can include embedded spaces following a +, -, D, or E. See SLALIB (SUN/67) routines SLA_DFLTIN and SLA_FLOTIN for details of the conversion from free-format to floating-point value. Missing columns that are not numeric cannot have these embedded spaces.

Prior Requirements :

- The Fortran text file must already be opened.

KPG1_RGLMT
Gets the range limits of a range specification

Description:

This routine returns begin and end limits of a range specified by a range word.

Invocation:

```
CALL KPG1_RGLMT( SCT, RNGWRD, BERNG, BELG, EDRNG, EDLG, STATUS )
```

Arguments:

SCT = CHARACTER∗(∗) (Given)

String which specifies a range with the range word specified by argument RNGWRD.

RNAWRD = CHARACTER∗(∗) (Given)

Gives a range word, such as " TO" , " -" , which is used in the range specification.

BERNG = INTEGER (Returned)

Begin limit of the range.

BELG = LOGICAL (Returned)

If the begin limit of the range has been omitted. it equals true, otherwise, false.

EDRNG = INTEGER (Returned)

End limit of the range.

EDLG = LOGINCAL (Returned)

If the end limit of the range has been omitted, it equals true, otherwise, false.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_RGNDF
Gets a group of existing NDFs

Description:

The supplied parameter is used to get a GRP group expression (see SUN/150) holding a list of existing NDFs (the syntax of the group expression is defined by the current default GRP control characters). If the group expression is flagged, then the current parameter value is cancelled, the string supplied in TEXT is displayed (if it is not blank) and another group expression is obtained. The NDFs specified by the second group expression are added to the group holding the NDFs specified by the first group expression. The group continues to be expanded in this way until a group expression is obtained which is not flagged, or a null value is given, or the limit on the number of NDFs (MAXSIZ) is reached. If any of the specified NDFs does not exist, the user is warned, and re-prompted. If the final group contains more than MAXSIZ NDFs, then all but the first MAXSIZ NDFs are removed from the group. The user is warned if this happens. If MAXSIZ is supplied with the value zero no limit is imposed on the number of NDFs within the group. If the final group contains fewer than the minimum number of NDFs specified by argument MINSIZ, then the user is asked to supply more NDFs. All messages issued by this routine have a priority level of MSG__NORM.

Invocation:

```
CALL KPG1_RGNDF( PARAM, MAXSIZ, MINSIZ, TEXT, IGRP, SIZE, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The parameter (of type LITERAL).

MAXSIZ = INTEGER (Given)
The maximum number of NDFs which can be allowed in the returned group. If zero is supplied, no limit is imposed.

MINSIZ = INTEGER (Given)
The minimum number of NDFs which can be allowed in the returned group. If zero is supplied, then the returned group may contain no NDFs.

TEXT = CHARACTER * (*) (Given)
The text to display between issuing prompts for successive group expressions. If blank then no text is displayed.

IGRP = INTEGER (Returned)
The GRP identifier for the returned group holding all the specified NDFs. The group should be deleted using GRP_DELET when it is no longer needed. If an error occurs, the value GRP__NOID is returned.

SIZE = INTEGER (Returned)
The number of files in the output group. SIZE is returned equal to 1 if STATUS is returned not equal to SAI__OK.

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

KPG1_RMAPx
Remaps an array's values according to an histogram key

Description:

This routine remaps all the values in an array on to new values, according to a key. The key locates element values with respect to an histogram that has been transformed from a linear one, i.e. has equal-sized bins. For example, in histogram equalisation the bin width varies so as to give approximately equal numbers of values in each bin.

Invocation:

```
CALL KPG1_RMAPx( EL, INARR, VALMAX, VALMIN, NUMBIN, MAP, OUTARR, STATUS )
```

Arguments:

EL = INTEGER (Given)
The dimension of the arrays.

INARR(EL) = ? (Given)
The array to be remapped.

VALMAX = ? (Given)
Maximum value data value used to form the linear histogram. Data values greater than this will be set bad in the output array.

VALMIN = ? (Given)
Minimum value data value used to form the linear histogram. Data values less than this will be set bad in the output array.

NUMBIN = INTEGER (Given)
Number of bins used in histogram. A moderately large number of bins is recommended so that there is little artifactual quantisation is introduced, say a few thousand except for byte data.

MAP(NUMBIN) = INTEGER (Given)
The key to the transform of the linear histogram. MAP stores the new bin number for each of the linear bin numbers. A bin number in the linear histogram is defined by the fractional position between the minimum and maximum data values times the number of bins.

OUTARR(EL) = ? (Returned)
The array containing the remapped values.

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays and the histogram limits supplied to this routine must have the data type specified.

References :

Gonzalez, R.C. and Wintz, P., 1977, " Digital Image Processing" , Addison-Wesley, pp. 118–126.

KPG1_RMSx
Finds the RMS of the good data values in an array

Description:
Finds the RMS of the good data values in an array.

Invocation:

```
CALL KPG1_RMSx( N, DATA, RMS, STATUS )
```

Arguments:

N = INTEGER (Given)
Number of elements in the array.

DATA(N) = ? (Given)
The data array.

RMS = ? (Returned)
The RMS value. Set to VAL__BAD<T> if no good data are found.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each of the standard numeric types. Replace "x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The array and RMS arguments supplied to the routine must have the data type specified.
- The summation is carried out in double precision.

KPG1_RNORM
Returns a set of random samples from a normal distribution

Description:

This routine returns a set of random samples from a normal distribution with mean=0.0 and standard deviation=1.0. It uses the default GSL random number generator type.

Invocation:

```
CALL KPG1_RNORM( EL, ARRAY, SEED, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of samples to return.

ARRAY(EL) = DOUBLE PRECISION (Returned)
The array in which to return the values.

SEED = INTEGER (Given)
The seed to use. If zero or negative, the value of environment variable STAR_SEED is used if set, and a non-repeatable value is used if STAR_SEED is not set.

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

KPG1_SATKC
Substitutes alphabetic (TRANSFORM) character tokens into a string

Description:

This routine parses the expression in EXPRES looking for tokens of the name PREFIX//[A-Z]. If one is located an attempt to access a value for this tokens if made using the ADAM parameter PREFIX//[A-Z]. If a value is obtained then it is substituted into the string EXPRES.

New character tokens (functions) may contain references to other character tokens which will be either prompted for, or, if it is a token which has already been given a value this will be substituted.

Invocation:

```
CALL KPG1_SATKC( PREFIX, EXPRES, STATUS )
```

Arguments:

PREFIX = CHARACTER * (*) (Given)

The prefix of the tokens. Valid tokens are ones with any trailing single alphabetic character.

EXPRES = CHARACTER * (*) (Given and Returned)

On entry this contains a TRANSFORM algebraic-like expression which may contain tokens which need to be substituted by other expressions (functions). References to other functions within functions are allowed and prompts will be made until all tokens are absent from the expression, however, later uses of the same tokens will be replaced with the same value.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_SATKD
Substitutes alphabetic (TRANSFORM) numeric tokens into a string

Description:

This routine parses the expression in EXPRES looking for tokens of the name PREFIX//[A-Z]. If one is located an attempt to access a value for this tokens if made using the ADAM parameter PREFIX//[A-Z]. If a value is obtained then it is substituted into the string EXPRES.

Invocation:

```
CALL KPG1_SATKD( PREFIX, EXPRES, STATUS )
```

Arguments:

PREFIX = CHARACTER * (*) (Given)

The prefix of the tokens. Valid tokens are ones with any trailing single alphabetic character.

EXPRES = CHARACTER * (*) (Given and Returned)

On entry this contains a TRANSFORM algebraic-like expression which may contain tokens which need to be substituted either for values (constants).

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_SAXAT
Sets attributes of a Frame axis to describe an NDF array component

Description:

This routine sets the Symbol, Units and Label attributes of a specified axis in an AST Frame so that they describe the values in a specified array component of an NDF. The Symbol is set to the name of the NDF array component (note, this is short and has no spaces so that it can be used as a column name in a catalogue), the Label is set to the NDF Label component, and Units is set to the NDF Units component.

Invocation:

```
CALL KPG1_SAXAT( INDF, COMP, AXIS, LOG, FRAME, STATUS )
```

Arguments:

INDF = INTEGER (Given)
An identifier for the NDF being described.

COMP = CHARACTER * (*) (Given)
The name of the NDF array component being described.

AXIS = INTEGER (Given)
The index of the Frame axis to be modified.

LOG = LOGICAL (Given)
If .TRUE., then the attribute values are modified so that they are appropriate for a logarithmic axis. The modified attribute values are of the form " Log10(...)" where " ..." is the attribute value used for non-logarithmic axes. PGPLOT escape characters are included in the attribute values to produce a sub-scripted " 10" .

FRAME = INTEGER (Given)
The identifier for the AST Frame to be modified.

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

KPG1_SCALE
Determines nominal WCS axis scales in an array

Description:

This routine determines (and formats) the nominal scales of the WCS axes in the current Frame of the supplied FrameSet. The scales are determined at a selection of points within the supplied base Frame bounds, and the median scales are returned. At each point, the scale for a specific WCS axis is the WCS axis increment produced by moving a unit distance within the base Frame away from the point, along the WCS axis.

Invocation:

```
CALL KPG1_SCALE( IWCS, LBND, UBND, SCALE, VALUE, UNIT, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
The FrameSet.

LBND(*) = DOUBLE PRECISION (Given)
The lower bounds of the region within the base Frame of the supplied FrameSet in which the WCS scales are to be evaluated. The array should have one element for each base Frame axis.

UBND(*) = DOUBLE PRECISION (Given)
The upper bounds of the region within the base Frame of the supplied FrameSet in which the WCS scales are to be evaluated. The array should have one element for each base Frame axis.

SCALE(*) = DOUBLE PRECISION (Returned)
The returned WCS scales. Note, the scales for both celestial longitude and latitude axes are returned as an arc-distance in radians. The array should have one element for each WCS axis.

VALUE(*) = CHARACTER(*) (Returned)
The formatted WCS scales. Celestial axes are formatted as arc-seconds using a " G15.6" format. Time values are also formatted using G15.6 (the Format attribute in the current WCS Frame is ignored, since it may produce a calendar date), in what ever units are indicated in the current Frame. Other types of axes (including spectral axes) are formatted using the axis Format attribute in the current WCS Frame. The array should have one element for each WCS axis. Each element of the array should be at least 15 characters long. The returned text is left justified.

UNIT(*) = CHARACTER(*) (Returned)
Units strings that describe the values returned in VALUES. The array should have one element for each WCS axis.

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

KPG1_SCALX
Copies array values into another array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The input and output arrays can be of different data types.

Invocation:

```
CALL KPG1_SCALX( SCALE, ZERO, BAD, EL, ITYPE, IP1, OTYPE, IP2, BADOUT, NBAD, STATUS
)
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

ITYPE = CHARACTER * (*) (Given)
The HDS data type of the input array. All primitive HDS data types are supported.

IP1 = INTEGER (Given)
Pointer to the input array.

OTYPE = CHARACTER * (*) (Given)
The HDS data type of the output array. Must be an integer type (i.e. one of _INTEGER, _INT64, _WORD, _UWORD, _BYTE or _UBYTE).

IP2 = INTEGER (Given)
Pointer to the output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- This function simply wraps up the generic KPG1_SCL<T><T> routines.

KPG1_SCLBx
Copies array values into a BYTE array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The output array is of type BYTE but the input array can be of a different type.

Invocation:

```
CALL KPG1_SCLBx( SCALE, ZERO, BAD, EL, IN, OUT, BADOUT, NBAD, STATUS )
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

IN(EL) = ? (Given)
The input array.

OUT(EL) = BYTE (Returned)
The output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The IN argument supplied must have the data type specified.

KPG1_SCLIx
Copies array values into a INTEGER array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The output array is of type INTEGER but the input array can be of a different type.

Invocation:

```
CALL KPG1_SCLIx( SCALE, ZERO, BAD, EL, IN, OUT, BADOUT, NBAD, STATUS )
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

IN(EL) = ? (Given)
The input array.

OUT(EL) = INTEGER (Returned)
The output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The IN argument supplied must have the data type specified.

KPG1_SCLKx
Copies array values into a INTEGER∗8 array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The output array is of type INTEGER∗8 but the input array can be of a different type.

Invocation:

```
CALL KPG1_SCLKx( SCALE, ZERO, BAD, EL, IN, OUT, BADOUT, NBAD, STATUS )
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

IN(EL) = ? (Given)
The input array.

OUT(EL) = INTEGER∗8 (Returned)
The output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, K, R, UB, UW, or W as appropriate. The IN argument supplied must have the data type specified.

KPG1_SCLOF
Applies a simple scaling and base-line shift to the values contained in the input vector

Description:

The input data values are multiplied by the given factor and the given offset is then added on, to form the output data.

Invocation:

```
CALL KPG1_SCLOF( EL, IN, FACTOR, OFFSET, OUT, NBAD, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the input and output vectors.

IN(EL) = REAL (Given)
The input data vector.

FACTOR = DOUBLE PRECISION (Given)
The factor by which the input valus are scaled.

OFFSET = DOUBLE PRECISION (Given)
The offset by which the data values are shifted.

OUT(EL) = REAL (Given)
The output data vector.

BAD = LOGICAL (Returned)
True if any bad pixels found.

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

KPG1_SCLUBx
Copies array values into a BYTE array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The output array is of type BYTE but the input array can be of a different type.

Invocation:

```
CALL KPG1_SCLUBx( SCALE, ZERO, BAD, EL, IN, OUT, BADOUT, NBAD, STATUS )
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

IN(EL) = ? (Given)
The input array.

OUT(EL) = BYTE (Returned)
The output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The IN argument supplied must have the data type specified.

KPG1_SCLUWx
Copies array values into a INTEGER∗2 array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The output array is of type INTEGER∗2 but the input array can be of a different type.

Invocation:

```
CALL KPG1_SCLUWx( SCALE, ZERO, BAD, EL, IN, OUT, BADOUT, NBAD, STATUS )
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

IN(EL) = ? (Given)
The input array.

OUT(EL) = INTEGER∗2 (Returned)
The output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The IN argument supplied must have the data type specified.

KPG1_SCLWx
Copies array values into a INTEGER∗2 array, scaling them in the process

Description:

This routine copies values from one array to another, applying a linear scaling in the process. The output array is of type INTEGER∗2 but the input array can be of a different type.

Invocation:

```
CALL KPG1_SCLW<T>( SCALE, ZERO, BAD, EL, IN, OUT, BADOUT, NBAD, STATUS )
```

Arguments:

SCALE = DOUBLE PRECISION (Given)
The scale factor.

ZERO = DOUBLE PRECISION (Given)
The zero offset.

BAD = LOGICAL (Given)
If true there may be bad pixels present in the input array. If false it is safe not to check for bad values.

EL = INTEGER (Given)
The size of the input array.

IN(EL) = ? (Given)
The input array.

OUT(EL) = INTEGER∗2 (Returned)
The output array.

BADOUT = LOGICAL (Returned)
True if there are any bad pixels in the output array.

NBAD = INTEGER (Returned)
The number of good input pixels that could not be accomodated within the dynamic range of the output data type, and were consequently set bad in the output array.

STATUS = INTEGER (Given and Returned)
Global status value

Notes:

- There is a routine for the all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The IN argument supplied must have the data type specified.

KPG1_SDIMP
Obtains up to a number of significant dimensions of an NDF

Description:

This routine finds the dimensions which are significant in an NDF, i.e. those with greater than one element. The significant dimensions are recorded and returned. If the number of significant dimensions found is less than a specified value, the insignificant dimensions pad out the array of dimension indices returned; and all the dimensions are returned in order of increasing dimensionality. However, should the number of significant dimensions exceed the required number a bad status, SAI_ERROR, is returned. Likewise there is an error when there are no significant dimensions.

Invocation:

```
CALL KPG1_SDIMP( NDF, NDIM, DIMV, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

NDIM = INTEGER (Given)
The desired number of dimensions.

DIMV(NDIM) = INTEGER (Returned)
The significant dimensions i.e. the ones that are greater than one. There is an execption when there are fewer than NDIM present in the NDF, whereupon this array includes in dimension order those that are insignificant too.

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

Prior Requirements :

- The NDF must exist.

KPG1_SECSH
Shifts bounds of upper dimensions that have one element to index one

Description:

This routine shifts the origin to the default of one of each dimension of an NDF greater than a nominated dimension and whose size is one.

The routine is required in applications that require NDFs of a specific dimensionality and which also call NDF_SECT. This is because a user-defined section may be from an NDF of higher dimensionality, and a subsequent call to NDF_SECT can result in an array of bad data, when the bounds of higher dimensions are not one. It is intended to be used alongside KPG1_SGDIM.

Invocation:

```
CALL KPG1_SECSH( NDF, MXDIM, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

MXDIM = INTEGER (Given)
The dimension above which the shifts are to be made. This should be the last significant bound for the required dimensionality of NDF.

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

KPG1_SEED
Obtains a semi-random seed for random-number generation

Description:

This function uses the computer time from an arbitrary date, plus the current process id., to generate a non-repeatable seed.

Alternatively, a fixed seed can be used by assigning the seed value to environment variable STAR_SEED.

Invocation:

```
RESULT = KPG1_SEED( STATUS )
```

Arguments:

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

Returned Value:

KPG1_SEED = REAL
The seed for use in SLA_RANDOM. Note that it is not necessarily in the range 0 to 1.

KPG1_SGDIG
Determines the number of significant digits in a number

Description:

This routine takes a string containing a numerical value, and counts the number of significant digits in the value. Thus leading and trailing zeroes are excluded.

Invocation:

```
CALL KPG1_SGDIG( STRING, NSDIG, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)

The string containing the numerical value. An error is returned if the value cannot be converted to a double-precision value without error.

NSDIG = INTEGER (Returned)

The number of significant digits in the string if it is treated as a numerical value.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_SGDIM
Determines the number of significant dimensions in an NDF

Description:

This routine finds the number of significant dimensions, i.e. those with greater than one element, in an NDF. If the number found is not equal to a specified number a bad status, SAI_ERROR, is returned. Likewise should there be no significant dimensions. The significant dimensions are recorded and returned.

Invocation:

```
CALL KPG1_SGDIM( NDF, NDIM, DIMV, STATUS )
```

Arguments:

NDF = INTEGER (Given)
The NDF identifier.

NDIM = INTEGER (Given)
The desired number of dimensions.

DIMV(NDIM) = INTEGER (Returned)
The significant dimensions i.e. the ones that are greater than one.

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

Prior Requirements :

- The NDF must exist.

KPG1_SHORT
Checks whether a string matches a supplied abbreviation template

Description:

This routine returns .TRUE. if the supplied test string matches the supplied abbreviation template. All characters in the template must be matched by the corresponding characters in the test string, with the exception that it is permissable for the test string to end anywhere following the first occurrence of the mark string (the mark string itself should not be matched in the test string). The template sub-string in front of the first mark string thus gives the minimum abbreviation which the test string can use.

For instance, if MARK='*' and TEMPLT='VECT*ORS', then the test string matches if the first 4 characters are 'VECT' and any remaining characters match 'ORS'.

Invocation:

```
RESULT = KPG1_SHORT( TEMPLT, TEST, MARK, CASE, STATUS )
```

Arguments:

TEMPLT = CHARACTER * (*) (Given and Returned)
The abbreviation template. Trailing blanks are ignored.

TEST = CHARACTER * (*) (Given and Returned)
The test string. Trailing blanks are ignored.

MARK = CHARACTER * (*) (Given and Returned)
The string which marks the end of the minimum abbreviation in TEMPLT. Trailing blanks are ignored.

CASE = LOGICAL (Given)
Should the match be case sensitive?

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

Function Value :

KPG1_SHORT = LOGICAL Returned .TRUE. if and only if the supplied test string matches the template.

KPG1_SLICE
Finds and removes any NDF slice specification from a name

Description:

A slice specification is taken to be anything between the first opening and the first closing parenthesis, so long as the closing parentheis is the last character in the name.

Invocation:

```
CALL KPG1_SLICE( NAME, SLICE, START, STATUS )
```

Arguments:

NAME = CHARACTER (Given and Returtned)

The name to be checked. On exit, any NDF slice specification contained in the name on entry, is removed.

SLICE = CHARACTER (Returned)

The slice specification including opening and closing parenthesise. If the input name contains no slice specification, then SLICE is returned blank.

START = INTEGER (Returned)

The position (within the original name) of the opening parethesis. If the name does not containa slice specification, then START is returned pointing to the first character beyond the end of the name.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_SNKTA
Writes AST_ data as text to a GRP group

Description:

This is a service routine to be provided as a " sink" routine for the AST_CHANNEL function. It takes data in the form of text (in response to writing an AST_ object to a Channel) and delivers it to a GRP group for storage.

This routine has only a STATUS argument, so it communicates with other KPG routines via global variables stored in the KPG_AST common blocks. These are described below under " Global Variables used as Arguments" .

Invocation:

```
CALL KPG1_SNKTA( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Global Variables used as Arguments :

ASTGRP = INTEGER (Given) A GRP identifier for the group which is to store the data. ASTTSZ = INTEGER (Given) The maximum length of text which should be stored in a single element of the group. This should be less than or equal to GRP__SZNAM. ASTLN = INTEGER (Given and Returned) This must initially be set to the value 1, to indicate that data will be written starting at the first element of the group (note the routine will not operate correctly unless 1 is the initial value - you cannot start writing at another point in the group if you have previously written to a different group). On exit it will be incremented by the number of elements used to store data, so that it identifies the first element to be used on the next invocation.

KPG1_SQSUx
Finds the sum of the squares of an array

Description:

This routine sums the squares of a supplied array and return the sum. This might used to sum the residuals for a minimisation. Bad values are ignored.

Invocation:

```
CALL KPG1_SQSUx( EL, ARRAY, SUMSQ, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the array to sum.

ARRAY(EL) = ? (Given)
The array whose sqaured values are to be summed.

SUMSQ = ? (Returned)
The sum of the squared array values.

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

Notes:

There is a routine for integer, real, and double precision data types: replace " x" in the routine name by I, R, or D respectively. The ARRAY and SUMSQ arguments supplied to the routine must have the data type specified.

KPG1_SRCTA
Reads AST_ data as text from a GRP group

Description:

This is a service routine to be provided as a " source" routine for the AST_CHANNEL function. It reads data from a GRP group (in response to reading from an AST_ Channel) and delivers it to the AST_ library for interpretation.

This routine has only a STATUS argument, so it communicates with other KPG routines via global variables stored in the KPG_AST common blocks. These are described below under " Global Variables used as Arguments" .

Invocation:

```
CALL KPG1_SRCTA( STATUS )
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Global Variables used as Arguments :

ASTGRP = INTEGER (Given) A GRP identifier for the group which holds the data. ASTLN = INTEGER (Given and Returned) This must initially be set to the value 1, to indicate that data will be read starting at the first element of the group (note the routine will not operate correctly unless 1 is the initial value - you cannot start reading at another point in the group if you have previously read from a different group). On exit it will be incremented by the number of elements used to obtain data, so that it identifies the first element to be used on the next invocation. ASTTSZ = INTEGER (Given) The number of characters to use from each GRP element.

KPG1_SSAZx
Applies a simple scaling and base-line shift to create the output vector

Description:

Pixel indices are multiplied by the given factor and the given offset is then added on, to form the output data. The first pixel has a value equal to the offset.

Invocation:

```
CALL KPG1_SSAZx( EL, FACTOR, OFFSET, OUT, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number elements in the returned array.

FACTOR = DOUBLE PRECISION (Given)
The factor by which the array indices are scaled.

OFFSET = DOUBLE PRECISION (Given)
The offset by which the array indices are shifted.

OUT(EL) = ? (Returned)
The output data vector.

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

Notes:

- There is a routine for the data types real or double precision: replace " x" in the routine name by R or D respectively, as appropriate. The returned array should be of the same type.
- There is no exception handler if the evaluated value exceeds the machine floating-point range.

KPG1_SSCOF
Applies a simple scaling and base-line shift to create the output vector

Description:

Pixel indices are multiplied by the given factor and the given offset is then added on, to form the output data. The first pixel has value equal to the offset.

Invocation:

```
CALL KPG1_SSCOF( EL, FACTOR, OFFSET, OUT, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number elements in the returned array.

FACTOR = DOUBLE PRECISION (Given)
The factor by which the array indices are scaled.

OFFSET = DOUBLE PRECISION (Given)
The offset by which the array indices are shifted.

OUT(EL) = REAL (Given)
The output data vector.

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

Notes:

There is no exception handler if the evaluated value exceeds the machine floating-pint range.

KPG1_STATx
Computes simple statistics for an array

Description:

This routine computes simple statistics for an array, namely: the number of valid pixels, the minimum and maximum pixel values (and their positions), the pixel sum, the mean, and the standard deviation. Iterative K-sigma clipping may also be optionally applied.

Invocation:

```
CALL KPG1_STATx( BAD, EL, DATA, NCLIP, CLIP, NGOOD, IMIN, DMIN, IMAX, DMAX, SUM, MEAN,
STDEV, NGOODC, IMINC, DMINC, IMAXC, DMAXC, SUMC, MEANC, STDEVC, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether checks for bad pixels should be performed on the array being analysed.

EL = INTEGER (Given)
Number of pixels in the array.

DATA(EL) = ? (Given)
Array to be analysed.

NCLIP = INTEGER (Given)
Number of K-sigma clipping iterations to apply (may be zero).

CLIP(NCLIP) = REAL (Given)
Array of clipping limits for successive iterations, expressed as standard deviations.

NGOOD = INTEGER (Returned)
Number of valid pixels in the array before clipping.

IMIN = INTEGER (Returned)
Index where the pixel with the lowest value was (first) found before clipping.

DMIN = DOUBLE PRECISION (Returned)
Minimum pixel value in the array before clipping.

IMAX = INTEGER (Returned)
Index where the pixel with the highest value was (first) found before clipping.

DMAX = DOUBLE PRECISION (Returned)
Maximum pixel value in the array before clipping.

SUM = DOUBLE PRECISION (Returned)
Sum of the valid pixels before clipping.

MEAN = DOUBLE PRECISION (Returned)
Mean of the valid pixels before clipping.

STDEV = DOUBLE PRECISION (Returned)
Standard deviation of the valid pixels before clipping.

NGOODC = INTEGER (Returned)
Number of valid pixels in the array after clipping.

IMINC = INTEGER (Returned)
Index where the pixel with the lowest value was (first) found after clipping.

DMINC = DOUBLE PRECISION (Returned)
Minimum pixel value in the array after clipping.

IMAXC = INTEGER (Returned)
Index where the pixel with the highest value was (first) found after clipping.

DMAXC = DOUBLE PRECISION (Returned)
Maximum pixel value in the array after clipping.

SUMC = DOUBLE PRECISION (Returned)
Sum of the valid pixels after clipping.

MEANC = DOUBLE PRECISION (Returned)
Mean of the valid pixels after clipping.

STDEVC = DOUBLE PRECISION (Returned)
Standard deviation of the valid pixels after clipping.

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

Notes:

- There is a routine for each of the standard numeric types. Replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The data type of the array being analysed must match the particular routine used.
- If no clipping is performed (i.e. if NCLIP = 0) then the values of arguments which return results after clipping will be the same as for those returning results before clipping.
- If NGOOD or NGOODC is zero, then the values of all the derived statistics will be undefined and will be set to the " bad" value appropriate to their data type (except for the pixel sum, which will be zero).

KPG1_STDSx
Displays statistics generated by KPG1_STATx, KPG_OSTAx, and KPG1_HSTAx

Description:

This routine displays the statistics of pixel data as generated by the routines KPG1_STATx, KPG_OSTAx, and KPG1_HSTAx.

Invocation:

```
CALL KPG1_STDSx( IWCS, NDIM, EL, NGOOD, DMIN, MINP, MINC, DMAX, MAXP, MAXC, SUM, MEAN,
STDEV, SKEW, KURTOS, MEDIAN, MODE, NUMPER, PERCNT, PERVAL, MAXWCS, MINWCS, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
Pointer to WCS FrameSet.

NDIM = INTEGER (Given)
Number of dimensions of the array whose statistics are being displayed.

EL = INTEGER (Given)
Total number of pixels in the array.

NGOOD = INTEGER (Given)
Number of valid pixels which contributed to the statistics.

DMIN = DOUBLE PRECISION (Given)
Minimum pixel value.

MINP(NDIM) = INTEGER (Given)
Pixel indices at which the minimum pixel value was (first) found.

MINC(NDIM) = DOUBLE PRECISION (Given)
The co-ordinate values of the centre of the minimum pixel.

DMAX = DOUBLE PRECISION (Given)
Maximum pixel value.

MAXP(NDIM) = INTEGER (Given)
Pixel indices at which the maximum pixel value was (first) found.

MAXC(NDIM) = DOUBLE PRECISION (Given)
The co-ordinate values of the centre of the maximum pixel.

SUM = DOUBLE PRECISION (Given)
Pixel sum.

MEAN = DOUBLE PRECISION (Given)
Pixel mean value.

STDEV = DOUBLE PRECISION (Given)
Pixel standard deviation value.

SKEW = DOUBLE PRECISION (Given)
Skewness of the pixel values.

KURTOS = DOUBLE PRECISION (Given)
Pixel kurtosis.

MEDIAN = DOUBLE PRECISION (Given)
Median value.

MODE = DOUBLE PRECISION (Given)
Modal value.

NUMPER = INTEGER (Given)
Number of percentiles values to report.

PERCNT(NUMPER) = REAL (Given)
The array of percentiles levels corresponding to the values given by PERVAL. They should be in the range 0.0 to 100.0, and preferably in ascending order. If there are none to report, set NUMPER to 1 and pass the bad value in PERCNT(1).

PERVAL(NUMPER) = DOUBLE PRECISION (Given)
Percentile values corresponding to the percentile fractions in PERCNT.

MAXWCS = CHARACTER * (*) (Returned)
The formatted WCS co-ordinates at the minimum position.

MINWCS = CHARACTER * (*) (Returned)
The formatted WCS co-ordinates at the minimum position.

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

Notes:

- There is a routine for both real and double-precision statistics: replace " x" in the routine name by R or D as appropriate. Note that the two routines have identical argument lists (with floating-point qualitities in double precision) but differ in the precision with which the results are displayed.
- If the value of NGOOD is not positive, then this routine will assume that all the derived statistics (except for the sum) are undefined and will not display them.
- If a statistic is undefined, i.e. has the bad value, it is not reported. In the case of arrays, the first value is checked. For the co-ordinates and pixel indices of the extreme values both of the first elements of these must be good to display these positions.

KPG1_STFLx
Writes statistics generated by KPG1_STATx, KPG_OSTAx, and KPG1_HSTAx to a text file

Description:

This routine writes the statistics of pixel data as generated by the routines KPG1_STATx, KPG_OSTAx, and KPG1_HSTAx to a text file.

Invocation:

```
CALL KPG1_STFLx( IWCS, NDIM, EL, NGOOD, DMIN, MINP, MINC, DMAX, MAXP, MAXC, SUM, MEAN,
STDEV, SKEW, KURTOS, MEDIAN, MODE, NUMPER, PERCNT, PERVAL, IFIL, STATUS )
```

Arguments:

IWCS = INTEGER (Given)
Pointer to WCS FrameSet.

NDIM = INTEGER (Given)
Number of dimensions of the array whose statistics are being displayed.

EL = INTEGER (Given)
Total number of pixels in the array.

NGOOD = INTEGER (Given)
Number of valid pixels which contributed to the statistics.

DMIN = DOUBLE PRECISION (Given)
Minimum pixel value.

MINP(NDIM) = INTEGER (Given)
Pixel indices at which the minimum pixel value was (first) found.

MINC(NDIM) = DOUBLE PRECISION (Given)
The co-ordinate values of the centre of the minimum pixel.

DMAX = DOUBLE PRECISION (Given)
Maximum pixel value.

MAXP(NDIM) = INTEGER (Given)
Pixel indices at which the maximum pixel value was (first) found.

MAXC(NDIM) = DOUBLE PRECISION (Given)
The co-ordinate values of the centre of the maximum pixel.

SUM = DOUBLE PRECISION (Given)
Pixel sum.

MEAN = DOUBLE PRECISION (Given)
Pixel mean value.

STDEV = DOUBLE PRECISION (Given)
Pixel standard deviation value.

SKEW = DOUBLE PRECISION (Given)
Skewness of the pixel values.

KURTOS = DOUBLE PRECISION (Given)
Pixel kurtosis.

MEDIAN = DOUBLE PRECISION (Given)
Median value.

MODE = DOUBLE PRECISION (Given)
Modal value.

NUMPER = INTEGER (Given)
Number of percentiles values to report.

PERCNT(NUMPER) = REAL (Given)
The array of percentiles levels corresponding to the values given by PERVAL. They should be in the range 0.0 to 100.0, and preferably in ascending order. If there are none to report, set NUMPER to 1 and pass the bad value in PERCNT(1).

PERVAL(NUMPER) = DOUBLE PRECISION (Given)
Percentile values corresponding to the percentile fractions in PERCNT.

IFIL = INTEGER (Given)
FIO_ file descriptor for the output file.

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

Notes:

- There is a routine for both real and double-precision statistics: replace " x" in the routine name by R or D as appropriate. Note that the two routines have identical argument lists (with floating-point qualitities in double precision) but differ in the precision with which the results are displayed.
- If the value of NGOOD is not positive, then this routine will assume that all the derived statistics (except for the sum) are undefined and will not display them.
- If a statistic is undefined, i.e. has the bad value, it is not reported. In the case of arrays, the first value is checked. For the co-ordinates and pixel indices of the extreme values both of the first elements of these must be good to display these positions.

KPG1_STORx
Stores a value in an array

Description:
The supplied value is stored in the array at the given index.

Invocation:
```
CALL KPG1_STORx( EL, INDEX, VALUE, DATA, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of elements in the array.

INDEX = INTEGER (Given)
The index at which to store the supplied value.

VALUE = ? (Given)
The value to be stored in the array.

DATA(EL) = ? (Given and Returned)
The array.

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

Notes:

- There is a routine for all numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The VALUE and DATA arguments must have the data type specified.

KPG1_TDLIx
Applies a constant-determinant transformation to an array by linear interpolation

Description:

This routine creates a new n-dimensional array from an input m-dimensional array by applying a constant-determininant transformation to all the elements of the output array. Each output value is calculated by linear interpolation between the elements in the input array that surround each transformed co-ordinate. The transformation must convert from output co-ordinates to input pixel indices. The two arrays may have different numbers of dimensions. In addition a variance array may be created likewise.

This operates with double-precision co-ordinates. Use KPG1_TRLIx for single precision.

Invocation:

```
CALL KPG1_TDLIx( NDIMI, IDIMS, INARR, VAR, INVAR, TRID, FLUX, AXES, OEL, NDIMO, OLBND,
ODIMS, OUTARR, OUTVAR, COIN, COOUT, INDICE, STATUS )
```

Arguments:

NDIMI = INTEGER (Given)
The dimensionality of the input arrays. It must be greater than one. To handle a one-dimensional array, give it a second dummy dimension of 1.

IDIMS(NDIMI) = INTEGER (Given)
The dimensions of the input n-D arrays.

INARR($*$) = ? (Given)
The input n-D data array.

VAR = LOGICAL (Given)
If VAR is .TRUE. there is a variance array to create from an input variance array.

INVAR($*$) = ? (Given)
The input n-D variance array. When VAR is .FALSE., this can contain an arbitrary number of elements. When VAR is .TRUE. it must have the shape of the input data array.

FLUX = DOUBLE PRECISION (Given)
The factor to multiply the values in the output arrays to preserve the flux. This will be the determinant of the transformation. Set this to 1.0 if no flux conservation is required.

AXES($*$) = DOUBLE PRECISION (Given)
The concatenated axis co-ordinates of the input array. This array should therefore have a dimension at least as large as the sum of the input array's dimensions.

OEL = INTEGER (Given)
The first dimension of the work arrays. It should be at least ODIMS(1).

NDIMO = INTEGER (Given)
The dimensionality of the output arrays.

OLBND(NDIMO) = INTEGER (Given)
The lower bounds of the output n-D arrays.

ODIMS(NDIMO) = INTEGER (Given)
The dimensions of the output n-D arrays.

OUTARR(*) = ? (Returned)
The transformed data array.

OUTVAR(*) = ? (Returned)
The variance array of the transformed data.

COIN(OEL, NDIMI) = DOUBLE PRECISION (Returned)
Workspace used to store the co-ordinates of a row of points in the input arrays.

COOUT(OEL, NDIMO) = DOUBLE PRECISION (Returned)
Workspace used to store the co-ordinates of a row of points in the output arrays.

INDICE(OEL, NDIMI) = DOUBLE PRECISION (Returned)
Workspace used to store the floating-point pixel indices of a row of points in the input arrays.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The input and output data and variance arrays must have the data type specified.
- There is no protection against overflows when the absolute data values are very large.

Bugs:
{note_bugs_here}

KPG1_THRSx
Sets pixels in array to defined new values outside limits

Description:

This routine takes an array and sets all values above a defined upper threshold to a new defined value, and sets all those below a defined lower threshold to another defined value. In practice, all values outside the two thresholds may be set to zero or the bad value, for example.

If the lower threshold is greater than or equal to the upper threshold, the values between the thresholds are set to the supplied lower replacement value, and the upper replacement value is ignored. In this case the number of replaced pixels is returned in NREPLO, and NREPHI is returned as -1.

Invocation:

```
CALL KPG1_THRSx( BAD, EL, INARR, THRLO, THRHI, NEWLO, NEWHI, OUTARR, NREPLO, NREPHI,
STATUS )
```

Arguments:

BAD = LOGICAL (Given)
The bad-pixel flag. If it is .TRUE., tests are made for bad array values. When .FALSE., no tests are made for bad values, and any encountered are treated literally.

EL = INTEGER (Given)
Dimension of the input and output arrays.

INARR(EL) = ? (Given)
Input data to be thresholded.

THRLO = ? (Given)
Upper threshold level.

THRHI = ? (Given)
Lower threshold level.

NEWLO = ? (Given)
Value to which pixels below THRLO will be set.

NEWHI = ? (Given)
Value to which pixels above THRHI will be set.

NREPLO = ? (Returned)
The number of values less than the lower threshold and substituted.

NREPHI = ? (Returned)
The number of values greater than the upper threshold and substituted.

OUTARR(EL) = ? (Returned)
Output thresholded data.

STATUS = INTEGER (Given)
Global status.

Notes:

- There is a routine for each numeric data type: replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays and values supplied to the routine must have the data type specified.

KPG1_TRALx
Finds the extreme co-ordinates of an n-d array after being transformed

Description:

This routine applies a forward mapping to the co-ordinates of all pixel corners in an n-dimensional array in order to find the limits of the array after a transformation (forward mapping) has been applied. It assumes equally spaced co-ordinates which will be true for pixel co-ordinates and many sets of data co-ordinates, but should be adequate even for unevenly spaced data co-ordinates. It will only give poor results for strange transformations with singularities.

Invocation:

```
CALL KPG1_TRALx( NDIMI, IDIMS, LBND, UBND, TRID, IEL, NDIMO, COIN, COOUT, COMIN, COMAX,
STATUS )
```

Arguments:

NDIMI = INTEGER (Given)
The dimensionality of the input array.

IDIMS(NDIMI) = INTEGER (Given)
The dimensions of the associated input n-dimensional data array.

LBND(NDIMI) = ? (Given)
The co-ordinates of the lower bounds of the associated input n-dimensional array.

UBND(NDIMI) = ? (Given)
The co-ordinates of the upper bounds of the associated input n-dimensional array.

TRID = INTEGER (Given)
The TRANSFORM identifier of the mapping.

IEL = INTEGER (Given)
The first dimension of the work arrays. It should be at least IDIMS(1) + 1.

NDIMO = INTEGER (Given)
The dimensionality of the output array.

COIN(IEL, NDIMI) = DOUBLE PRECISION (Returned)
Workspace used to store the co-ordinates of a row of points in the input array.

COOUT(IEL, NDIMO) = DOUBLE PRECISION (Returned)
Workspace used to store the co-ordinates of a row of points in the output array.

COMIN(NDIMO) = ? (Returned)
The minimum co-ordinate along each dimension of the output array.

COMAX(NDIMO) = ? (Returned)
The maximum co-ordinate along each dimension of the output array.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace "`x`" in the routine name by D or R as appropriate. The input co-ordinate work arrays and the output limiting-co-ordinate arrays must have the data type specified.

Bugs:

{note_bugs_here}

KPG1_TRBOx
Finds the extreme co-ordinates of an n-d array after being transformed

Description:

This routine applies a forward mapping to the pixel co-ordinates of test points in an n-dimensional array in order to find the limits of the array after a transformation (forward mapping) has been applied. The test points are the vertices and the midpoints between them.

Invocation:

```
CALL KPG1_TRBOx( NDIMI, LBND, UBND, TRID, NDIMO, COMIN, COMAX, STATUS )
```

Arguments:

NDIMI = INTEGER (Given)
The dimensionality of the input array.

LBND(NDIMI) = ? (Given)
The co-ordinates of the lower bounds of the input n-dimensional array.

UBND(NDIMI) = ? (Given)
The co-ordinates of the upper bounds of the input n-dimensional array.

TRID = INTEGER (Given)
The TRANSFORM identifier of the mapping.

NDIMO = INTEGER (Given)
The dimensionality of the output array.

COMIN(NDIMO) = ? (Returned)
The minimum co-ordinate along each dimension of the output array.

COMAX(NDIMO) = ? (Returned)
The maximum co-ordinate along each dimension of the output array.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by D or R as appropriate. The input co-ordinate bounds and the output limiting-co-ordinate arrays must have the data type specified.

Bugs:

{note_bugs_here}

KPG1_TRIGx
Applies a trigonometric function to each element of a vectorised array

Description:

The routine applies a specified trigonometric function to each element of a vectorised array. Any associated variance values are also modified appropriately.

Invocation:

```
CALL KPG1_TRIGx( BAD, VAR, TRIGFN, EL, DIN, VIN, DOUT, VOUT, NBAD, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether to check for bad values in the input arrays.

VAR = LOGICAL (Given)
Have associated variances been supplied?

TRIGFN = CHARACTER∗(∗) (Given)
The required trig function. This should be one of SIN, COS, TAN, SIND, COSD, TAND, ASIN, ACOS, ATAN, ASIND, ACOSD, ATAND.

EL = INTEGER (Given)
Number of array elements to process.

DIN(EL) = ? (Given)
Input data array.

VIN(EL) = ? (Given)
Input variance array. Only accessed if VAR is .TRUE.

DOUT(EL) = ? (Given)
Output data array.

VOUT(EL) = ? (Given)
Output variance array. Only accessed if VAR is .TRUE.

NBAD(2) = INTEGER (Returned)
Element 1 has the number of bad values stored in DOUT, and Element 2 has the number of bad values stored in VOUT.

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

Notes:

- There is a routine for real and double-precision data types: replace " x" in the routine name by R or D respectively. The input and output data and variance arrays supplied to the routine must have the data type specified.

KPG1_TRLIx
Applies a constant-determinant transformation to an array by linear interpolation

Description:

This routine creates a new n-dimensional array from an input m-dimensional array by applying a constant-determininant transformation to all the elements of the output array. Each output value is calculated by linear interpolation between the elements in the input array that surround each transformed co-ordinate. The transformation must convert from output co-ordinates to input pixel indices. The two arrays may have different numbers of dimensions. In addition a variance array may be created likewise.

This routine operates with single-precision co-ordinates. Use KPG1_TDLIx for double precision.

Invocation:

```
CALL KPG1_TRLIx( NDIMI, IDIMS, INARR, VAR, INVAR, TRID, FLUX, AXES, OEL, NDIMO, OLBND,
ODIMS, OUTARR, OUTVAR, COIN, COOUT, INDICE, STATUS )
```

Arguments:

NDIMI = INTEGER (Given)
The dimensionality of the input arrays. It must be greater than one. To handle a one-dimensional array, give it a second dummy dimension of 1.

IDIMS(NDIMI) = INTEGER (Given)
The dimensions of the input n-D arrays.

INARR($*$) = ? (Given)
The input n-D data array.

VAR = LOGICAL (Given)
If VAR is .TRUE. there is a variance array to create from an input variance array.

INVAR($*$) = ? (Given)
The input n-D variance array. When VAR is .FALSE., this can contain an arbitrary number of elements. When VAR is .TRUE. it must have the shape of the input data array.

FLUX = DOUBLE PRECISION (Given)
The factor to multiply the values in the output arrays to preserve the flux. This will be the determinant of the transformation. Set this to 1.0 if no flux conservation is required.

AXES($*$) = REAL (Given)
The concatenated axis co-ordinates of the input array. This array should therefore have a dimension at least as large as the sum of the input array's dimensions.

OEL = INTEGER (Given)
The first dimension of the work arrays. It should be at least ODIMS(1).

NDIMO = INTEGER (Given)
The dimensionality of the output arrays.

OLBND(NDIMO) = INTEGER (Given)
The lower bounds of the output n-D arrays.

ODIMS(NDIMO) = INTEGER (Given)
The dimensions of the output n-D arrays.

OUTARR(*) = ? (Returned)
The transformed data array.

OUTVAR(*) = ? (Returned)
The variance array of the transformed data.

COIN(OEL, NDIMI) = REAL (Returned)
Workspace used to store the co-ordinates of a row of points in the input arrays.

COOUT(OEL, NDIMO) = REAL (Returned)
Workspace used to store the co-ordinates of a row of points in the output arrays.

INDICE(OEL, NDIMI) = REAL (Returned)
Workspace used to store the floating-point pixel indices of a row of points in the input arrays.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace " x" in the routine name by B, D, I, R, UB, UW, or W as appropriate. The input and output data and variance arrays must have the data type specified.
- There is no protection against overflows when the absolute data values are very large.

Bugs:
{note_bugs_here}

KPG1_TRPIx
Finds vectorised pixel indices after applying a transformation to an array's pixel co-ordinates

Description:

This routine applies a transformation to the pixel co-ordinates of the elements of an (output) n-dimensional array of a specified shape. It then determines the nearest-neighbour element in the transformed (input) m-dimensional array, as a vector index, which is returned. This array of vector indices may then be used by another routine to fill the output array with values from the input array.

Invocation:

```
CALL KPG1_TRPIx( NDIMI, IDIMS, TRID, AXES, OEL, NDIMO, OLBND, ODIMS, COIN, COOUT, FRIND,
INDICE, STATUS )
```

Arguments:

NDIMI = INTEGER (Given)
The dimensionality of the input array.

IDIMS(NDIMI) = INTEGER (Given)
The dimensions of the input n-D array.

TRID = INTEGER (Given)
The TRANSFORM identifier of the mapping.

AXES(*) = ? (Given)
The concatenated axis co-ordinates of the input array. This array should therefore have a dimension at least as large as the sum of the input array's dimensions.

OEL = INTEGER (Given)
The first dimension of the work arrays. It should be at least ODIMS(1).

NDIMO = INTEGER (Given)
The dimensionality of the output array.

OLBND(NDIMO) = INTEGER (Given)
The lower bounds of the output n-D array.

ODIMS(NDIMO) = INTEGER (Given)
The dimensions of the output n-D array.

COIN(OEL, NDIMI) = ? (Returned)
Workspace used to store the co-ordinates of a row of points in the input array.

COOUT(OEL, NDIMO) = ? (Returned)
Workspace used to store the co-ordinates of a row of points in the output array.

FRIND(OEL, NDIMI) = ? (Returned)
Workspace used to store the floating-point pixel indices of a row of points in the input array.

INDICE(*) = INTEGER (Returned)
The vector indices of the nearest-array element of the transformed positions of the output array. This array should therefore have a dimension at least as large as the product of the output array's dimensions.

STATUS = INTEGER (Given and Returned).
Global status value

Notes:

- There is a routine for the following numeric data types: replace "`x`" in the routine name by D or R as appropriate. The workspace and axis arrays must have the data type specified.

Bugs:

{note_bugs_here}

KPG1_TRSPx
Transposes a two-dimensional array

Description:
This routine creates a new two-dimensional array containing a transposed copy the supplied array.

Invocation:
```
CALL KPG1_TRSPx( M, N, IN, OUT, STATUS )
```

Arguments:

M = INTEGER (Given)
Number of columns in the input array and the number of lines in the output array.

N = INTEGER (Given)
Number of lines in the input array and the number of columns in the output array.

IN(M, N) = ? (Given)
The input array to be transposed.

OUT(N, M) = ? (Returned)
The transposed array.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The arrays supplied to the routine must have the data type specified.

KPG1_UNZ2x
Unzips a two-dimensional co-ordinate array into two one-dimensional arrays

Description:

This routine takes an array of dimension 2 by EL elements and puts the two columns into separate arrays.

Invocation:

```
CALL KPG1_UNZ2x( EL, IN, OUT1, OUT2, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of lines in the input array, and elements in each of the output arrays.

IN(2, EL) = ? (Given)
The array to be 'unzipped'.

OUT1(EL) = ? (Returned)
The vector to contain first column of the input array.

OUT2(EL) = ? (Returned)
The vector to contain second column of the input array.

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

Notes:

- There is a routine for real and double-precision data types: replace "x" in the routine name by R or D respectively. The routine arguments IN, OUT1, and OUT2 must have the data type specified.

KPG1_VASVx
Assigns values to an output array from an input array using a list of indices

Description:

This routine assigns values to an output vector from an input vector. The values are selected using a list of indices in the input vector, there being one index per output value. A bad value or a value outside the bounds of the array in the list of indices causes a bad value to be assigned to the output array.

Invocation:

```
CALL KPG1_VASVx( OEL, INDICE, IEL, INARR, OUTARR, NBAD, STATUS )
```

Arguments:

OEL = INTEGER (Given)
The dimension of the output vector and also the list of indices.

INDICE(OEL) = INTEGER (Given)
The indices in the input array that point to the values to be assigned to the output vector.

IEL = INTEGER (Given)
The dimension of the input vector.

INARR(IEL) = ? (Given)
The vector containing values to be given to the output vector.

OUTARR(OEL) = ? (Returned)
The vector containing values copied from the input vector according to the list of indices.

NBAD = INTEGER (Returned)
The number of bad values in the output array.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The input and output vectors supplied to the routine must have the data type specified.

KPG1_VEC2N
Converts vectorised array indices into n-dimensional form

Description:

This routine converts pixel indices which refer to a pixel in a vectorised array into sets of indices which identify the same pixel when the array is regarded as n-dimensional, with specified lower and upper pixel-index bounds for each dimension.

Invocation:

```
CALL KPG1_VEC2N( NVEC, VEC, NDIM, LBND, UBND, IDIM, STATUS )
```

Arguments:

NVEC = INTEGER (Given)
Number of vectorised array indices to convert.

VEC(NVEC) = INTEGER (Given)
Array of vectorised pixel indices to be converted.

NDIM = INTEGER (Given)
Number of array dimensions.

LBND(NDIM) = INTEGER (Given)
Lower pixel-index bounds for each array dimension.

UBND(NDIM) = INTEGER (Given)
Upper pixel-index bounds for each array dimension.

IDIM(NDIM, NVEC) = INTEGER (Returned)
Returns a set of NDIM pixel-indices for each vectorised index supplied.

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

Notes:

The maximum number of dimensions which can be handled by this routine is equal to the symbolic constant NDF__MXDIM.

KPG1_VERB
Determines whether the specified package should report verbose messages

Description:

This routine returns a logical flag indicating if a specified applications package should report verbose information. This is the case if the environment variable <PACK>_VERBOSE is defined (the value assigned to the environment variable is immaterial).

Invocation:

```
CALL KPG1_VERB( VERB, PACK, STATUS )
```

Arguments:

VERB = LOGICAL (Returned)

Should the package run in verbose mode? Returned .FALSE, if an error has already occurred, or if this routine should fail for any reason.

PACK = CHARACTER * (*) (Given)

The name of the package (e.g. " KAPPA" , " POLPACK").

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- This routine attempts to execute even if STATUS is set to an error on entry.

KPG1_WCATW
Writes an AST Object to a catalogue

Description:

This routine stores a textual representation of the supplied AST Object within the supplied catalogue. The information is stored within COMMENT strings which are appended to the supplied catalogue's textual information. Lines of AST information which are too long to fit in a single COMMENT are split into several lines. Continuation lines are marked by having a "+" in the first column. Each line of AST information is preceeded with the string " !!" , which is used to mark the end of any leading blanks etc. added by AST when the string is read back.

Note, at the moment CAT reports errors if textual information is added to a catalogue which contains no rows of data. For this reason, this routine should normally be called just before closing the catalogue, since this will normally ensure that the catalogue contains some data.

Invocation:

```
RESULT = KPG1_WCATW( IAST, CI, STATUS )
```

Arguments:

IAST = INTEGER (Given)
An AST pointer to the Object.

CI = INTEGER (Given)
A CAT identifier for the catalogue.

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

Function Value :

KPG1_WCATW = INTEGER Returned equal to 1 if an Object was written to the catalogue, and zero otherwise.

KPG1_WCAXC
Obtains co-ordinates for an axis from a WCS component FrameSet

Description:

This routine returns an array of world co-ordinates along a nominated axis, that are derived from the Current Frame in the supplied FrameSet.

Invocation:

```
CALL KPG1_WCAXC( INDF, FS, AXIS, EL, CENTRE, STATUS )
```

Arguments:

INDF = INTEGER (Given)
The NDF identifier.

FS = INTEGER (Given)
An AST pointer for a FrameSet.

AXIS = INTEGER (Given)
The number of the axes array to obtain.

EL = INTEGER (Returned)
The number of elements in the centre array.

CENTRE(EL) = DOUBLE PRECISION (Returned)
The axis centres.

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

KPG1_WCFAX
Obtains the co-ordinate of an axis in the current WCS Frame at every pixel centre in an NDF

Description:

This routine returns the world co-ordinate along a nominated WCS axis for each pixel centre of an NDF. The co-ordinates are defined within the current Frame in the supplied FrameSet. It is assumed that the nominated axis is independent of the other axes.

Invocation:

```
CALL KPG1_WCFAX( LBND, UBND, MAP, JAXIS, IAXIS, CENTRE, WORK, STATUS )
```

Arguments:

LBND(*) = INTEGER (Given)

The lower pixel index bounds of the NDF. This array should have one element for each NDF pixel axis.

UBND(*) = INTEGER (Given)

The upper pixel index bounds of the NDF. This array should have one element for each NDF pixel axis.

MAP = INTEGER (Given)

Mapping from PIXEL coords in the NDF to current WCS coords.

JAXIS = INTEGER (Given)

The number of the pixel axis which is varied.

IAXIS = INTEGER (Given)

The number of the WCS axis whose values are returned.

CENTRE(*) = DOUBLE PRECISION (Returned)

The current-Frame co-ordinate along the nominated axis at each pixel centre. The size and shape of this array should be the same as the NDF.

WORK(*) = DOUBLE PRECISION (Returned)

A work array with the same length as the requested pixel axis.

STATUS = INTEGER (Given and Returned)

The global status.

KPG1_WGNDF
Gets a group of output NDF names

Description:

The supplied parameter is used to get a GRP group expression (see SUN/150) holding a list of NDFs which are to be created by the calling application (the syntax of the group expression is defined by the current default GRP control characters). Modification elements within the group expression are based on the group identified by IGRP0. If the group expression is flagged, then the current parameter value is cancelled, the string supplied in TEXT is displayed (if it is not blank) and another group expression is obtained. The NDFs specified by the second group expression are added to the group holding the NDFs specified by the first group expression. The group continues to be expanded in this way until a group expression is obtained which is not flagged, or a null value is given, or the limit on the number of NDFs (MAXSIZ) is reached. If the final group contains more than MAXSIZ NDFs, then all but the first MAXSIZ NDFs are removed from the group. The user is warned if this happens. If MAXSIZ is supplied with the value zero no limit is imposed on the number of NDFs within the group. If the final group contains less than MINSIZ NDFs then the user is told to supply more, and is re-prompted for further NDF names. All messages issued by this routine have a priority level of MSG__NORM.

Invocation:

```
CALL KPG1_WGNDF( PARAM, IGRP0, MAXSIZ, MINSIZ, TEXT, IGRP, SIZE, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
The parameter (of type LITERAL).

IGRP0 = INTEGER (Given)
The NDG identifier for a group containing a set of NDF names to be used as the basis for any modification elements contained within the group expressions. If this is supplied equal to GRP__NOID then modification elements are left un-expanded.

MAXSIZ = INTEGER (Given)
The maximum number of NDFs which can be allowed in the returned group. If zero is supplied, no limit is imposed.

MINSIZ = INTEGER (Given)
The minimum number of NDFs which can be allowed in the returned group. If zero is supplied, then the returned group may contain no NDFs.

TEXT = CHARACTER * (*) (Given)
The text to display between issuing prompts for successive group expressions. If blank then no text is displayed.

IGRP = INTEGER (Returned)
The NDG identifier for the returned group holding all the specified NDFs.

SIZE = INTEGER (Returned)
The number of files in the output group. Returned equal to 1 if STATUS is not equal to SAI__OK.

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

KPG1_WMODx
Estimates the mean of a number of normally distributed data values, some of which may be corrupt

Description:

The routine is based on maximising the likelihood function for a statistical model in which any of the data points has a constant probability of being corrupt. The data points have weights, to allow for different intrinisic errors. A weighted mean is chosen according to the deviation of each data point from the current estimate of the mean. The weights are derived from the relative probabilities of being valid or corrupt. A sequence of these iterations converges to a stationary point in the likelihood function. The routine approximates to a k-sigma clipping algorithm for a large number of data points and to a mode-estimating algorithm for fewer data points.

Invocation:

```
CALL KPG1_WMODx( X, W, NX, PBAD, NITER, TOLL, XMODE, SIGMA, :  STATUS )
```

Arguments:

X(NX) = ? (Given)
An array of data values.

W(NX) = REAL (Given)
An array of data weights for each data value. The weights are inversely proportional to the square of the relative errors on each data point.

NX = INTEGER (Given)
The number of data values.

PBAD = REAL (Given)
An estimate of the probability that any one data point will be corrupt. (This value is not critical.)

NITER = INTEGER (Given)
The maximum number of iterations required.

TOLL = REAL (Given)
The absolute accuracy required in the estimate of the mean. Iterations cease when two successive estimates differ by less than this amount.

XMODE = REAL (Returned)
The estimate of the uncorrupted mean.

SIGMA = REAL (Returned)
An estimate of the uncorrupted normalised standard deviation of the data points. An estimate of the standard deviation of any one point is: SIGMA / SQRT(W) where W is its weight.

STATUS = INTEGER (Returned)
The global status.

Notes:

There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B, UB as appropriate. The array supplied to the routine must have the data type specified.

KPG1_WRAST
Writes AST_ data as text to an HDS object

Description:

This is a service routine to be provided as a " sink" routine for the AST_CHANNEL function. It takes data in the form of text (in response to writing an AST_ object to a Channel) and delivers it to an HDS object for storage.

This routine has only a STATUS argument, so it communicates with other KPG routines via global variables stored in the KPG_AST common blocks. These are described below under " Global Variables used as Arguments" .

Invocation:

```
CALL KPG1_WRAST
```

Arguments:

STATUS = INTEGER (Given and Returned)

The global status.

Global Variables used as Arguments :

ASTLC = CHARACTER * (DAT__SZLOC) (Given) A locator for the HDS object which is to store the data. This must be a one-dimensional _CHAR array, whose initial size and character string length will be determined via this locator. Write access to the object must be available via this locator, but the locator itself is not altered by this routine. ASTLN = INTEGER (Given and Returned) This must initially be set to the value 1, to indicate that data will be written starting at the first element of the HDS array (note the routine will not operate correctly unless 1 is the initial value - you cannot start writing at another point in the array if you have previously written to a different array). On exit it will be incremented by the number of elements used to store data, so that it identifies the first element to be used on the next invocation. ASTPT = INTEGER (Given and Returned) A pointer to the contents of the HDS object, initially mapped in ' WRITE' mode. This pointer may be modified by the routine (and re-mapped in ' UPDATE' mode) if it needs to extend the size of the object to accommodate the data written.

KPG1_WRCAT
Writes a set of positions to a text file as a CAT catalogue

Description:

This routine is equivalent to KPG1_WRLST, except that it provides the option of storing a textual label with each position, via the extra argument LABS, and extra arbitrary columns via the extra argument KEYMAP.

Invocation:

```
CALL KPG1_WRCAT( PARAM, ARRDIM, NPOS, NAX, POS, IFRM, IWCS, TITLE, ID0, IDENTS, KEYMAP,
LABS, HIST, NULL, STATUS )
```

Arguments:

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

ARRDIM = INTEGER (Given)
The size of the first dimension of the positions array. This must be larger than or equal to NPOS.

NPOS = INTEGER (Given)
The number of positions to store in the file.

NAX = INTEGER (Given)
The number of axes in the Frame specified by IFRM.

POS(ARRDIM, NAX) = DOUBLE PRECISION (Given)
The positions to store in the file. POS(I, J) should give the axis J value for position I.

IFRM = INTEGER (Given)
The index of the Frame within IWCS to which the supplied positions relate. Can be AST__BASE or AST__CURRENT.

IWCS = INTEGER (Given)
A pointer to an AST FrameSet to store with the positions.

TITLE = CHARACTER * (*) (Given)
A title to store at the top of the text file. Ignored if blank.

ID0 = INTEGER (Given)
The integer identifier value to associate with the first supplied position. Identifiers for subsequent positions increase by 1 for each position. If this is supplied less than or equal to zero, then its value is ignored and the identifiers supplied in array IDENTS are used instead.

IDENTS(NPOS) = INTEGER (Given)
The individual integer identifiers to associate with each position. Only accessed if ID0 is less than or equal to zero.

KEYMAP = INTEGER (Given)
An optional AST KeyMap containing data for extra columns to add to the catalogue. It can be used (for instance) to add character columns to the catalogue. If a value of AST__NULL is supplied, no extra columns are added to the catalogue. Otherwise, the column names and values can be specified using either of the following two schemes:

- If the KeyMap contains an entry called " COLNAMES" , it is assumed to be a vector entry holding the names of the columns. For each of these column names, the value to store in a particular row of the catalogue is assumed to be held in a scalar KeyMap entry with key " <colname>_<row number>" (" _1" for the first row). If no such entry exists for a particular row, then the value is marked as bad in the catalogue. The data type of the column is determined from the first row value found for the column.
- If the KeyMap does not contain an entry called " COLNAMES" , it is assumed that each entry in the KeyMap contains a vector holding all the values for a single column, in row order. The entry key is used as the column name, and the column data type is determined from the entry data type.

LABS = INTEGER (Given)
A GRP group identifier containing the labels to be associated with the positions. The number of elements in this group should be equal to NPOS. If GRP__NOID is supplied, no label column will be created.

HIST = INTEGER (Given)
A GRP group identifier containing history text to store with the catalogue. If GRP__NOID is supplied, no history information will be stored with the catalogue.

NULL = LOGICAL (Given)
Is the user allowed to supply a null value? If so, the error status will be annulled before returning.

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

KPG1_WREAD
Reads an AST Object from an HDS object

Description:

This routine reads an AST Object from a component of the supplied HDS object. The component name is specified by the caller. The component must have a type of WCS, must be scalar, and must contain a single one-dimensional array component with name DATA and type _CHAR. AST__NULL is returned in IAST, and no error is reported if the named component does not exist.

Invocation:

```
CALL KPG1_WREAD( LOC, NAME, IAST, STATUS )
```

Arguments:

LOC = CHARACTER * (*) (Given)
The locator to the HDS object.

NAME = CHARACTER * (*) (Given)
The name of the component within the HDS object to read. If a blank name is supplied, the object itself is used.

IAST = INTEGER (Returned)
Pointer to the AST Object returned. Returned equal to AST__NULL if no Object can be read.

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

KPG1_WRLST
Writes a set of positions to a text file as a CAT catalogue

Description:

This routine saves a set of positions in a text file as a CAT catalogue (see SUN/181). Information describing associated co-ordinate Frames can also be stored in the file as textual information, allowing subsequent applications to interpret the positions. Files written with this routine can be read using KPG1_RDLST (and also XCATVIEW etc.).

The positions are stored in the file in a Frame selected by the user using hardwired parameters CATFRAME and CATEPOCH. This Frame defaults to a SKY Frame if present, otherwise a PIXEL Frame if present, otherwise the original Base Frame within the supplied FrameSet. The positions can be supplied within any of the Frames in the FrameSet and will be Mapped into the required Frame if necessary.

If the ID atttribute of the FrameSet is set to " FIXED_BASE" , then the user is not allowed to change the base Frame using parameters CATFRAME and CATEPOCH.

See also KPG1_WRTAB, which is like this routine but allows a textual label to be associated with each position.

Invocation:

```
CALL KPG1_WRLST( PARAM, ARRDIM, NPOS, NAX, POS, IFRM, IWCS, TITLE, IDO, IDENTS, NULL,
STATUS )
```

Arguments:

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

ARRDIM = INTEGER (Given)
The size of the first dimension of the positions array. This must be larger than or equal to NPOS.

NPOS = INTEGER (Given)
The number of positions to store in the file.

NAX = INTEGER (Given)
The number of axes in the Frame specified by IFRM.

POS(ARRDIM, NAX) = DOUBLE PRECISION (Given)
The positions to store in the file. POS(I, J) should give the axis J value for position I.

IFRM = INTEGER (Given)
The index of the Frame within IWCS to which the supplied positions relate. Can be AST__BASE or AST__CURRENT.

IWCS = INTEGER (Given)
A pointer to an AST FrameSet to store with the positions.

TITLE = CHARACTER * (*) (Given)
A title to store at the top of the text file. Ignored if blank.

ID0 = INTEGER (Given)
The integer identifier value to associate with the first supplied position. Identifiers for subsequent positions increase by 1 for each position. If this is supplied less than or equal to zero, then its value is ignored and the identifiers supplied in array IDENTS are used instead.

IDENTS(NPOS) = INTEGER (Given)
The individual integer identifiers to associate with each position. Only accessed if ID0 is less than or equal to zero.

NULL = LOGICAL (Given)
Is the user allowed to supply a null value? If so, the error status will be annulled before returning.

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

KPG1_WRTA2
Puts a set of positions into a text file as a CAT catalogue

Description:

This routine writes the supplied positions to a CAT catalogue (see SUN/181). A dump of the supplied FrameSet (if any) is included in the text file as a set of " text" lines. A column is created with name " PIDENT" to contain the integer identifiers. A column is also created for each axis of the Base Frame, with a name equal to the Symbol attribute of the Axis (AXIS_<n> is used if the Symbol is blank). The catalogue can be read using KPG1_RDLST (and also XCATVIEW etc.).

Invocation:

```
CALL KPG1_WRTA2( PARAM, ARRDIM, NPOS, NAX, POS, IWCS, TITLE, ID0, IDENTS, KEYMAP, LABS,
HIST, STATUS )
```

Arguments:

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

ARRDIM = INTEGER (Given)
The size of the first dimension of the positions array. This must be larger than or equal to NPOS.

NPOS = INTEGER (Given)
The number of positions to store in the file.

NAX = INTEGER (Given)
The number of axes for each position.

POS(ARRDIM, NAX) = DOUBLE PRECISION (Given)
The positions to store in the file. POS(I, J) should give the axis J value for position I. The positions should be in the Base Frame of the FrameSet supplied using argument IWCS.

IWCS = INTEGER (Given)
A pointer to an AST FrameSet to store with the positions.

TITLE = CHARACTER * (*) (Given)
A title to store at the top of the text file.

ID0 = INTEGER (Given)
The integer identifier value to associate with the first supplied position. Identifiers for subsequent positions increase by 1 for each position. If this is supplied less than or equal to zero, then its value is ignored and the identifiers supplied in array IDENTS are used instead.

IDENTS(NPOS) = INTEGER (Given)
The individual integer identifiers to associate with each position. Only accessed if ID0 is less than or equal to zero.

KEYMAP = INTEGER (Given)
An optional AST KeyMap containing data for extra columns to add to the catalogue. It can be used (for instance) to add character columns to the catalogue. If a value of AST__NULL is supplied, no extra columns are added to the catalogue. Otherwise, the column names and values can be specified using either of the following two schemes:

- If the KeyMap contains an entry called " COLNAMES" , it is assumed to be a vector entry holding the names of the columns. For each of these column names, the value to store in a particular row of the catalogue is assumed to be held in a scalar KeyMap entry with key " <colname>_<row number>" (" _1" for the first row). If no such entry exists for a particular row, then the value is marked as bad in the catalogue. The data type of the column is determined from the first row value found for the column.
- If the KeyMap does not contain an entry called " COLNAMES" , it is assumed that each entry in the KeyMap contains a vector holding all the values for a single column, in row order. The entry key is used as the column name, and the column data type is determined from the entry data type.

LABS = INTEGER (Given)
A GRP group identifier containing the labels to be associated with the positions. The number of elements in this group should be equal to NPOS. If GRP__NOID is supplied, no label column will be created.

HIST = INTEGER (Given)
A GRP group identifier containing history text to store with the catalogue. If GRP__NOID is supplied, no history information will be stored with the catalogue.

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

KPG1_WRTAB
Writes a set of positions to a text file as a CAT catalogue

Description:

This routine is equivalent to KPG1_WRLST, except that it provides the option of storing a textual label with each position, via the extra argument LABS.

Invocation:

```
CALL KPG1_WRTAB( PARAM, ARRDIM, NPOS, NAX, POS, IFRM, IWCS, TITLE, ID0, IDENTS, LABS,
HIST, NULL, STATUS )
```

Arguments:

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

ARRDIM = INTEGER (Given)
The size of the first dimension of the positions array. This must be larger than or equal to NPOS.

NPOS = INTEGER (Given)
The number of positions to store in the file.

NAX = INTEGER (Given)
The number of axes in the Frame specified by IFRM.

POS(ARRDIM, NAX) = DOUBLE PRECISION (Given)
The positions to store in the file. POS(I, J) should give the axis J value for position I.

IFRM = INTEGER (Given)
The index of the Frame within IWCS to which the supplied positions relate. Can be AST__BASE or AST__CURRENT.

IWCS = INTEGER (Given)
A pointer to an AST FrameSet to store with the positions.

TITLE = CHARACTER * (*) (Given)
A title to store at the top of the text file. Ignored if blank.

ID0 = INTEGER (Given)
The integer identifier value to associate with the first supplied position. Identifiers for subsequent positions increase by 1 for each position. If this is supplied less than or equal to zero, then its value is ignored and the identifiers supplied in array IDENTS are used instead.

IDENTS(NPOS) = INTEGER (Given)
The individual integer identifiers to associate with each position. Only accessed if ID0 is less than or equal to zero.

LABS = INTEGER (Given)
A GRP group identifier containing the labels to be associated with the positions. The number of elements in this group should be equal to NPOS. If GRP__NOID is supplied, no label column will be created.

HIST = INTEGER (Given)
A GRP group identifier containing history text to store with the catalogue. If GRP__NOID is supplied, no history information will be stored with the catalogue.

NULL = LOGICAL (Given)
Is the user allowed to supply a null value? If so, the error status will be annulled before returning.

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

KPG1_WTM3D
Forms the weighted median of a list of ordered data values. Incrementing the contributing pixel buffers and estimating the variance change

Description:

This routine finds a value which can be associated with the half- weight value. It sums all weights then finds a value for the half-weight. The comparison with the half-weight value proceeds in halves of the weights for each data point (half of the first weight, then the second half of the first weight and the first half of the second weight etc.) until the half weight is exceeded. The data values around this half weight position are then found and a linear interpolation of these values is the weighted median. This routine also uses the order statistic covariance array (for a population NENT big) to estimate the change in the variance from a optimal measurement from the given population, returning the adjusted variance.

Invocation:

```
CALL KPG1_WTM3D( ORDDAT, WEIGHT, VAR, NENT, COVAR, RESULT, RESVAR, STATUS )
```

Arguments:

ARR(NENT) = DOUBLE PRECISION (Given)
The list of ordered data for which the weighted median is required

WEIGHT(NENT) = DOUBLE PRECISION (Given)
The weights of the values.

VAR(NSET) = DOUBLE PRECISION (Given)
The variance of the unordered sample now ordered in ARR.

NENT = INTEGER (Given)
The number of entries in the data array.

COVAR($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size NENT.

RESULT = DOUBLE PRECISION (Returned)
The weighted median

RESVAR = DOUBLE PRECISION (Returned)
The variance of result.

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

Notes:

- the routine should only be used at real or better precisions.

Prior Requirements :

- The input data must be ordered increasing. No BAD values may be present.

KPG1_WTM3R
Forms the weighted median of a list of ordered data values

Description:

This routine finds a value which can be associated with the half- weight value. It sums all weights then finds a value for the half-weight. The comparison with the half-weight value proceeds in halves of the weights for each data point (half of the first weight, then the second half of the first weight and the first half of the second weight etc.) until the half weight is exceeded. The data values around this half weight position are then found and a linear interpolation of these values is the weighted median. This routine also uses the order statistic covariance array (for a population NENT big) to estimate the change in the variance from a optimal measurement from the given population, returning the adjusted variance.

Invocation:

```
CALL KPG1_WTM3R( ORDDAT, WEIGHT, VAR, NENT, COVAR, RESULT, RESVAR, STATUS )
```

Arguments:

ORDDAT(NENT) = REAL (Given)
The list of ordered data for which the weighted median is required

WEIGHT(NENT) = REAL (Given)
The weights of the values.

VAR(NSET) = REAL (Given)
The variance of the unordered sample now ordered in ARR.

NENT = INTEGER (Given)
The number of entries in the data array.

COVAR($*$) = DOUBLE PRECISION (Given)
The packed variance-covariance matrix of the order statistics from a normal distribution of size NENT.

RESULT = REAL (Returned)
The weighted median

RESVAR = REAL (Returned)
The variance of result.

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

Notes:

- The routine should only be used at real or better precisions.

Prior Requirements :

- The input data must be ordered increasing. No BAD values may be present.

KPG1_WWRT
Writes WCS information to an HDS object

Description:

This routine stores a supplied AST Object in a new component of a supplied HDS object. The new component has type WCS and contains a single component named DATA, of type _CHAR. DATA is a one-dimensional array holding lines of text which can be interpreted by a simple AST Channel.

Invocation:

```
CALL KPG1_WWRT( IAST, NAME, LOC, STATUS )
```

Arguments:

IAST = INTEGER (Given)
A pointer to an AST Object.

NAME = CHARACTER * (*) (Given)
The name of the WCS component to add into the supplied HDS object.

LOC = CHARACTER * (*) (Given)
A locator for an HDS structure object. This object is modified by adding a component of type WCS, with name given by NAME.

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

KPG1_XYD2W
Converts linear data co-ordinates to world co-ordinates

Description:

The co-efficients of the linear transformation from world co-ordinates to data co-ordinates are supplied in arguments SCALE and OFFSET. The inverse of this transformation is used to transform each supplied position from data to world co-ordinates.

Invocation:

```
CALL KPG1_XYD2W( SCALE, OFFSET, NPOINT, XP, YP, STATUS )
```

Arguments:

SCALE(2) = DOUBLE PRECISION (Given)
The scale factors in the linear relationships between axis co-ordinates and pixel co-ordinates.

OFFSET(2) = DOUBLE PRECISION (Given)
The offsets in the linear relationships between axis co-ordinates and pixel co-ordinates.

NPOINT = INTEGER (Given)
The number of points specified.

XP(NPOINT) = REAL (Given and Returned)
Array holding the x co-ordinate of each point.

YP(NPOINT) = REAL (Given and Returned)
Array holding the y co-ordinate of each point.

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

Notes:

- The supplied values of SCALE and OFFSET are such that:

DATA = SCALE(I) $*$ PIXEL $+$ OFFSET(I)

where PIXEL is a pixel co-ordinate for the I' th dimension, and DATA is the corresponding axis co-ordinate.

KPG1_XYZWx
Converts a two-dimensional array into a list of x-y co-ordinates, values and weights

Description:

This routine converts a two-dimensional array into a list of x-y co-ordinates, data values and weights for each of the good pixels in the array. The x-y co-ordinates come from the axis arrays supplied. Bad pixels are ignored if BAD is .TRUE.. If VARWTS is .TRUE., weights are calculated from the reciprocal of the variance for each pixel, otherwise the weights are returned 1.0.

Invocation:

```
CALL KPG1_XYZWx( DIM1, DIM2, ARRAY, XAXIS, YAXIS, BAD, VARWTS, VAR, SIZE, X, Y, Z, W,
NGOOD, XMIN, XMAX, YMIN, YMAX, STATUS )
```

Arguments:

DIM1 = INTEGER (Given)
The first dimension of the two-dimensional array.

DIM2 = INTEGER (Given)
The second dimension of the two-dimensional array.

ARRAY(DIM1, DIM2) = ? (Given)
The input data array.

XAXIS(DIM1) = DOUBLE PRECISION (Given)
X axis co-ordinates for the input data array.

YAXIS(DIM2) = DOUBLE PRECISION (Given)
Y axis co-ordinates for the input data array.

BAD = LOGICAL (Given)
Flag indicating whether bad values are likely to be present.

VARWTS = LOGICAL (Given)
Flag indicating if the variance array contains valid data.

VAR(DIM1, DIM2) = ? (Given)
An optional array containing the variance of the values in the input data array, used to generate a weight for each element. It is only used if VARWTS is .TRUE..

SIZE = INTEGER (Given)
The total number of pixels.

X(SIZE) = DOUBLE PRECISION (Returned)
The mean x positions for each pixel

Y(SIZE) = DOUBLE PRECISION (Returned)
The mean y positions for each pixel.

Z(SIZE) = DOUBLE PRECISION (Returned)
The data value for each pixel.

W(SIZE) = DOUBLE PRECISION (Returned)
The weight for each pixel.

NGOOD = INTEGER (Returned)
The number of good pixels.

XMIN = DOUBLE PRECISION (Returned)
Minimum x co-ordinate.

XMAX = DOUBLE PRECISION (Returned)
Maximum x co-ordinate.

YMIN = DOUBLE PRECISION (Returned)
Minimum y co-ordinate.

YMAX = DOUBLE PRECISION (Returned)
Maximum y co-ordinate.

STATUS = INTEGER (Given and Returned)
Global status value.

Notes:

- There is a routine for each numeric data type: replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The ARRAY and VAR arrays supplied to the routine must have the data type specified.
- Uses the magic-value method for bad or undefined pixels.

KPG_BLONx
Smooths an n-dimensional image using box filter

Description:

The routine smooths an n-dimensional array using an n-dimensional box filter; each pixel is replaced by the mean of those good neighbours which lie within a box of specified size.

Invocation:

```
CALL KPG_BLONx( BAD, SAMBAD, VAR, NDIM, DIMS, A, IBOX, NLIM, WDIM, B, BADOUT, ASUM,
NSUM, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether it is necessary to check for bad pixels in the input image.

SAMBAD = LOGICAL (Given)
If a .TRUE. value is given for this argument, then bad input pixels will be propagated to the output image unchanged (a smoothed output value will be calculated for all other pixels). If a .FALSE. value is given, then the NLIM argument determines whether an output pixel is good or bad. The value of SAMBAD is not relevant if BAD is .FALSE..

VAR = LOGICAL (Given)
If a .FALSE. value is given for this argument, then the smoothing applied will be appropriate to a data image. If a .TRUE. value is given, then the smoothing will be appropriate to an image containing variance values. In the latter case the output values will be (on average) smaller than the input values to take account of the variance-reducing effect which smoothing produces.

NDIM = INTEGER (Given)
The number of dimensions of the array to be smoothed.

DIMS(NDIM) = INTEGER (Given)
The dimensions of the array to be smoothed.

A($*$) = ? (Given)
Input array to be smoothed. Its dimensions are given by argument DIMS.

IBOX(NDIM) = INTEGER (Given)
Half-size of the smoothing box in pixels along eeach axis (the actual size of the ith axis' s box used will be 2$*$IBOX(i)+1 pixels).

NLIM = INTEGER (Given)
Minimum number of good pixels which must be present in the smoothing box in order to calculate a smoothed output pixel. If this minimum number is not satisfied, then a bad output pixel will result. A value between 1 and the total number of pixels in the smoothing box should be supplied.

WDIM = INTEGER (Given)
The dimension of the ASUM and NSUM workspaces. It must be at least 1 + DIMS(1) for a two-dimensional array , and at least 1 + (DIMS(1) $*$ ... (1 + DIMS(NDIM-1)) ...) for an NDIM-dimensional array. For example a 20x16x27-element array would need WLIM not fewer than 1+(20$*$(1+16)=341 elements.

B($*$) = ? (Returned)
The smoothed output array. It has the same dimensions as A.

BADOUT = LOGICAL (Returned)
Whether bad pixels are present in the output image.

ASUM(*) = ? (Returned)
Workspace for the pixel sums.

NSUM(*) = INTEGER (Returned)
Workspace for counting good pixels.

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

Notes:

- There is a routine for processing single- and double-precision arrays; replace " x" in the routine name by R or D as appropriate. The data type of the A, B and ASUM arguments must match the routine used.
- The routine uses recursion of the standard filter algorithm for each dimension moving from the highest to lowest. Recursion ends when there is only the integrated value and the corresponding number of contributing pixels sums used to find the output smoothed value for the current element of the input array. At each dimension the filter box is initialised with the required number of values for the box width, and then as the routine progresses through each position in the current dimension a new section is incorporated into the summations, while a section leaving the box is subtracted.

KPG_DIMLS
Writes the dimensions of an array in standard notation

Description:

This routine returns a character variable containing the dimensions of an array in the usual notation, that is the dimensions separated by commas and bounded by parentheses. There is an exception; if there is only one dimension, no parentheses and comma are included. This enables calling code to report the dimensions of an array using only one message call.

Invocation:

```
CALL KPG_DIMLS( NDIMS, DIMS, NCHDIM, DIMSTR, STATUS )
```

Arguments:

NDIMS = INTEGER (Given)
Number of dimensions of the array.

DIMS(NDIMS) = INTEGER (Given)
The dimensions of the array.

NCHDIM = INTEGER (Write)
The number of characters in DIMSTR, excluding trailing blanks.

DIMSTR = CHARACTER∗(∗) (Write)
A character string containing the dimension list enclosed in parentheses.

STATUS = INTEGER (Given)
Global status value.

KPG_ENV0C
Reads a string value from an environment variable, using a default value if the variable is undefined

Description:

This routine reads a string value of up to forty characters from a specified environment variable. No error occurs should the environment variable not be defined, and the supplied value is returned unchanged.

Invocation:

```
CALL KPG_ENV0C( VARNAM, CVAL, STATUS )
```

Arguments:

VARNAM = CHARACTER * (*) (Given)
The environment variable to check.

CVAL = CHARACTER * (*) (Given and Returned)
The string.

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

KPG_FISEx
Substitutes a constant value in a defined section of an array

Description:
This fills a rectangular section of a multi-dimensional array with a constant.

Invocation:
```
CALL KPG_FISEx( VALUE, NDIM, DIMS, LBND, UBND, ARRAY, STATUS )
```

Arguments:

VALUE = ? (Given)
Value to be substituted in every pixel within the defined section.

NDIM = INTEGER (Given)
The number of dimensions of the array, up to NDF__MXDIM (defined in NDF_PAR).

DIMS(NDIM) = INTEGER (Given)
Dimensions of the array to be edited.

LBND(NDIM) = INTEGER (Given)
Lower bounds of the section whose elements are assigned to VALUE. These are in the range 1 to DIMS(I) for the Ith dimension.

UBND(NDIM) = INTEGER (Given)
Upper bounds of the section whose elements are assigned to VALUE. These are in the range LBND(I) to DIMS(I) for the Ith dimension.

ARRAY($*$) = ? (Given and Returned)
The array to be edited.

STATUS = INTEGER (Given)
The global status.

Notes:

- There is a routine for each of the standard numeric types. Replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The data type of the arguments VALUE and ARRAY must match the particular routine used.

KPG_GTFTS
Obtains FITS header information from an NDF

Description:

The routine reads the FITS extension from an NDF and returns an AST pointer to a FitsChan which contains this information. The information may then be accessed using routines from the AST library (SUN/211).

Invocation:

```
CALL KPG_GTFTS( INDF, FCHAN, STATUS )
```

Arguments:

INDF = INTEGER (Given)
NDF identifier.

FCHAN = INTEGER (Returned)
An AST pointer to a FitsChan which contains information about the FITS headers associated with the NDF.

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

Notes:

- It is the caller's responsibility to annul the AST pointer issued by this routine (e.g. by calling AST_ANNUL) when it is no longer required.
- If this routine is called with STATUS set, then a value of AST__NULL will be returned for the FCHAN argument, although no further processing will occur. The same value will also be returned if the routine should fail for any reason.
- Status is set to KPG__NOFTS if no FITS extension is found.

KPG_IMMMx
Finds the mean, median and mode iteratively using the Chauvenet rejection criterion

Description:

This finds the mean, median, and mode (mmm) of a sample of data taken from a background region of an astronomical image of the sky. It should not be used as a general statistical routine as there are special features that are peculiar to the problem of contamination by stars. The routine also returns the standard deviation and the skew of the sample.

It makes a first estimate of the statistics, and then refines these by including or excluding values at either end of the sorted values, deciding which values to reject using the Chauvanet criterion. Iterations stop once no more values are rejected; or the statistics stabilise; or there is an oscillation between two states, in which case the returned statistics are the averages found in these two states; or there are too few values remaining.

Invocation:

```
CALL KPG_IMMMx( EL, ARRAY, MEAN, MEDIAN, MODE, SIGMA, SKEW, NSAMP, NINVAL, STATUS )
```

Arguments:

EL = INTEGER (Given)
The number of data in the sample.

ARRAY(EL) = REAL (Given)
Vector of data samples.

MEAN = REAL (Write)
The mean of the sample.

MEDIAN = REAL (Write)
The median of the sample.

MODE = REAL (Write)
The mode of the sample from the 3∗ median - 2 ∗ mean formula.

SIGMA = REAL (Write)
The standard deviation of the sample.

SKEW = REAL (Write)
The skewness of the sample from the formula MEAN - MODE / SIGMA.

NSAMP = INTEGER (Write)
The population of the sample after the iterative procedure.

NINVAL = INTEGER (Write)
The number of bad pixels in the sample.

STATUS = INTEGER(Given and Returned)
Global status value.

Notes:

- There is a routine for each of the standard numeric types. Replace "`x`" in the routine name by D, R, I, W, UW, B or UB as appropriate. The data type of the array being analysed must match the particular routine used.

- The supplied array need not be sorted.
- The minimum sample size throughout is five.
- The magic-value method is used for bad pixels.
- The initial statistics start from the median and a symmetric interval about that. Deviations from that median derive the initial mean and variance, and a new median.

In the iterations the new symmetric value range is about the mode, the half width being the Chauvenet number of standard deviations plus half the average separation of mean and mode. The median averages the central nine values to derive a more robust estimate.

KPG_ISEQN
Increments a sequence string

Description:

This routine takes an input string comprising two parts: an alphanumeric string ending in an non-numeric character; followed by a sequence number of digits, possibly with leading zeroes. The sequence number is incremented or decremented by a supplied number in a returned string of the same length as the input string. For example, u20100320_0084 would become u20100320_0085 for an increment of 1, and u20100320_0074 for an increment of -10.

An error results if the supplied string is not of the correct form, or the increment takes the sequence counter beyond its range. The sequence number cannot be negative or require additional characters. For example incrementing string DATA999 by one would be rejected as it would require an extra character.

Invocation:

```
CALL KPG_ISEQN( IN, INCREM, OUT, STATUS )
```

Arguments:

IN = CHARACTER * (*) (Given)
String containing a trailing sequence number.

INCREM = INTEGER (Given)
The increment to apply to string IN. It may be positive or negative. A value of 0 causes IN to be merely copied to OUT.

OUT = CHARACTER * (*) (Returned)
The supplied string but with its sequence number incremented by INCREM. It must have at least the length of IN.

STATUS = INTEGER (Given and Returned)
Global status value. An SAI__ERROR is returned, should any of the errors listed above be detected.

Notes:

The sequence number can have up to twelve digits.

KPG_LD2Ax
Converts a sparse form of a two-dimensional array into its two-dimensional counterpart

Description:

A list of double-precision x-y positions and values are converted to a complete two-dimensional array. Missing elements take the bad value.

Invocation:

```
CALL KPG_LD2Ax( NX, NY, SX, SY, NBIN, X, Y, Z, ARRAY, STATUS )
```

Arguments:

NX = INTEGER (Given)
The first dimension of the two-dimensional array.

NY = INTEGER (Given)
The second dimension of the two-dimensional array.

SX = REAL (Given)
The co-ordinate scale factor (i.e. the length of a pixel) in the x direction. It is used to determine which pixel a given x-y position lies. Normally, it will have the value 1.

SY = REAL (Given)
The co-ordinate scale factor (i.e. the length of a pixel) in the y direction. It is used to determine which pixel a given x-y position lies. Normally, it will have the value 1.

NBIN = INTEGER (Given)
The number of bins in the pixel list.

X(NBIN) = DOUBLE PRECISION (Given)
The x position of the pixel in the list.

Y(NBIN) = DOUBLE PRECISION (Given)
The y position of the pixel in the list.

Z(NBIN) = DOUBLE PRECISION (Given)
The value of the pixel in the list.

ARRAY(NX, NY) = ? (Returned)
The expanded two-dimensional array formed from the list.

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

Notes:

- There is a routine for each of the standard floating-point types. Replace "`x`" in the routine name by D or R as appropriate. The data type of the ARRAY argument must match the particular routine used.
- It uses eight-byte integers for array addressing.

KPG_LR2Ax
Converts a sparse form of a two-dimensional array into its two-dimensional counterpart

Description:

A list of single-precision x-y positions and values are converted to a complete two-dimensional array. Missing elements take the bad value.

Invocation:

```
CALL KPG_LR2Ax( NX, NY, SX, SY, NBIN, X, Y, Z, ARRAY, STATUS )
```

Arguments:

NX = INTEGER (Given)
The first dimension of the two-dimensional array.

NY = INTEGER (Given)
The second dimension of the two-dimensional array.

SX = REAL (Given)
The co-ordinate scale factor (i.e. the length of a pixel) in the x direction. It is used to determine which pixel a given x-y position lies. Normally, it will have the value 1.

SY = REAL (Given)
The co-ordinate scale factor (i.e. the length of a pixel) in the y direction. It is used to determine which pixel a given x-y position lies. Normally, it will have the value 1.

NBIN = INTEGER (Given)
The number of bins in the pixel list.

X(NBIN) = REAL (Given)
The x position of the pixel in the list.

Y(NBIN) = REAL (Given)
The y position of the pixel in the list.

Z(NBIN) = REAL (Given)
The value of the pixel in the list.

ARRAY(NX, NY) = ? (Returned)
The expanded two-dimensional array formed from the list.

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

Notes:

- There is a routine for each of the standard floating-point types. Replace " x" in the routine name by D or R as appropriate. The data type of the ARRAY argument must match the particular routine used.
- It uses eight-byte integers for array addressing.

KPG_NORVx
Returns a supplied value with normally distributed noise added

Description:

This routine takes as input a number and returns a value that is the input number plus or minus a random amount of normally distributed noise. It uses a Box-Mueller algorithm to generate a fairly good normal distribution.

Invocation:

```
CALL KPG_NORVx( BAD, EL, INARR, SIGMA, SEED, OUTARR, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether checks for bad pixels should be performed.

EL = INTEGER (Given)
Number of pixels in the array.

INARR(EL) = ? (Given)
Input array to which noise is to be added to each pixel.

SIGMA = ? (Given)
Standard deviation of the normal distribution.

SEED = REAL (Given & Returned)
Seed for the random-number generator SLA_RANDOM.

OUTARR(EL) = ? (Returned)
Output array which has random noise added.

STATUS = INTEGER (Given)
Global status value.

Notes:

- There is a routine for each of the standard floating-point types. Replace "`x`" in the routine name by D or R as appropriate. The data type of the INARR, SIGMA, and OUTARR arguments must match the particular routine used.

KPG_NXWRD
Finds the next word in a string

Description:

The routine looks for the start of the next word, after OFFSET characters, in the given string. Words are assumed to be delimetered by spaces, commas or tabs. The routine is really a wrap round calls to CHR_FIWE and CHR_FIWS trapping any status returns.

Invocation:

```
CALL KPG_NXWRD( STRING, OFFSET, FIRST, LAST, NOTFND, STATUS )
```

Arguments:

STRING = CHARACTER * (*) (Given)
The string to be searched for a 'word'. The word is looked for in STRING(OFFSET:).

OFFSET = INTEGER (Given)
The offset into the given string after which the word is to located.

FIRST = INTEGER (Returned)
First character of the located word. Offset into STRING.

LAST = INTEGER (Returned)
Last character of the located word. Offset into STRING.

NOTFND = LOGICAL (Returned)
If a next word is not located then this is set true, otherwise it is set false.

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

Notes:

- If an error has already occurred, or if an error occurs during this routine, NOTFND is returned .FALSE., and FIRST and LAST are returned equal to one.

KPG_OSTAx
Computes simple statistics for an array

Description:

This routine computes simple statistics for an array, namely: the number of valid pixels; the minimum and maximum pixel values (and their positions); the pixel sum; the mean; and the population standard deviation, skewness, and excess kurtosis. Iterative K-sigma clipping may also be optionally applied.

It uses a one-pass recursive algorithm for efficiency using the formulae of Terriberry (2007).

Invocation:

```
CALL KPG_OSTAx( BAD, EL, DATA, NCLIP, CLIP, ISTAT, DSTAT, ISTATC, DSTATC, STATUS )
```

Arguments:

BAD = LOGICAL (Given)
Whether checks for bad pixels should be performed on the array being analysed.

EL = INTEGER (Given)
Number of pixels in the array.

DATA(EL) = ? (Given)
Array to be analysed.

NCLIP = INTEGER (Given)
Number of K-sigma clipping iterations to apply (may be zero).

CLIP(NCLIP) = REAL (Given)
Array of clipping limits for successive iterations, expressed as standard deviations.

ISTAT(3) = INTEGER (Returned)
The integer statistics before clipping. The meanings of the elements in order are as follows.

- Number of valid pixels
- Index where the pixel with the lowest value was (first) found
- Index where the pixel with the highest value was (first) found

DSTAT(7) = DOUBLE PRECISION (Returned)
The floating-point statistics before clipping derived from the valid pixel values in DATA. The meanings of the elements in order are as follows.

- Minimum value
- Maximum value
- Sum
- Mean
- Population standard deviation
- Population skewness
- Population excess kurtosis. This is zero for a Gaussian.

ISTATC(3) = INTEGER (Returned)
The integer statistics after clipping derived from the valid pixel values in DATA. The attributions of the elements are the same as for argument ISTAT. If NCLIP is zero, the array will contain the same values as ISTAT.

DSTATC(7) = DOUBLE PRECISION (Returned)
The floating-point statistics after clipping derived from the valid pixel values in DATA. The attributions of the elements are the same as for argument DSTAT. If NCLIP is zero, the array will contain the same values as DSTAT.

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

Notes:

- There is a routine for each of the standard numeric types. Replace " x" in the routine name by D, R, I, W, UW, B or UB as appropriate. The data type of the array being analysed must match the particular routine used.
- If no clipping is performed (i.e. if NCLIP = 0) then the values of arguments which return results after clipping will be the same as for those returning results before clipping.
- If ISTAT(1) or ISTATC(1) is zero, then the values of all the derived statistics will be undefined and will be set to the " bad" value appropriate to their data type (except for the pixel sum, which will be zero).

References :

Terriberry, T.B., 2007, Computing Higher-order Moments Online, http://people.xiph.org/∼tterribe/notes/homs.html

KPG_PTFTS
Stores FITS header information into an NDF

Description:

The routine stores the contents of an AST FitsChan into an NDF by creating (or replacing) the FITS extension in the NDF.

Invocation:

```
CALL KPG_PTFTS( INDF, FCHAN, STATUS )
```

Arguments:

INDF = INTEGER (Given)

Identifier of NDF to receive the .FITS extension.

FCHAN = INTEGER (Given)

An AST pointer to a FitsChan which contains information about the FITS header to be associated with the NDF.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- If a .MORE.FITS extension already exists it will be completely replaced by this routine.

KPG_STOCx
Calculates accurate order statistics by sorting an array

Description:

This routine calculates the median and optionally up to one-hundred percentiles. It achieves this by using Quicksort to order the good array values, and hence provide correct values (unlike the faster histogram approximation used by KPG1_HSTAx).

A clipped range may be supplied, such as found by KPG_OSTAx, to derive ordered statistics after clipping of outliers.

Invocation:

```
CALL KPG_STOCx( EL, ARRAY, NGOOD, NUMPER, PERCNT, RANGE, MEDIAN, PERVAL, STATUS )
```

Arguments:

EL = INTEGER (Given)
Total number of pixels in the array.

ARRAY(EL) = ? (Given)
The vectorised array of values whose ordered statistics are to be calculated.

NGOOD = INTEGER (Given)
Number of valid pixels which contributed to the statistics.

NUMPER = INTEGER (Given)
Number of percentiles values to report. This should be in the range 1 to 100. Set this to 1 and PERCNT(1) to VAL__BADR if percentiles are not required.

PERCNT(NUMPER) = REAL (Given)
The percentiles to derive. Valid percentiles must be in the range 0.0 to 100.0, and preferably in ascending order. If the first element is set to the bad value, then no percentiles are calculated.

RANGE(2) = DOUBLE_PRECISION (Read)
The clipping limits between which to statistics are to be determined, lower then upper. A bad value means no limit is needed at its respective end.

MEDIAN(2) = DOUBLE PRECISION (Returned)
Median value. If there is an even number of good values present in the array, the median is the average of the middle pair. The second value is the clipped median after application of the RANGE bounds. If both RANGE values are bad no clipping is performed and the clipped median is set to the bad value.

PERVAL(NUMPER, 2) = DOUBLE PRECISION (Returned)
Percentile values corresponding to the percentile fractions in PERCNT. The second set of percentiles are the clipped values after application of the RANGE bounds. If both RANGE values are bad no clipping is performed and the clipped percentiles are set to the bad value.

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

Notes:

- There is a routine for byte, double-precision, integer, 64-bit integer, real, and word data types: replace " x" in the routine name by B, D, I, K, R, or W as appropriate. The data type of the ARRAY argument must match the particular routine used.

- If the value of NGOOD is not at least two, then this routine will abort. The median and percentiles will have the bad value.
- The sorting is recorded in an index leaving the order of the supplied array values intact.

KPG_STOSx
Calculates accurate order statistics by sorting an array

Description:

This routine calculates the median and optionally up to one-hundred percentiles. It achieves this by using Quicksort to order the good array values, and hence provide correct values (unlike the faster histogram approximation used by KPG1_HSTAx).

Invocation:

```
CALL KPG_STOSx( EL, ARRAY, NGOOD, NUMPER, PERCNT, MEDIAN, PERVAL, STATUS )
```

Arguments:

EL = INTEGER (Given)
Total number of pixels in the array.

ARRAY(EL) = ? (Given)
The vectorised array of values whose ordered statistics are to be calculated.

NGOOD = INTEGER (Given)
Number of valid pixels which contributed to the statistics.

NUMPER = INTEGER (Given)
Number of percentiles values to report. This should be in the range 1 to 100. Set this to 1 and PERCNT(1) to VAL__BADR if percentiles are not required.

PERCNT(NUMPER) = REAL (Given)
The percentiles to derive. Valid percentiles must be in the range 0.0 to 100.0, and preferably in ascending order. If the first element is set to the bad value, then no percentiles are calculated.

MEDIAN = DOUBLE PRECISION (Returned)
Median value. If there is an even number of good values present in the array, the median is the average of the middle pair.

PERVAL(NUMPER) = DOUBLE PRECISION (Returned)
Percentile values corresponding to the percentile fractions in PERCNT.

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

Notes:

- There is a routine for byte, double-precision, integer, 64-bit integer, real, and word data types: replace " x" in the routine name by B, D, I, K, R, or W as appropriate. The data type of the ARRAY argument must match the particular routine used.
- If the value of NGOOD is not at least two, then this routine will abort. The median and percentiles will have the bad value.
- The sorting is recorded in an index leaving the order of the supplied array values intact.

KPG_TYPSZ
Returns the number of bytes used to store an item of any of the HDS primitive data types

Description:

If the input TYPE is one of the HDS primitive numeric data types, i.e. one of _REAL, _DOUBLE, _INTEGER, _INT64, _WORD, _UWORD, _BYTE or _UBYTE, then the number of bytes used by that data type is returned as NBYTES. The values are those stored as the symbolic constants VAL__NBx in the PRM_PAR include file. (See SUN/39.) If the TYPE is not one of the above, an error results and STATUS is set.

Invocation:

```
CALL KPG_TYPSZ( TYPE, NBYTES, STATUS )
```

Arguments:

TYPE = CHARACTER * (*) (Given)
The HDS data type.

NBYTES = INTEGER (Returned)
The number of bytes used for the supplied data type.

STATUS = INTEGER (Given and Returned)
Global status. Bad status is returned should TYPE not be a valid HDS data type.

Q C-only Routine Descriptions

kpg1Axcpy
Copies an NDF AXIS structure from one NDF to another

Description:

This function copies all AXIS information describing axis " ax1" in " indf1" , into " indf2" axis " ax2" .

Invocation:

```
void kpg1Axcpy( int indf1, int indf2, int ax1, int ax2, int *status )
```

Arguments:

indf1
The source NDF.

indf2
The destination NDF.

ax1 The one-based index of the axis to be copied within " ndf1" . Must be less than or equal to the number of pixel axes in " ndf1" .

ax2 The one-based index of the new axis to create within " ndf2" . Must be less than or equal to the number of pixel axes in " ndf2" .

status
The inherited status.

kpg1Config
Creates an AST KeyMap holding a set of configuration parameter values

Description:

This function first creates a KeyMap by reading the values from a specified text file (argument " def"). This text files specifies the complete list of all allowed config parameters and their default values. The KeyMap is then locked by setting its MapLocked attribute to a true value. A GRP group is then obtained from the environment using the specified parameter (argument " param"). The config settings thus obtained are stored in the KeyMap. An error is reported if the user-supplied group refers to any config parameters that were not read earlier from the default file.

Both the defaults file and the user-supplied configuration may contain several sets of " alternate" configuration parameter values, and the set to use can be specified by the caller via the supplied " nested" keymap. For instance, if a basic configuration contains keys " par1" and " par2" , the user could supply two alternate sets of values for these keys by prepending each key name with the strings " 850" and " 450" (say). Thus, the user could supply values for any or all of: " par1" , " par2" , " 450.par1" , " 450.par2" , " 850.par1" , " 850.par2" . Note, values supplied without any prefix take priority over values supplied with a prefix.

The " nested" keymap supplied by the caller serves two functions: 1) it defines the known alternatives, and 2) it specifies which of the alternatives is to be used. Each key in the supplied " nested" keymap should be the name of an allowed alternative. In the above example, a KeyMap containing keys " 850" and " 450" could be supplied. No error is reported if the user-supplied configuration fails to provide values for one or more of the alternatives. The value associated with each key in the " nested" keymap should be an integer - the alternative to be used should have a non-zero value, and all other should be zero.

So first, the configuration parameters specified by the supplied defaults file are examined. Any parameter that starts with the name of the selected alternative (i.e. the key within " nested" that has an associated value of 1) has the name of the alternative removed (but only if no value has been supplied for the parameter without an alternative prefix - thus " filt=10" takes priority of " 450.filt=20"). Any parameter that starts with the name of any of the other alternatives is simply removed from the configuration. Thus, using the above example, if " nested" contains two entries - one with key " 850" and value " 1" , and the other with key " 450" and value " 0" - the " 850.par1" and " 850.par2" entries in the defaults file would be renamed as " par1" and " par2" (so long as no values already existed for " par1" and " par2"), and the " 450.par1" and " 450.par2" entries would be deleted.

Next, the same process is applied to the user-supplied configuration obtained via the specified environment parameter.

Finally, the values in the user-supplied configuration are used to replace those in the defaults file.

The benefits of using this function are that 1) the user gets to know if they mis-spell a config parameter name, and 2) the default parameter values can be defined in a single place, rather than hard-wiring them into application code at each place where the config parameter is used.

Invocation:

```
AstKeyMap *kpg1Config( const char *param, const char *def, AstKeyMap *nested, int null,
int *status )
```

Arguments:

param = const char * (Given)

The name of the environment parameter to use.

def = const char * (Given)
The path to a file containing the default value for every allowed config parameter. For instance, " $SMURF_DIR/dimmconfig.def" . May be NULL.

nested = AstKeyMap * (Given)
If non-NULL, used to determine which nested keys might be in the config and which should be merged with the base keymap. The values in the keymap should be non-zero to indicate merging.

null = int (Given)
Controls what happens if a null (!) value is obtained for the specified parameter. If " null" is non-zero, no error is reported and the returned KeyMap holds the values (if any) specified by " def" . If " null" is zero, an error is reported, status is set to PAR__NULL and a NULL KeyMap pointer is returned.

status = int * (Given & Returned)
The inherited status.

Returned Value:

A pointer to the AST KeyMap, or NULL if an error occurrs.

Notes:

- The KeyError attribute is set non-zero in the returned KeyMap so that an error will be reported by astMapGet<X> if the requested key does not exist in the KeyMap.

kpg1Kygp1
Creates a GRP group holding keyword/value pairs read from an AST KeyMap

Description:

This function is the inverse of kpg1Kymp1. It extracts the values from the supplied AST KeyMap and creates a set of " name=value" strings which it appends to a supplied group (or creates a new group). If the KeyMap contains nested KeyMaps, then the " name" associated with each primitive value stored in the returned group is a hierarchical list of component names separated by dots.

Invocation:

```
void kpg1Kygp1( AstKeyMap *keymap, Grp **igrp, const char *prefix, int *status )
```

Arguments:

keymap

A pointer to the KeyMap. Numerical entries which have bad values (VAL__BADI for integer entries or VAL__BADD for floating point entries) are not copied into the group.

igrp

A location at which is stored a pointer to the Grp structure to which the name=value strings are to be appended. A new group is created and a pointer to it is returned if the supplied Grp structure is not valid.

prefix

A string to append to the start of each key extracted from the supplied KeyMap. If NULL, no prefix is used.

status

Pointer to the inherited status value.

Notes:

- This function provides a private implementation for the public KPG1_KYGRP Fortran routine and kpg1Kygrp C function.
- Entries will be stored in the group in alphabetical order

kpg1Kymp1
Creates an AST KeyMap holding keyword/value pairs read from a GRP group

Description:

This function checks each non-comment, non-blank line in the supplied GRP group. An error is reported if any such lines do not have the form " keyword = value" , where the keyword name can be a hierarchical list of component names separated by dots. The returned KeyMap has an entry for each component name found at the start of any keyword name. The value associated with the entry will either be a primitive value (if the keyword name contained no other components) or another KeyMap (if the keyword name contained other components).

For example, consider a group containing the following lines:

gaussclumps.epsilon = (0.001,0.002) gaussclumps.contrast = 2.3 clumpfind.naxis = 2 clumpfind.deltat = 2.0 method = gaussclumps

The returned KeyMap will contain 3 entries with keys " gaussclumps" , " clumpfind" and " method" . The value associated with the " gaussclumps" entry will be another KeyMap containing keys " epsilon" (a primitive vector entry containing the values 0.001 and 0.002) and " contrast" (a primitive scalar entry with value " 2.3"). The value associated with the " clumpfind" entry will be another KeyMap containing keys " naxis" and " deltat" , which will have primitive scalar values " 2" and " 2.0" . The value associated with the " method" entry will be the primitive scalar value " gaussclumps" .

Assigning the value " <def>" (case insensitive) to a keyword has the effect of removing the keyword from the KeyMap. For example:

^global.lis method = <def>

reads keyword values from the file " global.lis" , and then ensures that the KeyMap does not contain a value for keyword " method" . The calling application should then usually use a default value for " method" .

Assigning the value " <undef>" (case insensitive) to a keyword has the effect of forcing the value to be undefined. This can be useful in defining defaults where the keymap is locked after being populated.

Invocation:

```
void kpg1Kymp1( const Grp *igrp, AstKeyMap **keymap, int *status )
```

Arguments:

igrp
A GRP identifier for the group of text strings to be analysed.

keymap
A location at which to return a pointer to the new KeyMap, or NULL if an error occurs. A valid pointer to an empty KeyMap will be returned if the supplied group contains nothing but comments and blank lines.

status
Pointer to the global status variable.

Notes:

- Vector elements should be separated by commas and enclosed within parentheses (commas and closing parentheses can be included literally in a vector element by preceeding them with a backslash).
- This function provides a private implementation for the public KPG1_KYMAP Fortran routine and kpg1Kymap C function.
- Component names must contain only alphanumerical characters, underscores, plus and minus signs [a-zA-Z0-9_+\-],
- Any lower case characters contained in a component name will be translated to the upper case equivalent.
- If the last non-blank character in a value is a backslash (" \"), the backslash will be reomved, together with any white space following it, and the entire next line will be appended to the value.

kpg1Kymp2
Parses a " keyword = value" string for kpg1Kymp1 and add to a KeyMap

Description:

This is a service function for kps1Kymp1. It parses the supplied " keyword = value" string into a keyword and value. It then parses the keyword into a list of dot-separated component names. It then adds the value into the supplied KeyMap at the correct point.

Invocation:

```
void kpg1Kymp2( const char *string, const char *ind, AstKeyMap *keymap, int *status
);
```

Arguments:

string
The null-terminated " keyword=value" string to be parsed.

ind The GRP indirection character - only used in error messages.

keymap
A pointer to the KeyMap in which to store the value.

status
Pointer to the global status variable.

Notes:

- If the value is a comma-separated list of values, enclosed in parentheses, then a vector entry will be added to the KeyMap. Otherwise, a scalar entry will be created.
- To include a comma or a closing parenthesis literally in a vector value, preceed it with a backslash.
- If the string has the form " keyword=<def>" (case insensitive), then any entry for the specified keyword is removed from the KeyMap.
- Component names must contain only alphanumerical characters, underscores, plus and minus signs [a-zA-Z0-9_+\-],
- Any lower case characters contained in a component name will be translated to the upper case equivalent.

kpgGetOutline
Retrieve an STC polygon describing the spatial extent of an NDF

Description:

If The NDF contains an OUTLINE extension, it is expected to be a character array copntaining an STC-S description of a polygon. If this is the case, the polygon is returned as the function value. Otherwise a NULL pointer is returned.

Invocation:

```
AstRegion *kpgGetOutline( int indf, int *status )
```

Arguments:

indf = int (Given)

Identifier for the NDF.

status = int * (Given and Returned)

Pointer to global status.

kpgGtfts
Obtains FITS header information from an NDF

Description:

The routine reads the FITS extension from an NDF and returns an AST pointer to a FitsChan which contains this information. The information may then be accessed using routines from the AST library (SUN/211).

Invocation:

```
kpgGtfts( int indf, AstFitsChan ** fchan, int * status );
```

Arguments:

indf = int (Given)
NDF identifier.

fchan = AstFitsChan ** (Returned)
An AST pointer to a FitsChan which contains information about the FITS headers associated with the NDF.

status = int * (Given and Returned)
The global status.

Notes:

- It is the caller's responsibility to annul the AST pointer issued by this routine (e.g. by calling AST_ANNUL) when it is no longer required.
- If this routine is called with STATUS set, then a value of AST__NULL will be returned for the FCHAN argument, although no further processing will occur. The same value will also be returned if the routine should fail for any reason.
- Status is set to KPG__NOFTS if no FITS extension is found.

Return Value :

Returns the status.

kpgPtfts
Stores FITS header information into an NDF

Description:

The routine stores the contents of an AST FitsChan into an NDF by creating (or replacing) the FITS extension in the NDF.

Invocation:

```
kpgPtfts( int indf, AstFitsChan * fchan, int * status );
```

Arguments:

indf = int (Given)

Identifier of NDF to receive the .FITS extension.

fchan = const AstFitsChan * (Given)

An AST pointer to a FitsChan which contains information about the FITS header to be associated with the NDF.

status = int * (Given and Returned)

The global status.

Notes:

- If a .MORE.FITS extension already exists it will be completely replaced by this routine.

Return Value :

Returns the status.

kpgPutOutline
Create and store an STC polygon describing the spatial extent of an NDF

Description:

If the NDF is 3D it is collapsed onto the spatial pixel axes (which must be pixel axes 1 and 2), and all pixels with less than the specified fraction of good values in each pixel are set bad. An AST Polygon is then created describing the good values in the collapsed array. This polygon is mapped into the SKY Frame and converted to an STC-S description, which is stored in an extension called " OUTLINE" in the supplied NDF (the extension is a character array).

Invocation:

```
void kpgPutOutline( int indf, float wlim, int convex, int *status )
```

Arguments:

indf = int (Given)

Identifier for the NDF. Must be 2-D or 3-D. The WCS must contain a SkyFrame, and the first two pixel axes must map onto the two sky axes.

wlim = float (Given)

Minimum fraction of good values per pixel required when collapsing a 3D NDF. Ignored if the NDF has only two significant pixel axes.

convex = int (Given)

Indicates the nature of the polygon. If zero, the polygon will be a simple outline that hugs the edges of the good pixels. If the NDF contains multiple dis-contiguous regions of good pixels, the outline will be of a randomly selected contiguous clump of good pixels. If " convex" is non-zero, the polygon will be the shortest polygon that encloses all good pixels in the NDF. Such a polygon will not in general hug the edges of the good pixels.

status = int * (Given and Returned)

Pointer to global status.